Robert B. Gramacy

Monty the Null Hippopotamus

Exploring Statistical Foundations Through Simulation

Best fam ever

Leah, Natalia, Kaspar

Contents

Preface

You're probably wondering about the book title. Here's how I like to stylize it: Monty the $\mathcal{H}\text{ipp}_0$. Why? I don't want to spoil it before we even get started. Check out Chapter 1. In Chapter 2 and beyond I use the word hippopotamus exactly once, in lieu of a different word. See if you can find it. Treat it like a game of "Where's Waldo?"[1]

My hope with this book is to provide an unconventional approach to a first course in statistics that resonates with an audience who grew up in the age of computing. It aims to be friendly with new concepts, while not talking down to the uninitiated. You can be the judge as to whether or not I've succeeded in that.

Many professionals, when speaking to me in passing, report that "Stats 101" was their least favorite/hardest class they took toward their degree. I think that's because stats is taught in a boring, old-fashioned way that focuses on memorization, and carrying out complex and inscrutable formulas that lack intuition. Folks remember working on calculators or spreadsheets, and combing through tables of quantiles and percentiles squirreled away in textbook appendices. Maybe those things made sense at one time. Now they're out of touch with how people experience data analytics in a modern context, and a disservice to an important and interesting area of scientific inquiry.

My target audience is second-year college students who've already taken a class on probability, and who are comfortable with calculus. It's alright if you're not a probability expert, but it will help a lot to be comfortable with core concepts like mass, density, distribution, expectation, conditioning, independence, etc. Derivative calculus is essential, and integral calculus is helpful. You must have some experience coding. The book is written to help you learn R. Supplementary Python is also provided. You need not be an expert in either, but some familiarity will help. If you've never coded in a language like one of those (e.g., MATLAB ®, C, C++, Julia, Rust), then you'll have a hard time with the material here. I hope that this book is also a good fit for master's-level students who came from another, non-statistical undergraduate program.

This isn't the kind of book you can skim for formulas. It must be read to understand what you're meant to learn/do, and the context behind why. Like acquiring any real skill, you'll need practice and time for reflection to get good at it. Again, *this book is intended to be read*. That seems like a silly thing to say about a book, but I think it's important here. Many students treat textbooks as reference manuals. But this one is not intended as a reference manual. My focus is on concepts not recipes. You can get a lot from the code examples, but if you don't read the words around the code, and around the math, you'll miss out. You'll miss out not just on real understanding, but you also won't get the homework completely right and you'll probably lose points. This book is a few hundred pages. If you can't be bothered to read twenty pages a week in a fifteen-week college semester, then you probably aren't ready for college.

At the risk of beating a dead horse, I've got two more paragraphs on this. Although this

[1] https://en.wikipedia.org/wiki/Where's_Wally?

isn't a reference text, about 75% of the homework questions can be done by simply cutting-and-pasting code from earlier in the chapter with minor modifications – a few characters here and there. The trick is to know what to change/adapt. Students will need help recognizing this, especially if they're not reading the book, which means not reading this Preface either. A common mistake is to use Chapter 4 methods (approximation) on Chapter 3 questions (exact). This happens because students prefer AI-assisted web search over reading the chapter that the questions are in.[2] It sometimes happens as early as Chapter 1 because students think they know how to do it already, like from an earlier stats class, but they're not noticing that what I'm teaching – and thus encouraging them to practice – is actually a new skill. When I use this material at Virginia Tech[3] (VT), students lose lots of points on early homework sets despite warnings. It takes until Chapter 5 before I start getting taken seriously, and I've yet to figure out how to short-circuit that journey.

Presuming that students have prior experience with statistical analysis and probability lets me move a little faster, but it presents a hazard for understanding. I want to show you/your students a new way of doing things, and if you don't pay attention you'll likely miss out, and end up circling back later. Try to un-know what you think you know, especially if it falls outside of the scope of the presentation. Keep your mind open. There's no harm in doing outside research. However, if you're using something found outside of the text, with the exception of pointers I've made explicitly via links, then you're definitely not doing it how I intended, and you may be doing it wrong. Each homework section has a preamble where I outline how to approach the problems. Usually it's a long-winded way of saying "use methods from the current chapter, not from somewhere else; only use libraries and other sources to check your work".

I make many references to other materials that can be found on the internet, either via footnotes (printed versions) or hyperlinks (HTML version). Many of those are Wikipedia links, which is a wonderful resource. You are encouraged to click on them, but mostly they're there so you know you *can* find details, and more precise definitions, somewhere. The idea to is add color and context without derailing the message. Only visit those links if you're curious. Clicking on them is *not* required to follow the development, which is intended to be self-contained. This is a book about ideas, not trivia. There are many good sources, including other textbooks, and my linking to Wikipedia is expedient and exemplary. I always encourage students to track things down if they're curious. I hope those links are a good place to start.

My narrative favors on process over precision. I'm a little defensive about this, and apologize for it multiple times in the text. There will be times where I need to make seemingly arbitrary adjustments so that a calculation matches output from a software library. I'll be the first to admit when I don't know why it works that way. There may be a good reason for it, or it may simply be convention. I arrived at some of the adjustments that I show you by peeking under the hood at the R implementation. Sometimes those codes aren't particularly well documented and your guess is as good as mine. Often a good guess can be made forensically. Sometimes it's best to move on.

You might think writing a textbook in such a cavalier way is a cop-out. But I genuinely believe it's not worth investing neurons on minutia. I realize that where I draw the line on that is arbitrary and may not be the same place you would. I can be overly compulsive about stuff that I can't pin down, believe me. I tell myself that edge cases are inevitable,

[2] Although this may change over time, not much of what I'm teaching in this book is AI-searchable at the time of writing.

[3] https://www.vt.edu/

and usually you can't address all of them and keep your sanity. Precision can be the enemy of understanding. I want my readers to get a feeling for how statistics works. Sometimes you can draw comfort from seeing how numbers line up with some benchmark. Yet in many cases that benchmark is an approximation, and often the sense in which that approximation is good (or eventually converges asymptotically) is either unrealistic, or is based on assumptions that are unverifiable. I hope to impress upon you that my preferred approach – more on that in Chapter 1 – suffers fewer limitations because it doesn't cut corners. Getting an accurate answer might require a lot (more) computer work. But it's not your work; the transistors are doing it for you once the code is written.

I'm not going to list the subjects that are covered in this book. That's what the Contents is for. I do want to say something about broad themes. There are five modules, as it were, though I don't make a show of that with any special headings. The first two chapters are a warm-up. The idea is to give you a feel for how I like to think about statistical inference. The next two add mathematical rigor to, and expand upon, ideas from the first two chapters. The third module branches out into more complex methodology, filling out a "classical" parametric approach to basic applied statistics, all the way through to simple linear regression (SLR). Then I transition to nonparametric (non-P) methods. This is unconventional for an introductory statistics text, but I think that my unique approach (still not giving that away here) makes non-P accessible. Non-P methods are fascinating and beautiful, both conceptually and in practice. Many of the topics from the first half of the book are revisited here, through a new lens. This helps reinforce understanding, while broadening the toolkit. The book concludes with a final chapter on "fancy" regression methods, including transformations and multiple linear regression. I hope this is a "treat", to whet your appetite for a second course in statistics.

There are many subjects that are not covered here, but very well could be. Perhaps the most important omission is power. I allude to the power of statistical tests from time to time, and point to Appendix A.2 for a glimpse at details. I wont deny the importance of power analysis, but I don't teach it in any of my introductory classes on statistics. My feeling is that it's too in the weeds for a first pass. Before you have a solid grasp of the fundamentals involved – what comprises a statistical test, operationally speaking – it doesn't make much sense to discuss which tests are "better" than others in terms of power.

Every subject herein is paired with illustration in worked code. There's not a single table or figure (which is not a drawing) in the book that's not supported by code on the page. Everything is fully reproducible via `bookdown` (Xie, 2018a) on CRAN, combining `knitr` (Xie, 2015, 2018b) and `rmarkdown` (Allaire et al., 2018) packages. What you see is what you get (WYSIWYG). Many examples and some data sets are based on pseudo-random numbers. This means that when you run the code on your machine, you won't get the same thing that I print in the book. Any time random numbers are involved, I encourage you to run the example multiple times to see how answers may vary from one random "instance" to another. It's impossible to fully remove randomness from the experience of engaging with this book's material. That invariably results in hedged verbiage, at times, on my end.

Each chapter ends with homework exercises that have been vetted over three semesters of teaching at VT. Some of those questions are mathematical in nature, but most are practical implementation exercises and data analysis. Before jumping into any data analysis, I encourage students to code up their own "library version" of the procedures required, which usually means tests and confidence intervals. Like the examples that accompany every illustration in the main body of the chapters, data for homeworks come from a variety of sources. Some of them are totally fabricated; others come from repositories like the Data

and Story Library (DASL)[4]. Some have been passed down from colleagues of mine over the years; others are borrowed from the academic literature. Some I have found on my own, or pulled from other resources on the web. Whenever possible, links to origins and citations to research papers are provided.

Although my presentation exclusively favors R, supplementary Python code is provided on the book webpage[5]. Unfortunately, many of the library procedures that are showcased for comparison in R are not available for Python. All of the bespoke code – that means the code that I wrote from scratch for this book – is realtively easy to translate. Many thanks to Anya Raval[6], a talented student at VT, for the Python code files linked from the book webpage. I hope it would be similarly straightforward, if tedious, to port things over to other languages like or MATLAB or Julia. Most of the code in this book involves simple `for` loops, random number generation and vectorized aggregation (say via `mean` or `var`). All of the homework questions can be solved in any language, but it could be handy to have R to check your work. R is an easy language to learn if you already have familiarity with Python, MATLAB or Julia. Let me encourage you to use this book as a tutorial and, at the end, add another language to your résumé.

Code readability, reliability and reproducibility of output are at least as important as efficiency. R code in this book is peppered with ample commentary, both as coded `## comments` and as prose around the code. Partly to ensure that the code is human-digestible; I don't make use of Tidyverse[7]. Everything is ordinary R. Tidyverse is a very important part of the R ecosystem, but its target audience is data wranglers. This is not a data-wrangling book; it is an algorithms and procedures text. I want anyone with coding experience, not only R/`tidy` experts, to be able to follow the examples. I also aimed for the bare minimum of dependencies, in terms of other R packages. I wanted most of the code to be easy to translate to other languages, and to be executable without an internet connection (to install packages, etc.). In part for this reason, I use basic R plotting as opposed to `ggplot` (Wickham, 2016). Colleagues sometimes tell me that `ggplot` makes prettier plots than base R does. Aesthetics are a matter of opinion. For what I need in this book, base R plotting is sufficient and provides nice-looking graphics with simple code.

I'm ready to get started, are you? I have a few folks to thank before jumping in. I am grateful to the VT Department of Statistics[8] and the Computational Modeling and Data Analytics (CMDA)[9] program at VT for allowing me to teach CMDA 2006 on multiple occasions. The material from this book came primarily from that course, but also from a nonparametric statistics class for VT/Stats, and from classes that I taught while at the University of Chicago[10]. The first version of CMDA 2006 was during the Covid-19 pandemic, which turned out to be a blessing for this material/book. Online delivery meant that I had to be careful about the presentation. I needed a record of all of code and bookwork, which made it easier to re-purpose for this project. I'm not saying I'm grateful to a virus, but it is what it is. I'm saying this book is lemonade.

I am grateful for my family. They've been patient with me, right from the moment their eyes bulged out of their skulls when I said I was thinking I'd write another book. Fortunately, this one was easier than the first one. Practice and experience makes everything easier and

[4] https://dasl.datadescription.com/
[5] https://bobby.gramacy.com/hipp0
[6] https://www.linkedin.com/in/anya-raval
[7] https://www.tidyverse.org/
[8] https://stat.vt.edu
[9] https://data.science.vt.edu/programs/cmda.html
[10] https://www.uchicago.edu

more efficient. Most important of all, though, is a comfortable base of operations, and that's what family is for. Mine's the mostest bestest. Mama and kiddos: I hope that I help you be your best the way you do for me.

Robert B. Gramacy
Blacksburg, VA

1

Toss up: coin flips

The simplest kind of statistical experiment involves flipping coins to detect if the coin is fair: equally likely to turn up heads or tails. Many statistics texts start here. Although my goal is to upend statistical pedagogy with this book, I'll start here too. One reason, besides convention, is that my preferred, simulation-based approach, is simple to describe and execute in this setting. It makes for a nice warm-up. We can do the entire analysis without any fancy math. Now, connecting to what other textbooks and software libraries provide *does* require math, and we won't shy away from that in this book. But I'll kick that can down the road a little bit to later chapters, beginning in Chapter 3.

It's amazing what you can do with a computer and a simple R program. This is true for coin flipping experiments, as described here, but also for much more involved scenarios. Convincing you of that is one of the goals of this book. Giving you the tools and intuition required to do it, and connecting you to results and nomenclature involved in a more conventional analysis, is another. If you've taken a university-level course in statistics (or perhaps a high-school level advanced placement class), then you may have heard the term "null hypothesis". This concept is very important to statistical inquiry. Perhaps because it helps make a connection to the scientific method[1] where the formulation and (usually) rejection of hypotheses play a central role. Statistical methods are often involved in the advance of science.

Yet students find hypothetical formalism off-putting. Consequently, I don't like it, especially not as pertains to an early introduction of an otherwise digestible subject for someone who is quantitatively inclined. Like you, I hope. So I poke fun at it with my title, "null hippopotamus" (or `hipp0` for short). I prefer to instead ponder, "what if's". So that's how we'll start.

I'll bring some of the formalism in too, so that connections can be made to convention. I don't want to hang you out to dry when it comes to communicating with others, and understanding analysis in the literature. Throughout, my goal is to keep it simple. That doesn't mean there won't be rigor. Yet the focus will be more squarely on statistical ideas, and on the intuition that they provide through data. I'm no pedant for scientific vocabulary.

Alright, enough disclaimer. Suppose your buddy flipped a coin thirty times, and this was the outcome, where H means "heads" and T means "tails".

```
data <- c("H", "T", "H", "H", "H", "T", "T", "H", "T", "T", "H", "H", "H",
  "T", "T", "H", "H", "H", "H", "T", "H", "H", "H", "T", "H", "H", "T",
  "H", "T", "T")
```

And now you want to know: is the coin fair? You're not allowed to inspect the coin, flip the

[1] https://en.wikipedia.org/wiki/Scientific_method

coin more times, nothing. All you've got from your buddy is the outcome of those thirty flips. Those are your data. Your observations.

1.1 Simulation

But that doesn't mean you can't bring something of your own to the table. You could always flip a different coin – one you know to be fair – and compare what you get to the data. I know what you're thinking. Why would I do that when I already know what the outcome would be? I'd get about the same number of heads as tails, right? Wrong. That's the thing about chance. Anything could happen. You could flip that fair coin thirty times and get all heads, or all tails, or you could alternate heads, tails, etc., fifteen times, and all other things in between. Some such events[2] are rare. A run of all heads has probability $1/2^{30}$, which is about one in 10 billion. But, if everyone in the world did it, then chances are one person would get "lucky". (At the time of writing, there are 8 billion souls on Earth.)

This isn't a book on probability. I'll assume you have some experience, and quickly review relevant concepts as necessary, often pointing to Wikipedia[3], like I did above for several key vocabulary words. This book, and perhaps you might say the enterprise of statistical inquiry more generally, is about combining probability with data to draw conclusions about how things (might) work. It's a deep subject, but many of its tenets are simple, and many basic recipes are both intuitive and work well.

Ok, so suppose we flip our own coin thirty times. Let's not actually do that with a real coin, because that's super monotonous. Instead, let's get our computer to do it for us. There are a lot of ways to do this. For example, each flip could go like this:

```
u <- runif(1)   ## generate single pseudo-random number uniformly in [0,1]
f <- "T"
if(u >= 0.5) f <- "H"
f
```

```
## [1] "T"
```

The outcome is random and may be different when you try it on your own. It certainly will if you do it over and over again, and I encourage you to do that. I'm not going to explain how pseudo-random number generators[4] work, but I will note that most use a uniform $[0, 1]$ generator[5], or $u \sim \mathcal{U}[0, 1]$, as a building block, or subroutine, just like I did above. Notice how I converted $u \geq 1/2$ into "heads", and otherwise "tails".

A more streamlined way to do that in R would be to use the binomial generator, `rbinom`.

```
c("T", "H")[rbinom(1, 1, 0.5) + 1]
```

```
## [1] "H"
```

[2] https://en.wikipedia.org/wiki/Event_(probability_theory)
[3] https://en.wikipedia.org/wiki/Scientific_method
[4] https://en.wikipedia.org/wiki/Pseudorandom_number_generator
[5] https://en.wikipedia.org/wiki/Continuous_uniform_distribution

Again, the outcome is random and I encourage you to repeat that (many times) and see what you get. The output of `rbinom(1, 1, 0.5)` is either 0 or 1. You can read up on it with `?rbinom` in R, but here are the highlights that are relevant for now. A call to `rbinom(n, x, p)` will "flip a coin" `x` times with probability `(1 - p, p)` of `(0, 1)`, respectively, record the sum of those flips, and repeat that `n` times and report all the sums. Under the hood, it uses `runif`. You can download the source code from https://cran.r-project.org if you're curious and see how it all works. So if you provide `n=1` and `x=1`, it's basically the same as a coin flip, but with outcome of 0 or 1 rather than heads or tails. My code above makes the conversion from binary[6] to characters "T" or "H".

Quick digression on terminology. Flipping (weighted) coins is known formally as sampling from a Bernoulli distribution[7]. We'll talk in more detail about that later. I capitalized Bernoulli[8] because it's named after a Swiss mathematician. A binomial distribution[9] is the sum of repeated Bernoulli trials, or flips. I just wanted to tell you that so you could understand where the function name `rbinom` comes from. We'll come back to the binomial distribution later too.

Alright, back to flipping coins. Adjusting the `n` argument to `rbinom` gets you more flips of a single coin.

```
sim <- c("T", "H")[rbinom(30, 1, 0.5) + 1]
sim
```

```
##  [1] "H" "T" "T" "H" "H" "H" "H" "T" "T" "T" "T" "H" "T" "T" "T" "H"
## [17] "H" "H" "H" "H" "T" "T" "T" "T" "T" "H" "T" "H" "T" "T"
```

You get a similar outcome by holding fixed `n=1` and adjusting `x=30`, but this is a little more complex since it requires a vector `p` argument. Perhaps tinker with that on your own if you're curious.

Ok, now we have thirty flips that come from a fair coin and thirty that come from our buddy who may or may not have used a fair coin. What do we do? It makes sense to somehow compare them to one another, but there are lots of ways we could do that. I'm sure you have a way in mind – probably the same as mine – but just to convince you there are others, I'll list a few silly ones.

1. Look at each pair of first flips and see if they're the same.
2. Look at each pair of second flips, third, etc.
3. Measure the length of the longest "run" of consecutive tails (or heads) for each.
4. Measure the length of the second longest "run" for each.
5. Measure the length of the longest alternating "T", then "H" run for each.

And so on like that. These are pretty bonkers, admittedly, but none of them is a wrong choice. Some of them are quantitatively inferior to others, in a way that will take a little while to explain in more detail.

For now, it stands to reason that if you only look at the first flip, or the second flip, then you're not being very efficient about things because you're not using all of the collected information. That inefficiency will likely lead to the wrong conclusion just by chance. Moreover, if you only pair your first flip with your buddy's first flip, or second with second, then you're

[6] https://en.wikipedia.org/wiki/Binary_number
[7] https://en.wikipedia.org/wiki/Bernoulli_distribution
[8] https://en.wikipedia.org/wiki/Daniel_Bernoulli
[9] https://en.wikipedia.org/wiki/Binomial_distribution

artificially imposing an order to otherwise probabilistically independent[10] events. That's not ideal either.

A sensible choice, and likely the one you're thinking of, is to count the number of heads (or equivalently tails) separately for both sets of thirty flips. In code …

```
Hd <- sum(data == "H")
Hs <- sum(sim == "H")
c(buddy=Hd, sim=Hs)
```

```
## buddy   sim
##    18    13
```

Any numerical summary of data, like the sensible one above but also the silly ones earlier, is called a statistic[11]. Intuitively, the best statistics summarize the data with the fewest quantities possible without losing any relevant information. I'll make this more specific later. For now, counting the number of heads is pretty good on this account since knowing that number, along with the total number of flips (30), tells you just about everything from the experiment. You can deduce from that the number of tails (30 minus the count of heads). With those two numbers, you know about the outcome of the entire experiment, except the order in which the flips came in. But if we regard flips as independent, order doesn't really matter. None of the other silly statistics above let you do that.

We actually have two statistics. One summarizes the observational data (`Hd`), where some aspect of the data-generating mechanism is unknown to us/beyond our control, and about which we desire to make an inference. (Is the coin fair?) The other comes from a simulation (`Hs`) where we know the "coin" was fair. Both are subject to randomness: the data from the chaotic dynamics of coin flipping, and the simulation from pseudo-random numbers. These are not the same thing, so we acknowledge that by saying that the simulated data are from, or a realization of, a model for actual coin flips. We could have chosen a more elaborate model, perhaps one based on the physics of coin tossing (Stefan and Cheche, 2016), but that's for a different book.

Keeping it simple for now, let's presume our simulation is a decent stand-in for the actual data-generating mechanism. No model is perfect, but this one seems good enough to be useful.[12] Our simulated and observed statistics are not equal, and together they tell a strange story. We expect[13] fifteen heads in a sample of thirty when the coin is fair. When we knew the coin was fair, in our simulation, we got 13 (less than that). The data, where we don't know if the coin was fair, gave us 18 (more than that). What gives? How is any of this helpful?

We now have a sense of the inherent randomness in the problem: the spread of what *could* happen when flipping thirty coins. To take things to the next level, we must flip even more (sets of thirty) coins. I know, strange right? If doing it once is perplexing, how could doing it even more times make things better, not worse? Hold that thought for a moment. Before I show you what to do, and how it works and why, I'll need to invest in a little notation. What I'm about to show you works for pretty much everything, not just flipping coins. In a way, the entire book is here in the first chapter. The rest of the book is just practice appropriating the idea for other scenarios.

[10] https://en.wikipedia.org/wiki/Independence_(probability_theory)
[11] https://en.wikipedia.org/wiki/Statistic
[12] https://en.wikipedia.org/wiki/All_models_are_wrong
[13] https://en.wikipedia.org/wiki/Expected_value

1.2 Notation and nomenclature

You might not like it, but it's helpful here to make a brief digression and invest in some mathematical notation. This will help make things concrete and generic at the same time. That in a nutshell, I think, encapsulates the investment in mathematical reasoning. It begins by making a simple thing, like coin flips, harder and possibly inscrutable to the uninitiated. But then you gain some familiarity through experience and value emerges. The additional sophistication that thoughtful mathematical notation and reasoning provides not only makes things look more elegant, but also allows us to be much more specific about the calculations involved. Don't worry, there isn't much math required for the time being.

The letter I prefer for random experimental outcomes (i.e., observations) – and preferred in most statistical circles, whether that be mainstream stats, machine learning and artificial intelligence, or data science more broadly – is y. We capitalize Y to refer to a random, as yet unknown outcome (like for a coin flip). We would say that Y is a random variable[14], which is explicitly or implicitly endowed with some probabilistic distribution[15] that prescribes the chances that it could take on particular values, once observed.

Quick digression on Y. You might say, "we used X in my other stats class." That may be, and X is good too, but Y is better because of regression (Chapters 7, 12 and 13). Take my word for it. If you're tempted to substitute X everywhere I use Y, please don't. You'll regret it. Suspend disbelief and force yourself to learn a new way. You'll be smarter for it. X will come back later.

Alright, back to random variables. Particular values random Y could take on are notated in lowercase, like y. Lowercase is also used to refer to values already observed (say, in data). For example, we could refer to $\mathbb{P}(Y = y)$, which is the probability of observing that random variable Y takes on value y. We can refer to the outcome of a random event or experiment that hasn't happened yet as Y, or one that has already (and we've seen the outcome) as y.

In the case of flipping coins, the convention is to notate $y = 1$ for heads and $y = 0$ for tails. That way, we can introduce a parameter θ that controls the rate of heads and describe the distribution as follows:

$$\mathbb{P}(Y = y) = f(y; \theta) = \theta^y (1 - \theta)^{1-y}, \quad \theta \in [0, 1], \tag{1.1}$$

where $f(y; \theta)$ could generically stand in for any probability density (pdf)[16] or mass function (pmf)[17], but above a Bernoulli mass is used for binary data. Check that under this setup $f(1; \theta) = \theta$ and $f(0; \theta) = 1 - \theta$, which means that the probability of heads is θ and one minus that for tails, as it should. In our experiment we wondered if the coin is fair, and that scenario corresponds to $\theta = 0.5$.

Quick digression on lettering. If you follow the link to the Bernoulli distribution on Wikipedia[18], you'll see that the letter p is used instead of θ. This is non-standard in most statistical circles where Greek letters are preferred for parameters. Roman letters are for data quantities. Parameters are unknown quantities that must be estimated from data. So I

[14] https://en.wikipedia.org/wiki/Random_variable
[15] https://en.wikipedia.org/wiki/Probability_distribution
[16] https://en.wikipedia.org/wiki/Probability_density_function
[17] https://en.wikipedia.org/wiki/Probability_mass_function
[18] https://en.wikipedia.org/wiki/Bernoulli_distribution

think of parameters as ideals we'll aspire to learn but never achieve exactly. We shall never ascend to enlightenment (think lofty Greek gods and philosophers), but with data we can dial in estimates and move things along from a practical perspective (think practical Roman imperialism, though they also had their gods). I will follow this throughout in the book.

Alright, back to coins. Mass $f(y;\theta)$ is for just one coin. What about thirty coins? The letter that's usually used in statistics for a number of experimental trials, or number of data points, is n. There are $n = 30$ in our coin flipping experiment. This is a good point to convert our code from §1.1 to use this new notation. I always find it helpful when object names, which are called symbols[19] in computer programming vernacular, match up with math (the names of the variables) as much as possible.

```
y <- as.numeric(data == "H")
n <- length(y)
Ys <- as.numeric(sim == "H")
```

Operator `==` creates what in R is called a logical vector, composed of `TRUE` and `FALSE` values. I encourage you to try that part separately, before `as.numeric` which coverts Boolean[20] values to zeros and ones.

Note that `y` and `Ys` are `n`-length vectors, not scalars. They hold observed data values $y_1, \ldots, y_n$ and simulated versions, respectively. Instead of `Y` for simulated values, as an analog of random variable Y, I prefer `Ys`. There are a number of reasons. One is that simulated `Ys` holds not unknown, random, values but actual ones I generated. Another is that I like to remind myself that they came from a simulation. So the name I prefer for the symbol is a compromise between the notion it represents and the manner in which the values were generated. A final thing worth mentioning is that the code above for `Ys` is basically yielding what `rbinom(30, 1, 0.5)` does, when we first generated `sim` above, before mapping those values to `"H"` and `"T"`.

For experiments like this, it's common to make the assumption that experimental units – each coin flip – are independent of one another and that they follow the same probabilistic (Bernoulli) distribution. In other words, they are independent and identically distributed[21], abbreviated as iid (usually written in lowercase). The first "i" in iid, for independence, means that the joint probability[22] of all n observations (flips) factorizes as the product of the probability of the individual ones. Recall that, by definition, two events A and B are independent[23] if

$$\mathbb{P}(A, B) = \mathbb{P}(A)\mathbb{P}(B).$$

That may be extended, by repeated application, to all n events ($n = 30$ coin flips) as follows.

$$\mathbb{P}(Y_1 = y_1, \ldots, Y_n = y_n) \overset{\text{iid}}{=} \prod_{i=1}^{n} \mathbb{P}(Y_i = y_i) \tag{1.2}$$

If they are also identically distributed, the second "id" in iid, then $\mathbb{P}(Y_i = y_i) = f(y_i;\theta)$. In other words, each Y_i has the same mass or density function f, and with identical parameter

[19] https://en.wikipedia.org/wiki/Symbol_(programming)
[20] https://en.wikipedia.org/wiki/Boolean_data_type
[21] https://en.wikipedia.org/wiki/Independent_and_identically_distributed_random_variables
[22] https://en.wikipedia.org/wiki/Joint_probability_distribution
[23] https://en.wikipedia.org/wiki/Independence_(probability_theory)

θ, for all $i = 1, \ldots, n$. In the case of a Bernoulli (1.1), it would be common to write the statistical model out in shorthand as follows.

$$\text{Model:} \qquad Y_i \overset{\text{iid}}{\sim} \text{Bern}(\theta), \quad i = 1, \ldots, n$$

Check in this setup that

$$\mathbb{P}(Y_1 = y_1, \ldots, Y_n = y_n) = \theta^{\sum y_i}(1-\theta)^{n-\sum y_i}, \tag{1.3}$$

where $\sum y_i \equiv \sum_{i=1}^{n} y_i$. Dropping the bounds is a common shorthand especially when sums reside in powers.

Notice that $\sum y_i$, which is the same as the number of heads in our n flips, is an important statistic. We came across this quantity by intuition earlier, and now it has some mathematical justification. I see this as a bonus. There's nothing wrong with intuition as the basis for deciding on something. It also doesn't hurt to have mathematical reasoning to point to, especially if someone doubts your intuitive chops. We'll revisit this in much more detail later. In general, a statistic[24] $s_n = s(y_1, \ldots, y_n)$ is some function that summarizes your data. Often it distills n quantities down to one scalar, but not always.

Now let's circle back to the specific coin flipping problem, collect thoughts, and codify what we have and what we want to know in mathematical terms. We have $n = 30$ flips, summarized by the count of the number of heads $s_{30} = 18$. Again, endowing our code with the new symbols …

```
s <- sum(y)            ## observed statistic
Ss <- sum(Ys)          ## simulated statistic
c(s=s, Ss=Ss, n=n)
```

```
##  s Ss  n
## 18 13 30
```

Nothing has really changed; these are just new names for things in code to match the math.

Now, I want to know if the coin is fair ($\theta = 0.5$) or not ($\theta \neq 0.5$). The statistical, mathematical way to describe this "want to know" is through the following formal hypotheses:

$$\begin{aligned} \mathcal{H}_0 &: \theta = \tfrac{1}{2} && \text{null hypothesis ("what if")} \\ \mathcal{H}_1 &: \theta \neq \tfrac{1}{2} && \text{alternative hypothesis.} \end{aligned} \tag{1.4}$$

The way that we use evidence, which comes from observed data via our statistic s_n, is by making a contradiction argument[25]. Assume the opposite of what the data is telling you, that the null hypothesis $\mathcal{H}_0$ is true even though you didn't get what you expected, and see where that leads compared to quantities actually calculated from observed data. Or, you can think about it as "innocent until proven guilty"[26]. The simpler, null hypothesis corresponds to the accused being innocent of a crime. It is up to a prosecutor to convince a jury that something more complicated, more nefarious, is in play.

It's hard to carry two imperfect metaphors through to fruition at once, so bear with me.

[24] https://en.wikipedia.org/wiki/Statistic
[25] https://en.wikipedia.org/wiki/Proof_by_contradiction
[26] https://en.wikipedia.org/wiki/Presumption_of_innocence

Here's what we've got. For fair coin flips we might expect $s_n = \sum_{i=1}^{n} y_i = n/2$, or 15 for 30 flips.[27] So $\mathcal{H}_0$ is the "what if". And if it turns out this null hypothesis is false, you should be able to propagate that, logically speaking, to some blatant mathematical contradiction. Carrying this out, and coming to a conclusion, is what is meant by a statistical hypothesis test[28].

It's worth remarking that it's certainly possible that we *could* get $s_n = 15$ for any θ, fair coin or no. The opposite is also true: even with $\theta = 0.5$ we *could* observe $s_n \neq 15$, and we did when we simulated coin flips in §1.1. But usually the null hypothesis/"what if" is a simplifying choice of convenience – something favoring parsimony[29] – rather than something we generally believe to be true.

For example, having every/any minted coin be exactly fair is preposterous. Surely it must be that each coin slightly favors heads or tails in the long run, due to manufacturing irregularities. Yet as long as quality control at the mint is relatively high, we probably wouldn't have the patience to flip the coin enough times to tell. So a sensible baseline "what if" is that the coin is fair, versus the alternative that it is not. For an actually (or close to) fair coin a prosecutor will need a lot of evidence – a lot of flips – to tell. But for our friend's particular coin, which maybe was purchased at a magic shoppe, a good lawyer may be able to build a case with fewer.

Algorithm 1.1 "Want to know what if" (or, testing statistical hypotheses)

Assume you have some **data** $y_1, \ldots, y_n$ and a statistical **model** caricaturing the data generating mechanism.

Require complementary null ($\mathcal{H}_0$) and alternative ($\mathcal{H}_1$) **hypotheses** encoding what you want to know, and a **statistic** $s_n = s(y_1, \ldots, y_n)$ that you can calculate from your data.

Then

1. Pretend $\mathcal{H}_0$ is true (your "what if") and under those conditions (i.e., with your model), calculate or derive the sampling distribution of $S_n = s(Y_1, \ldots, Y_n)$.
2. Compare the observed statistic s_n to the distribution of S_n.
   - If s_n looks typical under S_n, then $\mathcal{H}_0$ is probably reasonable ("fail to reject $\mathcal{H}_0$");
   - otherwise, "reject" $\mathcal{H}_0$ in favor of the alternative H_1.

Report your conclusions/evidence.

Enough mixing of metaphors. The actual procedure is outlined in Alg. 1.1. It's framed generically, because it has nothing to do with coin flips. The idea is to have something to circle back to later, and over and over again. Notice what is "assumed" and "required", which are both things that the practitioner must provide, in a sense. You need some data, obviously, and you pair that with a probabilistic model for how the data is generated. That model will likely have some (unknown) parameters, like θ, but may not always.

Then, the practitioner must specify what he or she "wants to know" in terms of competing

[27] Technically, I should be using uppercase S_n and $Y_1, \ldots, Y_n$ because I am entertaining a hypothetical, with a random variable as opposed to actual data values. But for now, I'm not doing that because I don't want to involve probabilistic expectations. At least not yet; hold your horses for Chapter 3.

[28] https://en.wikipedia.org/wiki/Statistical_hypothesis_test

[29] https://en.wikipedia.org/wiki/Occam's_razor

hypotheses. These are usually opposites of one another. The null one ($\mathcal{H}_0$) is simpler or more constrained, in some sense, than the alternative ($\mathcal{H}_1$) which represents the unconstrained version. Often these have to do with a parameter θ, but again not always. For example, Eq. (1.4) contrasts $\theta = 0.5$ with $\theta \neq 0.5$, and these are complements of one another.

Finally, the practitioner needs to choose a statistic, $s_n = s(y_1, \ldots, y_n)$ which provides the wedge that (potentially) forms evidence for an alternative against the null. Like in a trial by jury, a prosecutor must present specific evidence that distills a complex situation (lots of data) down to a few things s_n (DNA from blood on a knife) to focus scrutiny. Whoops, I used a metaphor again. Sometimes s_n is referred to as a test statistic[30] because it is a statistic that is used to conduct a hypothesis test.

The precise relationship between null and alternative hypotheses, and what it means about the procedure and the interpretation of its outcome, is subtle and we'll get into more detail on that later. The real stars of the show, for now and in great generality, are the observed data, model, null hypothesis $\mathcal{H}_0$ and statistic s_n. These are in bold in Alg. 1.1; you must have these four ingredients before you can get started.

The null describes how observed data *could* be generated under the chosen probabilistic model, comprising our "what if" in Step 1. Those choices can be parlayed into a distribution over *possible* values of the statistic S_n. This is known as the sampling distribution[31] of the statistic. It is also known as the null distribution[32], making a direct link to $\mathcal{H}_0$ involved in a statistical test. All statistical tests involve comparing the observed value of the statistic, s_n, calculated on the actual data, to the distribution of possible values S_n under the null.

S_n under $\mathcal{H}_0$ (with the model) is a random variable, which is why it is capitalized. $Y_1, \ldots, Y_n$ values that could arise in the "what if" scenario that $\mathcal{H}_0$ is true yields $S_n = s(Y_1, \ldots, Y_n)$. A function $s(\cdot)$ of random variables Y is itself a random variable S. Note that I'm using subscripts a little differently for Y and y than for S and s. For Y-values the subscript indicates the sample record number, or index into a vector. For S, which is often a scalar but almost always of smaller dimension than the sample size n, subscript n is intended to remind you of how many data points the statistic is calculated from.

In old fashioned statistics, you would use math – sometimes a lot of fancy advanced math that's way over my head – to derive the form of the sampling distribution of S_n under $\mathcal{H}_0$. We'll see some of that later. When the math is doable, I'll do it. When it's not, I'll show you where to look. Perhaps one of the most important deliverables of this book, however, is that you don't need fancy math if you have a computer.

Before we transition to that, let me address the hippopotamus in the room. What? I think that the use of the term "hypothesis" in statistics is awkward. We don't really believe that any of this stuff is "true", null or alternative. I already said that we don't believe that coin flips come from a Bernoulli distribution. Because they don't. So it's a bit harebrained to hypothesize that they do. Yet the Bernoulli is a useful simplifying assumption, because it makes things concrete probabilistically. And anyways, all models are wrong.

I already said I think we use the word hypothesis in statistics in order to connect with tenets of the scientific method[33]. That's a noble enterprise. Also, scientists seem to have a

[30] https://en.wikipedia.org/wiki/Test_statistic
[31] https://en.wikipedia.org/wiki/Sampling_distribution
[32] https://en.wikipedia.org/wiki/Null_distribution
[33] https://en.wikipedia.org/wiki/Scientific_method

fetish for Greek words, like algorithm and logarithm, which (sorry!) there will be a lot of in this book.[34]

This book is about statistical evidence and procedures like Alg. 1.1 that try to adjudicate between one option or another for how observational data we have in hand could have arisen. For that, I prefer the simpler, less snooty, plain English "want to know" or "what if". I want to know: could data generated in a certain way have given rise to statistics that are similar to what I observed from actual data? The way I do that is to see "what (happens) if" synthetic data are generated that way, and then make a comparison. That comparison can happen via the sampling distribution, and that's what we'll do next.

So if, like me, you don't like the word hypothesis, just substitute it with hippopotamus. If you're not advanced enough to do the hard math that's sometimes needed to derive the sampling distribution, I'll show how you can do it by Monte Carlo sampling from $\mathcal{H}_0$. In other words, "Monty the null hippopotamus". That's easy to remember, and easy to do.

1.3 Monte Carlo

Here I focus on Steps 1–2 of Alg. 1.1 by returning to the simulation idea from §1.1, but doing it a lot! In other words, we build up an understanding of the sampling distribution of a statistic under the null hypothesis by sampling from that distribution. That sounds vacuous, and therefore potentially meaningless, but perhaps that's because as a notion it's so simple. And after you see what I mean, because we're going to do it momentarily, (I hope) you'll think "oh yeah, that's obvious and easy to do" on a computer. And, in the case of flipping coins, you could even do it without a computer but it would be a bit tedious.

People used to do stuff like this without a computer. To understand the chances that certain random occurrences would/could happen, they would repeatedly try them over and over again and keep a tally of the number of times they got various outcomes. Nowhere is this more widely understood, and perhaps more important to operations, than with games of chance that involve money. I mean gambling in casinos. Those casinos want to make sure that they make money – to make sure the house wins on aggregate – so they invest a lot of resources into understanding the odds. Then they try to pair those odds with payouts that seem lucrative for the client, to entice them to play, but also ensure profitability not only in the long run, but also in the worst case.

One of the most prominent modern-day gambling spots, and home to some of the oldest casinos that are still in operation today, is Monte Carlo, Monaco[35]. Monte Carlo is a strange and fascinating place, and I encourage you to read more about it. There are casinos there that have been open since the 19th century. Back in the day, they didn't have computers to help visualize things and make it cheap to explore odds. So they would play the games of chance over and over again and keep tabs. That's what we're going to do here, but on a computer.

Although we'll use Monte Carlo (MC) sampling primarily for statistical purposes, the Monte Carlo method[36] has broad applicability in mathematical calculations with computers, most

[34] Actually, logarithm was coined by a Scottish mathematician, and algorithm is named after a Persian one.

[35] https://en.wikipedia.org/wiki/Monte_Carlo

[36] https://en.wikipedia.org/wiki/Monte_Carlo_method

notably integration[37]. In fact, there is a sense that what we're doing here is basically an integral[38] – I mean from calculus – but we'll talk more about that later. For now, let's just see how it works for entertaining "what if's".

Alright, head back to §1.1 where we simulated $n = 30$ coin flips and compared those to observed data. We converted those to y- and s-values in §1.2. And now we're going do that lots more times, comprising a MC simulation experiment. Without further ado, the code below gathers $N = 10,000$ samples of the test statistic under $\mathcal{H}_0$ by repeatedly flipping fair coins $Y_1, \ldots, Y_n$ and calculating S_n for those values by summing them up.

```
N <- 10000                          ## number of MC "trials"
Ss <- rep(NA, N)                    ## storage for sampled test statistics
for(i in 1:N) {                     ## each MC trial
  Ys <- rbinom(n, 1, 0.5)           ## generate n (=30) virtual coin flips
  Ss[i] <- sum(Ys)                  ## calculate the test statistic
}
```

Notice how each sample of the statistic in the vector `Ss` is saved, but not all of the $N \times n$ coin flips. Once we've summed them up, we don't need the flips anymore.

Ok, now what to do with that? There are many ways to inspect samples from a distribution. My favorite generally, and the best for our purposes here, is a histogram[39]. Figure 1.1 provides that visual, overlayed with some additional information that I shall discuss momentarily. A histogram provides one way of visually inspecting empirical distribution[40], which basically means that its form depends on all of the numerical values in the data (in this case samples of S_n), rather than some summary of them or a (possibly estimated) parameter. So you could say that the histogram of Figure 1.1 comprises an empirical sampling distribution.[41] More on empirical distributions, if you're curious, is in §A.1. But I'm hoping you can get the gist with experience.

```
hist(Ss, main="")
abline(v=c(s, n-s), lty=1:2, col=2, lwd=2)
legend("topright", c("s", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
```

The vertical red lines in the figure are particularly important as they represent different views on the value of the statistic s_n calculated from observed coin flip data. The position of the solid one is exactly at s_n on the x-axis. For me, its position indicates that the value of s_n we observed from actual data is not atypical under the sampling distribution for S_n derived from the null hypothesis.

Flipping a fair coin $n = 30$ times gives rise to $s_{30} = 18$ heads regularly, and even more (extreme) counts. That's not a rare occurrence at all! Therefore, I can't "reject" the hypothetical "what if" that the actual observational data $y_1, \ldots, y_n$ could have been generated from conditions described by the assumed (Bernoulli model) data generating mechanism under $\mathcal{H}_0$. I must be willing to accept that the null hypothesis may be true.

[37] https://en.wikipedia.org/wiki/Monte_Carlo_integration
[38] https://en.wikipedia.org/wiki/Integral
[39] https://en.wikipedia.org/wiki/Histogram
[40] https://en.wikipedia.org/wiki/Empirical_distribution_function
[41] A histogram technically provides a mass, sometimes approximating a density, not a distribution which is the sum or integral of a mass or density.

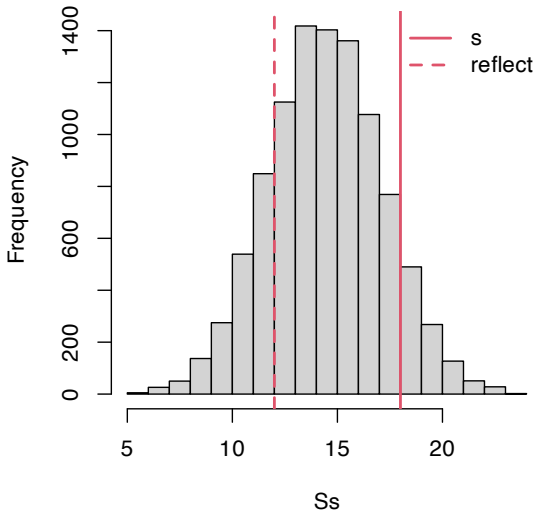

FIGURE 1.1: Histogram of samples S_n from the sampling distribution under $\mathcal{H}_0$, and the actual s_n (and it's "two-tailed" reflection) for comparison, for the coin flipping experiment.

Quick digression on terminology here. Alg. 1.1 says that, in this case, I should "fail to reject $\mathcal{H}_0$", but I said "willing to accept". Many instructors would take off points for such blasphemy. Not me. You can say whatever you want in my book. I sure will. You can read many opinions online, like this one[42], asserting that acceptance "implies that the null hypothesis is by nature true, and it is proved". Poppycock! That's not how the word "accept" is used in the English language. I don't know how people used it in the 19th century, when stats was in its early days. Today people "accept" an idea when they don't want to quibble. They want the conversation to keep moving. It's a matter of politeness, not conviction or gospel truth. I'm not going to quibble with you. I'm cool if you accept when you mean you weren't able to reject, given the evidence in the data.

Remember, this modeling framework is all a convenient cartoon. Nothing in real life boils down to coin flips, not even coin flips. It's a simplifying abstraction. There is no true, mathematical specification of the data generating mechanism. The null is never "by nature" anything, let alone true or false. Acknowledging that, we might choose to reject every $\mathcal{H}_0$ without even looking at the data. That's not helpful. So when we fail to reject in our simplified, cartoon land, I'm ok with accepting however you want to tell me that.

Alright, back off my high horse. That's basically it. Yet, anything important bears repeating. We sampled a statistic via synthetic data from the null distribution. If the actual observed version of the statistic looks typical (as it does in Figure 1.1) then we cannot reject the notion that $\mathcal{H}_0$ is true. If, rather, it looks rare or extreme, then we reject $\mathcal{H}_0$ in favor of the alternative $\mathcal{H}_1$, about which I shall say more momentarily.

Statisticians don't like to rely purely on visuals, because "looking" lacks precision, and is beholden to human judgment which can be whimsical. They'd rather have hard figures. You can probably imagine many reasonable ways of quantifying the sentiment that an observed s_n "looks typical or not" under the null distribution for S_n. Statisticians prefer to do that by calculating the probability of observing a more extreme s_n under $\mathcal{H}_0$, where "more

[42] https://www.jove.com/science-education/v/14102/hypothesis-accept-or-fail-to-reject

extreme" means toward the nearest tail (in the direction away from the mean) of the null distribution.

Such a quantity is easy to calculate with samples. We have $N = 10{,}000$ of them. Some are out in that tail (bigger than s_n), and some aren't. That proportion (how many bigger over how many total) is approximating a probability that statisticians call a p-value[43]. As $N \to \infty$ that proportion becomes exact for the p-value.

```
pval <- mean(Ss >= s)   ## same as sum(Ss >= s)/N, or (count/total)
pval
```

```
## [1] 0.1735
```

Something that happens 17% of the time isn't rare at all. For historical reasons that are ultimately arbitrary and not of much interest to me (and thus to this book), statisticians have converged on a significance level[44] of 5% as the threshold delimiting rare from typical. If the p-value is not less than 5%, then we can't reject the null. I shall stick with that in this book and not give it much more thought. This value is sometimes written as $\alpha = 0.05$, suggesting one could choose another α-value if they wanted to.

This α controls what is known as Type I error[45], which is pretty inscrutable as terminology goes. Basically, α controls the probability that you incorrectly reject $\mathcal{H}_0$. If you estimate $p \approx \alpha$, then you'll be wrong $(100 \times \alpha)\%$ of the time. This presumes that all the other assumptions you've made, like about the statistical model generating the data, are true. (And we know they're not.)

I wanted you to be aware of α and Type I error, but this book isn't really about those details. Now that I've dispensed with them, I won't bring them up again. I promise. But statisticians do another funny thing that I do need to tell you about, because it addresses an issue I brushed off earlier in §1.2 and promised to come back to: the alternative hypothesis $\mathcal{H}_1$. I'll preface by saying that I find some of the logic here pretty unsatisfying, and easy to take issue with. Yet it's not totally without sense and forms the basis of another important convention. Conventions are good because they let you get on with things by leveling the playing field. So we've gotta talk about it.

Our alternative hypothesis (1.4) was $\mathcal{H}_1 : \theta \neq 0.5$, which means that the coin could be biased in either direction: more heads with $\theta > 0.5$ or more tails with $\theta < 0.5$. Our p-value calculation only focused on the "greater" side of the sampling distribution when evaluating the chances of seeing a more extreme statistic under $\mathcal{H}_0$. Looking only in the right tail corresponds to one-tailed or one-sided test[46].

Being biased either way ($\theta \neq 0.5$) allows for more wiggle room to accept the null – less likely to reject it – because the alternative is less specific. We saw more heads than would be expected under the null, but it just as easily could have been more tails. Performing a similar calculation on the other side of the sampling distribution provides the other tail. This is the meaning of the dashed vertical red line in Figure 1.1, which has been "reflected" into the left tail. Its value corresponds to `n - s` heads or `s` tails.

Combining one-tailed p-values, by adding together that of the original statistic and its

[43] https://en.wikipedia.org/wiki/P-value
[44] https://en.wikipedia.org/wiki/Statistical_significance
[45] https://en.wikipedia.org/wiki/Type_I_and_type_II_errors
[46] https://en.wikipedia.org/wiki/One-_and_two-tailed_tests

reflection, yields a two-tailed test appropriate for an alternative hypothesis like $\mathcal{H}_1 : \theta \neq 0.5$. The reflected one is hypothetical, representing the other tail.

```
pvals <- c(right=mean(Ss >= s), left=mean(Ss <= n - s),
       both=mean(Ss >= s) + mean(Ss <= n - s))
pvals
```

```
##  right   left   both
## 0.1735 0.1881 0.3616
```

Note that the two-tailed p-value above is very similar, but not identical to, two times the one-tailed calculation. Sometimes you'll see this described in textbooks, or on Wikipedia[47].

```
2*mean(Ss >= s)
```

```
## [1] 0.347
```

This does not give the same p-value because our sampling distribution is discrete, asymmetric, and composed of (way) fewer than N unique values.

```
length(unique(Ss))
```

```
## [1] 20
```

There are only so many unique sums of n coin flips – at most n – but some, like with very few heads or tails, are highly improbable. In many situations, the two calculations agree (as $N \to \infty$), although the notion of a "reflection" is more nuanced, as we shall see later. This is just something to be aware of for now. Either way you calculate it, the two-sided p-value is larger than the one-sided one, so we still cannot reject $\mathcal{H}_0$.

Many statistical texts make a big deal about one- and two-sided tests, especially in homework exercises. A fair bit of the challenge in those problems involves sussing out which tail should be scrutinized. (A hint for working on problems like that: if your one-tailed p-value is bigger than $\frac{1}{2}$, then you're probably working in the wrong tail.) I'm not going to give you those exercises here. This is not that kind of textbook. I want you to internalize the idea behind tests, and how you can do them in many settings through MC simulation. Figuring out whether you want a one- or two-tailed test is not really a statistical topic in my opinion.

All tests in this book will be two-sided by default. You can usually work out the one-sided version by eliminating the reflection. There are many tests, however, where there is no two-tailed test that makes sense. We'll talk more about that when we get to it.

1.4 Wrapping up

Believe it or not, that's it! Now you know how to do a hypothesis test for coin flips, or any binary/Boolean data from a Bernoulli process. You'll have a chance to get some practice with homework problems in §1.5.

[47] https://en.wikipedia.org/wiki/P-value

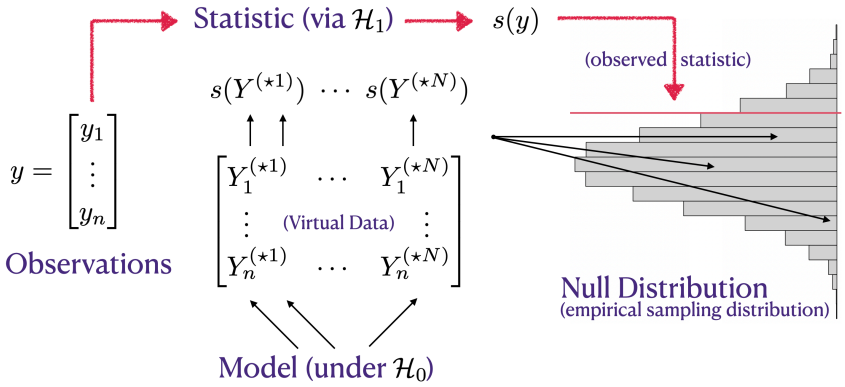

FIGURE 1.2: Hypothesis testing as a flow chart, capturing the relationship between key ingredients – data (observations), model, hypotheses, statistic – and the sampling distribution, represented via histogram.

I made a little diagram that I hope will help augment the formal description in Alg. 1.1. See Figure 1.2. The diagram borrows a histogram representation of the (empirical) sampling distribution from the coin-flipping example, turned on its side in order to limit crossing of lines. So the diagram favors a MC adaptation of Alg. 1.1, but otherwise the flow is the same. The story behind the diagram, viewing it from left to right, goes like this.

You have some data y and something you want to know about it. You must choose a probabilistic model for that data (e.g., coin flips with probability θ of heads), and frame your "want to know" as competing hypotheses. The null hypothesis ($\mathcal{H}_0$), the simpler or more restrictive of the two hypotheses, along with the model, informs the simulation. The model is used to generate virtual data, $Y^{(\star \cdot)}$ in the figure, or `Ys` in the code. The alternative hypothesis ($\mathcal{H}_1$) is more complex, providing evidence against the null, which may be helpful in determining an appropriate testing statistic $s(\cdot)$. Those two things combine to form the sampling distribution, allowing a comparison to be made between observed and simulated statistics. Details, like how to draw that comparison quantitatively, are ancillary. One way to do that is with a p-value.

I bet you can figure out how to adapt this to other settings on your own. To help, I'll flesh out a lot of the details in later chapters. Eventually, we'll need to invest in some higher-powered mathematics. There are two reasons for this. One is to help automate the choice of statistic s_n, which may be the single most important ingredient. The other is so that you can appreciate how my preferred MC approach compares to the old-school way of doing things, which is what most software packages provide. That comparison is not merely for culture. It's important to relate to others, and see where strengths, weaknesses, and opportunities for synergy are. An old-school approach is complementary and has merits. And if you already have some experience doing things that way, I hope that this book will teach you a new – and I think highly transferable – skill.

Speaking of software. Suppose we wanted to encapsulate our work so we can use it later, for data from a different coin-flipping experiment. (Say, for your homework.) I mean a function that automates everything. Wrap it all up in a neat little bow. It might look something like `monty.bern`, below, where you can specify any $\theta \in [0, 1]$.

```
### monty.bern:
###
### calculate a two-sided Bernoulli test via Monte Carlo and return
### a p-value, along with optional visual
###
### s: test statistic from ...
### n: coin flips
### theta: null hypothesis proportion
### N: Monte Carlo effort
### vis: optional visualization
###
### uses auxiliary pval.discrete function provided on book webpage

source("pval_discrete.R")  ## details in Appendix B

monty.bern <- function(s, n, theta=0.5, N=10000, vis=FALSE)
{
  ## sanity checks
  if(length(s) != 1 && s >= 0) stop("s should be a positive scalar")
  if(length(n) != 1 && n >= s) stop("need scalar n >= s")

  ## MC code copied from earlier
  S.dens <- rep(0, n+1)
  for(i in 1:N) {                          ## each MC trial
    Ys <- rbinom(n, 1, theta)              ## n virtual coin flips
    sY <- sum(Ys)                          ## calculate test stat
    S.dens[sY + 1] <- S.dens[sY + 1] + 1   ## accumulate density/mass
  }

  ## calculate p-value from discrete distribution
  p <- pval.discrete(s, 0:n, S.dens, vis)

  ## calculate p-value and return
  return(list(stat=s, n=n, N=N, p=p))
}
```

Notice how it first performs some checks, to encourage appropriate use. Those are pretty bare-bones. Sanity checks can always be beefed up. It's so hard to anticipate how people will misuse code. Another thing you'll notice, if you look carefully, is that the strategy for storing sampled statistics S_n is somewhat different than our earlier MC in §1.3. There are only $n+1$ possible outcomes, so we only need to store $n+1$ quantities, not $N \gg n$ of them. In `monty.bern` I instead accumulate (empirical) "density" for `Ss` in `S.dens`, which represents sufficient information, a topic I'll return to in Chapter 3. "Density" is in quotes because it's really mass, but that's an esoteric distinction. More on that later. An added benefit of this tally involves an optional visual.

Code for that visual, and for the reflection and ultimate p-value calculation is hidden in `pval.discrete`, via pval_discrete.R[48] which is explained in more detail in §B.1 and may be downloaded from the book webpage. Some of the plotting code resembles my use of

[48] https://bobby.gramacy.com/hipp0/pval_discrete.R

`stepfun` in §3.2. Calculating the reflection is rather more involved when $\theta \neq \frac{1}{2}$, and there are also several reasonable choices, including foregoing the reflection entirely and returning double the one-tailed p-value. What I have coded up mimics what an R library does, which I shall introduce momentarily. This forms the reflection from the location in the opposite tail which has the largest mass not exceeding the mass of the observed statistic s_n.

Ultimately this choice is arbitrary, and I don't think it's important to delve into the details, which is why some of the code is obscured from your view here. The most important thing is the sampling distribution, and where the test statistic s_n – the one actually calculated from data – falls in that distribution. You lose information, and are faced with other arbitrary choices, when distilling that down to a single number like a p-value.

Let's try `monty.bern` on our data, but not exactly our data. Suppose that heads and tails were flipped. This shouldn't change the result, but it could change the visual slightly because now the observed statistic is in the left tail and its reflection on the right. The coin is either fair or it isn't; it doesn't matter what we call heads or tails. I'll let you explore the visual on your own, saving some space here. We'll see lots more histograms characterizing empirical null distributions going forward.

```
out <- monty.bern(n - s, n)
out$p
```

```
## [1] 0.3566
```

Note that this p-value is not identical to the one calculated earlier. Both are based on pseudo-random numbers, so they are themselves random quantities. But this randomness, known as *Monte Carlo error*, can be reduced with larger N. It has nothing to do with flipping from heads to tails. Discussion of MC error is peppered throughout the book, as needed. For more details, see §A.3.

There are already functions built into R to perform a Bernoulli test. Throughout the book I'll refer to these as *library* functions, or routines. Sometimes we'll need to load them in with a `library` command in R. We don't for a Bernoulli test since the one we need comes in the `stats` library which is automatically loaded at startup. The function we need is called `binom.test`. Earlier in §1.1 we talked about the relationship between Bernoulli and binomial distributions. The latter is for sums of Bernoulli trials. Recall that all we need is the sum of binary y_i variables, which is the same as the count of the number of ones. So although I would have preferred the built-in library function were called `bern.test`, or some such, it's not too confusing the way it is. Check it out:

```
c(heads=binom.test(s, n)$p.value, tails=binom.test(n - s, n)$p.value)
```

```
##  heads  tails
## 0.3616 0.3616
```

Notice how these give the same p-value each time, because the calculations are deterministic[49], not stochastic[50] like our `monty.bern`. Those are antonyms, meaning not random and (Greek for) random, respectively. Both calculations give the same answer when you round to the second decimal place, which is plenty accurate for most purposes. You could try increasing N in `monty.bern` to see if agreement increases. The library routine provides

[49] https://en.wikipedia.org/wiki/Deterministic_system
[50] https://en.wikipedia.org/wiki/Stochastic_process

other output, besides the p-value, and we'll talk about that later. The most important of these is a confidence interval for θ, which is something I'll revisit in §3.3.

If you're ever curious about how an R function works, you can always inspect its source code. In the case of `binom.test`, just type the function name (i.e., its symbol, without parentheses or arguments), and its contents will be printed to the console. That won't be the case for all library functions, as some may involve compiled code that's not human-readable in its installed form. But even for those functions, you can dig down to find the actual code (which might be in R or C) by downloading the source archive from CRAN[51], the Comprehensive R Archive Network. It takes some skill and familiarity to decipher some of those codes because they're written to support many edge-cases[52], and have plenty of checks (like our "sanity checks" in `monty.bern`) to help guard against misuse.

How about testing for some θ\$ other than $\frac{1}{2}$? That way I can show you how things come out a little differently, visually speaking. Figure 1.3 uses $\theta = 0.8$ with the same data. The first thing you'll notice is the upgraded visual, using a step function to clearly show each mass-point, whereas the histogram in Figure 1.1 involves binning.

```
out <- monty.bern(s, n, 0.8, vis=TRUE)
```

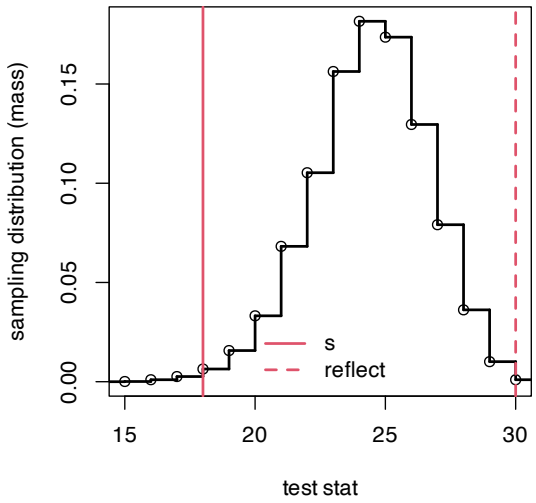

FIGURE 1.3: Using the `monty.bern` function on the coin-flipping data with $\theta = 0.8$.

Also notice that the statistic, s_n, which is in the same location as the previous experiment – it hasn't changed! – but the sampling distribution is different, as is the reflection of the statistic into the other tail. Even without inspecting the p-value, it's clear that the test is a "close call". The statistic and its reflection are near the tails, but not way out in the tails. We can make that precise with a p-value, but that might not help much in making a final decision. That value is below, along with a comparison from the R library.

```
c(monty=out$p, lib=binom.test(s, n, p=0.8)$p.value)
```

[51] https://cran.r-project.org
[52] https://www.uxtweak.com/ux-glossary/edge-case/

```
##   monty     lib
## 0.01110 0.01073
```

If faithful to the default $\alpha = 0.05$, then reject $\mathcal{H}_0$. But I wouldn't bet too much money on it.

Well, there you have it. Now you know how to perform a hypothesis test: just Monte Carlo sample a statistic from the sampling distribution implied by the null hypothesis, and check if your observed statistic is typical (fail to reject) or rare (reject) under that distribution. That's a mouthful and hard to remember, so I came up with a silly alternative. Just "Monty the null hippopotamus". You'll never forget.

It really is just that easy. There's some art to choosing good statistics – ones with good power[53], as they say. Power is an important statistical concept, but one that is largely glossed over in this book. In my opinion, it belongs in a second or third course in statistics, for after you have a handle on the notion of the sampling distribution of a statistic. I think that just by following your nose, you can choose a sensible statistic – one with good power – and concern yourself with fine-tuning later. I've put a bit about power analysis in §A.2 in case you're curious, and for completeness. I shall refer to that from time to time so as not to derail focus from other more pressing themes.

There's also some art to model choice, depending on what data you have. Coin flips are pretty easy in both regards. More widely, there are simple principles that help. I'll get to those in §3.1, but first in Chapter 2 I'll show you one more important class of models and tests with the tools we already have. You'll get lots of practice, beginning with some homework exercises on coin flips.

More reading

To read about the origins of computer simulation, you might want to have a look at Hammersley and Handscomb (1964) and Torcher (1969). Both are out of print, but you can find scans on the internet. It's interesting to see how people thought of computer simulation in an age where computers were beasts that could only be found in well-funded research labs. You can find a lot of good, more modern books with a simple web search. One I really like is by Jones et al. (2009), in part because it uses R. I think that what I'm presenting in this book is unique because it focuses on simulation as a means of statistical inference, which is *not* how people have done things in the past. Though some of that is changing. Many of the principles elucidated here are derived from simulation as a means of broader scientific inquiry. So you may find it interesting to broaden your exposure by checking out one of those references.

Some of the very earliest MC experiments using computers were part of post-war, Manhattan project[54] research into atomic weapons and energy at Los Alamos National Laboratory (LANL)[55]. MC simulation still features prominently in LANL research today. It's one of the ways the USA "tests" its aging nuclear arsenal without violating test-ban treaties. They do it virtually on computers.

[53] https://en.wikipedia.org/wiki/Power_(statistics)
[54] https://en.wikipedia.org/wiki/Manhattan_Project
[55] https://www.lanl.gov

1.5 Homework exercises

These exercises help gain experience with Bernoulli/binomial tests. We shall return to some of them later with an expanded toolkit.

Ensure that you are using methods outlined in this chapter. Many students will have experience working with Bernoulli/binomial tests in other contexts, from previous classes, other books, or materials found online. Those will be discussed and evaluated in later chapters and homeworks. Deploying non-Chapter 1 methods here will likely not earn full credit.

It is intended that you do these problems with your own code, or code from this book/chapter. I encourage you to check your work with library functions like `binom.test`*. However, check with your instructor first to see what's allowed. Unless stated explicitly otherwise, avoid use of libraries in your solutions.*

#1: More flips

What would you conclude for the coin-flips experiment if more coins were flipped, but the outcomes had the same proportion? That is, suppose you got …

- 36 heads out of 60 flips (double-sized),
- 180 heads out of 300 flips (10×).

#2: Silicone wafers

A semiconductor manufacturer claims that 10% of the silicone wafers they produce are defective. In a random sample of 400 wafers, 50 were found to be defective. What do you think?

#3: PVC or copper piping

A building inspector claims that 70% of all homes in Richmond, VA use newer PVC water piping as opposed to copper or other legacy materials. In a random survey of homes in an older neighborhood, 8 out of 15 had PVC. Is this in line with the building inspector's claims?

#4: GPS speed vs. speedometer

It has been claimed that smartphone maps using GPS underestimate highway driving speeds compared to in-dash speedometers. An experiment was conducted with a team of professional drivers. The driving team recorded the mapping app, GPS-estimated speed when the car was set to cruise at a speedometer reading of 70 mph. Of the 50 readings, 29 of them were too slow (i.e., < 70 mph). What do you conclude?

#5: Simulation as a means of calculation

This question has nothing to do with data or statistics, but more generally explores the concept of Monte Carlo for calculating probabilities of certain events.

In a certain process, the probability of producing a defective component is 0.07. Calculate the following probabilities using *only* virtual coin flips (i.e., by Monte Carlo).

- In a sample of 250 randomly chosen components, what is the probability that fewer than 20 of them are defective.

- In a sample of 10 randomly chosen components, what is the probability that one or more of them is defective?
- In a sample of 12 randomly chosen components, what is the probability that the number of defective ones is odd?
- *(This one is for brownie points.)* To what value must the probability of a defective component be reduced such that only 1% of lots of 250 components contain 20 or more that are defective?

#6: One-liner

It is possible to sample from the sampling distribution (generate `Ss`) via MC for a binomial test with one line of code in R, and crucially without any `for` loops. Can you figure out how to do that? Calculating the p-value can be done, cleanly, with a second line (e.g., using `pval.discrete` or by hand for $\theta = 0.5$). Maybe you could code both into one long line if you insist.

Hint: generate `N*n` flips all at once, arrange them in a $N \times n$ `matrix` and use `rowSums` on that matrix.

2

Location: mean modeling

According to search results I got when I Googled "United States height average and standard deviation", adult American women are about 64.5 tall inches with a standard deviation of 2.5 inches. Suppose you surveyed women in your stats class and obtained n height measurements, in inches. Now you "want to know", are women in your class similar to adult women in the USA at large? In particular, do they have the same average height?

I encourage you to do this with data collected from your class, or other women you know. I'm not going to guess at what you got. Here's some data from when I ran this experiment with one of my classes.

```
y <- c(65, 65, 69, 67, 66, 66, 67, 64, 68, 68, 68, 69, 70, 65, 70,
  67, 65, 62, 69, 68, 65, 68, 64, 64)
n <- length(y)
```

In the remainder of the chapter I shall answer this question using tools from Chapter 1, along with a new modeling construct based on a particular example of location models, which are a special case of the location–scale family[1] of probability distributions. I aim to show you that you basically know how to do this already. So we'll be able to move through things quickly, although like with coin flips some detail will be tabled for later.

2.1 Modeling

Technically we have data from two populations: adult US women and women in my class. For now, let's assume the survey of women in the USA was very large, and so we may take the mean and standard deviation estimates from that population as accurate proxies for the gold-standard truth. In other words, the mean height of women in the USA *is* $\mu = 64.5$, and the standard deviation *is* $\sigma = 2.5$.

```
mu <- 64.5
sigma2 <- 2.5^2  ## for sigma^2; more on that in a moment
```

Quick digression on means versus averages, which are estimates of means. They're not the same thing. One (mean) refers to a population quantity, like $\mu = \mathbb{E}\{Y\}$, where Y is a random variable representing American women's heights. The other (average) is an estimate obtained from a (possibly small) n-sized sample from that population. They're not the same

[1] https://en.wikipedia.org/wiki/Location–scale_family

unless you sample everyone in the population, i.e., if $n = 168$ million, according to Google. I just wanted to make sure that's clear before we move on.

To possibly muddy the waters, I'm taking a number I got from Google and treating it as a mean. Google is almost certainly providing an average from a small, or modestly sized, sample. They didn't ask everyone, and for the purposes of the example I am treating an estimate as a population quantity: a mean μ, and a variance σ^2. I'm doing this to compare it to students in my class, who are clearly a much narrower (and possibly not representative) sample of the larger sample, and of the population. Partly, I'm doing this because we're not yet equipped to deal with two samples yet. That's Chapter 5. Hold your horses.

Alright, back to modeling. The first step of any statistical inference exercise is to specify a probabilistic model for the data, our 24 measurements in `y`. We chose a Bernoulli model for coin flips in Chapter 1, and now we need something appropriate for heights in inches. In other words, we need a probability distribution for how heights, such as those observed for American women, *could* have been generated. We don't need to believe this is how they *were* generated, because of course that's preposterous. No model will be perfect, and ultimately this is a choice of convenience. We want something that is not egregiously wrong. Moreover, it helps to choose a family of distributions that can be "matched up" with the information we have, summarized by μ and σ^2.

Suppose we assume that adult heights are Gaussian[2], i.e., that they follow a so-called normal distribution. This is easy to quibble with, but not ridiculous. For one thing, you'd never get any duplicates if that were the case, and there are some duplicated heights in `y`. That's a more involved discussion for later; we're still in the warm-up phase of the book. Again, this isn't a text on probability, which I presume you already have some experience with. If not, that Wikipedia link should help. I'll provide some of the necessary detail here, and more in later chapters.

If we notate each measurement as y_i, with random analog Y_i, then we may characterize our model for n independent and identically distributed measurements as follows,

$$\text{Model:} \qquad Y_1, \ldots, Y_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2), \tag{2.1}$$

where parameters μ and σ^2 dictate the shape of the bell curve of the Gaussian density function. Parameter μ determines the center of the density, or *location*, and can take on any (finite) real value. It specifies the mean[3]: $\mathbb{E}\{Y_i\} = \mu$. Parameter σ^2 determines the spread, with, or *scale* of the bell. It also determines the variance[4]: $\sigma^2 = \mathbb{V}\text{ar}\{Y_i\} = \mathbb{E}\{(Y_i - \mu)^2\} = \mathbb{E}\{Y_i^2\} - \mathbb{E}\{Y_i\}^2$. Without the square, σ is known as standard deviation[5]. You may recall that about 95% of Y_i values lie within $\mu \pm 2\sigma$ under a Gaussian distribution.

I want to assume a Gaussian could characterize the height of a random woman in the USA, i.e., $Y \sim \mathcal{N}(64.5, 2.5^2)$ using mean and standard deviation information gleaned from a Google search. Figure 2.1 shows what that bell curve looks like. The mathematical convention is to notate variances $\sigma^2 \equiv$ `sigma2` (in code), especially for Gaussian distributions. In R, however, standard deviation is used for dialing in the density of a normal/Gaussian distribution. So you'll see in the code that I'm always careful to use the number 2 in my variable name for σ^2 (i.e., `sigma2`), but then I take the square root of that quantity to pass

[2] https://en.wikipedia.org/wiki/Normal_distribution
[3] https://en.wikipedia.org/wiki/Expected_value
[4] https://en.wikipedia.org/wiki/Variance
[5] https://en.wikipedia.org/wiki/Standard_deviation

it to `dnorm`, the density function. You'll probably find that to be pedantic, but it's a great way to avoid bugs. Take my word for it.

```
ygrid <- seq(50, 80, length=1000)
plot(ygrid, dnorm(ygrid, mu, sqrt(sigma2)), type="l",
  xlab="y", ylab="density N(64.5, 2.5^2)", lwd=2)
points(y, rep(abs(rnorm(n, 0, 0.005))))
abline(v=mu + c(-2, 2)*sqrt(sigma2), col=2, lty=2, lwd=2)
legend("topright", "observed", pch=21, bty="n")
legend("topleft", "95%", col=2, lty=2, lwd=2, bty="n")
```

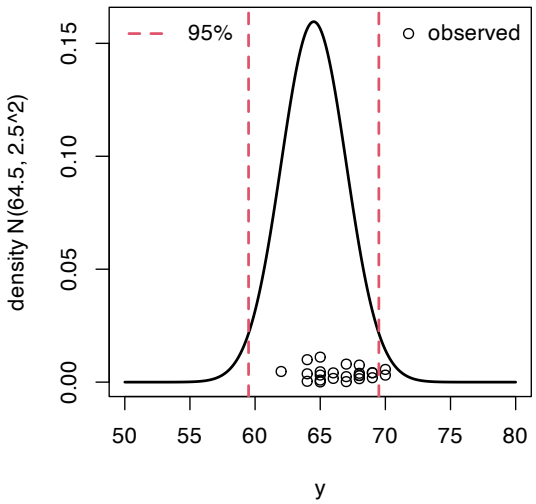

FIGURE 2.1: Visualizing data on women's heights from class versus the US population of women's heights at large. Vertical jitter is added to more easily see duplicated heights.

Overlayed on the density plot in the figure are the 24 observed measurements from women in my class. Basically the "want to know" question boils down to: could it be that those $n = 24$ heights, from my class, were drawn from the same distribution as women in the USA as characterized by the density drawn on the plot? Visually, it doesn't seem implausible. Sure, the observed values concentrate a little to the right of $\mu = 64.5$. And, 8% of them are outside of the 95% interval, which is a little high (only 5% should). But it's not too crazy, right?

How can we make a call in a more principled, and precise manner? How likely is it to see data that we got by chance, in my class, as opposed to anywhere else? Are women taking stats at VT special? (They will be once this class is over! Though not because of their height. They'll have mad stats skills!)

Quick digression before answering that question. People use "Gaussian" and "normal" interchangeably, and so will I. They mean the same thing. There are other members of the location–scale family; and there are many good choices of outside that family where you can still match moments[6] to μ and σ^2 to set parameters. You'll get to tinker with that in the homework exercises of §2.6. My particular Gaussian choice here simplifies many matters, which again I'll return to later.

[6] https://en.wikipedia.org/wiki/Method_of_moments_(statistics)

Alright, back to testing, but I need one last thing. In what follows, I'm going to make a common "baby steps" simplifying assumption that the standard deviation (or variance) is known, and fixed to $\sigma^2 = 2.5^2$ for all heights of women regardless of population. This is not correct, and I will fix it in §3.1. For now, it allows me to focus our investigation on mean μ, on location.

2.2 Testing apparatus

We have our data and our model. We need to clarify hypotheses and choose a testing statistic. Then we can follow Alg. 1.1 via Monte Carlo (MC), just like in Chapter 1. I wish to know if the mean from my class is the same as the mean from the population at large. That can be written as a set of competing hypotheses much in the same way as Eq. (1.4).

$$\begin{aligned} \mathcal{H}_0 &: \mu = 64.5 && \text{null hypothesis ("what if")} \\ \mathcal{H}_1 &: \mu \neq 64.5 && \text{alternative hypothesis} \end{aligned} \tag{2.2}$$

That's the easy part. The harder part, and where it can help to be creative, is to choose a test statistic s_n. Recall that it is the sampling distribution of S_n, contrasted against observed s_n, which plays an integral role in a contradiction argument behind the process of Alg. 1.1. I bet you already know what statistic you want to use: the sample average $s_n = \bar{y}_n = \frac{1}{n}\sum_{i=1}^{n} y_i$. The sample average is an estimator[7] of the mean, a topic we'll return to in more detail in §3.2, and the mean μ is the focus of our hypotheses (2.2). Almost certainly, the sample average will differ from the value of μ specified by the null hypothesis; therefore, it supports the alternative $\mathcal{H}_1$.

```
s <- mean(y)
s
```

```
## [1] 66.62
```

If intuition is failing you, or if it brought you down some other path (there are plenty of reasonable choices!), let me again whet your appetite for §3.1. There, I'll provide an automatic method for choosing a test statistic s_n for tests that are based on parameters describing distributions underlying the data-generating model (2.2). For now, allow me to appeal to reasoning that an estimate (sample average) is a good proxy for a population quantity (mean). If fact, it's so good that people often believe they are one and the same thing when they're not, hence my disclaimer earlier, at the start of the chapter.

Quick digression to foreshadow a little and indulge in a couple of quick technical explanations for why the sample average is a good test statistic for the mean. Under a Gaussian distribution, each $\mathbb{E}\{Y_i\} = \mu$, for all $i = 1, \ldots, n$. By linearity of expectations[8], the expected value of a (weighted) sum of random variables is equal to the sum of their expected values. We can use that with $S_n = \bar{Y}_n$.

[7] https://en.wikipedia.org/wiki/Estimator
[8] https://en.wikipedia.org/wiki/Expected_value#Properties

$$\mathbb{E}\{S_n\} = \mathbb{E}\left\{\frac{1}{n}\sum_{i=1}^{n} Y_i\right\} = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\{Y_i\} = \frac{1}{n} \times n\mu = \mu. \tag{2.3}$$

In other words, the expected value of the statistic is equal to the parameter of interest. We say that the sample average is an unbiased estimator[9] for the mean. Eliminating bias is a good thing, especially in a data-poor situations. So the sample average has one feather in its cap.

Let's look at the variance of our statistic. Under a Gaussian distribution, each $\mathbb{V}\text{ar}\{Y_i\} = \sigma^2$ for all $i = 1, \ldots, n$. By linearity of variance[10], the variance of a (weighted) sum of *independent* random variances is the sum of the (squared weights) variances. For us:

$$\mathbb{V}\text{ar}\{S_n\} = \mathbb{V}\text{ar}\left\{\frac{1}{n}\sum_{i=1}^{n} Y_i\right\} = \left(\frac{1}{n}\right)^2 \sum_{i=1}^{n}\mathbb{V}\text{ar}\{Y_i\} = \frac{1}{n^2} \times n\sigma^2 = \frac{\sigma^2}{n}. \tag{2.4}$$

So as n increases, the variance of S_n, which is the same as its squared expectation around its mean (and which we just learned is $\mathbb{E}\{S_n\} = \mu$), decreases. If you're following along with those Wikipedia links, note that the $\mathbb{C}\text{ov}$ term you see on that page is not present in Eq. (2.4) because I assumed independence. Independent random variables do not co-vary; they have a covariance of zero.

Whenever information goes up (e.g., from more data n) and uncertainty goes down (2.4), that means you're learning something! So S_n is good for μ because it is unbiased, and because it extracts information, decreasing uncertainty with n. Observe that as $n \to \infty$ we have that $\mathbb{V}\text{ar}\{S_n\} \to 0$. Eventually, with enough data n, S_n reveals μ precisely, and without uncertainty. This is not true of all choices of S_n.

For example, what if we chose $S_n = Y_1$? That is, just estimate that the mean is equal to the value of the first observation, ignoring the other $n - 1$. This might seem silly, but it has some good properties. It is unbiased since $\mathbb{E}\{Y_1\} = \mu$. But it does not do a good job of extracting information: $\mathbb{V}\text{ar}\{Y_1\} = \sigma^2 \ll \frac{\sigma}{n}$ for all $n > 1$. No matter how much data you gather, n, you'll never learn μ with improved precision.

There is a sense in which $S_n = \bar{Y}_n$ is optimal for μ under a Gaussian distribution because it minimizes mean-squared error (MSE)[11]. MSE is a hybrid statistic that combines bias and variance in a particular way. But that's a technical topic that is best left for a more advanced text.

Alright, back to the task at hand. We've been able to use some math to understand the central moments[12] of S_n. But we need the whole sampling distribution to adjudicate between hypotheses in Eq. (2.2).

[9] https://en.wikipedia.org/wiki/Bias_of_an_estimator
[10] https://en.wikipedia.org/wiki/Variance#Linear_combinations
[11] https://en.wikipedia.org/wiki/Mean_squared_error
[12] https://en.wikipedia.org/wiki/Central_moment

2.3 Sampling distribution

In fact, you can use math to work out the sampling distribution in closed form. It's not terrible, but you guessed it: I'll delay that until §3.2. Because it's much simpler, and more easily adjusted to other settings, we'll instead simulate here from the sampling distribution by MC. The model is Gaussian (2.1), so all you have to do is repeatedly draw $n = 24$ Y_i-values from a Gaussian pseudo-random number generator and save averages. Here we go.

```
N <- 10000                          ## number of MC "trials"
Ss <- rep(NA, N)                    ## storage for sampled test stats
for(i in 1:N) {                     ## each MC trial
  Ys <- rnorm(n, mu, sqrt(sigma2))  ## change 1: generate n Gaussian samps
  Ss[i] <- mean(Ys)                 ## change 2: calculate test stat
}
```

Note that there are only two changes to the MC for coin flips in §1.3. It bears repeating: be careful to take the square root of σ^2 when using R's normal generators (or density functions, or anything else). This is a common source of bugs in code even for seasoned R professionals. When coding in-real-time, or under the pressure of being at a projector in front of class, I make this mistake all the time. You can score some brownie points with your professor by helping fix a problem like this by calling it out before it really becomes one.

What to do with those samples? The same exact thing we did for coin flips. Once we have samples from the sampling distribution for S_n, we can (optionally) visualize them to contrast with observed s_n, or jump straight to a two-tailed p-value calculation. Figure 2.2 shows a view of the empirical distribution, overlaying s_n and its reflection over the mean as vertical lines. Again, empirical distributions are covered in more detail in §A.1. Here, I just mean that the histogram is summarizing a spread of sampled values, showing how they distribute on the x-axis.

```
hist(Ss, main="", xlim=range(s, 2*mu - s, Ss))
abline(v=c(s, 2*mu - s), lty=1:2, col=2, lwd=2)
legend(65, 1650, c("s", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
```

A simple, means-based reflection is fine since Gaussians are symmetric. This involves finding the value which is the same distance from the middle of the distribution as s_n is, but put it on the other side. Notice I used `mu` for the reflection instead of `mean(Ss)`. Both work and are correct, but `mu` is more accurate for finite MC effort N since I already know that $\mathbb{E}\{S_n\} = \mu$. That was the basis of our argument that S_n is unbiased (2.3). Any time you can use something that you know is exact, mathematically, as opposed to being calculated via MC you'll get a more accurate calculation. I encourage you to try swapping out for `2*mean(Ss) - s` instead.

Clearly both s_n and its reflection represent unlikely observations under the null hypothesis. It doesn't look like any of our $N = 10{,}000$ simulations of S_n is more extreme than what we observed in the data. This means that a calculation in R for the p-value with these samples will show zero.

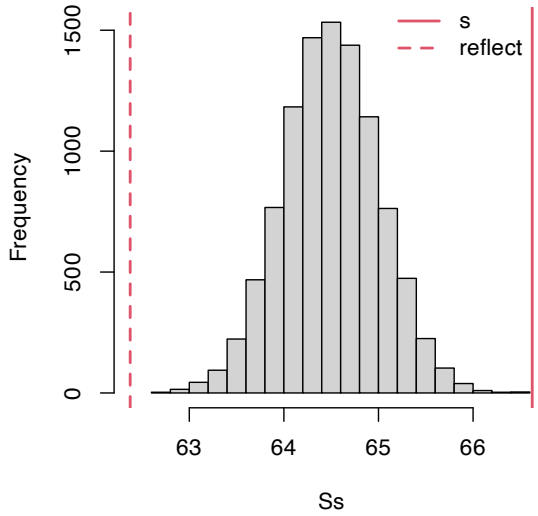

FIGURE 2.2: Histogram of samples S_n from their sampling distribution under $\mathcal{H}_0$, and the actual s_n (and it's "two-tailed" reflection) for comparison, for the heights experiment.

```
pval <- mean(Ss < 2*mu - s) + mean(Ss > s)
pval
```

```
## [1] 0
```

That isn't quite right, without acknowledging that the calculation is sensitive to the amount of MC sampling effort N. A better answer is that the p-value, often notated as ψ, in this case follows $\psi \leq 1/N \lessapprox 10^{-4}$. If your threshold is $\alpha = 0.05$ then even without that improved precision you've got a pretty clear reject of $\mathcal{H}_0$, and no further action is necessary.

Based on this, I conclude that women in my class are not like the population at large, at least not based on their height. I can probably explain that on socioeconomic grounds. Women who attend college are more affluent than those who do not, which means better nutrition among other things, so they're likely to be taller. Yet that sort of thing is an indulgence without further information. It'll be fun to ponder such things in this book, but I'm not serious about it. (I am serious about the stats though!)

If you want more precision out of an MC p-value calculation, increase N and keep going. I had to increase N one-hundred-fold to get more than one hundred extreme values. Your mileage may vary as you'll get different random draws than me.

```
N2 <- N*100
Ss <- c(Ss, rep(NA, N2-N))
for(i in (N+1):N2) {
  Ys <- rnorm(n, mu, sqrt(sigma2))
  Ss[i] <- mean(Ys)
}
p100 <- mean(Ss < 2*mu - s) + mean(Ss > s)
p100
```

```
## [1] 2.6e-05
```

Finally, since Gaussians are symmetric, so will be the sampling distribution. More detail is available in Chapter 3. This means that we'll get the same p-value, up to error inherent in a small MC sample, by doubling a one-tailed p-value.

```
2*mean(Ss > s)
```

```
## [1] 3e-05
```

However, if you don't know for sure that your sampling distribution is symmetric, then you can always do the following instead. Calculate the empirical quantile[13] of s_n in the distribution for S_n, and then match it with a value in the opposite tail.

```
pl <- mean(Ss >s)
sr <- as.numeric(quantile(Ss, pl))
c(dist=2*mu - s, quantile=sr)
```

```
##     dist quantile
##    62.38    62.42
```

They're not the same, but pretty close. The quantity `s` lies in some percentile of `Ss`, which may be calculated as `mean(S > s)`, and in this case lies in the right tail (implicitly `pr`). One minus that is `pl = mean(Ss >= s)` in the left tail. The command `quantile(Ss, pl)` extracts the value of `Ss` that lies the `pl` percentile: the equivalent extreme value observed in the opposite tail of the sampling distribution. So the reflection `sr` is like the left-tail image of the statistic we observed in the right tail, in this instance. The idea is that the statistic and its reflection have the same amount of (more extreme) tail probability.

Then calculate the p-value as follows.

```
mean(Ss < sr) + mean(Ss > s)
```

```
## [1] 3e-05
```

So we get the same thing as doubling the one-tailed p-value. It's that way by construction. We made it so the quantiles match exactly.

2.4 Wrapping up

See, that wasn't so bad. It's basically the same thing as in Chapter 1 except with a different distribution – one appropriate for real-valued measurements. In Chapter 1, I built a library function and then compared to calculations built into R. Some of the details involved in finding a reflected statistic for a two-tailed p-value were buried in a subroutine because of the discrete, asymmetric nature of the sampling distribution.

This time it's easier because of symmetry, so I'm going to let you make the library function as a homework exercise. I suggest calling it `monty.z`. What we're doing here is known as

[13] https://en.wikipedia.org/wiki/Quantile

a z-test. The reason for this is that (standardized) Gaussian deviates[14], which is another name for random variable, are often notated with the letter Z or z. You may have heard of a t-test[15], which is the generalization of a z-test. A z-test uses a known variance σ^2, whereas a t-test uses estimated σ^2 and consequently differs in a few other details. Wait for Chapter 3.

These z-tests are a great place to start – often right after binomial tests but sometimes the other way around – because they're relatively simple. They're not particularly common in practice, at least for data under a Gaussian model, because usually you *do* want to estimate σ^2. There's no z-test built into R, but you can get one from a library on CRAN. There are many, and I have chosen one from the BSDA (Basic Statistics and Data Analysis) package (Arnholt and Evans, 2023) somewhat arbitrarily.

```
## install.packages("BSDA")  ## only do this once per machine
library(BSDA)
z.test(y, mu=mu, sigma.x=sqrt(sigma2))$p.value
```

```
## [1] 3.125e-05
```

Observe that this p-value is similar to the second one calculated above, based on $N = 10^6$ MC samples. Note that you only need to install an R package once per machine, but it must be loaded with the `library` command each time you start a new session. Since I have already installed the package, the `install.packages` command is commented out above. You'll need to do that (un-comment it, once on your machine) if you haven't already.

2.5 Return on investment

I used to work in the Booth School of Business[16] teaching MBAs at The University of Chicago[17]. When I started working there, some colleagues passed me a cool data set that was tied to a statistical analysis provided by a major consulting company. The clients of that company based an aggressive, negative advertising (smear) campaign against their competition, on that analysis. I'm being a little coy here not to mention the names of these companies, or the statistical consultancy, because I don't want anyone annoyed at me. But I am going to share the data involved in that analysis with you, because you can do (a better job of) that analysis. It's not that hard. Find the file as roi.RData[18] on the book webpage.

```
load("roi.RData")    ## defines variable roi, which is a list
roi
```

```
## $company
##  [1]  21.1  15.8   1.2  61.2  15.2  26.4  66.9   5.4  12.2   4.9  20.8
## [12]  23.3  13.2  14.5  27.3  17.0 -41.1  35.9 116.4   8.1   7.9   6.7
## [23]  32.7  11.6  17.1 -47.5  23.8   1.1  26.7   6.9  13.4  14.9  14.9
```

[14] https://en.wikipedia.org/wiki/Standard_normal_deviate
[15] https://en.wikipedia.org/wiki/Student's_t-test
[16] https://www.chicagobooth.edu
[17] https://www.uchicago.edu/
[18] https://bobby.gramacy.com/hipp0/roi.RData

```
## [34]  14.6 -15.4   4.6  43.9  15.8   6.4  13.1  14.6   8.4   8.9   8.8
## [45]  23.1 -91.8  45.8   7.3  18.9  22.8  -7.7   6.8  -9.2 -62.6  18.7
## [56]  27.3   8.3  16.7  18.4  28.8  25.6  16.9   0.3  -0.7 -38.5   8.3
## [67]   8.3   8.3   8.3   6.2   6.2  -0.5  18.6   0.9  47.4  27.3   4.7
## [78]   2.8  41.4  26.0  14.6
##
## $industry
##  [1]  8.7 19.5 19.5 22.0 19.6 22.2 10.6  3.7 19.5 22.2 14.0 11.9 14.9
## [14]  8.8 18.2 10.2 16.3 15.3 14.9 10.9 10.9  9.8  9.5  6.4 15.3 13.3
## [27] 20.2 26.0 26.1 14.0  8.6
```

These are real data about real companies. I verified that with some of my own digging, at first not trusting the narrative of my colleagues. It used to be that you could download a PDF document containing these data from the website of the statistical consultancy. They were proud of these data and their analysis. Now you can't find it anymore, or at least I couldn't. I won't speculate why, but I can insinuate by helping you conduct some analysis of your own. These data focus on a financial variable called return on investment (ROI)[19]. Higher ROI is better. In a down-year for sales, say, it might be low or negative. If ROI doesn't (eventually) trend positive on aggregate, year-on-year, then the company isn't going to stay in business long.

When you run that `load` command, your environment is populated with an object called `roi` that is comprised of an R `list` object with two vectors. One contains ROI numbers for particular companies that use a particular IT product that I won't name. These are found in `roi$company`. There is one number for each company. Think Apple, Google, Bayer and AstraZeneca, which may or may not have been part of the study. The other vector, `roi$industry`, contains similar ROI numbers for the sectors that these companies operate in. Think of these numbers as aggregate ROIs for all of the companies in that industry. There would be one for high tech companies (Apple, Google) and one for pharma (Bayer, AstraZeneca), for example and among others.

Figure 2.3 provides a visual of these two, related data sets. The first thing folks notice is that the spread of values for `$industry` is much narrower than `$company`. This makes sense because the former is based on an aggregate – on all companies in a particular industry – and the latter are single companies representing part of that whole. Since not all companies use the particular IT product in question, there are some companies represented in the `$industry` aggregate that are not represented in the `$company` vector.

```
hist(roi$company, main="", xlab="ROI")
hist(roi$industry, col=2, add=TRUE)
legend("topright", c("company", "industry"), fill=c("gray", "red"),
  bty="n")
```

The other thing folks notice is that the entire red histogram is positive. This also makes sense. These industries make money. If they didn't, they would collapse. Individual companies, with their much wider spread of ROIs, have more extreme values, particularly negative ones.

The focus of the consultant's statistical analysis, and their client's subsequent advertising campaign involved sample averages.

[19] https://en.wikipedia.org/wiki/Return_on_investment

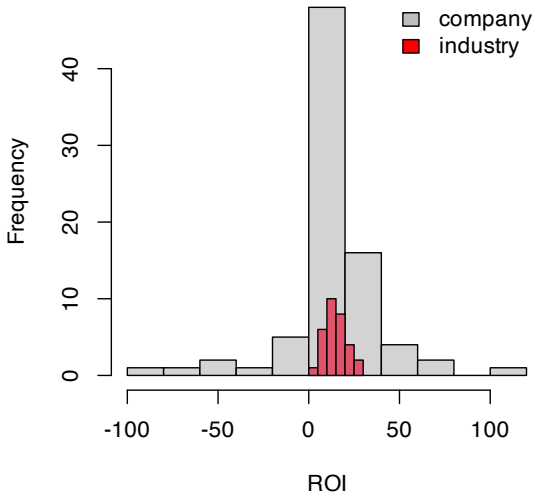

FIGURE 2.3: Histogram of ROI for individual companies and the industries they are part of, at large.

```
ybar <- c(com=mean(roi$company), ind=mean(roi$industry))
ybar
```

```
##   com   ind
## 12.64 14.94
```

From these you can see that it's less profitable to be a company who uses a particular IT software than it is to be a company in the population at large. By taking the ratio of the first number over the second, the consultancy concluded that companies using that IT software were 15% less profitable. Their client pounced on this and proclaimed, on billboard advertisements at bus stops and subway stations, that "Companies who use XXX are 15% less profitable than their peers". Quite a smear.

Is it true? Taking some sample averages and dividing them is a *statistic*, but it is not doing Statistics, capital "S". What are the chances that you could get a ratio like that by pure chance? Could it be that the `$company` population actually has the same mean as the `$industry` at large? Can we reject that hippopotamus?

I'll let you decide in a series of homework problems, some in this chapter and some in later chapters. We'll start simple and then build up sophistication. I'll give you a little spoiler alert: the evidence isn't very compelling. It's a wonder nobody was sued. Maybe they settled out of court.

2.6 Homework exercises

These exercises help gain experience with z-tests, and extensions, for location. We shall return to some of them later with an expanded toolkit.

Ensure that you are using methods outlined in this chapter. Many students will have experi-

ence working with z-tests in other contexts, from previous classes, other books, or materials found online. Those will be discussed and evaluated in later chapters and homeworks. Deploying non-Chapter 2 methods here will likely not earn full credit.

Do these problems with your own code, or code from this book. In some cases you can check your work with `z.test` *from* `BSDA`*. However, check with your instructor first to see what's allowed. Unless stated explicitly otherwise, avoid use of libraries in your solutions.*

#1: Library function

Write a library function like `monty.bern` from §1.4 but for testing μ with observations modeled as Gaussian with a particular, fixed σ^2 parameter. Call it `monty.z`. Include optional visuals when `vis=TRUE` and return a similar `list` of relevant quantities.

#2: Fewer heights

What would you conclude for the women's heights experiment if, on the particular day the data was collected, fewer women came to class. That is, suppose you got …

- `y14 <- y[1:14]  ## (first fourteen women)`
- `y7 <- y[1:7]  ## (first seven women)`

#3: Logistic distribution

Revisit the (full) heights data but instead of a Gaussian distribution, use a logistic distribution[20], which is another member of the location–scale family. The logistic has two parameters, mean μ and scale σ (notated as s on that Wikipedia page). The mean has the same interpretation as the Gaussian, where $\mathbb{E}\{Y_i\} = \mu$ for all $i = 1, \ldots, n$. But σ is not the standard deviation. Wikipedia explains that $\mathbb{V}\text{ar}\{Y_i\} = \sigma^2\pi/3$. It probably won't escape you that $\pi/3 \approx 1$, but not exactly. So you'll need to do a little work to use standard deviation information from the US population of women with the logistic parameter σ. See `rlogis` in R.

#4: Gamma distribution

Revisit the (full) heights data but instead of a Gaussian distribution, use a gamma distribution[21], which is *not* a member of the location–scale family. Use the shape α and rate λ version from Wikipedia and via `rgamma` in R. You'll need to match moments by first solving the following system of two equations and two unknowns in order to use the US population information.

$$\mathbb{E}\{Y_i\} \equiv \mu = \frac{\alpha}{\lambda} \quad \text{and} \quad \mathbb{V}\text{ar}\{Y_i\} \equiv \sigma^2 = \frac{\alpha}{\lambda^2}$$

You may still express your hypotheses (2.2) in terms of μ, but you'll have to figure out what this means in terms of α and λ so you can virtualize via MC simulation with `rgamma`.

#5: First ROI analysis

Using $\sigma^2 = 658$, do you think that ROI for companies using the particular IT product (i.e., from `y <- roi$company`) could have the same mean as that from the industry at large (i.e., $\mu = 15$)?

[20] https://en.wikipedia.org/wiki/Logistic_distribution
[21] https://en.wikipedia.org/wiki/Gamma_distribution

#6: Pushing the ROI envelope

What are the chances that things are exactly the opposite of what they seem, from the crude analysis via `ybar` in §2.5? In other words, can you reject the hypothesis that ROI for companies using the particular IT product is actually $\approx 15\%$ larger (not smaller) than the industrial average, i.e., with $\sigma^2 = 658$ can you reject $\mu = 17$?

#7: One-liner

It is possible to sample from the sampling distribution (generate `Ss`) via MC for a z-test with one line of code in R, and crucially without any `for` loops. Can you figure out how to do that? Calculating the p-value can be done, cleanly, with a second line, but you could probably combine into one long line if you insist.

Hint: generate ***`N*n`*** *Gaussian draws all at once, arrange them in a* $N \times n$ ***`matrix`*** *and use* ***`rowMeans`*** *on that matrix.*

#8: Bouncy balls

Data provided in R below represent heights of the first bounce of a batch of bouncy balls[22] fresh from the factory, when released from 5 feet above the ground.

```
y <- c(53, 53, 53, 54, 54, 54, 52, 52, 54, 53, 52, 50, 52, 52, 56, 54,
  54, 54, 54, 51, 53, 54, 54, 55, 54, 54, 53, 53, 54, 54, 55, 55, 55)
```

Manufacturer specifications are that the mean height of the first bounce should be 90% of the dropped height for a typical person. Assume that the standard deviation is 1. Does this batch of balls meet those criteria?

#9 Homework questions

My goal was to have about ten homework questions per chapter. Regarding the number of questions in each chapter as a random draw from a Gaussian distribution, could you reject the null hypothesis that $\mu = 10$? In other words, how'd I do? Use a sensible choice of σ^2.

Hint: you might need to do a little sleuthing to collect the data. How could you find out how many questions each chapter has?

[22] https://en.wikipedia.org/wiki/Bouncy_ball

3

A little math: inference

There's not too much math in this book, I promise. Like 99% of it is in Chapters 3-4. Mostly my goal in this book is to show you how much of statistics can be math-free with a computer. Yet, a lot of statistical fundamentals are rooted in math, and I want to show you how people usually do things. So we've gotta do this. I'll try to make it as painless as possible.

At times I'll use the words analytic[1] or closed form[2] to describe the methods in this chapter. Sometimes both at the same time (analytic closed form). This is just me being fancy with big words. By either or both, I simply mean "using math", as opposed to numerical methods like Monte Carlo (MC). My use of "analytic" in this context is highly colloquial, and wouldn't align well with the technical definition at that Wikipedia link. "Closed form" is appropriate, since it basically means "you get a function, expression or number", as opposed to "you get samples" (MC) or "you iterate until convergence" (other numerical procedures). Asymptotic methods, like those in Chapter 4, also provide closed forms, but those are approximate. The ones in this chapter are exact.

Almost everything here is an application of something you know already from high school math and probability. That is, assuming you took calculus in high school. I didn't, believe it or not. I wasn't the best high school student; I thought I was an athlete not a scholar. Hoping to satisfy a "gen-ed" requirement with as little hassle as possible, I re-took pre-calculus in my third quarter at UC Santa Cruz[3] (UCSC). I was surprised to like it. I give the credit to an excellent professor. I didn't take the calculus sequence at UCSC until my second year. Wouldn't you know it, I ended up a math major, and eventually a college professor.

Anyways, most of the stuff in this chapter is pretty straightforward conceptually, but let me warn you that getting good at it will take practice. You have to practice this stuff, like anything else in math (or life), to feel comfortable with it. There will be a few things that I'll waive my hands at because the math is hard, but the result is too important to omit. That importance is either conceptual, in the sense that it reveals a beautiful truth about how numbers and randomness work. Or it's a linchpin, in the sense that we can't move on without it, but otherwise it's pretty trivial and not very interesting. There's like one or two of each of those coming up here and in Chapter 4.

My goal is two-fold: (1) to give you a cookbook/recipe framework for the rest of the book; (2) to circle back to the coin-flipping and location examples of Chapters 1–2, showing how `binom.test` and `z.test` library functions really work. Then we'll talk about t-tests, the most famous of all tests, and about confidence intervals.

[1] https://en.wikipedia.org/wiki/Analytic_function
[2] https://en.wikipedia.org/wiki/Closed-form_expression
[3] https://www.ucsc.edu/

3.1 Maximum likelihood

It helps to have an "automatic" procedure for estimating unknown quantities, like parameters in probabilistic models. Think θ for Bernoulli data in Chapter 1 or μ and σ^2 for Gaussian data in Chapter 2. Here, I'll generically use θ as the unknown parameter. This quantity may be a scalar, like in the Bernoulli case, or it may be a p-vector, like in the Gaussian case where $\theta = (\mu, \sigma^2)$ and $p = 2$.[4] In the next little bit, it's fine to think of θ as a single, unknown number. What's that? Did someone say Hessian? No? Oh, never mind.

Let $f(y;\theta)$ denote the probability mass or density function for random variable Y, as determined by parameter θ. That's for a single y. Usually we have a data set of n observations, notated as $y_1, \ldots, y_n$, modeled as all having the same distribution (the same probability mass or density function), independent of one another conditional on a common setting for the parameter θ. The short hand for that is iid. So far, I'm just rehashing §1.2. Hold tight.

Recall that a joint density or mass factorizes as a product (1.2) under an iid assumption. I'll rewrite things here to be more generic with f as a density or mass, whereas Eq. (1.2) favors a discrete random variable via $\mathbb{P}(Y_i = y_i)$. This product is known as the likelihood[5]

$$L(\theta) \equiv L(\theta; y_1, \ldots, y_n) = f(y_1, \ldots, y_n; \theta) = \prod_{i=1}^{n} f(y_i; \theta).$$

So the likelihood *is* the joint mass or density, and it's a product of individual densities when observations are iid. Why have a special name for it? One reason is to elevate its importance. Another reason is because of how we are meant to think about it in the context of parameter θ. A density or mass f is saying something about Y or Y_i for particular θ. With f we think of y or Y as varying given fixed θ. For L we think of varying θ for fixed $y_1, \ldots, y_n$. A lot of times we abbreviate with $L(\theta) \equiv L(\theta; y_1, \ldots, y_n)$ to emphasize focus on θ, not y.

For different values of θ, the quantity $L(\theta)$ may be larger or smaller depending on how probable observed $y_1, \ldots, y_n$ values are. It makes sense to target the value of θ that, under the model f, makes the observed data most probable.

$$\hat{\theta} = \text{argmax}_\theta \, L(\theta) \tag{3.1}$$

This is called maximum likelihood estimation (MLE)[6], and $\hat{\theta}$ is the maximimum likelihood estimate. The E in MLE is interchangeably estimation, estimate and estimator (more on that later). It's common to put hats on estimates of parameters in statistics, whereas unhatted ones are stand-ins for other values like the "true", unknown value generating the data. The argmax is just a fancy way of saying "the argument (θ) that maximizes".

In the case of a Bernoulli distribution, we saw in Eq. (1.3) that $L(\theta) = \theta^{\sum y_i}(1-\theta)^{n-\sum y_i}$. Recall that $\sum y_i$ is short for $\sum_{i=1}^{n} y_i$. I mentioned how this sum is all you need to characterize the joint distribution, or in this case to evaluate (or maximize) the likelihood. It is said that $\sum_i y_i$ is a sufficient statistic[7], or sometimes we say "sufficient for θ", because it's all you

[4] Careful not to confuse the index p, counting parameters, with p-value. If I ever need to put a p-value, generically, into a mathematical expression, I will use the Greek ψ to avoid confusion.

[5] https://en.wikipedia.org/wiki/Likelihood_function

[6] https://en.wikipedia.org/wiki/Maximum_likelihood_estimation

[7] https://en.wikipedia.org/wiki/Sufficient_statistic

need from the data to decide on θ through the likelihood. You just need the number of heads, or equivalently the number of tails (and n).

How to do that? How to solve for $\hat{\theta}$ in Eq. (3.1)? The answer is derivatives. Optimization[8] is one of the most important applications of differential calculus[9]. The basic program is this. Take the derivative of the function you wish to optimize with respect to the variable you want to dial in, set equal to zero and solve. (And don't forget to check that the critical point[10] you've found is actually a minimum.)

When maximizing likelihoods, there's actually one more, or rather one preliminary, step that's required. It can be difficult to take the derivative of $L(\theta)$, like in Eq. (1.3), in its raw form because it's a product of exponentiated terms. That means messy applications of product and power rules for differentiation. It's even harder – can even be impossible – to solve after setting to zero. Instead, practitioners take the logarithm of the likelihood, working with $\ell(\theta) = \log L(\theta)$ before taking derivatives. The log-likelihood is also known as the score[11]. This doesn't really change the answer since

$$\text{argmax}_\theta \, \ell(\theta) = \text{argmax}_\theta \, L(\theta) \qquad \text{where} \qquad \ell(\theta) \equiv \log L(\theta)$$

because log is a monotonic transformation[12], which preserves the locations of critical points.

Note that I'm generically writing log, since it doesn't matter what base. Any of $\log_{10}$, $\log_2$ or $\ln \equiv \log_e$ would be fine. You can change bases by multiplying by a constant, and that doesn't change the location of critical points. In statistics, when someone writes log without a base, they mean natural log ($\ln \equiv \log_e$), unless otherwise clarified, because many densities involve the inverse of ln which is exp. We'll come back to that in just a minute.

Logarithms have many nice properties[13]. They're both beautiful and intimidating. The main thing they do for us, in the context of maximizing likelihoods, is turn products into sums. They also bring powers down as multiples, which is really the same thing. Sums are easier to work with than products when it comes to derivatives; same for powers/multiples. Both make solving for critical points easier as well. Let me show you.

Toss up MLE

Consider a log Bernoulli (1.3) likelihood, and derivative-based maximization for the MLE.

$$\begin{aligned}
\text{log-likelihood} \qquad & \ell(\theta) = \left(\sum_{i=1}^n y_i\right) \log\theta + \left(n - \sum_{i=1}^n y_i\right) \log(1-\theta) \\
\text{derivative} \qquad & \ell'(\theta) = \frac{\sum_{i=1}^n y_i}{\theta} - \frac{n - \sum_{i=1}^n y_i}{1-\theta} \\
\text{solve} \qquad & 0 \overset{\text{set}}{=} \frac{\sum_{i=1}^n y_i}{\hat{\theta}} - \frac{n - \sum_{i=1}^n y_i}{1-\hat{\theta}} = \sum_{i=1}^n y_i - n\hat{\theta} \\
& \hat{\theta} = \frac{1}{n}\sum_{i=1}^n y_i \equiv \bar{y}
\end{aligned} \tag{3.2}$$

[8] https://en.wikipedia.org/wiki/Differential_calculus#Optimization
[9] https://en.wikipedia.org/wiki/Differential_calculus
[10] https://en.wikipedia.org/wiki/Critical_point_(mathematics)
[11] https://en.wikipedia.org/wiki/Informant_(statistics)
[12] https://en.wikipedia.org/wiki/Monotonic_function#Monotonic_transformation
[13] https://en.wikipedia.org/wiki/Logarithm

What a cool result! The *rate* of heads, i.e., the average of the 0s and 1s representing the n coin flips, is the value of θ that makes the observed $y_1, \ldots, y_n$ most probable.

Recall that the statistic we chose, a sum of binarized flips (or count of heads), is the same as $s_n = n\hat{\theta}$. I used this s_n as a statistic in a hippopotamus test (1.4) for θ. In the code below, I'm using `s` and `n` variables defined in Chapter 1 for our coin-flipping data.

```
that <- s/n     ## s and n defined in Chapter 1
that
```

```
## [1] 0.6
```

So not only does an MLE give us an automatic way to estimate a parameter, with a nice (most probable) interpretation, but it gives us an automatic statistic for testing as well. I could have taken $s_n = \hat{\theta}$, i.e., dividing the count by n, but that wouldn't have changed things much beyond scaling labels on the x-axis in Figure 1.1. Table that thought for the moment so I can present another example first, and then we'll come back to it.

Location MLE

Now take a look at the Gaussian location model (2.1) from Chapter 2. I talked about some of the properties of the intuitive estimator $\hat{\mu} = \bar{y}$, like unbiasedness and efficiency. I never actually wrote down the joint distribution for all $y_1, \ldots, y_n$. This is a good thing, because it allows me to work here from scratch and show you everything from start to finish.

Recall that the model is $Y_i \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$, for $i = 1, \ldots, n$. So we have:

$$
\begin{aligned}
&\text{1: joint via iid} && f(y_{1:n}; \mu, \sigma^2) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(y_i - \mu)^2}{2\sigma^2}\right\} \\
&\text{2: re-label and distribute} && L(\mu, \sigma^2) = (2\pi\sigma^2)^{-n/2} \exp\left\{-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2\right\} \\
&\text{3: log and simplify} && \ell(\mu, \sigma^2) = c - \frac{n}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2 \\
&\text{4: differentiate} && \frac{\partial \ell}{\partial \mu} = -\frac{2}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mu)(-1) \\
&\text{5: set to 0 and simplify} && 0 \overset{\text{set}}{=} -n\hat{\mu} + \sum_{i=1}^{n} y_i \\
&\text{6: solve} && \hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} y_i \equiv \bar{y}.
\end{aligned}
\tag{3.3}
$$

Would you look at that! The MLE is, *again*, the simple, intuitive estimate. Is that going to happen every time? No. I'll show you in just a moment. First, a little commentary.

When I say "distribute", like in Step 2, I mean push the product into the exponent, where it becomes a sum. This isn't strictly necessary, but I think it looks a little neater. When you take the log in Step 3, you can do it on the product (which will become a sum) without hassle. You'll have a little more work to simplify things. And when I say "simplify", this relies on your algebra skills and could sometimes be laborious, often depending on how you set things up earlier. The constant in Step 3 is really $c = -\frac{n}{2}\log 2\pi$. Since this doesn't

depend on any of the parameters, μ or σ^2, it's not really relevant. When taking a derivative, like in Step 4, these terms evaluate to 0. Don't forget your chain rule! Finally, you may not always be able to do Step 6, the solving part. It's not about you. I shall limit our focus in this book to situations where this is possible.

Just to have it coded up for stuff coming later, here is the $\hat{\mu}$-value using data from the heights analysis of Chapter 2. This is the same value as what I was calling `s` earlier. Our MLE and test statistic for "location" are the same, which happens a lot. In the code below, I'm using `y` defined in Chapter 2 from our heights analysis.

```
muhat = mean(y)    # y defined in Chapter 2, where s = muhat
muhat
```

```
## [1] 66.62
```

That was just for one of the two parameters, μ. What about σ^2? We have everything we need up to Step 3. Things change from Step 4 because we must now differentiate with respect to σ^2 instead.[14]

$$\begin{aligned}
&\text{4: differentiate} && \frac{\partial \ell}{\partial \sigma^2} = -\frac{n}{2\sigma^2} - \frac{1}{2(\sigma^2)^2}\sum_{i=1}^{n}(y_i - \mu)^2 \qquad (3.4)\\
&\text{5: set to 0 and simplify} && 0 \overset{\text{set}}{=} -n + \frac{1}{\sigma^2}\sum_{i=1}^{n}(y_i - \mu)^2\\
&\text{6: solve} && \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \mu)^2
\end{aligned}$$

These two derivatives – technically partial derivatives[15] – form a gradient[16] $\nabla \ell$. Setting the gradient to zero and simultaneously solving for the MLE of all/both parameters at once is the same as doing them one at a time, when possible, if we also plug in the ones we've found along the way. In other words, plug in $\hat{\mu}$ when solving for $\hat{\sigma}^2$. So the MLE for σ^2 in a Gaussian model is

$$\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2. \qquad (3.5)$$

Quick digression: it turns out that this estimator is biased because $\mathbb{E}\{\hat{\sigma}^2\} = \frac{n-1}{n}\sigma^2$. You're asked to show this as an exercise in §3.5. However, it's asymptotically[17] unbiased since $(n-1)/n \to 1$ as $n \to \infty$. We'll talk a lot more about asymptotics in Chapter 4. This "bias" is why statisticians prefer another estimate

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2, \qquad (3.6)$$

because this one is unbiased. Here is what we would calculate with our heights data from Chapter 2.

[14] Not with respect to σ. Treat (σ^2) as a single quantity, like x, and differentiate with respect to x.
[15] https://en.wikipedia.org/wiki/Partial_derivative
[16] https://en.wikipedia.org/wiki/Gradient
[17] https://en.wikipedia.org/wiki/Asymptotic_theory_(statistics)

```
s2 <- var(y)
c(s2, sigma2)
```

```
## [1] 4.592 6.250
```

R automatically divides by $n-1$ when estimating a variance. This is a common choice because of the bias correction, giving up some optimality for good average-case behavior. Notice that this `s2` value is different than the `sigma2` one used in Chapter 2. More on this in §3.2.

Another reason Eq. (3.6) is preferred over the MLE is because it makes sense when $n = 1$. Notice that you'll get 0 from the sum in this case because $\bar{y} = y_1$ and $n - 1$ evaluates to zero for $n = 1$. So that's a $0/0$ fraction which is undefined. If you use $\hat{\sigma}^2$ instead, you get $0/1 = 0$ which is just wrong. You can't have zero variance – the smallest possible variance – from just one observation. Nonsense! Better for variance to be undefined when $n = 1$. You need to see more than one different thing to detect how they can vary.

Alright, back to MLEs. Maximizing the likelihood gives you an automatic, procedural way to solve for "good" estimates of parameters to your model. You get the best fitting parameters to your data in a certain sense. This is so important, that I'm going to wrap it into Alg. 3.1. Therein I use a generic parameter θ and density or mass $f(y;\theta)$. The parameter θ may be scalar, as in a Bernoulli experiment, or it may be vectorized, e.g., in a Gaussian/location experiment where $\theta = (\mu, \sigma^2)$. When derivatives are taken, I use gradient/partial derivative terminology even if θ is scalar. If I do a good job here, you should be able to follow this procedure in any (iid) inferential setting.

Some notes on a couple of those steps. If you did Step 3 right, taking the log, then you shouldn't have any exponential (exp) or powers left in your expression. Now on to Step 6. In our Gaussian example, we solved for each parameter one at a time: first $\hat{\mu} \equiv \hat{\theta}_1$; then we plugged that value in to solve for $\hat{\sigma}^2 \equiv \hat{\theta}_2$. This approach is usually the easier of the two options. There is often an order that makes sense, creating a cascade of simplification for $\hat{\theta}_{j+1}$ when plugging in previously calculated $\hat{\theta}_j$. Whenever you plug some estimated parameters into a log-likelihood (or its derivatives), you get what's called a concentrated or profile (log-) likelihood[18]. The alternative in Step 6, of solving simultaneously, can be difficult but may be the only recourse in some situations.

Note that it's not always possible to do Step 6, either one-at-a-time or simultaneously, analytically (which means with paper and pencil). When that happens, you need a derivative-based numerical optimization, many of which are based on Newton's method[19]. We won't encounter any of these in this book, but this approach plays an integral role in many more advanced statistical modeling setups.

3.2 Sampling distribution and p-value

In order to test a hypothesis about a statistic s_n calculated from data, we must understand its distribution under the null. This is what MC is for in §1.3 and §2.3, and throughout all

[18] https://en.wikipedia.org/wiki/Likelihood_function#Profile_likelihood
[19] https://en.wikipedia.org/wiki/Newton's_method_in_optimization

Algorithm 3.1 Maximum Likelihood Estimation

Assume you have iid data $y_1, \dots, y_n$, under a statistical model described by density or mass function $f(y;\theta)$ for some parameter θ with p components.

Then

1. Characterize the joint density/mass as a product where "[write it out]" stands for using the mathematical expression for the density or mass within the product.

$$f(y_1, \dots, y_n; \theta) = \prod_{i=1}^{n} f(y_i; \theta) = [\text{write it out}]$$

2. Relabel as likelihood $L(\theta)$, distribute the product through so it becomes a sum (when possible) and simplify as much as you can.

$$L(\theta) = [\text{what you wrote out, but simplified with } \textstyle\sum \text{ instead of } \prod]$$

3. Take a logarithm and simplify even further. Look for constants c representing (possibly complex) additive terms that are not a function of θ.

$$\ell(\theta) = \log L(\theta) \qquad \text{where log means ln} = \log_e$$

4. Take the gradient with respect to θ, formed of partial derivatives over its components $\theta = (\theta_1, \dots, \theta_p)$.

$$\nabla \ell = \left(\frac{\partial \ell}{\partial \theta_1}, \dots, \frac{\partial \ell}{\partial \theta_p} \right)$$

5. Establish the likelihood equations[a] by setting that gradient to zero and simplifying.

$$0 \overset{\text{set}}{=} \nabla \ell \Big|_{\theta = \hat{\theta}}$$

6. Solve those likelihood equations, which you can do one of two ways.
   - One at a time for each θ_j, for $j = 1, \dots, p$, in any order.
   - Or simultaneously for all components of $\hat{\theta}$ at once.

Return $\hat{\theta}$.

[a] https://en.wikipedia.org/wiki/Likelihood_function#Likelihood_equations

of the other chapters of this book. Sometimes it's possible to mathematically deduce what that distribution is, especially when the statistic in question is based on a sum or average (which is a scaled sum) or is an MLE. Sometimes that deduction is exact, i.e., you get a closed-form expression, as I'll show you momentarily. Other times it is asymptotic, which is coming later.

Either way, the approach begins with expressing the statistic $s_n = s(y_1, \dots, y_n)$ as a random variable $S_n = s(Y_1, \dots, Y_n)$. We've done this before, both for MC sampling and for calculating means and variances to argue that our estimates are unbiased and efficient [§3.1]. This time we're going to try to identify the entire distribution, first for Bernoulli coin tossing and then for Gaussian heights.

In both of those examples we shall take our statistic to be the estimate of a parameter describing the distribution, like $\hat{\mu}$ or $\hat{\theta}$. When viewing that estimate as a function of random $Y_1, \ldots, Y_n$, it gets a new name: estimator[20]. I know that's confusing; don't shoot the messenger. The estimate is the value you can calculate from data. The estimator is a procedure for what you would do when that data arrives, and which we study as a random variable derived from random $Y_1, \ldots, Y_n$.

The distribution of the statistic or estimator is known as a *sampling distribution* and is what leads to a p-value. Recall that a p-value is the probability of obtaining a value of the statistic, under its sampling distribution, that is more extreme than the quantity calculated from observed data. Therefore, the p-value is an integral (for sampling distributions that are densities) or sum (for masses). I shall illustrate this concretely via our two examples.

Toss up p-value analytically

A binomial distribution[21] characterizes the sum of n Bernoulli draws. Don't take Wikipedia's word for it though. I all but derived it in §1.2. Eq. (1.3) began with the joint probability $\mathbb{P}(Y_1 = y_1, \ldots, Y_n = y_n)$, ultimately providing an expression that could be distilled down to the sufficient statistic $s_n = \sum_i y_i$. You can check that its form is within a multiplicative constant of the mass provided on that Wikipedia page. That multiplicative constant is known as the binomial coefficient[22] $\binom{n}{s_n} = n!/s_n!(n - s_n)!$. We have that

$$n\hat{\theta} = S_n \sim \text{Bin}(n, \theta). \tag{3.7}$$

So one can study the sampling distribution of $\hat{\theta}$ through the distribution of S_n/n. Figure 3.1 provides a visual for the coin-flipping experiment where the null hypothesis is specified as $\theta = 0.5$, a fair coin. The reflected value of the statistic utilizes $\mathbb{E}\{Y\} = n\theta$ under a Binomial distribution, so that $\mathbb{E}\{\hat{\theta}\} = \theta$ since $\hat{\theta}$ is unbiased.

```
sgrid <- 0:n
theta <- sgrid/n
sfun <- stepfun(theta[-1], dbinom(sgrid, n, 0.5))
plot(sfun, xlim=c(0.1, 0.9), xlab="theta", ylab="mass Binom(theta*n, n)",
  main="", lwd=2)
abline(v=c(that, (n - s)/n), lty=1:2, col=2, lwd=2)
legend("topleft", c("that", "reflect"), lty=1:2, col=2, lwd=2, bty="n")
```

Observe how this view is similar to the one from Figure 1.1, so the conclusion will be similar, too. A p-value adds precision. This is easier to express as a sum over possible s_n-values, since those are integers, than as θ-values. Let r denote the reflected value of s_n, and $f(s; n, \theta)$ represent the mass implied by Eq. (3.7). Then,

$$p\text{-value:} \quad \psi = \sum_{s=0}^{r} f(y; n, \theta) + \sum_{s=s_n}^{n} f(y; n, \theta), \quad \text{where } f \text{ is the binomial mass.} \tag{3.8}$$

This assumes the statistic s_n is in the right tail, and its reflection is in the left one. The bounds of those sums are reversed if the observed statistic is in the other tail.

[20] https://en.wikipedia.org/wiki/Estimator
[21] https://en.wikipedia.org/wiki/Binomial_distribution
[22] https://en.wikipedia.org/wiki/Binomial_coefficient

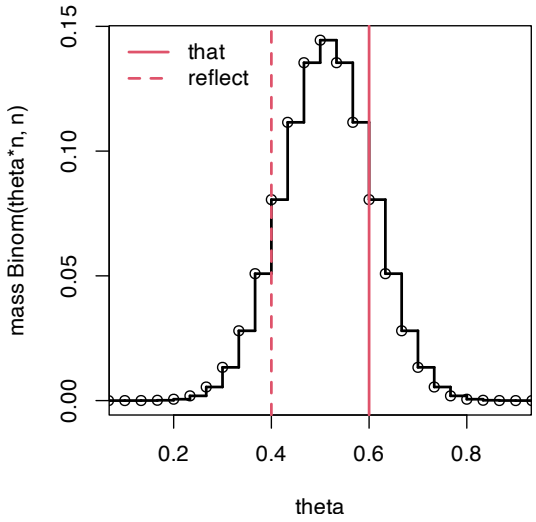

FIGURE 3.1: Exact $\theta = \frac{1}{2}$ sampling mass characterizing the distribution for S_n corresponding to a Bernoulli/binomial analysis of coin flips from Chapter 1.

In R code, Eq. (3.8) goes like this. For mass functions (like the binomial), `p*` functions (like `pbinom` below) sum up masses provided by analogous `d*` functions (like `dbinom`). We'll talk a little more in generality about this soon when we get to the Gaussian example.

```
pbinom(n - s, n, 0.5) + pbinom(s - 0.5, n, 0.5, lower.tail=FALSE)
```

```
## [1] 0.3616
```

Note that for the calculation in the right tail it's important to subtract off something in $(0, 1)$ – I did 0.5. Anything small will do. This is just a fluke of the way that the `pbinom` function is implemented in R. Without that, the calculation is off of what `binom.test` provides, which you can check against results in §1.4.

Quick digression on precision. It's easy to be off a bit by missing little details, both here in the "math" calculation, and also when doing MC. But that doesn't mean you're not doing it right. Here the issue boils down to how strict inequalities (like $<$ and $>$) are used in `p*` calculations, versus those that include $=$ (like $\leq$ and $\geq$). I try not to dwell too much on these edge cases. At the same time, I'd like to show you what's needed to match what software libraries provide, at least as far as possible. Ideally MC calculations would match up exactly too, but there are many factors that make that difficult. One is the MC error inherent to calculations with pseudo-random numbers (§A.3). That can be mitigated with bigger N. Other discrepancies can be explained by arbitrary choices (both by me and by the authors of those libraries).

The particular edge case in question here is actually more nuanced than what I have described above. The "fix" only works in this particular example, and others very similar to it. There are more such considerations – more edges – required to cover issues that arise in greater generality. I'll show you another one in just a moment. You can have a look at how those are handled in R by inspecting the `binom.test` function. (Entering `binom.test` at the command prompt dumps the code.) Such is the beauty of open source software, although

the code isn't documented as well as I wish it was. You won't get much of an explanation for why those adjustments are important.

Regardless, it is the process and transparency in calculations that is paramount, not the precision of the final answer. Any borderline case, like a p-value near $\alpha = 0.05$, or any other α, demands greater scrutiny. That might mean collecting more data, or it might mean thinking more about how the experiment is designed, what distributions are assumed, or what the right hypotheses are. If you're curious, I have some more thoughts along these lines in the Preface[23] to the book.

Alright, back to our example. Whether you find that easier or harder than MC perhaps depends on if you prefer math or computing. The mathematical solution is elegant, but I think the computing version is more intuitive, as it is faithful to the experience of "what if" modeling behind hypothesis tests. From a mathematical perspective, this is about as easy as it gets. For computing, the difficulty level is "medium". You'll recall from Chapter 2 that just about the same code works there for Gaussian samples. You'll see momentarily that the math for that case is more challenging.

As a final variation, consider the case of $\mathcal{H}_0 : \theta = 0.8$, as entertained with `monty.bern` via `pval.discrete` in §1.4. Recall that I described a rule determining the value of the reflected statistic, which was inspired by `binom.test`. Here's that rule played out with calculations described above. We still have that $S_n \sim \text{Bin}(n, \theta)$, but now $\theta = 0.8$. Our test statistic is the same too, $s_n = 0.6$, but now it's in the left tail. In R, and in `pval.discrete`, the reflected value is determined by the location of the largest mass point from the opposite tail (from the right) which is less than the mass corresponding to s_n. That is …

```
ms <- dbinom(s, n, 0.8)                      ## mass of s from data
mgrid <- dbinom(sgrid, n, 0.8)               ## mass of all s in 1:n
ti <- which(sgrid > 0.8*n & mgrid <= ms)     ## all right tail w lower mass
ri <- ti[which.max(mgrid[ti])]               ## index of largest
r <- sgrid[ri]                               ## extract reflection
```

Figure 3.2 provides a visual of the sampling distribution, the test statistic, and its two-tailed reflection. Notice how the view is similar to Figure 1.3, but this time based on math rather than MC. (Also, the x-axis is labeled differently, for s_n rather than θ.)

```
sfun <- stepfun(theta[-1], c(dbinom(sgrid, n, 0.8)))
plot(sfun, xlim=c(0.5, 1), xlab="theta", ylab="mass Binom(theta*n, n)",
  main="", lwd=2)
abline(v=c(that, r/n), lty=1:2, col=2, lwd=2)
legend("bottom", c("that", "reflect"), lty=1:2, col=2, lwd=2, bty="n")
```

The p-value calculation follows Eq. (3.8), but with different r and θ, and with r and s flipped because tails are reversed.

```
pbinom(s, n, 0.8) + pbinom(r - 0.5, n, 0.8, lower.tail=FALSE)
```

```
## [1] 0.01073
```

[23] https://bobby.gramacy.com/hipp0/index.html

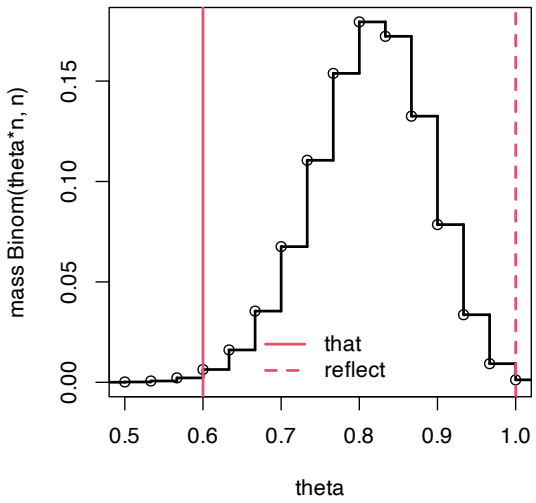

FIGURE 3.2: Exact $\theta = 0.8$ sampling mass, otherwise following the setup in Figure 3.1.

Check that this matches what was provided by `binom.test` at the end of Chapter 1. It's interesting, I think, that the hard part of this test is the calculation of the reflection. Deriving (and sampling from) the sampling distribution is the easy part.

Location p-value analytically

A sum of independent Gaussians is Gaussian[24]. This means that for $Y_i \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$, for $i = 1, \dots, n$, aggregate $\sum_i Y_i$ is Gaussian and thus so is $\hat{\mu} = \bar{Y} = \frac{1}{n}\sum_i Y_i$. Moreover, Gaussians are special in that their moments are in 1:1 correspondence with parameters μ, σ^2, and *uniquely determined* by those values. So if we know the mean and variance (i.e., the first and second moments), and we know that the distribution is Gaussian, then we know all we need to know to fully characterize the distribution. Regurgitating Eqs. (2.3) and (2.4), we have

$$\hat{\mu} = \bar{Y}, \quad \mathbb{E}\{\bar{Y}\} = \mu, \quad \text{and} \quad \mathbb{V}\text{ar}\{\bar{Y}\} = \frac{\sigma^2}{n}. \quad \text{Therefore} \quad \bar{Y} \sim \mathcal{N}(\mu, \sigma^2/n). \tag{3.9}$$

Rehashing: we know the sampling distribution of our estimator $\hat{\mu}$, which is the same as our chosen test statistic $S_n \equiv \hat{\mu}$. In the Gaussian analysis of heights data from §2.2 we presumed that σ^2 was known. We now have a way to estimate σ^2 via MLE, but hold that thought for a moment. I'll come back to it. For this discussion, and to make a direct contrast to what we got (by MC) in Chapter 2, it's better to first presume σ^2 known and keep the value we used earlier: $\sigma^2 = 6.25$. That way, we have everything we need to make a comparison between observed $\bar{y} = 66.62$ and its sampling distribution (3.9). Figure 3.3 has that visual.

```
ygrid <- seq(61.5, 67.5, length=1000)
plot(ygrid, dnorm(ygrid, mu, sqrt(sigma2/n)), type="l",
  xlab="ybar = muhat", ylab="density N(64.5, 2.5^2/n)")
abline(v=muhat, col=2, lwd=2)
legend("topleft", "observed", lty=1, col=2, lwd=2, bty="n")
```

[24] https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables

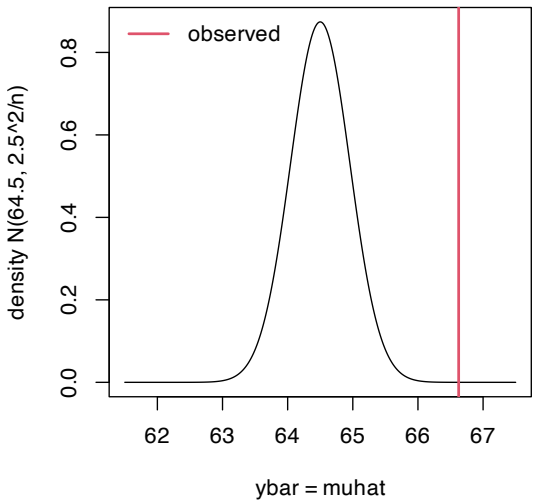

FIGURE 3.3: Exact sampling density characterizing the distribution for $\bar{Y}$ corresponding to the Gaussian analysis of heights data from Chapter 2.

See how similar things look here compared to the MC analog in Figure 2.2. Labeling on the y-axis is different and the histogram looks more discrete, but shapes and positioning on the x-axis (which is for $\bar{Y}$) are similar. In both cases, the observed value of the statistic (vertical red line) is far from the bulk of the distribution, nudging us toward rejecting the null hypothesis.

Having a Gaussian sampling distribution makes it easy to be precise with a p-value calculation. We don't need to "reflect" the observed statistic onto the other side of the density, because Gaussians provide symmetry. Once we know the p-value for one tail, we can double it to get a two-sided value.

The mathematical way to calculate the area under a curve – like we did by summing under the histogram or masses earlier – is to integrate[25]. Specifically in the Gaussian heights setting:

$$p\text{-value:} \quad \psi = 2 \times \int_{\bar{y}}^{\infty} f(y; \mu, \sigma^2/n)\, dy, \quad \text{where } f \text{ is the Gaussian density.}$$

Note that this assumes that $\bar{y} > \mu$ from the null hypothesis; otherwise, we must work in the left/lower tail with $\int_{-\infty}^{\bar{y}}$ instead. The computation required is not something that can be done with pen and paper unfortunately, but it can be done numerically in R via quadrature[26].

```
ny <- length(y) ## this is n from Chap 2, clashing with binomial n
pval <- 2*integrate(dnorm, muhat, Inf, mean=64.5, sd=sqrt(sigma2/ny))$value
pval
```

```
## [1] 3.125e-05
```

[25] https://en.wikipedia.org/wiki/Integral
[26] https://en.wikipedia.org/wiki/Numerical_integration

Be aware that `sigma2` was set in Chapter 2. Notice that this is exactly the same value we got from the `z.test` in §2.4. In a way, our MC alternative can be seen as a form of quadrature. MC integration[27] is a topic in and of itself, but largely for another book.

For important distributions like the Gaussian, R has a built-in way to numerically integrate densities. In fact, you may recall from a first class in probability that such an integral (or sum) is known as cumulative distribution function (cdf)[28]. Each generating function in R, beginning `r*` like `rnorm`, has a density version `d*` like `dnorm`, and a cdf version beginning with `p*` like `pnorm`. We can do:

```
2*pnorm(muhat, 64.5, sqrt(sigma2/ny), lower.tail=FALSE)
```

```
## [1] 3.125e-05
```

By default, `pnorm` and other cdf functions in R prefer to integrate in the lower tail, from $-\infty$ to the statistic $\hat{\mu}$, which is how a cdf is usually defined. But if you provide `lower.tail=FALSE`, they instead do $\hat{\mu}$ to ∞.

So why is this called a z-test? Because z is the letter people have settled on for standard normal deviates: $Z \sim N(0,1)$. Back in the day, textbooks included "z-tables" in the back so you could look up p-values (cdf evaluations) under a standard Gaussian and other distributions. Some still do, but I think that's old-fashioned now that we have R. Before you could use one of those tables for your particular test, you had to calculate a z statistic, sometimes called the standard score[29]. The standard score centers and re-scales your $\hat{\mu} \equiv \bar{y}$ under the null hypothesis by subtracting off mean μ and dividing by standard deviation $\sqrt{\sigma^2/n}$.

$$z = \frac{\bar{y} - \mu}{\sigma/\sqrt{n}} \qquad \text{with} \qquad Z \sim \mathcal{N}(0,1) \tag{3.10}$$

The quantity in the denominator $\sigma/\sqrt{n}$, which is the square root of the variance of the estimator, has a special name: standard error[30] or "se". When viewing $\bar{y}$ as a random quantity $\bar{Y}$, we obtain a standard Gaussian Z.

One way to view a z statistic, and many of the other test statistics I shall introduce later, is: estimate ("est") minus "want to know" ("wtk") divided by standard error.

$$\text{stat} = \frac{\text{est} - \text{wtk}}{\text{se}} \tag{3.11}$$

The "wtk" comes from the null hypothesis. Here is the z statistic for our heights analysis.

```
se <- sqrt(sigma2/ny)
z <- (muhat - 64.5)/se
z
```

```
## [1] 4.164
```

One nice thing about a z statistic, especially if you don't love looking up old-school tables, is that you can do a quick-and-dirty check by comparing to ± 2. If z is bigger than two in

[27] https://en.wikipedia.org/wiki/Monte_Carlo_integration
[28] https://en.wikipedia.org/wiki/Cumulative_distribution_function
[29] https://en.wikipedia.org/wiki/Standard_score
[30] https://en.wikipedia.org/wiki/Standard_error

absolute value, then reject $\mathcal{H}_0$ at the $\alpha = 0.05$ level. Alas, about 95% of a Gaussian density likes within $[-2, 2]$.

Paired with z is another special letter, now in Greek: $\phi(y) \equiv f(y; \mu = 1, \sigma^2 = 1)$. The integral of that density is $\Phi(z) = \int_{-\infty}^{z} \phi(y)\, dy$ so that

$$p\text{-value:} \quad \psi = 2 \times \Phi(-|z|). \tag{3.12}$$

The minus absolute-value part ensures a left-tail calculation, and $2\times$ makes it two-tailed by symmetry. If you took stats in the 20th century, you would look up $-|z|$ in a Φ table, but with R we can use its built-in numerical integration capability.

```
2*pnorm(-abs(z))
```

```
## [1] 3.125e-05
```

Same as before. An interesting side-fact, however, is that `pnorm` uses a table lookup to some extent, leaning on pre-computed "quadrature supports" that can be quickly linearly interpolated[31]. I thought you'd like to know. At least *you* don't have to do it; the computer does it for you. If you were a really clever undergrad in the 20th century you'd photocopy the table in the back of the book so that you didn't have to keep flipping back and forth. What are you going to do with all the time freed up by not having to do all that?

Infamous t-test

Brace yourself. This is going to be a long, but important "example".

You may have found it unsatisfying to fix σ^2. Our hypothesis test (2.2) focused on the location parameter μ, but since we also used a fixed σ^2 value in the analysis – whether in Chapter 2 or just above in the previous example – we were implicitly folding a value of σ^2 into the null hypothesis. Ideally, $\mathcal{H}_0$ would focus only on what's important. If only μ is of interest, our calculations should be agnostic to anything else, like σ^2.

I just showed you how to estimate σ^2, via MLE (3.5) or after bias correction (3.6). Let's stick with the latter going forward. Yet with focus on μ, it's not ideal to just "plug-in" s^2 everywhere that σ^2 is used. Two reasons: (1) $\hat{\mu} = \bar{y}$ is buried in s^2, so we're using the same info twice; (2) s^2 is just the setting that makes the observed data most probable, modulo bias correction. There are many other settings that could explain the data (almost) as well. It would be better to use s^2 for σ^2 in a way that acknowledges that a diversity of values are plausible, and that also compensates for the presence of $\hat{\mu}$ within s^2.

Quick digression. I'm *not* aiming for a sampling distribution for s^2. We'll look at things like that later, in Chapter 6, when testing for variances is of interest. Quantity σ^2 is a nuisance, not a main object of inquiry in this chapter. I want to make it vanish, in a sense. The statistical way to do that is to marginalize[32] it out by averaging over all plausible values. I wish I could show you the Bayesian[33] version, which would be more natural, but that's a whole different book.

Alright, back to what we *are* going to do. I'll teach you a little about sums-of-squares under a Gaussian distribution, which isn't too bad, and I'll wave my hands audibly when

[31] https://en.wikipedia.org/wiki/Linear_interpolation
[32] https://en.wikipedia.org/wiki/Marginal_distribution
[33] https://en.wikipedia.org/wiki/Bayesian_statistics

we get to the hard part. There's a very important result here that I want to get to, because I need to use it lots in the book. I'll use it even while avoiding math in favor of MC. It's so fundamental. Perhaps the most unsatisfying bit here is that we're going to do a lot of work, but it's not (usually) going to result in a different outcome. The reason for that is coming later in §4.1. But we'll feel better that we're making a full accounting of uncertainty. Or at least I will.

Ok, here we go. A really important identity that's clearly relevant to our discussion is written in the equation below. You might not have thought of it on your own, but it's not that hard to show. So I've left it for you as a homework exercise.

$$\sum_{i=1}^{n} \frac{(Y_i - \mu)^2}{\sigma^2} = \frac{(n-1)s^2}{\sigma^2} + \frac{(\bar{Y} - \mu)^2}{\sigma^2/n} \tag{3.13}$$

This is useful for two reasons. One is that it relates quantities we don't know, or don't want to commit to (σ^2), to those that can be calculated from data (s^2). The other is that (I'll show you) we know the distributions of two out of the three terms, so we can use that to deduce the third. Here we go.

If $Y_i \sim \mathcal{N}(0,1)$, then $Y_i^2 \sim \chi_1^2$ where χ_ν^2 is a chi-squared (χ^2) distribution[34] with ν degrees of freedom (DoF)[35]. This isn't something I need to prove to you. Think of it as a definition. Someone worked out some hard math to derive the density for Y_i^2 when Y_i is a standard Gaussian and decided to name it after a Greek letter. Go figure. I'm not going to show you the density function, because it's not important to what we're doing. You could easily simulate from a χ^2. Just draw $Y_i \sim \mathcal{N}(0,1)$ and square it. It's easier to use `rchisq` directly, and we're going to do that in just a moment. Before that, I want to generalize things a little.

The first way is pretty simple: if $Y_i \sim \mathcal{N}(\mu, \sigma^2)$ then $(Y_i - \mu)^2/\sigma^2 \sim \chi_1^2$. After standardizing, you get the standard normal result. Look back at the final term in Eq. (3.13). Since $\bar{Y} \sim \mathcal{N}(\mu, \sigma^2/n)$, I get to use this result immediately.

$$\frac{(\bar{Y} - \mu)^2}{\sigma^2/n} \sim \chi_1^2 \tag{3.14}$$

One way to interpret this, that's useful for understanding later, is that $\bar{Y}$ is worth one DoF ($\nu = 1$). That's one "piece of information", after all the y_i's are added up. I think of DoF as PoI, but I'm not going to use that acronym because I'm not that cavalier. I'll stick to DoF.

Here is the second generalization I promised you. If you have $Y_1, \ldots, Y_n \overset{\text{iid}}{\sim} \mathcal{N}(0,1)$, then $\sum_{i=1}^n Y_i^2 \sim \chi_n^2$. You can consider this to be an extension of the definition of χ_ν^2 that I talked about above. Basically this means you can add up χ_1^2's, accumulating DoF. And if we have $Y_1, \ldots, Y_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$ more generally, then

$$\sum_{i=1}^{n} \frac{(Y_i - \mu)^2}{\sigma^2} \sim \chi_n^2. \tag{3.15}$$

So $Y_1, \ldots, Y_n$ are worth n DoF; n pieces of information. That makes sense. Eqs. (3.14) and

[34] https://en.wikipedia.org/wiki/Chi-squared_distribution
[35] https://en.wikipedia.org/wiki/Degrees_of_freedom_(statistics)

(3.15) provide two of the three terms in Eq. (3.13), so we can solve for the last one through "additivity" of χ^2's. Referring back to Eq. (3.13),

$$\chi^2_n = [\chi^2_{n-1}] + \chi^2_1,$$

where the term in brackets is the one we're looking for. This is where, technically speaking, I'm waving my hands a little because proving that is beyond our reach. That's why "additivity" is in quotes. In case you're curious, the result is a consequence of Cochran's theorem[36]. The key part of it, in brackets, can be rearranged to give the result we need about σ^2 by solving across the $\sim$ as if it were an $=$:

$$\frac{(n-1)S^2}{\sigma^2} \sim \chi^2_{n-1} \quad \Rightarrow \quad \sigma^2 \sim \frac{(n-1)s^2}{\chi^2_{n-1}}. \tag{3.16}$$

In other words, n pieces of information is equal to $n-1+1$ pieces of information. Burying $\bar{y}$ inside s^2 costs one DoF because those data, $y_1, \ldots, y_n$, appear twice: once as each y_i and once as $\bar{y}$. The χ^2_{n-1} for s^2, after some scaling, corrects (or accounts for) this double-use by subtracting of one DoF for $\bar{y}$.

Figure 3.4 illustrates how the distribution for σ^2 in Eq. (3.16) compares to s^2. I use a sample of σ^2 values, and form an empirical distribution via histogram. That sample will be reused momentarily to expand our z-test.

```
sigma2s <- (ny - 1)*s2/rchisq(N2, ny - 1)
hist(sigma2s, main="", xlim=c(0, 20))
abline(v=c(s2, sigma2), lty=1:2, col=2, lwd=2)
legend("topright", c("s2", "sigma2"), lty=1:2, col=2, lwd=2, bty="n")
```

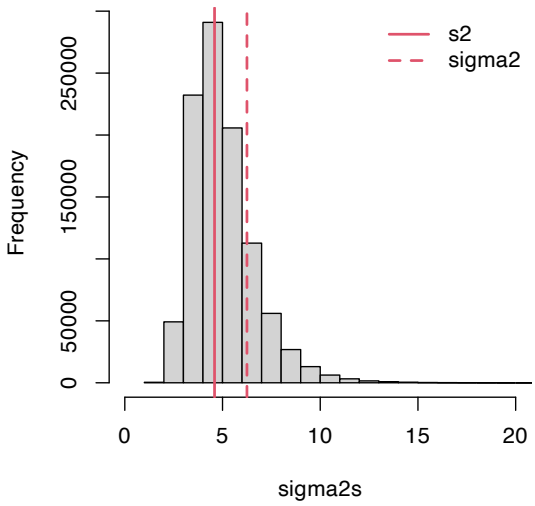

FIGURE 3.4: Distribution for σ^2 in the heights example from Chapter 1.

Quantity s^2, indicated by the solid red vertical line, is just the "central" value. There are many other nearby settings (both smaller and larger) that are reasonable. The same can be

[36] https://en.wikipedia.org/wiki/Cochran's_theorem

said for the value of σ^2 (dashed red line) used for the test in §2.2, tacitly part of $\mathcal{H}_0$. Using just one setting – any setting, but especially ones in the left tail of the distribution – will undercut any assessment of uncertainty about the main quantity of interest: μ. The value of σ^2 that I fixed for convenience in Chapter 2.2 is bigger than a substantial proportion of sampled σ^2 values.

```
mean(sigma2s < sigma2)
```

```
## [1] 0.8138
```

It may have been overly conservative; we were overestimating uncertainty in that analysis, although that doesn't take into account the values as large as twenty (or larger) in the right-hand tail in Figure 3.4. But there's no reason to wonder when we can *do*! It's easy to use these samples as part of the MC from §2.3.

```
Ss <- rep(NA, N2)
for(i in 1:N2) {
  ## sigma2s[i] <- (ny - 1)*s2/rchisq(1, ny - 1)   ## if not done above
  Ys <- rnorm(ny, mu, sqrt(sigma2s[i]))            ## only real change here
  Ss[i] <- mean(Ys)
}
```

I'm not going to show you a histogram of those samples, because it looks very similar to Figure 2.2. But check out the p-value calculation.

```
pval <- mean(Ss < 2*mu - muhat) + mean(Ss > muhat)  ## muhat=s from Ch 2
pval
```

```
## [1] 6.8e-05
```

Indeed, we get a smaller p-value because, on the whole, we're using smaller values of σ^2 now than earlier. What would happen with a z-test with s^2 plugged in instead? This time, I'll hit the easy button and use a library.

```
z.test(y, mu=mu, sigma.x=sqrt(s2))$p.value
```

```
## [1] 1.187e-06
```

Notice how this value is quite a bit smaller. Using s^2, while central, misses accounting for the possibility of much larger σ^2 values under its χ^2_{n-1} distribution.

Accounting for all reasonable σ^2 values in a z-test yields a t-test[37] and, like the z-test, that name will take some explaining. An English statistician named William Sealy Gosset[38], who worked as head brewer for Guinness[39], published a paper about some of the work he was doing for his employer under the pseudonym Student. Some of that work was about the form of Gaussian random variables sampled under a chi-squared variance. Specifically, Gosset was interested in

[37] https://en.wikipedia.org/wiki/Student's_t-test
[38] https://en.wikipedia.org/wiki/William_Sealy_Gosset
[39] https://en.wikipedia.org/wiki/Guinness

$$f_t(y) = \int_0^\infty f_\phi(y; \mu, \sigma^2) \times f_{\chi^{-2}}(\sigma^2; \nu)\, d\sigma^2, \tag{3.17}$$

where f_ϕ is the density of $\mathcal{N}(\mu, \sigma^2)$ and $f_{\chi^{-2}}$ is the $(n-1)s^2$ scaled inverse-χ_ν from Eq. (3.16). The technical term for forming a density by integrating over one of its parameters as a random effect[40] is called convolution[41]. By sampling σ^2 values and plugging them into a Gaussian generator, tens of thousands of times and then averaging, we have implemented a MC integral by convolution.

Gosset showed that the solution to this integral has an analytic closed form, and $f_t(y)$ is now known as the (density of the) Student-t distribution[42]. (Some say "Student's-t", but I think that's harder to verbalize and to write.) Using the form suggested by Eq. (3.17), this distribution has parameters: μ specifying location or mean, σ^2 for scale, and DoF ν. Together ν and σ^2 determine the variance of the random variable. Synthesizing $Y \sim \mathcal{N}(\mu, \sigma^2)$ and $\sigma^2 \sim (n-1)s^2/\chi^2_\nu$, one might write $Y \sim \mathrm{St}(\nu; \mu; s^2)$. I'm not going to bore you with the mathematical form of its density, since we don't need it for anything, but I will help you visualize it momentarily.

Like in a z-test, which uses a standardized Gaussian statistic, a t-test uses the Student-t in standardized form: $T \sim t_\nu \equiv \mathrm{St}(\nu; 0, 1)$ via:

$$t = \frac{\bar{y} - \mu}{s^2/\sqrt{n}} \qquad \text{with} \qquad T \sim t_\nu. \tag{3.18}$$

Notice how that follows the same generic test statistic setup as in Eq. (3.11): same estimate $\hat{\mu}$, same value $\mu = 64.5$ from $\mathcal{H}_0$, but different standard error $\sqrt{s^2/n}$. Viewing $\bar{Y}$ as a random quantity yields random T which is distributed as t_ν.

Contrasting a Gaussian with a Student-t is subtle. Figure 3.5 provides a visual based on standardized versions. Two variations on t_ν are provided. One, in dashed red, sets $\nu = n-1$ with particular n-value taken from our heights analysis ($n = 24$). The other shows $\nu = 5$, a more exaggerated DoF.

```
ztgrid <- seq(-5, 5, length=1000)
plot(ztgrid, dnorm(ztgrid), type="l", xlab="z or t",
    ylab="density N & St", lwd=2)
lines(ztgrid, dt(ztgrid, ny - 1), lty=2, col=2, lwd=2)
lines(ztgrid, dt(ztgrid, 5), lty=2, col=3, lwd=2)
legend("topright", c("Gauss", paste0("St_nu=", ny - 1), "St_nu=5"),
  lty=1:3, col=1:3, lwd=2, bty="n")
```

Observe how a Student-t has lower density for central values, near zero, and higher density in its tails. This is especially so when the DoF parameter ν is small, but it's also true in general for any ν-value. Eventually, however, as $\nu > 30$ or so, it's really hard to tell the difference between a t_ν distribution and a Gaussian one. But with a small data set – a small n – there could be a big difference. Since that difference manifests as heavier tails, a test based on a Student-t sampling distribution generally provides a more conservative

[40] https://en.wikipedia.org/wiki/Random_effects_model
[41] https://en.wikipedia.org/wiki/Convolution_of_probability_distributions
[42] https://en.wikipedia.org/wiki/Student's_t-distribution

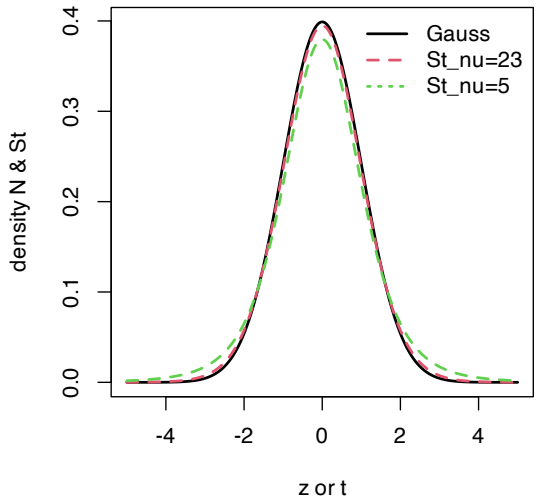

FIGURE 3.5: Contrasting ϕ and $t_{\nu=\{23,5\}}$ densities.

(i.e., larger) p-value, making one less likely to reject $\mathcal{H}_0$. It always yields a larger p-value compared to a z-test with $\sigma^2 = s^2$ plugged in.

Another way to contrast Gaussian and Student-t distributions is to look at their 95% quantiles.

```
q <- c(0.025, 0.975)                    ## central 95% is 2.5% and 97.5%
rbind(gauss=qnorm(q), t=qt(q, ny-1))
```

```
##         [,1]  [,2]
## gauss -1.960 1.960
## t     -2.069 2.069
```

The t_ν gives a wider 95% interval because it has heavier tails. But notice that, at least for $\nu = 23$, they both hover about the same amount around ± 2. So like with z-tests, $|t| > 2$ makes for a good threshold to quickly test a statistic and decide on $\mathcal{H}_0$ without bothering with a p-value.

Here's what we get for the t-test on the heights data.

```
se <- sqrt(s2/ny)            ## same as z, but with sigma2 = s2
t <- (muhat - 64.5)/se       ## same as z, different se above
t
```

```
## [1] 4.858
```

Like with the z-test, this is an easy reject. But, just to check that everything lines up with our earlier, MC setup, I'll finish with a p-value calculation. Again, symmetry of the Student-t simplifies a two-tailed integral.

```
pval.exact <- 2*pt(-abs(t), df=ny - 1)
pval.exact
```

```
## [1] 6.638e-05
```

Notice that this is very close to the MC alternative introduced first. That number is repeated below, along with the result from R's built-in `t.test` library function.

```
c(mc=pval, exact=pval.exact, lib=t.test(y, mu=mu)$p.value)
```

```
##        mc     exact       lib
## 6.800e-05 6.638e-05 6.638e-05
```

Now you know how to do a t-test, perhaps the most famous (or infamous) statistical test there is. It's really the same thing as a z-test, but averaging over the distribution of variances σ^2, rather than just plugging one in.

3.3 Confidence intervals

A confidence interval (CI)[43] can be thought of as the inverse of a hypothesis test. Let $\hat{\theta}$ generically denote an estimate (e.g., MLE) for a parameter θ of interest. Suppose you were to pick an α such that you would reject $\mathcal{H}_0 : \theta = \theta_0$ if your p-value satisfied $\psi(\theta_0, \hat{\theta}) < \alpha$. Then a CI is defined by the following set

$$\text{CI} = \{\theta_0 : \psi(\theta_0, \hat{\theta}) \geq \alpha\}.$$

Said another way, a CI contains all values θ_0 such that $\psi(\theta_0, \hat{\theta}) > \alpha$, i.e., θ_0's for which you would not reject $\mathcal{H}_0$. The bounds of that interval, which are what usually gets reported, can be identified by $\psi(\theta_0, \hat{\theta}) = \alpha$.

Technically that is just one definition of a CI; there are others which are less directly related to p-values. The one I've described is a *central* CI consistent with a two-sided hypothesis test. Perhaps the biggest issue with that definition is that it doesn't immediately tell you how to calculate one. As might not surprise you, I'm going to give you two ways: (1) by MC, which always works; and (2) one using math, which is sometimes doable, and sometimes not so much.

You can form a central CI (or just CI going forward) by revisiting the MC for a hypothesis test and plugging in the estimate of the parameter of interest $\hat{\theta}$ for the null θ. Use that value to simulate new $\tilde{\theta}(Y_1, \ldots, Y_n)$'s, calculated in the same way as $\hat{\theta}$ from the observed data, just with simulated observations. Once you're done with the simulation, report the lower $q_{\alpha/2}$ and upper $q_{1-\alpha/2}$ empirical quantiles[44] of that sample. So if you take $N = 10{,}000$ MC samples and $\alpha = 0.05$, report the 250th smallest and 250th largest (9,750th smallest) samples of $\hat{\theta}$ as your 95% CI.

I think that verbal description is sufficient and that we don't need a formal Algorithm environment, especially if I augment with a couple of examples.

[43] https://en.wikipedia.org/wiki/Confidence_interval
[44] https://en.wikipedia.org/wiki/Quantile

Toss up CI via MC

First for θ in the coin-flipping experiment.

```
thats <- rep(NA, N)
for(i in 1:N) {
  Ys <- rbinom(n, 1, that)              ## use that=ybar instead of theta
  thats[i] <- mean(Ys)                  ## use Ybar
}
CImc.theta <- quantile(thats, c(0.025, 0.975))
CImc.theta
```

```
##   2.5%  97.5%
## 0.4333 0.7667
```

The explanation here, or for any CI, is that there's a 95% chance that the true, but unknown θ value is inside this interval: $[0.43, 0.77]$. That is, when you make an interval like this 100 times, with 100 different observed data sets, about 95 of them will contain the true θ. We say that there's a 95% chance that the CI covers[45] the true value of the parameter. There's a little caveat, here, that I'll delve more into when we discuss the closed-form version, momentarily. If you prefer, you can explain the CI in the context of a hypothesis test. For any value of θ lying inside CI, we would fail to reject $\mathcal{H}_0$; for any value outside, we could reject at the specified α level.

Location CI via MC

Here the focus is on μ from the Gaussian heights example. I'll skip straight ahead to the full variance-integrated (3.17) version of the sampling distribution behind the t-test. We'll come back to z's later. Cutting-and-pasting that MC with minor modification …

```
muhats <- rep(NA, N2)
for(i in 1:N2) {
  Ys <- rnorm(ny, muhat, sqrt(sigma2s[i]))    ## use muhat instead of mu
  muhats[i] <- mean(Ys)
}
CImc.mu <- quantile(muhats, c(0.025, 0.975))
CImc.mu
```

```
##  2.5% 97.5%
## 65.72 67.53
```

So there's a 95% chance that the true, but unknown μ lies inside: $[65.72, 67.53]$. For any value of μ lying inside CI, we would fail to reject $\mathcal{H}_0$.

How do you calculate a CI analytically, using math rather than simulation? Well, that can be both more difficult, or easier depending on the situation. There's no generic approach like with MC. Everything is on a case-by-case basis. It turns out that the two cases we've been playing with so far have exact solutions that aren't too hard to work out.

[45] https://en.wikipedia.org/wiki/Coverage_probability

More widely, many CI constructions are based on asymptotics[46], which is the subject of Chapter 4. This involves reducing calculations to approximations based on Gaussian distributions and z statistics. So pay close attention when we get to the location CI example, because we'll be reusing that again later, and lots.

But first, coin flips. Actually, the standard CI for θ in a Bernoulli experiment is based on asymptotics, and I'll show you that later. I've always found strange that this is the case, because it can be done exactly in closed form.

Toss up CI analytically

Eq. (3.7) says that sums $S_n \sim \text{Bin}(n, \theta)$. So just plug in $\theta = \hat{\theta}$, extract 2.5% and 97.5% quantiles of that distribution, and divide them by n to get an interval for θ. The quantile function is the inverse of the cdf. Give it a probability, and it'll return the value of the cdf that achieves that probability.

```
CI.theta <- qbinom(c(0.025, 0.975), n, that)/n
rbind(mc=CImc.theta, exact=CI.theta)
```

```
##          2.5%  97.5%
## mc     0.4333 0.7667
## exact 0.4333 0.7667
```

This is identical to what we got by MC earlier. But it's not exactly the same as what R provides.

```
CI.theta.R <- as.numeric(binom.test(s, n)$conf.int)
CI.theta.R
```

```
## [1] 0.4060 0.7734
```

The reason for this is alluded to, but not entirely explained by, R's documentation for `binom.test`, which says (paraphrasing): CIs are obtained by a procedure first given in Clopper and Pearson (1934). I don't know about you, but I find this to be unsatisfying, almost 100 years later. Surely there's a better explanation.

I'll give it a go, based on my limited understanding of the situation from other reading. One issue is that a quantile-based CI can be problematic with a discrete distribution. The distribution may not have a quantile at exactly 2.5% or 97.5%, say. This is true for our particular binomial, whose quantiles are given by the following probabilities.

```
ps <- pbinom(1:n, n, that)
ps[ps > 0.001 & ps < 0.999]       ## focus away from the tails
```

```
##  [1] 0.002854 0.008302 0.021240 0.048112 0.097057 0.175369 0.285496
##  [8] 0.421534 0.568910 0.708528 0.823714 0.905989 0.956476 0.982817
## [15] 0.994341 0.998490
```

That being the case, one might wonder what that quantile command actually provided. Which of those percentiles did it use? More importantly, did the interval actually cover 95%?

[46] https://en.wikipedia.org/wiki/Asymptotic_theory_(statistics)

```
c(hand=diff(pbinom(CI.theta*n, n, that)),
  R=diff(pbinom(CI.theta.R*n, n, that)))
```

```
##   hand      R
## 0.9347 0.9616
```

Our by-hand one does not, and therefore neither does our MC one. Those are too small, but not by much. Our interval "under-covers" because the probability is below 95%. R's is more conservative. This is just something to be aware of. It's definitely fixable, by being more specific about the desired outcome. For example, if we said that the lower bound of the CI should be defined by the closest percentile below 2.5%, and the upper above 95%, then we could report the following.

```
lw <- qbinom(max(ps[ps <= 0.025]), n, that)/n
up <- qbinom(min(ps[ps >= 0.975]), n, that)/n
CI.theta2 <- c(lw, up)
CI.theta2
```

```
## [1] 0.4000 0.7667
```

This is much closer to R's, but not identical. (I'll leave it to you as an exercise to fix the MC interval similarly.) This new interval's coverage is, interestingly, identical even though it the interval itself is slightly different.

```
diff(pbinom(CI.theta.R*n, n, that))
```

```
## [1] 0.9616
```

Clopper and Pearson (1934) must be looking at something else. Appropriate coverage is only one piece of the puzzle. If I'm being honest here, this is getting pretty advanced, and you may even feel things are going off the rails. I won't ask you to pay this much attention to detail on your homework, but I do think it's important to be aware of.

A real problem arises when you observe zero heads. This isn't likely when the true θ is near $1/2$ and when n, the number of flips, is large. But it's not impossible. If n is small, you can get all tails in $n = 6$ flips more than one in 100 times. The trouble is that the procedure I've explained, either by MC or closed-form calculation, breaks down in that case because $\hat{\theta} = 0$. Check it out.

```
qbinom(c(0.025, 0.975), 6, 0)/6
```

```
## [1] 0 0
```

Having a confidence interval span 0 to 0 is nonsense. You'd get that for any n if there were no heads. This is a situation where the math breaks down, because the sampling distribution is degenerate. It provides no reasonable percentiles.

```
as.numeric(binom.test(0, 6)$conf.int)
```

```
## [1] 0.0000 0.4593
```

Everything covers 100%. I see this as a little like the denominator problem in $\hat{\sigma}^2$, and getting zero variance for $n = 1$. This was a problem noticed long ago, and there are several methods for correcting it. I'll show you another one later in §4.2. Clopper and Pearson (1934) provide one way of doing that.

I'm not going to explain it to you, because it would not be a fruitful digression. But look how big that interval is. It nearly covers half of the space of possible θ's. Suffice it to say that the mechanism is like a hedge – a sensible one in my opinion. If you took a small sample, say a blood sample testing for human immunodeficiency virus (HIV), you wouldn't want to conclude that HIV was non-existent in your population, with essentially zero variability, based on just a limited number of tests. You can read more about this here[47], and we'll revisit some of it in §4.1 around Eq. (4.6). As long as you have some "heads", and your $\hat{\theta} \neq 0$, then you don't have to worry about it.

Location CI analytically

Recall from Eq. (3.10) that $Z = \sqrt{n}(\bar{Y} - \mu)/\sigma \sim \mathcal{N}(0, 1)$ assuming σ^2 is fixed at a known value. For any Gaussian Z, we can find quantiles $q_{\alpha/2}$ and $q_{1-\alpha/2}$ bracketing the central $100 \times (1 - \alpha)\%$ of the distribution. That is,

$$1 - \alpha = \mathbb{P}(q_{\alpha/2} < Z < q_{1-\alpha/2}).$$

For example …

```
alpha <- 0.05
q <- qnorm(c(alpha/2, 1 - alpha/2))
q
```

```
## [1] -1.96  1.96
```

In this particular Gaussian case, we can write $q_p = \Phi^{-1}(p)$. Recall that the standard Gaussian cdf is notated as Φ.

Now, write $Z = \sqrt{n}(\bar{Y} - \mu)/\sigma$, and rearrange terms on both sides of both inequalities.

$$\begin{aligned} 1 - \alpha &= \mathbb{P}\left(q_{\alpha/2} < \frac{\bar{Y} - \mu}{\sigma/\sqrt{n}} < q_{1-\alpha/2}\right) \\ &= \mathbb{P}\left(\bar{Y} + \frac{\sigma}{\sqrt{n}} q_{\alpha/2} < \mu < \bar{Y} + \frac{\sigma}{\sqrt{n}} q_{1-\alpha/2}\right) \end{aligned}$$

So to build an interval for μ, which captures its value within its bounds $100 \times (1 - \alpha)\%$ of the time, use

$$\text{CI} = \left[\bar{y} + \frac{\sigma}{\sqrt{n}} q_{\alpha/2}, \bar{y} + \frac{\sigma}{\sqrt{n}} q_{1-\alpha/2}\right], \quad \text{or equivalently} \quad \bar{y} \pm \frac{\sigma}{\sqrt{n}} q_{1-\alpha/2}. \tag{3.19}$$

The last expression works because $-q_{1-\alpha/2} = q_{\alpha/2}$ due to symmetry.

It's worth reiterating the right interpretation here, because it's easy to get wrong. This interval, when we build it over-and-over again with different $\bar{Y}$ values calculated from Gaussian $Y_1, \ldots, Y_n$, will capture the true μ value 95% of the time if $\alpha = 0.05$. There's a hypothetical

[47] https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval

frequency argument behind many classical statistical methods. A MC alternative virtualizes that frequency hypothetical in computer code, actually repeating things over and over again.

Using $\bar{y} \equiv \hat{\mu}$ from the heights example, one can build the following CI.

```
CIz.mu <- muhat + q*sqrt(sigma2/ny)
CIz.mu
```

```
## [1] 65.62 67.63
```

Notice that I don't need the $\pm$ here because `q`, calculated above, is already both positive and negative. There's a perfect match with the library calculation.

```
as.numeric(z.test(y, sigma.x=sqrt(sigma2))$conf.int)
```

```
## [1] 65.62 67.63
```

Notice that I don't need to specify a `mu` argument if I only care about the CI. The quantity μ, which is unknown (but might be a "want to know"), does not feature in Eq. (3.19). The CI is telling you what μ's could not be rejected as null hypotheses.

Here is a helpful, generic version of Eq. (3.11) for CIs:

$$\text{CI} = \text{est} \pm q \times \text{se}, \tag{3.20}$$

where "est" is the estimated parameter, q is/are the quantile(s) of its distribution and "se" is the standard error. You could use this with a Student-t distribution, and s^2-based standard error, in order to average over σ^2 values. Use `qt` to get t quantiles similarly. I've left that for you as a homework exercise. Don't forget to check your work via `t.test(...)$conf.int`.

You might be worried that this CI does not actually achieve 95% coverage, because I made a big deal about that with the coin-flipping example. But don't worry, things are just fine.

```
diff(pnorm(CIz.mu, muhat, sqrt(sigma2/ny)))
```

```
## [1] 0.95
```

Bingo. Working with a continuous, and symmetric, distribution is much easier.

3.4 What is inference?

The title of this chapter is "A little math: inference". What does that word mean?[48]. I'm sure you can get a lot of perspectives on that depending on who you ask, whether two different statisticians, or one statistician and one machine learner. I don't want to put my foot in my mouth on that.

I think it's uncontroversial that statistical inference comprises at least three things: (1)

[48] https://en.wikipedia.org/wiki/Statistical_inference

dialing in parameter settings; (2) summarizing uncertainty about those parameters; and (3) parlaying estimates and uncertainties into conclusions or predictions. For (1), I like MLEs but there are plenty of other options, and many choices amount to more or less the same thing. For (2), I'm thinking both CIs and testing hypotheses. We talked about how those are related to/inverses of one another, and both amount to ruling out (or not) certain values for the unknown quantity as a function of the estimate (e.g., MLE) and its sampling distribution. For (3), it's about establishing a causal[49] link, or extrapolating or interpolating to novel experimental conditions. This book doesn't have as much of that, but we will dabble from time to time, especially with prediction.

There are even more intimate connections between MLEs, confidence intervals, and hypothesis tests. These are the subject of Chapter 4. In early drafts of this book, the contents of Chapter 4 were merged with Chapter 3, because both involve techniques for inference. But I realized that a single combined chapter would simply have been too long. It makes sense to stop halfway, digest, take stock, and get practice with some exercises.

3.5 Homework exercises

These exercises help gain experience with the math and closed-form calculations behind statistical inference carried out by Monte Carlo in earlier chapters, and with t-tests and confidence intervals, which are new to this chapter.

Ensure that you are using methods outlined in this chapter. Many students will have experience working with closed-form Bernoulli/binomal tests, t-tests, and others (like Poisson, exponential, etc.) in other contexts, particularly via approximation (i.e., converting to z-tests) from previous classes, other books, or materials found online. Those will be discussed and evaluated in later chapters and homeworks. Deploying non-Chapter 3 methods here will likely not earn full credit.

Do these problems with your own code, or code from this book. In some cases you can check your work, e.g., with with `t.test`*. However, check with your instructor first to see what's allowed. Unless stated explicitly otherwise, avoid use of libraries in your solutions. Throughout, I suggest you keep it simple and double a one-sided p-value calculation for two-tailed tests.*

#1: Poisson

Let $Y_1, \ldots, Y_n$ be iid Poisson[50] with unknown mean θ, i.e., $Y_i \overset{\text{iid}}{\sim} \text{Pois}(\theta)$. *Use Wikipedia to find facts about this distribution, like the form its mass, etc., but the idea is to show your work. Poissons are used for counts of events that happen in a certain period of time. Note that on Wikipedia, $\lambda \equiv \theta$ and $k \equiv y$.*

a. Derive the maximum likelihood estimator $\hat{\theta}$ for θ. What is the sufficient statistic for θ?
b. Derive the sampling distribution for $\hat{\theta}$ in closed form. *Hint: you may wish to use this fact[51] about sums of Poisson random variables.*

[49] https://en.wikipedia.org/wiki/Causal_inference
[50] https://en.wikipedia.org/wiki/Poisson_distribution
[51] https://en.wikipedia.org/wiki/Poisson_distribution#Sums_of_Poisson-distributed_random_variables

Use the data values for $y_1, \ldots, y_n$ given in R below to answer the remaining parts. These data are a count of the number of buses that stop at fifty randomly selected stops in a one-hour time period.

```
y <- c(3, 5, 6, 5, 2, 6, 6, 7, 8, 8, 7, 8, 0, 5, 7, 6, 6, 10, 6, 5, 6, 7,
  6, 9, 8, 4, 8, 7, 11, 9, 4, 4, 7, 9, 8, 6, 5, 6, 12, 10, 7, 13, 8, 12,
  9, 4, 10, 8, 4, 5)
```

Hint: try `poisson.test(sum(y), length(y), theta)`*, but only to check your work.*

c. A city planner asserts that the typical number of buses that pass by any particular stop in a given hour is 6. Do these data support this? Use both empirical (MC) and closed-form/exact calculations. *Hint: this is a hypothesis testing question.*

d. Provide a (central 95%) CI for θ. Show both empirical (MC) and closed-form/exact calculations. *Advanced: does your CI have appropriate coverage?*

#2: Exponential

Let $Y_1, \ldots, Y_n$ be iid exponential[52] with unknown rate parameter θ, i.e., $Y_i \overset{\text{iid}}{\sim} \text{Exp}(\theta)$. *Use Wikipedia to find facts about this distribution, like the form of the density, etc., but show your work! The exponential distribution can be thought of as the "inverse" of a Poisson: the waiting time for one thing to happen. Note that on Wikipedia, $\lambda \equiv \theta$ and $x \equiv y$.*

a. Derive the maximum likelihood estimator $\hat{\theta}$ for θ. What is the sufficient statistic for θ?
b. Derive the sampling distribution for $\hat{\theta}$ in closed form. *Hint: you may wish to use a fact from here[53] about sums of exponential random variables.*

Use the data values for $y_1, \ldots, y_n$ given in R below to answer the remaining parts. These are "times-to-failure" in years obtained from an accelerated life test[54] of a certain new smart-TV.

```
y <- c(3.71, 17.70, 23.49, 9.60, 16.16, 4.94, 10.69, 14.62, 11.27, 1.99,
  10.95, 4.87, 14.87, 26.37, 0.78, 1.60, 9.21, 18.73, 18.85, 6.71, 11.45,
  4.02, 7.57, 14.75, 3.48, 10.03, 16.21, 25.94, 3.11, 17.61)
```

Hint: I was unable to find any appropriate add-on packages to help check work, so you're kinda on your own with this one.

c. The TV manufacturer claims that the typical lifespan of their new smart-TV is twenty years. Do these data support this? Use both empirical (MC) and closed-form/exact calculations. *(This is a hypothesis testing question.)*

d. Provide a (central 95%) CI for θ, and interpret that as a CI for the number of years until failure. Show both empirical (MC) and closed-form/exact calculations.

[52] https://en.wikipedia.org/wiki/Exponential_distribution
[53] https://en.wikipedia.org/wiki/Exponential_distribution#Sum_of_two_independent_exponential_random_variables
[54] https://en.wikipedia.org/wiki/Accelerated_life_testing

#3: Scale-1 Pareto

Let $Y_1, \ldots, Y_n$ be iid Pareto[55] with scale fixed at $\beta = 1$, and unknown shape parameter $\theta > 0$. Specifically, $Y_i \overset{\text{iid}}{\sim} \text{Pareto}(1; \theta)$, with density $f(y; \theta) = \theta / y^{\theta+1}$ for $y \geq 1$ and zero otherwise. *Note that on Wikipedia, the scale is notated as x_m, and I am using $x_m = \beta = 1$. Shape is notated as α, and I am using $\alpha = \theta$.*

a. Derive the maximum likelihood estimator $\hat{\theta}$ for θ. What is the sufficient statistic for θ?

b. I'm not going to ask you to derive the sampling distribution in closed form. I think this is one that's lots easier by Monte Carlo. But I'm giving it to you so you can compare, if curious. Malik (1970) says that $1/\hat{\theta} \sim \text{Gamma}(n-1, n\theta)$. The calculation is similar to #2b.

Use the data values for $y_1, \ldots, y_n$ given in R below to answer the remaining parts. These are insurance claims, in millions of dollars, for trucking companies operating in 2024.

```
y <- c(1.15, 1.26, 2.98, 1.41, 1.01, 1.16, 1.02, 1.38, 1.40, 1.20, 1.017,
 1.71, 1.91, 1.00, 1.95, 1.14, 1.29, 1.11, 1.14, 1.29, 1.06)
```

c. A re-insurer asserts that the mean claim in 2024 is \$1.5M, which corresponds to $\theta = 3$. Do these data support this? *(This is a hypothesis testing question. The `Pareto` package on CRAN provides `rPareto` for random deviates.)*

d. Provide a (central 95%) CI for θ.

#4: Sufficient statistics for Gaussian/location model

What are the sufficient statistics for the Gaussian model? *Hint: there are two.*

#5: Biased MLE $\hat{\sigma}^2$

Show $\mathbb{E}\{\hat{\sigma}^2\} = \frac{n-1}{n}\sigma^2$, with $\hat{\sigma}^2$ given in Eq. (3.5). You may use that $\bar{Y}$ is unbiased for μ and any other definitions from earlier in the chapter. *Hint: start on the left-hand side, and make manipulations to arrive at the expression on the right-hand side. Use properties of expectation and variance.*

#6: Sums of squares decomposition

Show (3.13). *Hint: start on the left-hand side, and make manipulations to arrive at the expression on the right-hand side. A helpful first step is to replace $Y_i - \mu$ with $Y_i - \bar{Y} + \bar{Y} - \mu$ and then expand out the square.*

#7: Update `monty.bern` with exact calculations

Update `monty.bern` from §1.4 to include the exact calculations covered in §3.2 and CI calculations from §3.3, including optional visuals. Check the `N` argument. Interpret `N=0` as meaning no MC sampling, and in that case use exact calculations instead of MC. That way the same library function can be used for both. Test this new capability by checking that results agree on the homework questions in §1.5.

[55] https://en.wikipedia.org/wiki/Pareto_distribution

#8: Revisit Bernoulli

Revisit any of Exercises #2 (silicone wafers), #3 (PVC or copper piping), or #4 (GPS speed vs. speedometer) from §1.5, but do it the "math way" with closed-form calculations. Additionally provide CIs, checking MC simulation matches closed-form/math results. *(This will be pretty easy if you already did #7 above.)*

#9: Library function for t-test

Write a library function like `monty.z` from §2.6 but for t-tests. Call it `monty.t`. Implement both MC versions and closed form/exact calculations, which may be invoked when `N=0` is provided. See #7 above. Include optional visuals when `vis=TRUE` in both cases, MC and exact, and return a similar `list` of relevant quantities to `t.test` including s^2 and a CI. Test it out and make sure it works.

#10: ROI with t-test

Revisit §2.6 #5–6 with a t-test, i.e., estimate s^2 and test $\mu = \{15, 17\}$, separately. Provide a CI for μ. Use both empirical (MC) and closed-form/exact calculations *(which is pretty easy if you did #9 first)*.

#11: One-liner

Revisit exercise #7 from §2.6 and update it to sample from a Student-t sampling distribution by replacing `sigma2` with an appropriate `rchisq` call, producing a one-liner MC t-test. *(It'll be a long one-liner.)*

4

A little math: asymptotics

In statistics, asymptotic analysis[1] is the study of properties of an estimator or statistic as the training data size gets large: $n \to \infty$. Alas, that's an unrealistic setting. We never get infinite data. Asymptopia is at once a pipe dream and also irrelevant. The population of women in the United States is finite, so it's total nonsense to ponder a sample of infinitely many women's heights.

Nevertheless, contemplating the behavior of estimators or statistics as n gets large is helpful. You want those quantities to have good properties if and when you can collect more data, and it's interesting to wonder what happens in the limit as you take that to an infinite extreme. Quantities with good "asymptotic properties", in the lingo, have another feather in their cap, like unbiasedness or small mean-squared error, connecting back to §3.1. They lead to new statistical procedures and insight. Estimators adjusted to have good asymptotic properties form the basis of many classical statistical tests – in many cases, ones that are easier to perform but also sometimes harder to intuit.

Here I'll review two key forms of asymptotics. I say "review" because I'm not going to be able to prove everything to you. That's beyond the scope of the book and, if I'm being honest, my own capabilities. Yet both are beautiful, if a little antique. They're not – in my not-so-humble opinion – really necessary if you have a computer. Yet they reveal important truths about common procedures and enable you to make calculations mostly by hand, which can be valuable in a pinch, like when your laptop battery dies, and you need a good answer right away.

These two forms of asymptotics are related to one another, and so they often lead to the same place. One is a simple truth about averages (more specifically about sums), and the other about maximum likelihood. MLEs are often sums or averages, so there you go. You may have studied the first one before, and there's no harm in review. I'm pretty sure the second one will be new to you.

4.1 Central limit theorem

Recall our discussion – a digression really – from §2.1 about the distinction between an average and a mean. They're not the same thing, but they are related. An average is a quantity that you can calculate from data, whereas a mean is what you expect from averages if you calculate them over and over again. I showed you that an average is unbiased for the mean. Alright, that's the context we're working in, generally speaking. Here we go with

[1] https://en.wikipedia.org/wiki/Asymptotic_theory_(statistics)

more specifics. The central limit theorem (CLT)[2] says even more about those averages as they relate to the mean (and to the variance).

Consider n observations $y_1, \ldots, y_n$. Assume that each Y_i, for $i = 1, \ldots, n$, is independent and identically distributed (iid) with mean μ and variance σ^2, that is, $\mathbb{E}\{Y_i\} = \mu$ and variance $\mathbb{V}\text{ar}\{Y_i\} = \sigma^2$. Notice that I'm being very careful here. I'm saying that each Y_i is independent of the others and comes from the same distribution, and that the *moments* of that distribution are μ and σ^2. What's even more important is what I'm *not* saying: what the distribution actually is. In particular, I'm not saying it's Gaussian. It could be Poisson, Bernoulli, gamma, exponential, Pareto, Bobby (once they decide to name a distribution after me)[3], or whatever. It doesn't matter. In fact, the CLT is most useful when you don't know the distribution, but it can still be useful even when you do.

The CLT says the following. Let $\bar{Y}_n = \frac{1}{n}\sum_{i=1}^{n} Y_n$ denote the average of n random Y_i values. Previously I used $\bar{Y}$ for this without the n subscript. Here n matters a lot to the statement of the CLT, so I'm indicating, with my average notation via $\bar{Y}_n$, how many Y_i-values are involved.

$$\bar{Y}_n \sim \mathcal{N}(\mu, \sigma^2/n) \quad \text{as} \quad n \to \infty. \tag{4.1}$$

Let me help you appreciate, and interpret, this amazing result. No matter what distribution your data come from, the distribution of their average is Gaussian. But they must be iid and you've gotta have lots.

You never have infinite n, so the CLT provides approximation for whatever $n \ll \infty$ you do have, and the approximation is better the bigger n is. Moreover, you have the same unbiasedness and learning efficiency, via σ^2/n, as you did in the Gaussian case in §2.1. Just interpret things approximately.

Before turning to examples, I want to talk about two variations or special cases. You'll often see the CLT quoted for sums instead of averages. Just multiply everything by n and you get $\sum Y_i \sim \mathcal{N}(n\mu, n\sigma^2)$ as $n \to \infty$. I think this version is less useful for statistical inference where averages play a key role, but it does have important applications. A more useful variation involves standardization, like I did for testing with z statistics in Eq. (3.10).

$$Z \equiv \frac{\bar{Y}_n - \mu}{\sigma/\sqrt{n}} \sim \mathcal{N}(0,1) \quad \text{as} \quad n \to \infty. \tag{4.2}$$

So all averages are standard Gaussian, with pdf ϕ and cdf Φ, if you center them with their mean μ and divide by the standard error $\sigma/\sqrt{n}$. This is an approximation, but it's accurate when n is large. In some sense, a consequence of this result is that every statistical test based on averages boils down to a z-test, approximately. To make use of this, you must know, or work out, what μ and σ^2 are for the distribution you assume generates the data.

Remember, you get to choose the test statistic in Alg. 1.1. If you choose to do it with averages, you can use the CLT. Another consequence of Eq. (4.2) is that quantiles from a Gaussian distribution can be used to form an approximate confidence interval (CI). Eq. (3.19) can be used, and will be exact when $n \to \infty$.

How good are these approximations for particular n? How big should n be to trust the CLT? Those are, in some sense, the same question. Unfortunately, there's no simple answer,

[2] https://en.wikipedia.org/wiki/Central_limit_theorem

[3] And for some reason choose (the diminutive of) my first name rather than my last one.

requiring case-by-case analysis. It's also too technical for us, and would take up way too much space. Besides, the point is moot. You wouldn't use a CLT-based z-test or CI if you could do an exact one, making it an option of last resort. Still, the calculations behind tests in many software libraries utilize the CLT directly, or a CLT-adjacent argument. I'm mostly showing it to you here so that you can take in this awesome fact about averages, and (more importantly) so you understand and can duplicate results that you see from software packages.

Toss up CLT

According to Wikipedia, a Bernoulli distribution[4] with parameter θ has mean $\mu = \mathbb{E}\{Y_i\} = \theta$ and variance $\sigma^2 = \mathbb{V}\text{ar}\{Y_i\} = \theta(1-\theta)$. Therefore, our test statistic has the following asymptotic sampling distribution.

$$\bar{Y}_n \sim \mathcal{N}(\theta, \theta(1-\theta)/n) \quad \text{as} \quad n \to \infty \tag{4.3}$$

A standardized version follows Eq. (4.2) similarly.

The trouble with this is that we don't know θ, so there's a chicken or egg problem when it comes to the variance of the sampling distribution. Having θ in the mean of the Gaussian is fine. That's where the quantity of interest, from the null hypothesis, usually is. Things are quirky with a Bernoulli distribution, because it has just one parameter θ, doing double-duty to determine both location and scale.

Fortunately, our willingness to work in an asymptotic framework helps here. The weak law of large numbers (WLLN)[5] says that

$$\bar{Y}_n \to \mu \quad \text{as} \quad n \to \infty \tag{4.4}$$

Why is this true? The short answer is because of Eq. (4.1). As $n \to \infty$ the variance $\sigma^2/n \to 0$ and the Gaussian becomes a point spike density centered at μ. The long answer, supporting theory for laws of large numbers more broadly, is a topic for another text. What does it mean for us? It means we don't change the asymptotics by slotting in $\hat{\theta} \equiv \bar{Y}_n$ where θ appears in the variance.

In other words

$$\bar{Y}_n \sim \mathcal{N}(\theta, \bar{Y}_n(1-\bar{Y}_n)/n) \quad \text{or} \quad \frac{\bar{Y}_n - \theta}{\sqrt{\bar{Y}_n(1-\bar{Y}_n)/n}} \sim \mathcal{N}(0,1) \quad \text{as} \quad n \to \infty \tag{4.5}$$

which means we can use a z-test for θ as follows.

```
se <- sqrt(that*(1 - that)/n)                ## that (=ybar) & n in Ch 1&3
z <- (that - 0.5)/se                         ## testing H0: theta = 0.5
p.clt <- 2*pnorm(-abs(z))
c(clt=p.clt, exact=binom.test(s, n)$p.val)   ## s defined in Chapter 1
```

```
##    clt  exact
## 0.2636 0.3616
```

[4] https://en.wikipedia.org/wiki/Bernoulli_distribution
[5] https://en.wikipedia.org/wiki/Law_of_large_numbers#Weak_law

These aren't really that close. What gives? I guess $n = 30 \ll \infty$, so the approximation isn't very good. Figure 4.1 provides a comparison of these two sampling distributions, augmenting the view in Figure 3.1. Note the change of variables[6] in the figure that arises from a transformation between s_n and θ_n. This is a little advanced, but you don't really need to worry about it. It's only really there for the visual.

```
sgrid <- 0:n
theta <- sgrid/n
sfun <- stepfun(theta[-1], c(dbinom(sgrid, n, 0.5)))
plot(sfun, xlim=c(0.1, 0.9), xlab="theta", ylab="mass Binom(theta*n, n)",
  main="", lwd=2)
abline(v=c(that, (n - s)/n), lty=1:2, col=2, lwd=2)
clt <- dnorm(theta, 0.5, sqrt(that*(1 - that)/n))/n  ## change of variables
lines(theta, clt, col=3, lty=3, lwd=2)
legend("topleft", c("that", "reflect", "clt"), lty=1:3, col=c(2,2,3),
  lwd=2, bty="n")
```

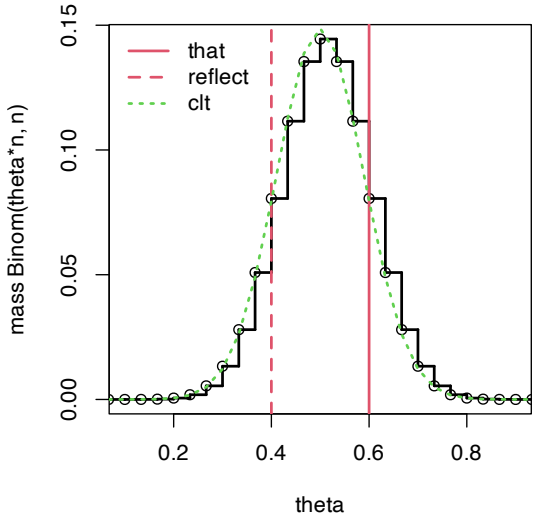

FIGURE 4.1: Extending Figure 3.1 to include a CLT-approximating sampling distribution. The factor `1/n` in the `clt` calculation arises from a transformation between `s` and `theta`.

That difference, between discrete and continuous, comprises at least some of the approximation error, which can be visualized in the figure as a gap between dotted green and solid black lines. The Gaussian CLT overestimates on the left, and under on the right. It is clear, however, that larger n, providing more steps in the discrete distribution, provides greater accuracy.

Completing the example, an approximate CI may be calculated as follows.

```
q <- qnorm(0.975)
CI.clt <- that + c(-1, 1)*q*se
rbind(clt=CI.clt, exact=binom.test(s, n)$conf.int)
```

[6] https://en.wikipedia.org/wiki/Probability_density_function#Scalar_to_scalar

```
##          [,1]   [,2]
## clt    0.4247 0.7753
## exact 0.4060 0.7734
```

Again, recall from §3.3 that the CI from `binom.test` is special because it has guardrails to cope with certain edge cases. Nevertheless, those CIs are in close agreement. This asymptotic CI goes by a special name, the Wald interval[7], and has known drawbacks, although those are not exhibited in this example. I'll let you read more on that Wikipedia page.

One correction that targets a degenerate case discussed in §3.3, when there are no "heads" and $\bar{y}_n = 0$ (or no "tails" and $\bar{y}_n = 1$), is known as an adjusted Wald interval (Agresti and Coull, 1998). At level α and using $q_{1-\alpha/2} = \Phi^{-1}(1-\alpha/2)$, an adjusted Wald interval can be interpreted as an interval formed from n data values plus four "hallucinated" y-values that contain two heads and two tails:

$$\tilde{\theta}_n \pm q_{1-\alpha/2}\sqrt{\tilde{\theta}_n(1-\tilde{\theta}_n)/n} \quad \text{where} \quad \tilde{\theta}_n = \frac{n}{n+4}\bar{y}_n + \frac{4}{n+4}\frac{1}{2}. \tag{4.6}$$

Adding those extra four flips doesn't change anything asymptotically. As $n \to \infty$ we have that $\tilde{\theta}_n \to \bar{y}_n$, so an adjusted Wald CI is the same as the Wald one, which converges to the exact interval under the CLT. Note that there isn't an $n+4$ in the denominator of the standard error because those "hallucinated" flips don't have real information in them, so you wouldn't want them to reduce the uncertainty or the size of the interval for fixed $n \ll \infty$.

```
ttilde <- (n/(n + 4))*that + (4/(n + 4))*0.5
setilde <- se <- sqrt(ttilde*(1 - ttilde)/n)
CI.awald <- ttilde + c(-1, 1)*q*setilde
CI.awald
```

```
## [1] 0.4121 0.7643
```

Having "imagined" flips helps guard against a degenerate $[0, 0]$ interval, but does not provide the same correction as the R library, due to Clopper and Pearson (1934).

```
ttilde <- (6/(6 + 4))*0 + (4/(6 + 4))*0.5
setilde <- se <- sqrt(ttilde*(1 - ttilde)/6)
CI.awald <- ttilde + c(-1, 1)*q*setilde
rbind(awald=CI.awald, clopper=binom.test(0, 6)$conf.int)
```

```
##             [,1]   [,2]
## awald   -0.1201 0.5201
## clopper  0.0000 0.4593
```

It's unsatisfying to get a CI for θ, a proportion in $[0, 1]$, that includes negative numbers. This is happening because $n = 6$ is very small compared to infinity. In such cases, a CLT approximation can be poor. However, if you report a version of the interval truncated to lie inside $[0, 1]$, you get a conservative approximation that is much better than reporting $[0, 0]$.

[7] https://en.wikipedia.org/wiki/Binomial_proportion_confidence_interval#Problems_with_using_a_normal_approximation_or_%22Wald_interval%22

```
if(CI.awald[1] < 0) { CI.awald[1] <- 0 }
if(CI.awald[2] > 1) { CI.awald[2] <- 1 }
CI.awald
```

```
## [1] 0.0000 0.5201
```

My preference would be to lean the Monte Carlo (MC) interval from §3.3, since I can control the accuracy. However, it could be sensible to borrow the adjusted $\tilde{\theta}_n$ estimator with its imaginary four flips. I've left exploring that to you as a homework exercise.

Location CLT

Maybe I could have skipped this example, but I wanted to continue the flow. It's nice to have two running examples. Their utility will soon run thin, but first I'd like to squeeze everything out.

When data are modeled as iid Gaussian, $Y_i \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$, we already know the sampling distribution. The one quoted by the CLT (4.1) is exact for any n, no asymptotics required. See Eq. (3.9) and more broadly §3.2. I showed you how the Gaussian becomes a Student-t when estimating σ^2, resulting in the infamous t-test and associated intervals (§3.3). Nothing new to add here.

But since I've got you, it's worth additionally remarking on one thing. The CLT is letting us know that, if n is large, we can approximate a t-test and CI with a z-analog. A similar WLLN argument gives that $\hat{\sigma}_n \to \sigma$ as $n \to \infty$, and likewise for s_n. See Eqs. (3.5) and (3.6), respectively. Therefore, $\bar{Y}_n \sim \mathcal{N}(\mu, \hat{\sigma}^2/n)$ as $n \to \infty$.

Back in the day, Student-t tables only had quantiles for a limited selection of ν-values. There would be some sort of note to use the Gaussian ones for larger ν. Recall that $\nu = n - 1$ for our location example. A CLT argument justifies that guidance. These days we can use `pt` and `qt`, like in R. So what I'm telling you is largely trivia.

As a practical matter, most folks don't bother with a Student-t when $\nu > 30$. (They often don't bother with z tables either, using the $|z| > 2$ rule to reject.) A rule of thumb of $n > 30$ is applied a lot more widely (i.e., beyond Gaussian modeling) by appealing to similar logic, regardless of the modeling distribution. Figure 4.1 looks pretty good for the Binomial, and in homework exercises in §4.4 you'll have a chance to look at some other examples.

4.2 Distribution of the MLE

The statistic you use with a hippopotamus is totally up to you. If you choose a sum or an average, then the CLT is your go-to, that is, assuming you don't have a computer handy or your mathematical chops don't lead you to a closed-form sampling distribution. Here I'll show you that if you limit yourself to MLEs, then you get a double automation benefit. The first is that with an MLE you know that you're estimating quantities optimally, in a sense. The second is this nice theorem, which I see as an extension of the CLT.

Let θ be a p-dimensional parameter to a family of distributions emitting log-likelihood $\ell(\cdot)$, and let $y_1, \ldots, y_n$ be modeled as iid from that family. Notice how I'm being coy here not to mention the actual distribution. Like the CLT, the beauty of this result lies in its general

applicability. Within reason, and with the details being a bit of a distraction, it doesn't matter what the distribution is. What matters is the log-likelihood and iid assumption.

Let $\hat{\theta}_n$ be the p-dimensional vector maximizing $\ell(\theta; y_1, \dots, y_n)$. You would find $\hat{\theta}_n$ by following the procedure outlined in Alg. 3.1, which involves taking partial derivatives. These are the likelihood equations:

$$\frac{\partial \ell}{\partial \theta_j}\Big|_{\hat{\theta}_n} = 0, \quad \text{for} \quad j = 1, \dots, p.$$

Then, under some technical conditions, that would (again) be a distraction,

$$\hat{\theta}_n \sim \mathcal{N}_p\left(\theta, \frac{1}{n} i_1^{-1}(\theta)\right) \quad \text{as} \quad n \to \infty. \tag{4.7}$$

Above, $\mathcal{N}_p$ is a p-variate multivariate normal distribution (MVN)[8], which is a vectorized extension of an ordinary Gaussian distribution. This concept is a little bit of a stretch for this book, but bear with me. Whereas a Gaussian is parameterized by its mean and variance, both scalars, an MVN is parameterized by a mean vector ($\mu = \theta$ above) and covariance matrix ($\Sigma = \frac{1}{n} i_1^{-1}(\theta)$). Notice that since the mean of the distribution for $\hat{\theta}_n$ is θ, this means that the MLE is asymptotically unbiased for the true, unknown parameter. Cool, right?

Quick digression for two notes on notation before explaining that i_1 thing. Many authors use a bold font for matrices and vectors, but I'm not going to do that here because we won't really need it. I'm only making a brief foray into vectorized probabilities. Don't panic. The main reason to introduce them here is to give you an impression of the bigness, and generality, of the result.

For now, just appreciate that Eq. (3.1) is similar to the CLT (4.1), but better, because it applies to any MLE and for all of its p parameters! The mean of the MVN is the true, but unknown parameter (i.e., $\hat{\theta}_n$ is asymptotically unbiased), and the covariance has an n in the denominator which means that you learn, having lower uncertainty, with more data. I'll say more about that in a moment.

My second note involves θ, which is pulling double duty as the *true*, unknown value that generated the data and as the free variable we differentiate with respect to in order to estimate $\hat{\theta}_n$. I hope you'll forgive me that transgression so that we can have things a little more streamlined by not introducing new quantities to keep track of. I'm betting you didn't notice until I said something, and you possibly still don't. This is mostly a disclaimer for my more pedantically minded colleagues.

Alright, back to that funny $i_1^{-1}(\theta)$. The (co-) variance of the derivative of the log-likelihood $\ell(\theta; Y_1, \dots, Y_n)$ is known as Fisher information (FI)[9]. Sometimes you'll hear it described as the variance of the score[10]. It turns out you can calculate this covariance via the Hessian[11]. Writing $Y \equiv Y_1, \dots, Y_n$ as a shorthand …

$$i(\theta) = \mathbb{C}\text{ov}\{\nabla \ell(\theta; Y)\} = -\mathbb{E}\{\nabla \nabla^\top \ell(\theta, Y)\}. \tag{4.8}$$

It's not too hard to show this is true, but I'd rather spend time pondering its meaning.

[8] https://en.wikipedia.org/wiki/Multivariate_normal_distribution
[9] https://en.wikipedia.org/wiki/Fisher_information
[10] https://en.wikipedia.org/wiki/Informant_(statistics)
[11] https://en.wikipedia.org/wiki/Hessian_matrix

The Hessian is a vectorized second derivative (second gradient). First derivatives tell you about slope, and second derivatives about curvature. So FI is expected curvature. Curvature tells you something about how "peaky" the function – in this case the likelihood – is. Peakiness and (co-) variance are one and the same thing, conceptually. For example, the smaller σ^2 for a Gaussian, the more peaked the bell curve. The more information you get from the data, measured by FI, the lower your uncertainty. So there should be an inverse relationship between FI and (co-) variance (4.8).

The most important thing about Eq. (4.8) is it explains how to calculate the variance of Gaussian approximation (4.7). An outer product[12] of $(p \times 1)$-vectors $(\nabla\nabla^\top)$ gives you a $p \times p$ matrix. You've already taken first derivatives when solving the likelihood equations. Just do it one more time. Here is another fact that helps, allowing me to complete Eq. (4.8). When data are iid, $\ell(\theta; y_1, \ldots, y_n) = \sum_{i=1}^n \ell(\theta; y_i)$. This is a logarithm turning products, via independence, into sums. In that case $i(\theta) = i_n(\theta) = ni_1(\theta)$ where i_1 is the FI of a single observation. Said another way, FI from one observation (e.g., Y_1) can be used to determine FI from all n observations. You don't have to hassle with any sums when taking second partial derivatives. That makes the bookwork a lot easier.

Examples are coming momentarily. But first, I want to put it all down procedurally in an Algorithm, augmenting Alg. 3.1 for the MLE. These are steps you would always do *after* finding $\hat{\theta}_n$.

Algorithm 4.1 Asymptotic Distribution of the MLE

Assume Steps 1–4 of Alg. 3.1 have already been carried out.

Then

7. Simplify to a one-sample version of the (partial) derivative of the log-likelihood $\nabla\ell_1 \equiv \nabla\ell(\theta; y_1)$.

$$\nabla\ell_1 = \left(\frac{\partial\ell_1}{\partial\theta_1}, \ldots, \frac{\partial\ell_1}{\partial\theta_p}\right)$$

8. Differentiate again with respect to all p-components, forming a Hessian.

$$\nabla\nabla^\top\ell_1 = \begin{pmatrix} \frac{\partial^2\ell_1}{\partial\theta_1\partial\theta_1} & \cdots & \frac{\partial^2\ell_1}{\partial\theta_1\partial\theta_p} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2\ell_1}{\partial\theta_p\partial\theta_1} & \cdots & \frac{\partial^2\ell_1}{\partial\theta_p\partial\theta_p} \end{pmatrix}$$

9. Interpret $\nabla\nabla^\top\ell_1 \equiv \nabla\nabla^\top\ell(\theta; y_1)$ as a random quantity by replacing y_1 with Y_1 and take its expectation with respect to the distribution of Y_1:

$$i_1(\theta) = -\mathbb{E}\{\nabla\nabla^\top\ell(\theta; Y_1)\}.$$

Return the asymptotic sampling distribution (4.7).

A few notes. The "Assume" part specifies Steps 1–4 of Alg. 3.1 because those are all that's required here. Presumably you also did Steps 5–6 calculate the MLE $\hat{\theta}_n \equiv \hat{\theta}$. You'll need that too, eventually. Step 7 involves inspecting the form of $\Delta\ell$ from Step 4 and dropping the sum, replacing y_i with $y \equiv y_1$. Step 8 is technically an integral, however, log-likelihoods are

[12] https://en.wikipedia.org/wiki/Outer_product

often a sum of different quantities involving y, which means you can just replace y-values (or Y-values) with $\mathbb{E}\{Y\}$ by linearity of expectations[13]. Step 10, which in generality would involve a matrix inverse, is just a reciprocal when θ is a $p = 1$ scalar, or when the $p \times p$ matrix $i_1(\theta)$ is diagonal.

Toss up asymptotic MLE

Eq. (3.2) provides an expression for the derivative of the log-likelihood. Reexpressing that quantity for a single observation y_1 serves as a jumping off point for Steps 7–9 in Alg. 4.1. There are some simplifications here since parameter θ is $p = 1$ dimensional. All gradients are scalar derivatives, and so the Hessian is scalar too.

$$
\begin{aligned}
\text{7: one-sample derivative} \qquad \ell_1'(\theta; y_1) &= \frac{y_1}{\theta} - \frac{1 - y_1}{1 - \theta} \\
\text{8: second derivative} \qquad \ell_1''(\theta; y_1) &= -\frac{y_1}{\theta^2} - \frac{1 - y_1}{(1 - \theta)^2} \\
\text{9: expectation} \qquad -\mathbb{E}\{\ell''(\theta, Y_1)\} &= \frac{\mathbb{E}\{Y_1\}}{\theta^2} + \frac{1 - \mathbb{E}\{Y_1\}}{(1 - \theta)^2} \\
&= \frac{\theta}{\theta^2} + \frac{1 - \theta}{(1 - \theta)^2} \\
\text{so} \qquad i_1(\theta) &= \frac{1}{\theta(1 - \theta)}
\end{aligned}
$$

Connecting back to Eq. (4.7), the sampling distribution is therefore

$$\hat{\theta} \sim \mathcal{N}(\theta, \theta(1 - \theta)/n) \quad \text{as} \quad n \to \infty.$$

This the same as Eq. (4.3) since $\hat{\theta} = \bar{Y}_n$. So the CLT and the asymptotic distribution of the MLE are the same, in this Bernoulli case. This happens because $\hat{\theta}_n = \bar{Y}_n$: same estimator or test statistic, same result for the asymptotic distribution.

One nice thing about the calculations above is that they're programmatic. Alg. 4.1 tells you exactly what to do. Notice how the two methods use similar, but not identical, information. For the CLT, you have to remember the mean and variance of Bernoulli random variables. (Not a big leap if you're using a computer connected to the internet.) To calculate the asymptotic distribution of the MLE, you must recall the Bernoulli likelihood, mean and some calculus. (Differentiation is a good transferable skill. Keep it fresh!)

It's nice when two disparate ideas lead to similar, or identical, conclusions. The same WLLN result (4.4) applies here, replacing θ with $\hat{\theta}$ for variance terms in order to test hypotheses (4.5) and construct CIs (4.6). It also means there's nothing more for me to do for this example, because I did it all already in §4.1.

Location asymptotic MLE

Continuing on from Eqs. (3.3) and (3.4), and working on both parameters μ and σ^2 simultaneously, we have …

[13] https://en.wikipedia.org/wiki/Expected_value#Properties

7: one-sample derivative
$$\frac{\partial \ell_1}{\partial \mu} = \frac{y_1 - \mu}{\sigma^2} \quad \text{and} \quad \frac{\partial \ell_1}{\partial \sigma^2} = -\frac{1}{2\sigma^2} + \frac{(y_1-\mu)^2}{2(\sigma^2)^2}$$

8: second derivative
$$\frac{\partial^2 \ell_1}{\partial \mu^2} = -\frac{1}{\sigma^2} \quad \text{and} \quad \frac{\partial^2 \ell_1}{\partial (\sigma^2)^2} = \frac{1}{2(\sigma^2)^2} - \frac{(y_1-\mu)^2}{(\sigma^2)^3}$$
$$\frac{\partial^2 \ell_1}{\partial \mu \partial \sigma^2} = -\frac{y_1 - \mu}{(\sigma^2)^2}$$

Next, collect everything in a matrix, using $\frac{\partial^2 \ell_1}{\partial \mu \partial \sigma^2} = \frac{\partial^2 \ell_1}{\partial \sigma^2 \partial \mu}$, and take expectations.

9: expectation
$$-\mathbb{E}\{\nabla \nabla^\top \ell(\theta, Y_1)\} = \begin{bmatrix} \frac{1}{\sigma^2} & \frac{\mathbb{E}\{Y_1\}-\mu}{(\sigma^2)^2} \\ \frac{\mathbb{E}\{Y_1\}-\mu}{(\sigma^2)^2} & -\frac{1}{2(\sigma^2)^2} + \frac{\mathbb{E}\{(Y_1-\mu)^2\}}{(\sigma^2)^3} \end{bmatrix}$$
$$\text{so} \quad i_1(\theta) = \begin{bmatrix} \frac{1}{\sigma^2} & 0 \\ 0 & \frac{1}{2(\sigma^2)^2} \end{bmatrix}$$

The last line, above, follows since $\mathbb{E}\{Y_1\} = \mu$ and $\mathbb{E}\{(Y_1 - \mu)^2\} = \mathbb{V}\text{ar}\{Y_1\} = \sigma^2$.

So what does this mean when plugging into Eq. (4.7)? Did we learn anything? The FI matrix is diagonal, and so then too is its inverse. This means that the components of $\hat{\theta} = (\hat{\mu}, \sigma^2)$ are uncorrelated with one another, and under a Gaussian means they're statistically independent. So they may be studied in isolation.

Looking just at the first component, we have $\hat{\mu} \sim \mathcal{N}(\mu, \sigma^2/n)$, as $n \to \infty$. We already knew that result was exact for any n; no asymptotics required. Looking at the second component,

$$\hat{\sigma}^2 \sim \mathcal{N}(\sigma^2, 2\sigma^4/n) \quad \text{as} \quad n \to \infty. \tag{4.9}$$

If we didn't know anything about χ^2_ν distributions, that could serve as an approximation. I didn't do this particular case in §3.2, focusing instead on s^2 in Eq. (3.16), but a similar calculation provides $\hat{\sigma}^2 \sim \frac{\sigma^2}{n}\chi^2_n$. Figure 4.2 offers a comparison of actual and approximate distributions for the – somewhat arbitrarily chosen – case of $n = 20$ and $\sigma^2 = 1$.

```
sigma2 <- 1
n <- 20
s2grid <- seq(0, 3, length=1000)
plot(s2grid, dchisq(s2grid*n, n)*n/sigma2, ## from change of variables
  type="l", lwd=2, ylab="density", xlab="s2hat")
lines(s2grid, dnorm(s2grid, sigma2, sqrt(2/n)), col=2, lty=2, lwd=2)
legend("topright", c("actual", "approx"), lty=1:2, col=1:2, lwd=2, bty="n")
```

Observe that the approximation is good, but not perfect. It would improve for larger n. I encourage you to try other σ^2 and n values.

I've shown you two examples of how a Gaussian, which is a distribution for real-valued random variables (both positive and negative), can approximate the distribution of two very different estimators. For coin tosses, it can approximate a discrete, binomial sampling distribution, where the estimator is a natural number[14] (non-negative integer) because it is

[14] https://en.wikipedia.org/wiki/Natural_number

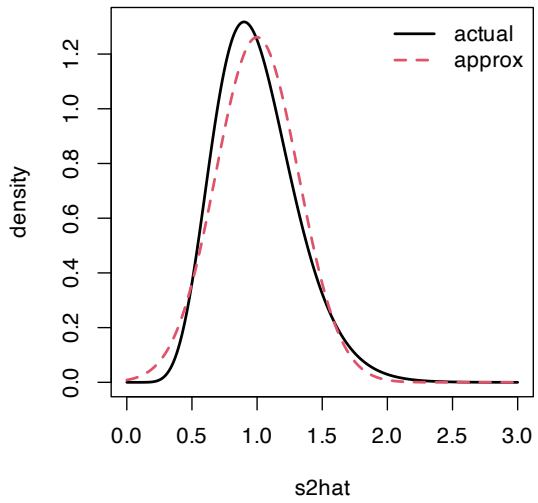

FIGURE 4.2: Comparing a Gaussian asymptotic approximation of the distribution of $\hat{\sigma}^2$ to the actual χ^2_n for $n = 20$ and $\sigma^2 = 1$. Factor `n/sigma2` in `dchisq` arises from a change of variables, scaling χ^2_n deviates by n/σ^2.

a sum of binary flips. For location data, it can approximate the distribution of a positive, variance quantity with a χ^2_ν distribution. If you're a nerd like me, you'll think that's pretty neat.

I want to make a final remark here on efficiency, mirroring a discussion from §2.2 where we studied uncertainty in estimators as a function of training data size, n. The Cramer-Rao lower bound (CRLB)[15] says that, under certain technical conditions that I won't bore you with, the variance of an unbiased estimator $\hat{\theta}_n$ of θ can be no smaller than the (squared) standard error of the asymptotic (Gaussian) sampling distribution of the MLE.

That is,

$$\mathbb{V}\text{ar}\{\hat{\theta}_n\} \geq \frac{1}{n} i_1^{-1}(\theta).$$

This is really cool! The CRLB says that the Gaussian sampling distribution (4.7) is efficient, at least asymptotically. It says you can do no better than $\frac{1}{n} i_1^{-1}(\theta)$, when it comes to uncertainty about $\hat{\theta}_n$, and Eq. (4.7) achieves that variance. Using the MLE gives you the optimal rate of learning for unbiased estimators, in a certain sense.

4.3 Asymptopia and Monte Carlo

There's an important distinction to make about asymptopia. Remember how I said in §1.3 that you get a better approximation with MC as $N \to \infty$. Here, with the CLT or asymptotic distribution of the MLE, I'm telling you that you get a better approximation with a Gaussian as $n \to \infty$. Both infinities are out of reach, past the end of the universe, but that doesn't mean they're the same thing.

[15] https://en.wikipedia.org/wiki/Cramér–Rao_bound

We have control over computational effort, N, in a MC simulation. So long as we're willing to wait a little longer, we can get a better approximation with bigger N. As statistical analysts, we almost never have control over n. The experiment has been performed, or the survey has been gathered, and no more data (bigger n) is coming any time soon. We're stuck with what we've got.

The CLT and approximate distribution of the MLE are beautiful results. For a long time they were very important, practical results. They saved a lot of hassle. For many tests, all you need is a z-table, and you're good to go. But times have changed. Lookup tables are out and computing is in. MC is intuitive, highly accurate (for large N), and as time goes on, will become even more powerful and easy to use. Computer hardware and software are getting better all the time.

Going forward, I'll do my best to navigate concepts with MC, while grounding outcomes by drawing comparisons to library output. Sometimes those libraries utilize exact, closed-form calculations. Other times they leverage asymptotics. I'll try to send a clear signal about which is which. When libraries leverage asymptotic results, it'll be hard to make a precise quantitative comparison to a MC alternative because both are approximations. However, I shall encourage you to draw comfort from the fact that MC can always be made to be more accurate with larger N. In my mind, that casts doubt on the precision of a calculation based on asymptotics. But that's only my opinion.

Taking things to extremes ($n \to \infty$) is a fine way to study a mathematical procedure, or to stress test an engineered system. Yet real decisions with real consequences are made from finite, and often limited information. When data are few (small n), statistical thinking and methodology shine the most. If it's done right.

4.4 Homework exercises

These exercises are more analytical than those in other chapters, although there's still some coding to do. Most questions involve revisiting exercises from §3.5 using the CLT or asymptotic distribution of the MLE.

At the risk of being too repetitive, use methods from this chapter, not previous chapters or other materials. Deploying non-Chapter 4 methods here will likely not earn full credit. Do these problems with your own code, or code from this book. I encourage you to check your work with library functions. However, check with your instructor first to see what's allowed. Unless stated explicitly otherwise, avoid use of libraries in your solutions.

#1: Poisson

Revisit exercise #1 from §3.5.

a. Approximate the sampling distribution for $\hat{\theta}_n$ using the CLT.
b. Now do it using the asymptotic distribution of the MLE.
c. Using data values provided in §3.5 #1, test the hypothesis in #1c using your results from parts #a and #b above.
d. Provide a 95% CI using your results from parts #a and #b above.

#2: Exponential

Revisit exercise #2 from §3.5.

a. Why might it be difficult to approximate the sampling distribution for $\hat{\theta}_n$ using the CLT? *(It's not impossible, but I would treat this as a challenge problem and move on to #b.)*
b. Approximate the sampling distribution for $\hat{\theta}_n$ using the asymptotic distribution of the MLE.
c. Using data values provided in §3.5 #2, test the hypothesis in #1c using your results from part #b above.
d. Provide a 95% CI using your results from part #b above.

#3: Scale-1 Pareto

Revisit exercise #3 from §3.5.

a. Why might it be difficult to approximate the sampling distribution for $\hat{\theta}_n$ using the CLT? *(It's not impossible, but I would treat this as a challenge problem and move on to #b.)*
b. Approximate the sampling distribution for $\hat{\theta}_n$ using the asymptotic distribution of the MLE.
c. Using data values provided in §3.5 #3, test the hypothesis in #1c using your results from part #b above.
d. Provide a 95% CI using your results from part #b above.

#4: Update `monty.bern` with asymptotic calculations

Update `monty.bern` from §1.4, and possibly also §3.5 #6, to include approximate Gaussian testing and CI calculations from this chapter, including optional visuals. Follow these specifications.

- Provide both ordinary *and* Wald CIs if the data have at least one head and one tail.
- Otherwise, provide an adjusted Wald interval, and be sure to note which interval is provided with your output.
- Update your MC CI implementation to use a modified $\hat{\theta}_n$ from the adjusted Wald interval in the case where the sample does not contain at least one head or tail.
- Check `N`, and if `N == -1` use an asymptotic approximation instead of MC or exact calculations. That way the same library function can be used for any of the methods you've learned so far.
- Test this new capability by checking that results are similar to those you got on earlier homeworks. *(Careful, they won't be identical.)*

#5: Revisit Bernoulli

Revisit any of Exercises #2 (silicone wafers), #3 (PVC or copper piping), or #4 (GPS speed vs. speedometer) from §1.5, but do it with an asymptotic approximation, including CIs. *(This will be pretty easy if you already did #4 above.)*

#6: Explore Wald adjustments

Study the adjusted Wald CI, and contrast it to the Clopper and Pearson (1934) interval provided by `binom.test` in R, when the observed data has all n heads, and no tails, for $n = 2, \ldots, 100$. Plot the upper and lower bounds for both intervals, on the same axes, as a

function of n. That's four lines on one plot (use color/line types). *(This is easier if you've already done #4 above.)*

5

Two samples

Sometimes, it's interesting to ask a question about data in a vacuum, comparing estimators to particular population quantities, meaning parameter values in a presumed distribution. Mostly that's a good warm-up in statistical textbooks. As a toolkit in practice, utility is limited. Far more often, data are collected from two or more populations which are then compared to one another to see if there's a difference. For example, in clinical trials we don't compare the effectiveness of a drug, given to patients (i.e., people) at large. Rather, we compare the treatment group (patients taking the drug) to a control group (taking a placebo), usually assigned at random[1].

There are many settings, outside of medicine, where treatment–control trials form a crucial tool of inquiry. One, recently popular variant is known as A/B testing[2]. IT companies like Meta, Amazon, and Google are performing A/B tests all the time, in order to see if certain content increases engagement or sales. In that context, the experimenter might randomly assign some users (or shoppers) one version (A) of a social media experience (or storefront), and another some other version (B) or a baseline. The idea is to see how changes to the experience affects behavior. Differences between one version or another might be as subtle as a choice of font or placement of an image.

Here, we'll begin our statistical exploration for data in this setting. I'll revisit this in later chapters as well, in different contexts. For now, focus is on just two samples. That'll be broadened to three or more later. Recall the ROI example from §2.5. I described one set of data collected from individual companies, and another aggregated by industry. We wished to know if those samples came from different populations, especially as regards their mean. We'll finally be able to answer that question in this chapter. You could view that in a treatment–control (or A/B) context as regards the software in question. Do businesses using that particular software have an ROI that is different than the industry at large?

Treatment/control analyses are a cornerstone of causal inference[3], which is an advanced topic that's beyond the scope of this book. What we're doing here is an important first step. I'll be careful to say things like "there's a difference" between the two populations, rather than attributing a causal relationship to some intervention, or treatment, on the population. In the parlance, we'll reject that the two samples come from the same population.

[1] https://en.wikipedia.org/wiki/Randomized_controlled_trial
[2] https://en.wikipedia.org/wiki/A/B_testing
[3] https://en.wikipedia.org/wiki/Causal_inference

5.1 Coin flips

Suppose you had data from two different campaigns of coin flips, possibly with the same coin. You're not interested in whether the coin(s) were fair. Rather, you wish to detect if the same coin was used in each campaign. Just to keep things tidy, I'll borrow coin flips from Chapter 1 for the first campaign. Recall that there were 18 heads in 30 flips.

```
sy <- sum(y)        ## number of heads
ny <- length(y)     ## number of coin flips
c(sy, ny)
```

```
## [1] 18 30
```

Notice how I'm labeling objects slightly differently here, as `sy` and `ny` rather than `s` and `n`. The reason for that will be apparent momentarily. Now, suppose a second campaign yielded the following result.

```
sx <- 19
nx <- 53
```

That is, out of 53 flips there were 19 heads. Is it likely that the same coin was used in both campaigns? That's our "want to know". Before framing that as hypotheses, choosing a statistic, and testing under the null – following Alg. 1.1 – the first step is to specify statistical models.

Re-notating a bit from Chapter 1, let $y_1, \dots, y_{n_y}$ denote the coin flips using the first coin. These are the same ones as in Chapter 1, but now I'm using $n_y \equiv n = 30$ to count the number of flips. An appropriate model is $Y_i \overset{\text{iid}}{\sim} \text{Bern}(\theta_y)$, which is also the same as before except $\theta_y \equiv \theta$. The sufficient statistic is now $s_y \equiv s = 18$, similarly.

We need similar notation for the second sample. Let $x_1, \dots, x_{n_x}$ denote a second set of flips, and $X_i \overset{\text{iid}}{\sim} \text{Bern}(\theta_x)$, with sufficient statistic $s_x = 19$. Summarizing, so it's easy to refer back to later, we have

$$\text{Model:} \qquad Y_1, \dots, Y_{n_y} \overset{\text{iid}}{\sim} \text{Bern}(\theta_y) \qquad \text{and} \qquad X_1, \dots, X_{n_x} \overset{\text{iid}}{\sim} \text{Bern}(\theta_x). \tag{5.1}$$

It is worth noting that we don't need $n_y = n_x$, although that could be the case. Models in hand, the "want to know" can be phrased as competing hypotheses as follows.

$$\begin{aligned} &\mathcal{H}_0 : \theta_y = \theta_x && \text{null hypothesis ("what if")} \\ &\mathcal{H}_1 : \theta_y \neq \theta_x && \text{alternative hypothesis} \end{aligned} \tag{5.2}$$

How to answer that question? Follow Alg. 1.1 and assume $\mathcal{H}_0$ is true, form the sampling distribution of a statistic of interest, and check for a probabilistic contradiction.

This leaves a lot of choices to the practitioner, although some are pretty automatic. For example, we know that (s_y, n_y) and (s_x, n_x) capture all that's needed in this modeling context about each sample. And we could choose to look at those numbers through the lens

of maximum likelihood estimation (MLE) $\hat{\theta}_y = s_y/n_y$ and $\hat{\theta}_x = s_x/n_x$. But perhaps most importantly is the choice of Monte Carlo (MC) versus closed-form or asymptotic calculation (i.e., the "math" way). Those will be explored in turn.

Two-sample Bernoulli test by MC

A natural statistic to look at for the test is $\hat{\theta}_\Delta = \hat{\theta}_x - \hat{\theta}_y$. Alternatively, one could look at $s_\delta = s_y - s_x$, but that choice has the downside that it doesn't take n_x and n_y into account. When they are equal ($n_y = n_x$) the two stats are equivalent. In that case, or generally for $\hat{\theta}_\Delta$, there is the advantage that the sampling distribution will be symmetric and centered at zero under the null: $\theta_y = \theta_x$, This helps with intuition. When $n_x \neq n_y$ all bets are off for s_Δ. Another reason to look at $\hat{\theta}_\Delta$ is that it connects better with the math version coming next.

```
that.d <- sy/ny - sx/nx
```

We must draw deviates from the sampling distribution under $\mathcal{H}_0 : \theta_x = \theta_y$. Let $\theta = \theta_x = \theta_y$ to make it easier to keep track of things. If a parameter takes on the same value, we don't need two different labels when one will do. Repeatedly, consider flipping n_y coins with probability θ of heads, and similarly n_x coins with the same θ. Of course, I don't mean to actually flip coins. Get your computer to do it virtually. In total, that's $n_y + n_x$ flips with probability θ of heads. Indeed, under the null hypothesis we have

$$Y_1, \ldots, Y_{n_y}, X_1, \ldots, X_{n_x} \overset{\text{iid}}{\sim} \text{Bern}(\theta). \tag{5.3}$$

This is sometimes referred to as a pooled[4] model for the pooled sample. But what θ? This is, again, a choice for the practitioner because that value is not pinned down in Eq. (5.2). Some choices will lead to tests with different power, which basically means differing ability to discern against the alternative. But I'm avoiding a discussion of power, formally speaking, in this book because it's pretty advanced. I've buried a brief intro to the back in §A.2. Rather, I'd prefer to appeal to your intuition. I think there's just one sensible choice.

The pooled model (5.3) has a θ that makes the observed $y_1, \ldots, y_{n_y}, x_1, \ldots, x_{n_x}$ most probable, it's MLE.

$$\hat{\theta}_{\text{pool}} = \frac{s_y + s_x}{n_y + n_x} \tag{5.4}$$

```
that.pool <- (sy + sx)/(ny + nx)
```

Why not use that setting for θ in the simulation? That way, the samples we get are as much like the observed values as possible, but under the null hypothesis that they came from the same distribution.

Alright, time to simulate. You may notice that I'm using a larger N this time. The number of MC "trials" will tend to drift higher as we get deeper into the book, and entertain more complex scenarios. Don't worry, things won't go off the rails, and will generally hover around $N = 100{,}000$. My determination of what to use is based on a quick assessment of MC error, discussed in more detail in §A.3.

[4] https://en.wikipedia.org/wiki/Pooled_analysis

```
N <- 100000                                 ## number of MC "trials"
That.ds <- rep(NA, N)                       ## storage
for(i in 1:N) {                             ## each MC iteration
  Sy <- sum(rbinom(ny, 1, that.pool))       ## could be combined ...
  Sx <- sum(rbinom(nx, 1, that.pool))       ## ... into a single sample
  That.ds[i] <- Sy/ny - Sx/nx               ## save sampled stat
}
```

Figure 5.1 shows the empirical sampling distribution along with the observed statistic and its reflection across the zero axis. This reflection is straightforward since, under the null hypothesis, we know that $\theta_y = \theta_x$, and therefore $\mathbb{E}\{\hat{\theta}_x - \hat{\theta}_y\} = 0$, because both are unbiased estimators.

```
hist(That.ds, main="", xlim=c(-0.65, 0.65))
abline(v=c(that.d, -that.d), lty=1:2, col=2, lwd=2)
legend("topright", c("that.d", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
```

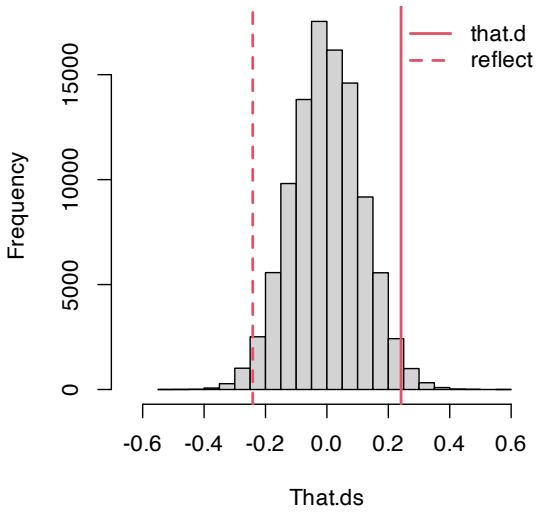

FIGURE 5.1: Histogram of sampled Bernoulli coin flip differences $\hat{\theta}_\Delta = \hat{\theta}_y - \hat{\theta}_x$ compared to the value observed from data, and its reflection into the opposite tail.

This means that a p-value may be calculated by doubling the one-tailed calculation.

```
pval.mc <- 2*mean(That.ds >= that.d)
pval.mc
```

```
## [1] 0.0355
```

It's a close call compared to the usual 5% threshold, but if we stick to that rule, then we would reject the null hypotheses and determine that $\theta_y \neq \theta_x$. The coins that were flipped in the two campaigns were likely not the same.

Two-sample Bernoulli test by math

Recall that by either of the methods in Chapter 4, central limit theorem (CLT) or MLE, we have the following asymptotic approximations:

$$\bar{Y} \equiv \hat{\theta}_y \sim \mathcal{N}(\theta_y, \theta_y(1-\theta_y)/n_y) \quad \text{and} \quad \bar{X} \equiv \hat{\theta}_x \sim \mathcal{N}(\theta_x, \theta_x(1-\theta_x)/n_x)$$

as $n_y \to \infty$ and $n_x \to \infty$. Assuming independence between the samples and using (again) the trick that sums of Gaussians are Gaussian[5], we have

$$\hat{\theta}_\Delta = \hat{\theta}_y - \hat{\theta}_x \sim \mathcal{N}\left(\theta_y - \theta_x, \frac{\theta_y(1-\theta_y)}{n_y} + \frac{\theta_x(1-\theta_x)}{n_x}\right) \quad \text{as} \quad n_y, n_x \to \infty.$$

The null hypothesis, providing $\theta \equiv \theta_y = \theta_x$, simplifies things substantially.

$$\hat{\theta}_\Delta \sim \mathcal{N}\left(0, \theta(1-\theta) \times \left(1/n_y + 1/n_x\right)\right) \quad \text{as} \quad n_y, n_x \to \infty$$

Just like in §4.1, this result is of little use with unknown θ in the variance, so it makes sense to replace it with an MLE. Remember, that's legit because of the weak law of large numbers (WLLN). And we're "Johnny on the spot" with $\hat{\theta}_{\text{pool}}$ from Eq. (5.4). This leads to a z-test after centering and scaling.

$$Z = \frac{\hat{\theta}_y - \hat{\theta}_x}{\sqrt{\hat{\theta}_{\text{pool}}(1-\hat{\theta}_{\text{pool}}) \times \left(1/n_y + 1/n_x\right)}} \sim \mathcal{N}(0,1) \quad \text{as} \quad n_y, n_x \to \infty$$

Foregoing visuals, an approximate p-value may be calculated by following a familiar routine.

```
se <- sqrt(that.pool*(1 - that.pool)*(1/ny + 1/nx))
z <- that.d/se
pval.z <- 2*pnorm(-abs(z))
c(mc=pval.mc, z=pval.z)
```

```
##      mc       z
## 0.03550 0.03345
```

That's in pretty close agreement with our MC calculation, above. Both are approximations, but the difference is that one can always entertain bigger N to make a MC more accurate. With the CLT, n_x and n_y are what they are, far from infinity, and we're stuck with that.

You may have wondered why I jumped straight to an asymptotic approximation. I did that because a closed-form distribution for $\hat{\theta}_y - \hat{\theta}_x$, or any other related quantity, is not known. Approximations like this are all you'll find in software. For example, in R the `prop.test` function provides a two-sample Bernoulli test.

```
pval.R <- prop.test(c(sx, sy), c(nx, ny), correct=FALSE)$p.value
pval.R
```

```
## [1] 0.03345
```

[5] https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables

This is identical to `pval.z` above. I had to provide `correct=FALSE` to make it match what we calculated. According to R's documentation, the default of `correct=TRUE` deploys a so-called Yates' continuity correction[6]. That page says "Yates's correction should always be applied, as it will tend to improve the accuracy of the p-value obtained." But this doesn't feel right to me for two reasons. One is that the examples on that page are for contingency table[7] tests for independence. This is related to our two-sample Bernoulli setting, but not identical. We'll talk about that more in Chapter 10. The second reason is that the correction doesn't make things better, on this example at least.

```
pval.Ry <- prop.test(c(sx, sy), c(nx, ny))$p.value
c(mc=pval.mc, z=pval.z, R=pval.R, Ry=pval.Ry)
```

```
##      mc       z       R      Ry
## 0.03550 0.03345 0.03345 0.05785
```

One of these things is not like the others and leads to a different conclusion at the 5% level. Of course, any time you're close to a threshold like 5%, whether that's 3.3% or 5.8%, it probably makes sense to proceed with caution especially when approximations are involved. I like that I can just make N bigger and get a more accurate MC test.

Sometimes, textbooks will discuss how to build confidence intervals (CIs) for θ_Δ. This is pretty straightforward, whether by MC or an asymptotic distribution. It's of somewhat limited utility in my opinion, but it makes for a nice homework exercise. I suppose it can be helpful, in some instances, to provide an estimate (including uncertainties) of how far apart two population parameters could be.

5.2 Location

Now consider the same situation but with location models. Just to keep the running example going – don't worry, we're approaching the limits of its utility – consider again heights of women in my class. Instead of comparing to the population at large, suppose we compared to women on the Blacksburg High School (BHS) Women's Varsity Basketball roster[8]. Data on that page are converted to inches below.[9]

```
x <- c(5*12 + 6, 5*12 + 8, 5*12 + 6, 5*12 + 8, 6*12 + 2, 5*12 + 7,
  5*12 + 10, 6*12 + 0, 5*12 + 7, 5*12 + 7)
nx <- length(x)
xbar <- mean(x)
s2x <- var(x)
```

As before, a vector `y` holds heights from my classroom (over-writing `y` for coin flips), but

[6] https://en.wikipedia.org/wiki/Yates's_correction_for_continuity
[7] https://en.wikipedia.org/wiki/Contingency_table
[8] https://www.maxpreps.com/va/blacksburg/blacksburg-bruins/basketball/girls/roster/
[9] These data were accessed on April 10, 2025. If you access them at a different time, you will probably get different numbers. It might be interesting to additionally try with whatever you find.

this is not printed to avoid duplication. Quantities `ny`, `ybar` and `s2y` are defined similarly. Figure 5.2 tries to help with a visual. Notice that there are two basketball players who are taller than the tallest woman from my class. Meanwhile, none are shorter than the shortest in my class. This is perhaps not surprising for a basketball team. However, that's merely anecdotal as statistical analysis is concerned. The real question is if we have enough data to distinguish means, and how confident we are about that.

```
plot(y, abs(rnorm(ny, 0, 0.1)), ylim=c(-0.5, 2.25),
  xlim=c(range(y, x)), xlab="heights", ylab="", yaxt="n", bty="n")
points(x, abs(rnorm(nx, 0.75, 0.1)), col=2, pch=19)
legend("top", c("class", "bball"), col=1:2, pch=c(21, 19),
  bty="n", horiz=TRUE)
```

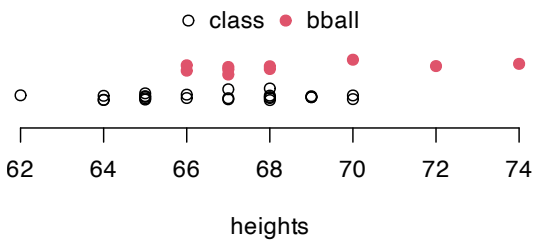

FIGURE 5.2: Comparing sample heights from my classroom and from the BHS basketball team. Vertical jitter is added to help visualize duplicate heights.

To get started on that "want to know", it helps to be concrete about the modeling apparatus, which is the first step (it's actually part of the "Assume") in Alg. 1.1. We can't get going without it.

$$\text{Model:} \qquad Y_1, \ldots, Y_{n_y} \overset{\text{iid}}{\sim} \mathcal{N}(\mu_y, \sigma_y^2) \qquad \text{and} \qquad X_1, \ldots, X_{n_x} \overset{\text{iid}}{\sim} \mathcal{N}(\mu_x, \sigma_x^2) \tag{5.5}$$

You might wonder if Gaussians are appropriate for these data, especially for a basketball team where we might expect a distribution of heights that is not symmetric: a long right-hand tail and a stubby left one. That's what Figure 5.2 shows. But, like in Chapter 2, let's table that worry for now. I'll get back to it later when I introduce nonparametric tests in Chapter 8. No modeling choice will be perfect. Gaussians are familiar, and they allow me to get on with explaining things.

Observe how this setup (5.5) is similar to Eq. (5.1) for coin flips, but with Gaussian location models. Hypotheses follow by analogy as well.

$$\begin{aligned} \mathcal{H}_0 &: \mu_y = \mu_x && \text{null hypothesis ("what if")} \\ \mathcal{H}_1 &: \mu_y \neq \mu_x && \text{alternative hypothesis} \end{aligned} \tag{5.6}$$

As in the single-Gaussian case (2.2), no specification is made for variances: σ_y^2 and σ_x^2. We could additionally presume they take on certain values, the same (unknown) value or different (unknown) values.

I'll develop the latter two of those three cases, where the first one (an uncommon case) is akin to a simpler z-testing analog of the third, t-testing case. I shall privilege that one, since it's the most common and likely the best suited to our situation, as pictured in Figure

5.2. The second one, known as the pooled variance[10] case where $\sigma^2 = \sigma_x^2 = \sigma_y^2$, is rather more niche. I believe it's popular because of its rather simpler mathematical development. I'll show you later. Via MC, however, the situation is reversed because the unpooled case is more compartmentalized: you basically just do two simulations like in §2.3 and combine them.

But I'm getting ahead of myself. MLE estimates (and bias-corrected variances) are straight out of §3.1. Let's notate these as $\hat{\mu}_y$, s_y^2, $\hat{\mu}_x$ and s_y^2. No need to reiterate their formulation. Just treat y's and x's separately and run the numbers two times over. When pooling variance, first estimate means separately, and then combine the samples to assess uncertainty via Eq. (3.6).

$$s_{\text{pool}}^2 = \frac{(n_y - 1)s_y^2 + (n_x - 1)s_x^2}{n_y + n_x - 2} = \frac{1}{n_y + n_x - 2}\left[\sum_{i=1}^{n_y}(y_i - \bar{y})^2 + \sum_{i=1}^{n_x}(x_i - \bar{x})^2\right] \tag{5.7}$$

So the pooled variance is a weighted average of individually calculated variances, with weights proportional to each sample size. Notice the -2 in the denominator. This is important, correcting bias that arises from estimates of two quantities, $\bar{Y}$ and $\bar{X}$, costing two degrees of freedom (DoF).

```
np <- nx + ny
s2.pool <- ((ny - 1)*s2y + (nx - 1)*s2x)/(np - 2)
```

Before jumping in we need one more thing, a test statistic. Above in §5.1 for coin flips we looked at differences in averages. By analogy, that's automatic here too. Again, the statistic is totally your choice as a practitioner, but it's my job to nudge you in the right direction conceptually. One of the points of Chapter 4 is that statistics based on averages and MLEs (often also averages) are automatic because they have desirable properties. Let $\mu_\Delta = \mu_y - \mu_x$, and under $\mathcal{H}_0$ we have $\mu_\Delta = 0$. A natural statistic for two-sample Gaussian testing is $\hat{\mu}_\Delta = \hat{\mu}_y - \hat{\mu}_x = \bar{Y} - \bar{X}$.

```
muhat.d <- ybar - xbar
```

Two-sample Gaussian test by MC

First, the unpooled ($\sigma_y^2 \neq \sigma_x^2$) case. The code below follows the `for` loop in §2.3, augmented with χ^2 variances for t-tests from §3.2. Loop innards are duplicated to include both y- and x-calculations, saving differences in MLEs. You could do this without `for` loops, following exercise #10 from §3.5, but I don't want to give that one away. The `for` loop way is a little slower, computationally, but more intuitive.

One caveat here is that, in order to respect $\mathcal{H}_0$, the means of the two Gaussians must be the same ($\mu = \mu_y = \mu_x$). It actually doesn't matter what value you use. My dad's favorite baseball player was Mickey Mantle[11] who wore #7 for the New York Yankees[12]. So I'm going to use that: $\mu = 7$. Try your favorite number and see what happens. If you had already sampled for one of the populations, say y from §3.2, then you could skip re-doing

[10] https://en.wikipedia.org/wiki/Pooled_variance
[11] https://en.wikipedia.org/wiki/Mickey_Mantle
[12] https://en.wikipedia.org/wiki/New_York_Yankees

that as long as you used the same μ-value as that simulation. I'm re-doing both samples below.

```
mu <- 7                                              ## RIP Dad and Mickey
Muhat.ds <- rep(NA, N)
for(i in 1:N) {
  sigma2ys <- (ny - 1)*s2y/rchisq(1, ny - 1)         ## Y population (class)
  Ys <- rnorm(ny, mu, sqrt(sigma2ys))
  sigma2xs <- (nx - 1)*s2x/rchisq(1, nx - 1)         ## X population (bball)
  Xs <- rnorm(nx, mu, sqrt(sigma2xs))
  Muhat.ds[i] <- mean(Ys) - mean(Xs)                 ## diff in averages
}
```

Figure 5.3 provides a comparison between the empirical sampling distribution of differences in averages compared to the observed statistic. Just eyeballing it, I don't think there's enough evidence here to reject the null. The observed statistic $\hat{\mu}_\Delta$, and its reflection, are partway into the tail(s), but not extreme relative to the sampling distribution.

```
hist(Muhat.ds, main="")
abline(v=c(muhat.d, -muhat.d), lty=1:2, col=2, lwd=2)
legend("topright", c("muhat.d", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
```

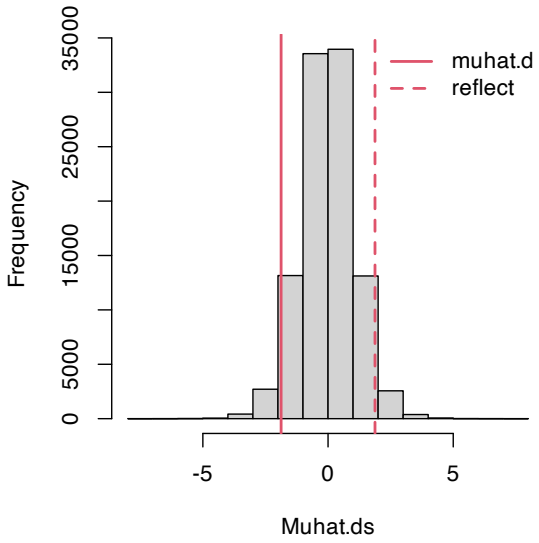

FIGURE 5.3: Histogram of sampled Gaussian average height differences $\hat{\mu}_\Delta = \hat{\mu}_y - \hat{\mu}_x$ compared to the value observed from data, and its reflection into the opposite tail.

Here's a more precise assessment through the lens of a p-value ...

```
pval.mc <- 2*mean(Muhat.ds < muhat.d)
pval.mc
```

```
## [1] 0.0807
```

… indicating that there may not be enough evidence to reject $\mathcal{H}_0$. We conclude that, although there are some tall women on the basketball team, those two samples could have come from (Gaussian) distributions with the same mean.

Code provided below goes back through the `for` loop implementing the pooled variance case.

```
Muhatp.ds <- rep(NA, N)
for(i in 1:N) {
  sigma2s <- (np - 2)*s2.pool/rchisq(1, np - 2)   ## pooled variance
  Ys <- rnorm(ny, mu, sqrt(sigma2s))              ## Y sample (class)
  Xs <- rnorm(nx, mu, sqrt(sigma2s))              ## X sample (bball)
  Muhatp.ds[i] <- mean(Ys) - mean(Xs)             ## diff in averages
}
pval.mcp <- 2*mean(Muhatp.ds < muhat.d)
pval.mcp
```

```
## [1] 0.03838
```

I've skipped the visual, but look what happens with the p-value. At the 5% level we would reject the null hypothesis, which is the opposite conclusion from earlier. So basically, what you decide comes down to a modeling choice. Pooling variances makes sense if you have a small amount of data, especially for one of the groups, and if you have reason to believe they're similar. When that happens, you get a sharper sampling distribution that makes it easier to reject the null.

In our example, there are few women on the basketball team, $n_x = 10$. Are variances similar? Looking at the spread of dots in Figure 5.2, perhaps they are. Both span about eight inches. But a choice about σ_x^2 or σ_y^2 is a modeling one. If you plan to use the data to help decide, then the right way to do that is via a statistical procedure that acknowledges sampling uncertainty. I plan to talk about that more in Chapter 6.

Two-sample Gaussian test by math

Quick digression. I know it's awkward to digress before even getting started, and also not so quick. It turns out that what I thought was the gospel truth in two-sample Gaussian testing, known as the Welch test[13], is actually an approximation. The Welch test is what `t.test` in R does when you give it two samples, but it doesn't warn you things might be off. Such betrayal! I would never have noticed if there wasn't such a discrepancy between my MC calculation and the `t.test` output. I'll show you in a moment; hold tight.

You'll find the Welch test in every intro stats book, but I doubt you'll find the straight story. Wikipedia gets it right. There's really no better source for information in my opinion. You could make the case that this book is just a lens on Wikipedia focused on basic stats. Narrowing that focus on the two-sample Gaussian test, additional insight is provided by the page for the Behrens–Fisher (BF) problem[14], along with some citations to other approximate solutions.

The F in BF is for Ronald Fisher[15], the same Fisher as Fisher information from §4.2. Ronald is/was a super famous statistician, and geneticist, with a lot of statsy things named

[13] https://en.wikipedia.org/wiki/Welch's_t-test
[14] https://en.wikipedia.org/wiki/Behrens-Fisher_problem
[15] https://en.wikipedia.org/wiki/Ronald_Fisher

after him. He's also a controversial figure. He picked a lot of fights with other scientists, and sometimes he was rude to them in print. Reading a Fisher paper is quite a trip. He had crazy ideas about racial issues[16] and has been canceled by many modern progressive scientists as a result.

Fisher was good with numbers, and invented a whole field of statistics called fiducial inference[17], basically to address the two-sample Gaussian testing problem. He really went down a rabbit hole. My understanding is that the German chemist Walter-Ulrich Behrens[18] solved the problem first (Behrens, 1928), though perhaps less elegantly.

I'm not going to explain to you Behrens' or Fisher's solution, or fiducial inference. I don't understand them. I did find an R package that will do the Behrens-Fisher test (Fay, 2023). That implementation involves a numerical procedure, so it too provides an approximation of sorts. Importantly, it's the kind of approximation that can be improved with better numerics (like MC, but unlike the CLT). I'm going to use that package to show you that it gives an answer closer to our MC result from above, than does the Welch test.

```
library(asht)
pval.bf <- bfTest(y, x)$p.value
pval.w <- t.test(y, x)$p.value
c(mc=pval.mc, bf=pval.bf, welch=pval.w)
```

```
##      mc      bf   welch
## 0.08070 0.07676 0.06916
```

Notice how the MC and BF outcome rounds to the same hundredths place, but the Welch number does not. If you use $10N$ MC simulations, then it will agree with BF to the one thousandths place. Considering how simple the MC approximation is, we can probably throw away the Welch test. Because it's so common, I'm going to quickly explain it to you anyways.

Alright, back to what I intended to present before I got sidetracked. Welch (1938) showed that

$$T = \frac{\bar{Y} - \bar{X}}{\sqrt{s_y^2/n_y + s_x^2/n_x}} \sim t_\nu \quad \text{where} \quad \nu = \left\lfloor \frac{(s_y^2/n_y + s_x^2/n_x)^2}{\frac{(s_y^2/n_y)^2}{n_y - 1} + \frac{(s_x^2/n_x)^2}{n_x - 1}} \right\rfloor,$$

approximately. The approximation improves as $n_x, n_y \to \infty$, but that's sort-of a moot point. Out at infinity you can just use a z-test via a CLT and not bother with that complicated DoF formula for ν. Wow, what a mouthful! That funny square bracket missing its top corners is the floor[19] function, returning the largest integer smaller than its argument. This part is optional. The `t.test` function in R does not use the floor.

Here it goes in code.

```
se2 = s2y/ny + s2x/nx                    ## squared standard error
se <- sqrt(se2)                          ## standard error
```

[16] https://en.wikipedia.org/wiki/Eugenics
[17] https://en.wikipedia.org/wiki/Fiducial_inference
[18] https://en.wikipedia.org/wiki/Walter-Ulrich_Behrens
[19] https://en.wikipedia.org/wiki/Floor_and_ceiling_functions

```
nu.denom <- (s2y/ny)^2/(ny - 1) + (s2x/nx)^2/(nx - 1)
nu <- se2^2/nu.denom                    ## optionally nu <- floor(nu)
t <- (ybar - xbar)/se                   ## t-stat
2*pt(-abs(t), nu)                       ## p-value
```

```
## [1] 0.06916
```

This agrees with `pval.w` above.

The pooled variance setup is simpler and less controversial, because getting it right mathematically is lots easier. You don't have to invent a whole new way to think about uncertainty to get to the bottom of it. Since

$$\sigma^2 \sim (n_y + n_x - 2)s^2_{\text{pool}}/\chi^2_{n_y+n_x-2},$$

following a similar logic as in §3.2, a t statistic is readily available for testing.

$$T = \frac{\bar{Y} - \bar{X}}{s_{\text{pool}}\sqrt{1/n_y + 1/n_y}} \sim t_{n_x+n_y-2} \tag{5.8}$$

You'll notice that this is the Gaussian-sample version of the pooled Bernoulli test above in §5.1. In code, along with the version from the `t.test` library function in R for comparison …

```
se.pool <- sqrt(s2.pool*(1/ny + 1/nx))
t <- (ybar - xbar)/se.pool
pval.tp <- 2*pt(-abs(t), ny + nx - 2)
c(mc=pval.mcp, math=pval.tp, lib=t.test(y, x, var.equal=TRUE)$p.value)
```

```
##      mc    math     lib
## 0.03838 0.03833 0.03833
```

Everyone pretty much agrees.

As with the Bernoulli case, some textbooks and many software libraries provide CIs for μ_Δ along with tests. The `t.test` function does. I think it's fairly basic, so I've left it to you as a homework exercise in §5.4. Remember, once you know the parameter estimate and standard error, which you do already once you have the sampling distribution for the test (3.11), there's a simple adaptation to form a CI (3.20). If using MC simulation, just go back through the sim with estimated quantities (averages/MLEs) instead of values from $\mathcal{H}_0$, which in this case means replacing 0.

5.3 Paired location

A special case that's common when testing differences in means, whether for Gaussian populations or beyond, involves so-called paired data[20]. This happens when experimental subjects in the y-group and x-group are linked together in some way as

$$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n),$$

and consequently $n \equiv n_x = n_y$. For example, suppose we asked each woman in my class (or, equivalently, each basketball player) to report their mother's height when they were in college (or high school). Then we would have paired data. The result of a test on these paired data could be very interesting because it could tell us about the impact of socio-economic, behavioral, environmental and nutritional factors on health.

Quick digression on broader themes. There are many interesting questions you can ask about paired data. I bet correlation and prediction comes to mind, that is, using information about x to predict y? This is a very important topic, but not exactly what I'm intending to do here. We'll get to that in Chapters 7 and 12. Here, I'm just considering two independent samples from related populations. It is the same setup as Eqs. (5.5) and (5.6), but different data collection environment.

Alright, back to what's special about this paired setting. I asked the women in my class to provide their mother's heights, and here are those values, below. I'm writing over the x's from those basketball players. Their mothers' heights aren't available on the internet, unfortunately.

```
x <- c(64, 63, 68, 66, 67, 66, 66, 63, 66, 67, 69, 68, 70, 64, 68,
  66, 65, 61, 69, 66, 61, 67, 63, 64)
nx <- length(x)
```

This time I'm not immediately jumping to parameter estimates for the two populations. There are two special, and cool, things about a paired test compared to the unpaired version. The first is that leveraging a paired structure eliminates some uncertainty, because there are fewer quantities to estimate. This gives you more power (§A.2) to reject the null. The second is that, when you're doing it right, you reduce the problem to a simpler one that you already know how to solve, from Chapter 2, with some of the mathematical details in §3.2.

Before inference and testing, a good first step is visual. Figure 5.4 uses a scatterplot[21] to inspect the mother-daughter relationship in the data. Mothers are on the x-axis and daughters on the y-axis, lining up with their variable names. Overlayed on the plot is a $y = x$ line as an indication of parity in heights.

```
plot(x + rnorm(nx, sd=0.1), y + rnorm(ny, sd=0.1),  ## jitter for visual
  xlab="mothers (x)", ylab="daughters (y)")
abline(0, 1)  ## intercept (a) = 0, slope (b) = 1 in y = ax + b.
legend("bottomright", c("y = x"), lty=1)
```

[20] https://en.wikipedia.org/wiki/Paired_difference_test
[21] https://en.wikipedia.org/wiki/Scatter_plot

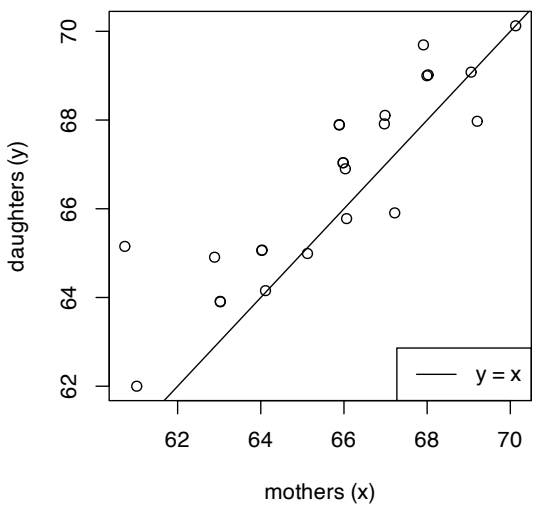

FIGURE 5.4: Scatter plot showing mothers and daughters, with a $y = x$ line overlayed for comparison.

If daughters tended to be the same heights as their mothers, then there would be about the same numbers of points above as below, with some lying exactly on the line in this case because data values are rounded to the nearest inch. Observe that there are more dots above the line than below (or on) the line. This suggests that these daughters are taller than their mothers. Is there enough evidence to reject the hippopotamus that they basically have the same mean height? That's the "want to know".

Assuming that x's and y's are uncorrelated – remember, we'll get to the correlated case later – and that they are each Gaussian, their difference is also Gaussian:

$$\text{Model:} \qquad D_i = Y_i - X_i \overset{\text{iid}}{\sim} \mathcal{N}(\mu_d, \sigma_d^2), \qquad i = 1, \ldots, n.$$

If we're willing to parameterize the individual populations (5.5), then we could work out μ_d and σ_d^2 in terms of those values using additivity of Gaussians, a trick that's hopefully becoming familiar. But that is not our primary interest. If mothers and daughters have the same mean, then that means $\mu_d = 0$, and so we have reduced our two-sample problem to a one-sample one.

```
d <- y - x
n <- length(d)
dbar <- mean(d)
s2d <- var(d)
```

Notice how only two quantities are being estimated, instead of four in §5.2. Now test …

$$\begin{array}{ll} \mathcal{H}_0 : \mu_d = 0 & \text{null hypothesis ("what if")} \\ \mathcal{H}_1 : \mu_d \neq 0 & \text{alternative hypothesis.} \end{array} \tag{5.9}$$

Since this has been reduced to the one-sample case, I won't show all variations. Let me

encourage you to cut-and-paste the MC from §2.3. I'll go straight for the math way from §3.2 along with a comparison to what the library provides in R.

```
se.d <- sqrt(s2d/n)
t.d <- dbar/se.d
pval.d <- 2*pt(-abs(t.d), n - 1)
c(math=pval.d, lib=t.test(y, x, paired=TRUE)$p.val)
```

```
##      math       lib
## 0.0003117 0.0003117
```

Those are identical. The conclusion, with data from this particular example, is that daughters are indeed taller than their mothers. (An easy reject of $\mathcal{H}_0$.)

You can similarly follow calculations from §3.3 to develop CIs if interested. No changes are required after reducing paired data (x_i, y_i) to differences $d_i = y_i - x_i$. In this case, the calculation is interesting, because it tells you how much taller daughters tend to be than their mothers.

```
q <- qt(c(0.025, 0.975), n - 1)
CI <- dbar + q*se.d
rbind(math=CI, lib=t.test(y, x, paired=TRUE)$conf.int)
```

```
##        [,1]  [,2]
## math 0.4691 1.364
## lib  0.4691 1.364
```

Again, our by-hand calculation and the R library agree. It looks like women in my class are between half-an-inch and 1.5 inches taller, very roughly speaking, than their mothers. Whether or not this is causal, and what the mechanism is, is a topic for another course.

5.4 Homework exercises

These exercises help gain experience with two-sample tests. Focus is on Bernoulli and Gaussian cases, as explained above, but there are some other unique settings too. My suggestion is to do each problem both ways: via MC and the math way (which might mean closed-form calculation or asymptotic approximation). Check with your instructor for details.

Do these problems with your own code, or code from this book. I encourage you to check your work with library functions like `prop.test` *and* `t.test`*. Check with your instructor first to see what's allowed. Unless stated explicitly otherwise, avoid use of libraries in your solutions.*

#1: CI for θ_Δ

Develop code for MC and asymptotic approximations for a CI for θ_Δ in the two-sample Bernoulli setup of §5.1. Provide CI estimates for the coin-flipping example.

#2: Update `monty.bern` to include two-sample capabilities

Update `monty.bern` from §1.4, and possibly also §3.5 #6 and §4.4 #4, to include two-sample options. Follow these specifications.

- Make an optional `x` argument with `x=NULL` being the default, in which case one-sample versions (from earlier chapters) are used.
- Interpret `theta` as θ_Δ in this case.
- Support both MC and asymptotic approximations by checking `N == -1`.
- Optionally, augment with CI calculations for θ_Δ via #1, above.

Hint: the next three questions are pretty easy once you've done this one.

#3: Drone communication

A squadron of drones maintains a peer-to-peer communication protocol as they carry out maneuvers. Whenever a drone receives a signal from home base, it must relay the message to other members of the squadron following that protocol. A team is testing two different protocols, A and B, for their effectiveness while carrying out a schedule of prescribed maneuvers. Of the 198 Protocol-A messages sent from home base to the nearest drone, 169 of them were received by every drone in the squadron after one minute. Of the 155 Protocol-B messages sent to the nearest drone, 124 of them were received by every drone in the squadron after one minute. Do these protocols have the same success rate? Optionally, provide a CI for the difference in rates.

#4: Not in my back yard (NIMBY)

Blacksburg is a town in Montgomery county, Virginia. The county intends to green-light a new affordable housing project within town limits to alleviate a county-wide shortage. However, there's concern that Blacksburg residents already have too many down-market developments, like those that cater to Virginia Tech students. A poll was taken to determine if there's any difference between support for the project within town limits and in the broader county (excluding the town). Of the 209 townies who took the pool, 118 are in favor. Of the 488 county (non-Blacksburg) residents who took the pool, 211 are in favor. Is there any difference in voting patterns between the two groups. Optionally, provide a CI for the difference in voting rates.

#5: Truck pollution

In 2027 the US EPA will/has tighten(d) nitrogen oxide(s) (NOx) emission limits on diesel powered semi-trucks to 200 mg/bhp-hr. A major truck manufacturer has two models of these trucks, involving different diesel engines and NOx suppression technologies. In a sample of one-hundred trucks of both types, 16 of the first kind exceeded the threshold, while 12 of the second did. Is there any difference between the two models as concerns NOx emissions compliance? Optionally, provide the difference in rates exceeding the threshold.

#6: CI for μ_Δ

Develop code for MC and Welch approximations for a CI for μ_Δ in the two-sample Gaussian setup of §5.2, and also for μ_d in the paired setup of §5.3. Provide CI estimates for women's heights comparisons from those sections, studying differences between basketball players and mothers, respectively.

#7: Update `monty.t` to include two-sample capabilities

Update `monty.t` from §3.5 #8 to include two-sample options. Follow these specifications.

- Make an optional `x` argument with `x=NULL` being the default, in which case one-sample versions (from earlier) chapters are used.
- Interpret `mu` as μ_Δ in this case.
- Support both MC and Wald approximations by checking `N == -1`.
- Support a `pooled=TRUE` option (with `FALSE` being the default).
- Support a `paired=TRUE` option (with `FALSE` being the default), implementing a paired t-test. Note that `pooled` and `paired` don't make sense at the same time. Be sure to check, in the paired case, that `length(y) == length(x)`.
- Optionally, augment with CI calculations for μ_Δ and μ_d via #6, above.

Hint: the next several questions are pretty easy once you've done this one.

#8: Physical therapy

Physical therapy (PT) is often recommended for patients recovering from surgery, but success can vary depending on many factors. In one study, two PT treatment centers were compared by measuring recovery time, in weeks, for knee replacement patients.

```
ptA <- c(38, 50, 41, 29, 45, 53, 46, 47, 39)
ptB <- c(61, 49, 50, 56, 58, 44, 65)
```

Is there a difference in mean recovery time for patients at the two PT centers? Optionally, provide a CI for the difference in mean recovery times.

#9: Dieting and weight loss

Twins who were interested in losing weight were recruited for a study of two different diets, paleo and keto, assigned to each sibling at random. The data below record the number of pounds (lbs) lost under each diet after six months. Each row corresponds to a pair of siblings.

```
paleo <- c(49, 6, 31, 14, 35, 31, 20, 64, 39)
keto <- c(25, 15, 17, 15, 33, 21, 32, 50, 14)
data.frame(paleo, keto)
```

```
##   paleo keto
## 1    49   25
## 2     6   15
## 3    31   17
## 4    14   15
## 5    35   33
## 6    31   21
## 7    20   32
## 8    64   50
## 9    39   14
```

Is there a difference in mean weight loss under the two diets? Optionally provide a CI for the difference in mean pounds lost.

#10: On the rocks

A bartender was curious about the shape of ice cubes in mixed drinks and performed an experiment. She wondered if one big ice cube was equivalent to several, small ice cubes of the same total volume, as affects the melting rate. (Melting rate is linked to both the temperature of the liquid in the cocktail, and how watered-down it tastes.) She filled identical tumblers with identical pours of room-temperature single-malt bourbon. One set of tumblers got a single one-inch cube of ice. The other got four half-inch cubes. She measured the amount of time (minutes) it took for all of the ice to dissolve, which is recorded below.

```
one <-  c(21.6, 20.1, 24.9, 18.8, 20.8, 25.2, 22.6, 28.3, 19.7, 28.7,
    22.4, 14.8, 23.2, 21.6, 20.6)
four <- c(22.9, 17.2, 18.3, 21.2, 21.7, 23.4, 23.4, 17.4, 18.6, 18.3,
    20.5, 21.4)
```

Is there is a difference in melting time for the different ice cubes? Does this change if we assume a pooled variance? Optionally, provide a CI for the difference in mean times.

#11: Return on investment

This is our first opportunity to really investigate the ROI claims from §2.5, which is available as roi.RData[22] on the book webpage.

```
load("roi.RData")
```

Using `roi$company` and `roi$industry`, does there seem to be any difference in mean ROI between companies that use the IT product in question and the industry at large? Optionally, provide a confidence interval for the mean difference in ROI.

#12: Paired ROI

Data stored in roip.csv[23] on the book webpage contains similar numbers, but company observations are paired, row-wise, with the industry observations that correspond to the ones they are associated with.

```
roip <- read.csv("roip.csv")
```

For example, Apple's ROI is paired with the industry aggregate for high tech companies. This means that there are fewer unique `roip$industry` numbers, but `roip$company` is the same as `roi$company`. How does a paired analysis of these data compare with the unpaired one from #10?

#13: Simulation: lifetimes

We did two questions like this in §1.5. These next two are a little more complex because they involve two populations. For both, use simulations of size $N = 10^6$.

The lifetime of a major manufacturer's name-brand LED light bulb in hours is $Y \sim \text{Exp}(\lambda =$

[22] https://bobby.gramacy.com/hipp0/roi.RData
[23] https://bobby.gramacy.com/hipp0/roip.csv

10^{-4}), and the generic, store-brand alternative is e^X where $X \sim \mathcal{N}(\mu = 6.5, \sigma^2 = 4.8)$, i.e., it's log-normal[24]. Answer the following.

a. What is the probability that the name-brand light bulb lasts longer than the generic, store-brand one?
b. What is the probability that it lasts two times longer?

#14: Simulation: intrusion kill chain

An intrusion kill chain[25] is a model that is popular in some cybersecurity circles. The idea is based on a weakest link-in-the-chain concept. Suppose we were studying a kill chain with four links, and the amount of time available to detect and thwart a breach by disrupting an attack at any of the links is distributed as $Y_i \sim \mathcal{N}(12, 2)$, for $i = 1, \ldots, 4$, in minutes. *Note: 2 is the variance, so the standard deviation is* $\sqrt{2}$. One way to measure the robustness of a security system is to multiply the shortest time in the chain by the number of links. Call this the "kill time".

a. Calculate the mean, median and standard deviation of the kill time.
b. To be considered robust enough for certain cybersecurity applications, the probability that the kill time is less than 30 minutes must be less than 0.01. Is this system robust enough?

#15: Poisson rate change

This one is challenging, but also fun.

The file changept.txt[26] contains $n = 200$ samples that we can assume are independent Poisson. They are collected over time $t = 1, \ldots, n$. In the parlance, the data comprise a time series[27].

```
y <- scan("changept.txt")
```

It is suspected that there is a change point[28] $m \in \{1, \ldots, 200\}$ somewhere along in time so that the first m observations are $Y_t \overset{\text{iid}}{\sim} \text{Pois}(\theta)$, for $t = 1, \ldots, m$, and the last $n - m$ are $X_t \overset{\text{iid}}{\sim} \text{Pois}(\phi)$, for $t = m + 1, \ldots, n$.

a. Plot the data and guess $\hat{m}$.
b. Using that $\hat{m}$ argue that $\theta \neq \phi$ via an appropriate statistical test that you design. *Hint, no recipe is provided explicitly in this chapter, but you can combine ideas here with those in Chapters 3–4.*
c. If you choose m differently – a little left or right, or a lot – how would your test from #b change?
d. How would you devise a statistical scheme to "dial in" a best m according to some well-defined statistical criteria?

[24] https://en.wikipedia.org/wiki/Log-normal_distribution
[25] https://en.wikipedia.org/wiki/Cyber_kill_chain
[26] https://bobby.gramacy.com/hipp0/changept.txt
[27] https://en.wikipedia.org/wiki/Time_series
[28] https://en.wikipedia.org/wiki/Change_detection

6

Analysis of variance

This chapter lumps two things together which are actually quite distinct. One is tests that focus on variance, like σ^2 in a Gaussian sample. We'll do this for one sample and for two samples. My own feeling is that this has somewhat limited utility compared to things we've covered up until this point. It certainly can be useful, like to check if one process is more variable – more wobbly in its behavior – than another. But variance is often a second-order concern. (It's already a second moment.)

The second thing is eponymous in the title: analysis of variance[1] (ANOVA). ANOVA *is* super useful. The trouble is, it's not really about variance. It's about means. True, you do analyze variance(s), but as a means to an end for means. You don't use analysis of variance to test for variance. I prefer to think of it as a pooled-variance extension of the two-sample Gaussian test of §5.2, but to three or more samples. (You can also do it for two samples, but it doesn't make sense for a single sample.)

Statisticians are notoriously bad at naming things. You might even argue that the choice of the word "statistics" was a mistake. Too much of a mouthful. Apologies to my Greek friends. Another great example is regression, which we'll cover in Chapter 7. Regression is a super important and exciting topic with a terrible name. No parent wants to hear that their child is regressing in school.

Machine learning (ML) researchers are much better at naming. Instead of "logistic regression"[2], or "logit regression" for short (those are stats terms), ML has the "perceptron"[3]. I don't actually cover either topic in this book, though they are both cool and important (maybe in volume two). I'm just trying to make a point here, and there are lots of other great examples. In Chapter 5 I said that two-sample t-testing is a form of A/B testing, which comes from ML. I don't think there is any direct analog of ANOVA in the ML world, but if there were, I think they would call it an "alphabet test", because it's like "A/B/C/D/… testing". Better right?

So why did I lump these two topics together? One reason is that I wanted to cover both, but two separate chapters would have been rather thin. I'll lump these two topics together for the same reason when we treat their nonparametric cousins in Chapter 11. The second reason is everyone thinks ANOVA is about variance, and therefore is right at home alongside other variance tests. That might be wrong, but it lets me spin a tidy yarn that I hope you won't forget.

[1] https://en.wikipedia.org/wiki/Analysis_of_variance
[2] https://en.wikipedia.org/wiki/Logistic_regression
[3] https://en.wikipedia.org/wiki/Perceptron

6.1 One sample

This will be quick, so I'm just going to jump right in. Here the model is the same as in §2.1, but I'll repeat it below for convenience.

$$\text{Model:} \qquad Y_1, \ldots, Y_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2) \tag{6.1}$$

This time interest lies in σ^2 not μ. Suppose there were some value σ_0^2 that you were wondering about. You could pose the following competing hypotheses.

$\mathcal{H}_0 : \sigma^2 = \sigma_0^2$	null hypothesis ("what if")
$\mathcal{H}_1 : \sigma^2 \neq \sigma_0^2$	alternative hypothesis

The value of μ isn't pinned down by the hypotheses. Any value will work for carrying out the test by Monte Carlo (MC). However, the usual statistic that's used for variance, s^2 from Eq. (3.6) has an implicit $\bar{y}$ built in.

To have something concrete for calculations, consider the following example. The owner of a major ice cream franchise, understandably, wants to control the variability of the size of ice cream scoops in his store. He's concerned, in particular, about one goofball employee whose carefree attitude would seem to be at odds with consistency in any aspect of his life. He recruits sixteen confederate customers who order ice cream at the shop, and who are served by this particular employee. Their scoops are weighed and the results are below.

```
y <- c(4.0, 5.5, 4.9, 3.8, 4.3, 4.4, 5.1, 4.1, 4.0, 3.0, 3.7, 3.4, 3.6,
  3.0, 3.5, 4.0)
n <- length(y)
s2 <- var(y)
```

Once you've calculated s^2 and extracted n, you don't need the individual y-values anymore. These quantities comprise sufficient information about the sample for the purposes of inference.

According to Google, a scoop of ice cream is typically around 4 oz. During training, all employees were taught how to scoop ice cream such that scoops were between 3 and 5 oz, 95% of the time. In other words, with variance $\sigma^2 = 0.5^2$.

```
mu <- 4
sigma2 <- 0.5^2
```

The franchise owner hired us as statistical consultants. What can we tell him about how these data related to those targets, in particular as regards σ^2?

One-sample variance test by MC

The code below takes n-sized samples from the null hippopotamus ($\sigma^2 = 0.5^2$) and calculates their variance, repeatedly in a MC fashion. The value taken for μ is as specified above, but again any μ will do.

```
N <- 100000
S2s <- rep(NA, N)
for(i in 1:N) {
  Ys <- rnorm(n, mu, sqrt(sigma2))
  s2s <- var(Ys)
  S2s[i] <- s2s
}
```

Figure 6.1 offers a visual of the empirical sampling distribution of those variances with the value of s^2 from the data (and its reflection) overlayed. That observed statistic is out in the tails, but not so far out. It's a close call.

```
hist(S2s, main="", xlim=range(c(s2, S2s)))
abline(v=c(s2, quantile(S2s, mean(S2s > s2))), lty=1:2, col=2, lwd=2)
legend("topright", c("s2", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
```

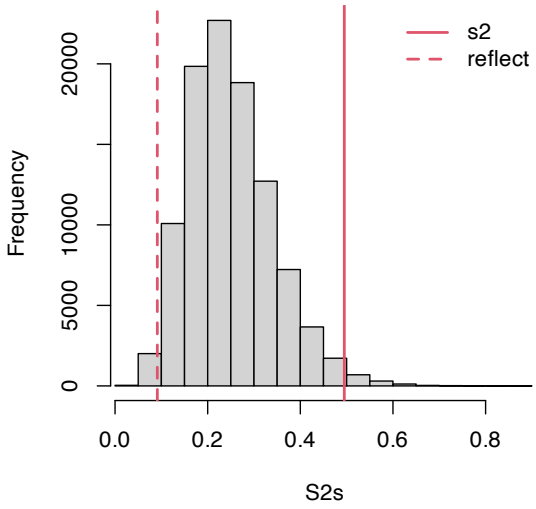

FIGURE 6.1: Histogram of sampled variances under $\mathcal{H}_0$ compared to the observed variance and its reflection.

To help quantify the outcome of the test, a p-value may be calculated as follows.

```
pval.mc <- 2*mean(S2s > s2)
pval.mc
```

```
## [1] 0.02572
```

This is comfortably below our usual 5% threshold. So we might decide to reject the null hypothesis and conclude that this particular employee's scooping is more variable than his boss would like. Yet I recommend *against* firing the poor bloke, yet. The situation is more nuanced than I have described. But before we get to that, how would you do this the old-school math way?

One-sample variance test by math

Recall from Chapter 3 that sums of Gaussian squares have a (scaled) χ^2 distribution. Those sums of squares are exactly what we simulated above, under $\mathcal{H}_0$. In particular, Eq. (3.16) provides that

$$V \equiv \frac{(n-1)S^2}{\sigma^2} \sim \chi^2_{n-1}.$$

Notice how that equation relays both the test statistic, $v = \frac{(n-1)s^2}{\sigma^2}$ using observed data values and for any σ^2 of interest (provided by $\mathcal{H}_0$), and its sampling distribution $V \sim \chi^2_{n-1}$.

```
v <- s2*(n - 1)/sigma2
```

Quick digression on notation. I chose the letter V because the quantity in question is a (scaled) variance. Many texts use a generic t for "test" statistics, but I think that's confusing because it clashes with the statistic involved t-tests, which have a t_ν distribution. Variances, and residual sums of squares more widely, usually have a χ^2_ν distribution.

Alright, back to our analysis. The visual for the statistic v and its distribution, provided in Figure 6.2, is similar to the empirical distribution of Figure 6.1, except scaled by $(n-1)/\sigma^2$.

```
x <- seq(0, 3*n, length(1000))
plot(x, dchisq(x, n-1), type="l", lwd=2, main="")
abline(v=c(v, qchisq(pchisq(v, n - 1, lower.tail=FALSE), n - 1)),
  lty=1:2, col=2, lwd=2)
legend("topright", c("v", "reflect"), lty=1:2, col=2, lwd=2, bty="n")
```

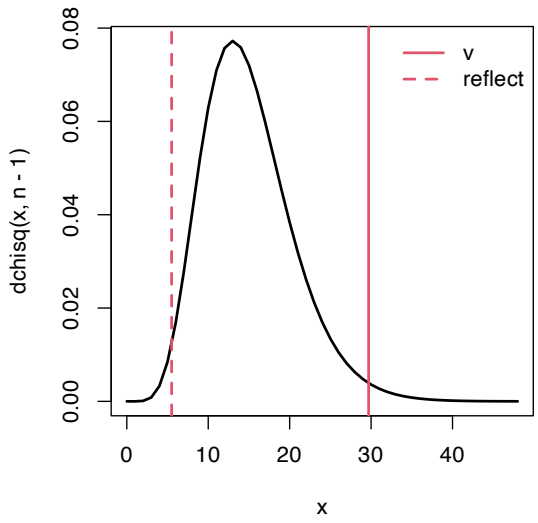

FIGURE 6.2: Sampling distribution and observed statistic for one-sample variance test.

We may calculate a p-value by integrating into the two tails as follows. Remember to be careful to get the direction of the tail right with the `lower.tail` argument. This could change depending on what side of the distribution your statistic is on. With this two-tailed

test, I'm presuming a reflected value lying at the opposite quantile, as shown in the figure. This is why I can get away with multiplying by 2.

```
pval.math <- 2*pchisq(v, n - 1, lower.tail=FALSE)
c(mc=pval.mc, math=pval.math)
```

```
##      mc    math
## 0.02572 0.02612
```

That p-value is in close agreement with our MC calculation earlier. There's no variance test built-into R; however, I found one in the `EnvStats` package on CRAN (Millard, 2013).

```
library(EnvStats)   ## you man need to install.packages("EnvStats") first
varTest(y, sigma.squared=sigma2)$p.value
```

```
## [1] 0.02612
```

The same p-value we calculated by hand above.

Confidence intervals

Before turning to two-sample tests, I want to make a quick note about confidence intervals (CIs). Remember, once you know the sampling distribution – whether empirically via MC or via math – it's easy to form a CI. In our current context, that means one of two things. Either replace estimated s^2 with σ^2 in the MC. (I've left that one for you as a homework exercise in §6.5.) Or, if you prefer the math way, follow the outline in §3.3. First find quantiles of the distribution, in this case χ^2_{n-1}. Then undo the re-scaling to solve of σ^2 on both sides of the inequality.

$$\begin{aligned} 1-\alpha = \mathbb{P}(q_{\alpha/2} < z < q_{1-\alpha/2}) &= \mathbb{P}\left(q_{\alpha/2} < \frac{(n-1)s^2}{\sigma^2} < q_{1-\alpha/2}\right) \\ &= \mathbb{P}\left(\frac{s^2(n-1)}{q_{1-\alpha/2}} < \sigma^2 < \frac{s^2(n-1)}{q_{1\alpha/2}}\right) \end{aligned}$$

Don't forget to flip around your inequalities when you divide all sides by σ^2 to put it in the numerator. Here are those calculations in code.

```
q <- qchisq(c(0.025, 0.975), n - 1)
CI.math <- s2*(n - 1)/rev(q)            ## rev to flip inequalities
```

Here it comes with the library-based calculation for comparison.

```
CI.lib <- as.numeric(varTest(y, sigma.squared=sigma2)$conf.int)
rbind(math=CI.math, lib=CI.lib)
```

```
##        [,1]  [,2]
## math 0.2701 1.186
## lib  0.2701 1.186
```

Those are identical.

I know I said, on multiple occasions, the math way to get a CI is est $\pm\, q \times$ se. (Use $q = 2$ for an approximate 95% interval). That's only if we have "est" and "se" and are willing to make a Gaussian approximation. We have the exact sampling distribution in this instance, so no need to approximate. But if you insist, check this out. Eq. (4.9) gives the following asymptotic approximation by way of the MLE for σ^2.

$$\hat{\sigma}^2 \sim \mathcal{N}(\sigma^2, 2\sigma^4/n) \quad \text{as} \quad n \to \infty.$$

Therein lies the standard error we require: se $= \sqrt{2(\hat{\sigma}^2)^2/n}$, after plugging in $\hat{\sigma}^2$ for σ^2 following the weak law of large numbers (WLLN).

```
s2hat <- s2*(n - 1)/n
se <- sqrt(2*s2hat^2/n)
CI.asy <- s2hat + c(-2, 2)*se
CI.asy
```

```
## [1] 0.1359 0.7921
```

This isn't a great approximation since $n = 16 \ll \infty$. You have to be particularly careful with approximate Gaussian CIs near zero. Both ends of the interval calculated above fit this description; the left one especially. In that situation, there's a non-negligible amount of density on negative values, but variances can't be negative. When the actual (χ^2_{n-1}) sampling distribution is substantially skewed – see Figure 6.2 – an approximately Gaussian-based CI estimate is biased toward zero, and is too small to have appropriate nominal coverage[4].

This is, in my opinion, all really cumbersome compared to the MC version which involves essentially one replacement: `sigma2 <- s2`. You may draw your own conclusion after trying for yourself in homework exercises. Coding effort aside, it bears repeating … When you must approximate numerically, MC trumps Gaussian asymptotics. The only exception might be when n is really large.

6.2 Two samples

Suppose I told you that the last seven scoops were not of ice cream, but instead sorbet. Sorbet is water-based and generally weighs less than dairy-based ice cream, per unit volume. So if our beleaguered employee is using the same scooper and technique, and producing the same volume of ice cream each time, it stands to reason that his ice cream scoops would weigh more than his sorbet ones, and also possibly have different variance.

For now, I plan to keep focus on variance. Means are next in §6.3. Here is the situation, with y's now capturing ice cream scoops, and x's for sorbet.

[4] https://en.wikipedia.org/wiki/Coverage_probability

```
x <- y[10:16]       ## sorbet
s2x <- var(x)
nx <- length(x)
y <- y[1:9]         ## ice cream
ny <- length(y)
s2y <- var(y)
```

It can be helpful to visualize these data just to fix ideas. See Figure 6.3. There's a clear location shift, but the question of variance is more subtle. Observe how the spread of red dots (x: sorbet) is a little narrower than the black open circles (y: ice cream). Is that difference statistically significant?

```
plot(y, abs(rnorm(ny, 0, 0.1)), ylim=c(-0.5, 2.25),
  xlim=c(range(y, x)), xlab="weights", ylab="", yaxt="n", bty="n")
points(x, abs(rnorm(nx, 0.75, 0.1)), col=2, pch=19)
legend("top", c("ice cream", "sorbet"), col=1:2, pch=c(21, 19),
  bty="n", horiz=TRUE)
```

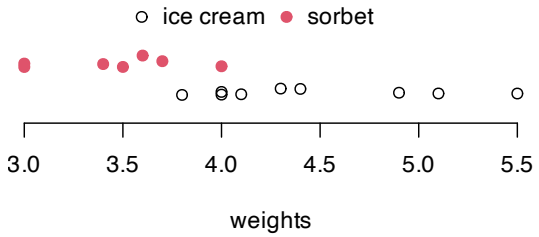

FIGURE 6.3: Comparing ice cream and sorbet weight data. Vertical jitter is added to help visualize duplicates.

We wish to determine if this employee's scoops of ice cream have the same weight variability as his scoops of sorbet. (Not the same mean.) The modeling setup is the same as Eq. (5.5) in §5.2, but I'll repeat it here for convenience.

$$\text{Model:} \quad Y_1, \ldots, Y_{n_y} \overset{\text{iid}}{\sim} \mathcal{N}(\mu_y, \sigma_y^2) \quad \text{and} \quad X_1, \ldots, X_{n_x} \overset{\text{iid}}{\sim} \mathcal{N}(\mu_x, \sigma_x^2)$$

Now, the competing hypotheses of interest, characterizing the "want to know", are different.

$$\begin{aligned} \mathcal{H}_0 &: \sigma_y^2 = \sigma_x^2 && \text{null hypothesis ("what if")} \\ \mathcal{H}_1 &: \sigma_y^2 \neq \sigma_x^2 && \text{alternative hypothesis} \end{aligned}$$

When inspecting means in §5.2 we pivoted from $\mu_y = \mu_x$ to $\mu_y - \mu_x = 0$. We might do that here too, but variances must always be positive, and so it's awkward to take differences because they might go negative. Instead, it's more natural to inspect ratios: $\sigma_y^2/\sigma_x^2 = 1$. There's another good reason for preferring a ratio over a difference that has to do with the math way of conducting the test, which I'll get to soon. A sensible test statistic is therefore

$$f = s_y^2/s_x^2,$$

or vice versa: $f = s_x^2/s_y^2$; it doesn't matter much, although I'll tell you that it's customary to put the larger one (in this case s_y^2) in the numerator. The reason for the letter f, and for this convention, will become apparent later. For now, yes, you guessed it: F is for Fisher.

```
f <- s2y/s2x
f
```

```
## [1] 2.542
```

This value is not, in isolation, very informative. Apparently, the variance of this employee's ice cream scoops is 2.54× larger than sorbet. One problem with that is that humans are bad at interpreting things in units of squared quantities. Back on the scale of standard deviations, which have units of ounces, it's more like 1.59× more. But more importantly, we don't know what sorts of F-values are likely under $\mathcal{H}_0$. We, as yet, have nothing to compare this f statistic to.

Two-sample variance test by MC

R code below samples from the empirical null distribution of F, the random analog of f. As in §5.2, it doesn't matter what values of μ_x and μ_y are used, and they may even be the same: $\mu \equiv \mu_y = \mu_x$. The same is true for $\sigma^2 \equiv \sigma_y^2 = \sigma_x^2$, except that they *must* be the same because $\mathcal{H}_0$ says so. Below, I arbitrarily stick with `mu` defined earlier, and keep it simple with $\sigma^2 = 1$.

```
sigma2 <- 1
Fs <- rep(NA, N)
for(i in 1:N) {
  Ys <- rnorm(ny, mu, sqrt(sigma2))
  Xs <- rnorm(nx, mu, sqrt(sigma2))
  s2ys <- var(Ys)
  s2xs <- var(Xs)
  Fs[i] <- s2ys/s2xs
}
```

When you think about this setup a little, you realize that the only quantities which change from one analysis to the next are n_y and n_x. Nothing else matters. Interesting, right? Figure 6.4 shows the empirical sampling distribution of F based on $n_y = 9$ and $n_x = 7$ from the scooping experiment.

```
hist(Fs[Fs < 8], main="", xlim=range(c(0,8)))  ## focus histogram near 0
abline(v=c(f, quantile(Fs, mean(Fs > f))), lty=1:2, col=2, lwd=2)
legend("topright", c("f", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
```

The sampling distribution of these F-values can be quite heavily right-tailed. This is the reason for `Fs[Fs < 8]`, focusing our attention near the origin. Without that, the x-axis of the histogram would extend out $\sim 50\times$ longer and, partly as a consequence, histogram binning would have low resolution. I'll give another reason why 8 is a good choice later.

```
summary(Fs)
```

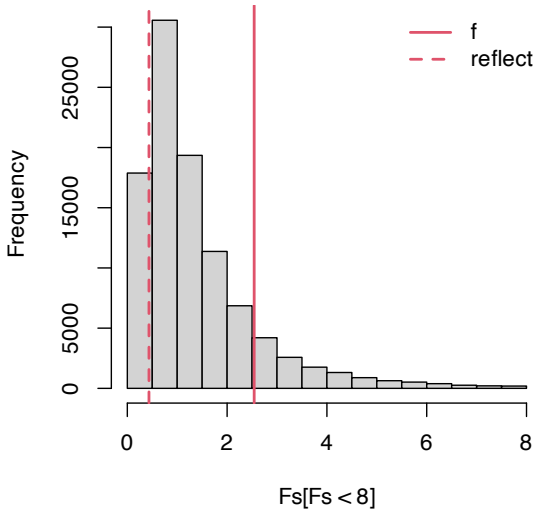

FIGURE 6.4: Empirical sampling distribution of F from the ice cream and sorbet scooping experiment.

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
##   0.0302   0.6066   1.0328   1.5041   1.7798 458.0424
```

It's pretty clear from Figure 6.4 that we do not have enough evidence to reject the null hypothesis. Putting the larger of s_y^2 and s_x^2 in the numerator, so that $f = s_y^2/s_x^2 > 1$, ensures that the statistic is also in the right tail. This leads to a more streamlined two-tailed p-value calculation because the inequality is always "`>`".

```
pval.mc <- 2*mean(Fs > f)
pval.mc
```

```
## [1] 0.2713
```

It turns out that, across these two types of tasty frozen desserts, this otherwise goofball employee in fact has a good, consistent scooping hand. The real issue is that one weight for all ice creams, sorbets, gelatos, fat-free ice creams, etc., might not be appropriate as a measure for quality control.

Two-sample variance test by math

By two applications of analogous one-sample results, ultimately tracing things back to Eq. (3.16),

$$S_y^2 \sim \frac{\chi^2_{n_y-1}\sigma_y^2}{n_y - 1} \quad \text{and} \quad S_x^2 \sim \frac{\chi^2_{n_x-1}\sigma_x^2}{n_x - 1}. \tag{6.2}$$

When studying their ratio, there is some cancellation under the null hypothesis, giving $\sigma_y^2 = \sigma_x^2$.

$$F = \frac{S_y^2}{S_x^2} \sim \frac{\chi^2_{n_y-1}/(n_y-1)}{\chi^2_{n_x-1}/(n_x-1)} \tag{6.3}$$

Ronald Fisher[5] may have been the first to extensively study ratios of normalized χ^2 random variables, but George Snedecor[6] is credited with characterizing a closed form for their distribution. Snedecor graciously named it after Fisher, and it is now known as the F distribution[7]. An F distribution has two parameters, affectionately named the numerator and denominator degrees of freedom (DoF), respectively, which come from the χ^2 statistics located in those positions of the ratio.

Summarizing, we have

$$F \sim F_{n_y-1, n_x-1}. \tag{6.4}$$

It's a little confusing to have a random variable, F, take on the same name and capitalization as F_{n_y-1,n_x-1}. Don't shoot the messenger. I'm not going to quote much else about the distribution, which you can anyways find on Wikipedia, except the following two nuggets. It is a distribution over positive quantities, because it involves ratios of positive (squared) values, and so it has a right skew. Pretty much for any DoF arguments bigger than a handful ($n_x, n_y > 4$ or so), 95% of the cumulative density lies to the left of four. This is similar to 95% of a standard Gaussian or t_ν distribution being in $[-2, 2]$. (And it's not a coincidence that $2^2 = 4$. More on this in §6.4.) So there's a nice rule of thumb for F-tests, like for t- and z-tests. You will likely reject at the 5% level if $f > 4$. It also means there's rarely much cause to plot the density much beyond four. I doubled that to eight in Figure 6.4, and again momentarily in Figure 6.5.

```
fgrid <- seq(0, 8, length=1000)
plot(fgrid, df(fgrid, ny - 1, nx - 1), type="l", lwd=2, main="")
abline(v=c(f, qf(pf(f, ny - 1, nx - 1, lower.tail=FALSE), ny - 1, nx - 1)),
  lty=1:2, col=2, lwd=2)
legend("topright", c("f", "reflect"), lty=1:2, col=2, lwd=2, bty="n")
```

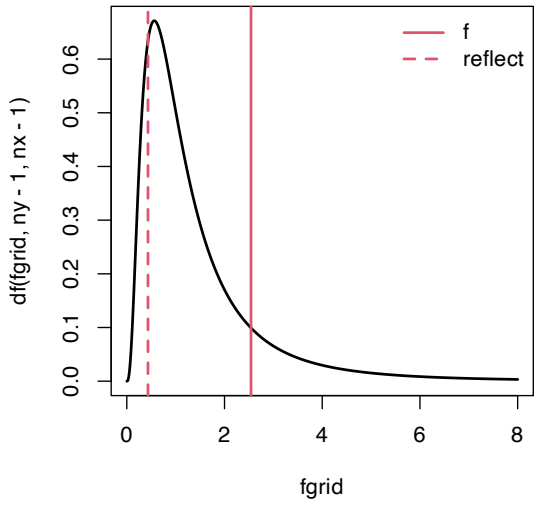

FIGURE 6.5: F distribution and observed f statistic for the ice cream and sorbet test.

[5] https://en.wikipedia.org/wiki/Ronald_Fisher
[6] https://en.wikipedia.org/wiki/George_W._Snedecor
[7] https://en.wikipedia.org/wiki/F-distribution

This view is similar to Figure 6.4, and therefore so is the p-value.

```
pval.f <- 2*pf(f, ny - 1, nx - 1, lower.tail=FALSE)
c(mc=pval.mc, f=pval.f)
```

```
##     mc      f
## 0.2713 0.2719
```

There's a built-in library function for two-sample variance tests in R, and it gives the same answer as our by-hand calculation.

```
var.test(y, x)$p.value
```

```
## [1] 0.2719
```

You can build a CI for a ratio of variances, but I'm not going to bore you with that, or even leave it to you as a homework exercise. You're welcome. If you're really curious, just follow the example(s) provided by the one-sample test(s).

A two-sample variance uses an f statistic under an F distribution, so you might think you're doing an F-test. Sort of, but that's not what most people mean when they say they're doing an F-test. They're thinking of ANOVA, which is what I'm about to cover next (§6.3), or they're thinking of a (multiple) linear model selection test which is the subject of §13.5. And those two things are related.

6.3 ANOVA

Alright, suppose we had one more set of scoops, this time for servings of fat-free ice cream. These numbers are stored in `w` below.

```
w <- c(3.5, 3.4, 3.7, 3.4, 3.9, 4.2, 4.0, 4.0, 3.8, 3.8)
```

But I want to reorganize things a little, because I want to get away from y, x, and w notation. That's not scalable. I need a data structure that can have any number of categories: $1, 2, 3, \ldots, m$ for arbitrary m. The first way is as a `list` in R.

```
ylist <- list(cream=y, sorbet=x, fatfree=w)
ylist
```

```
## $cream
## [1] 4.0 5.5 4.9 3.8 4.3 4.4 5.1 4.1 4.0
##
## $sorbet
## [1] 3.0 3.7 3.4 3.6 3.0 3.5 4.0
##
## $fatfree
```

```
##  [1] 3.5 3.4 3.7 3.4 3.9 4.2 4.0 4.0 3.8 3.8
```

Having a `list` makes it easy to extract information by "applying" a function over entries in the list, like using `length` with `sapply`.

```
nj <- sapply(ylist, length)
m <- length(nj)
n <- sum(nj)
nj
```

```
##   cream  sorbet fatfree
##       9       7      10
```

I'll explain those variables momentarily, and their names will make more sense in due course. R has many automations built in for different (common) statistical data types. Another way to represent similar information in R is as a `data.frame`.

```
ydf <- data.frame(weight=unlist(ylist), type=factor(rep(1:m, nj)))
```

I'm not going to print that one to the page because it takes up too much space. I encourage you to do that on your own. Many things that `sapply` can do for `list` objects, `tapply` accomplish on a `data.frame`. I'll show you later. Figure 6.6 demonstrates how both representations can be useful for visualization, with actual observations overlaid on a "box-and-whisker"[8] summary (a.k.a., "boxplot") of those values.

```
boxplot(ylist, border=1:3, ylab="weight")
jit <- rnorm(n, 0, 0.05)
points(as.numeric(ydf$type) + jit, ydf$weight, pch=19,
  col=ydf$type, cex=0.75)
```

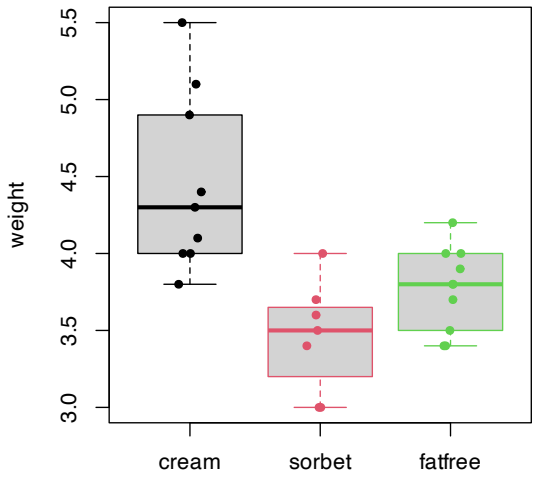

FIGURE 6.6: Visualizing the full ice cream scoop weight data set, with horizontal jitter.

Boxplots are my go-to for exploratory analysis of so-called "grouped" data in R. After plain

[8] https://en.wikipedia.org/wiki/Box_plot

`plot`, I use `boxplot` more than any visualization function in R. Let me encourage you read up on boxplots so you can see how the gray rectangle (the "box") and their interval extension ("whiskers") are defined. Be careful, the thin black line in the middle is the sample median, not the average. Sometimes, a small number of outliers are shown as open circles beyond the whiskers, but there aren't any data points meeting the criteria for that in the scoop data.

Depending on what you want to do, a `list` or `data.frame` representation of grouped data might be more handy. Both are flexible enough to hold the type of data we're considering here. Although there are only $m = 3$ groups in the scoop data, what I'm about to show you applies for arbitrary m, and any amount of data in each group, represented by `nj` in the code. Both the math and the code are intended to be generic enough for a wide class of problems/data of this type.

Now, as I said in the preamble to this chapter, an analysis of variance (ANOVA)[9] is really a study of means. I will show you how to use ANOVA to tell if the scooping weights of Figure 6.6 really come from populations with different means, that is, whether this employee (using a standard scooper) is making scoops of different weights, on average, depending on what type of frozen treat is being scooped. Informally, that's our "want to know". More generically, we wish to tell if a grouping variable – in this case, frozen scooped treat type – is a statistically significant *location* factor in the *variability* of the y-values – in this case, weights.

Notice how I'm being careful with my words in italics. In ANOVA we do study variability (the V in ANOVA). Still, I think it's a misnomer because means is what we really want to know about. That'll become more clear when I lay out the modeling apparatus and hypotheses.

ANOVA model, hypotheses and statistic

Since the number of groups, m, is arbitrary, we'll have to be a little more nimble with notation. Don't be frightened by the double-subscript indices. When you ponder it, they help with compactness. The model behind ANOVA is as follows.

$$\text{Model:} \qquad Y_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(\mu_j, \sigma^2) \qquad i = 1, \dots, n_j \;\text{ and }\; j = 1, \dots, m \tag{6.5}$$

Similarly, record observations as y_{ij} over the same indices. You can say that y_{ij} stores the i^{th} observation in the j^{th} group. As shorthand, it helps to extract quantities like

$$y_{\cdot j} \equiv y_{1,j}, \cdots, y_{n_j,j},$$

to refer to a whole collection of data values in a particular (say j^{th}) group.

An important thing to notice is that variance σ^2 doesn't have any subscripts, but means μ_j do. Each group has its own mean, but they all share a common variance. So Eq. (6.5) nests the pooled setup of §5.2 in the $m = 2$ case. In that context, we might rewrite the $j = 1$ group as $y_1, \dots, y_{n_y}$ with $n_y \equiv n_1$ and the $j = 2$ group as $x_1, \dots, x_{n_x}$ with $n_x \equiv n_2$.

That's the ANOVA model. An ANOVA system of competing hypotheses emphasizes distinguishing between the null that samples from all m groups *also* (in addition to variance) share the same mean, or the alternative that the sample from at least one group differs in its mean from some other.

[9] https://en.wikipedia.org/wiki/Analysis_of_variance

$$\begin{aligned} &\mathcal{H}_0 : \mu_1 = \mu_2 = \cdots = \mu_m && \text{null hypothesis (“what if”)} \\ &\mathcal{H}_1 : \mu_j \neq \mu_k \text{ for some } j \neq k \in \{1, \ldots, m\} && \text{alternative hypothesis} \end{aligned} \tag{6.6}$$

That complicated $\mathcal{H}_1$ is important. It says that there's at least one pair with differing means. There doesn't need to be more than one, and $\mathcal{H}_1$ doesn't say which ones. Conversely, $\mathcal{H}_0$ says *all* means are equal. So the null is stringent or “tight”, but the alternative is “loose”. If $m = 2$, covering the pooled case in §5.2, you'd have to clean up Eq. (6.6) a little because you'd have a redundant $\mu_2 = \mu_m$. I'm not going to worry about that and presume $m \geq 3$ going forward. I'll have more to say about the equivalence to a pooled two-sample t-test in the $m = 2$ case later, in §6.4.

We already know a lot about how to estimate these parameters, especially μ_j for $j = 1, \ldots, m$. Simply appropriate results from Chapters 3 and 5. Since there's independence all around, both within and between (or some say “across”) samples from each of m groups, MLE calculations are straightforward.

$$\hat{\mu}_j = \bar{y}_{\cdot j} \quad \text{where} \quad \bar{y}_{\cdot j} \equiv \frac{1}{n_j} \sum_{i=1}^{n_j} y_{ij}, \quad \text{for } j = 1, \ldots, m$$

Quantity $\bar{y}_{\cdot j}$ is sometimes called the “within-group” average. In R, this is easy to implement with an `sapply` over `mean` in the `list` setup.

```
ybarj <- sapply(ylist, mean)      ## or tapply(ydf$weight, ydf$type, mean)
ybarj
```

```
##   cream  sorbet fatfree
##   4.456   3.457   3.770
```

An estimate for pooled σ^2 may be found by extending Eq. (5.7).

$$s^2_{\text{pool}} = \frac{\sum_{j=1}^m (n_j - 1) s_j^2}{n - m} \quad \text{where} \quad s_j^2 = \frac{1}{n_j - 1} \sum_{i=1}^{n_j} (y_{ij} - \bar{y}_{\cdot j})^2 \quad \text{and} \quad n = \sum_{j=1}^n n_j \tag{6.7}$$

Forgive me for not trying to squeeze in a $j = 1, \ldots, m$ above. In R ...

```
s2j <- sapply(ylist, var)         ## or tapply(ydf$weight, ydf$type, var)
s2.pool <- sum((nj - 1)*s2j)/(n - m)
s2.pool
```

```
## [1] 0.1809
```

Actually, `s2.pool` isn't directly involved in testing, but it will be handy for a visual coming later.

Under the null hypothesis (6.6) we have that $\mu \equiv \mu_1 = \mu_2 = \cdots = \mu_m$. In other words, there's just one big Gaussian for everybody: $Y_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$. Double-indexing is a nuisance, but we know how to estimate parameters in that setting.

$$\hat{\mu} = \bar{y} = \frac{1}{n}\sum_{j=1}^{m}\sum_{i=1}^{n_j} y_{ij} \qquad \text{and} \qquad s^2 = \frac{1}{n-1}\sum_{j=1}^{m}\sum_{i=1}^{n_j}(y_{ij} - \bar{y})^2. \tag{6.8}$$

Quantity $\bar{y}$ is sometimes referred to as the "grand average" or the "across group average". Similarly, s^2 is a grand sample variance, but $s^2 \neq s^2_{\text{pool}}$. In R, calculations are simple after you strip away the list structure with `unlist`.

```
ybar <- mean(unlist(ylist))     ## or mean(ydf$weight)
s2 <- var(unlist(ylist))        ## or var(ydf$weight)
c(ybar=ybar, s2=s2)
```

```
##   ybar     s2
## 3.9231 0.3386
```

The testing apparatus behind ANOVA involves a statistic that targets a contrast between Eqs. (6.7) and (6.8) under $\mathcal{H}_0$. In other words, it involves analyzing the variance under the null and alternative models. But it does so in service of testing about means (6.6). Before I get to explaining that statistic, I want to draw a picture. I think it will help motivate the thought process behind the statistic.

Figure 6.7 is the most ambitious graphic I've attempted so far. I don't claim to be an artist! The setup is similar to Figure 6.6, but not with boxplots. Instead, alongside those data are m fitted Gaussian densities under the model in Eq. (6.5). Notice how `dnorm` is used to draw those densities. In fact, they should be Student-t densities (`dt`), but that code would have been more involved because `dt` doesn't have `mean` and `sd` arguments. Forgive me, or blame R, or let it pass. To the right of the dashed line is the combined data, and its Gaussian fit under $\mathcal{H}_0$. In each case, a $*$ indicates the location of the mean estimate, either $\bar{y}_{\cdot,j}$ or $\bar{y}$.

```
## prepping the plot with data under unrestricted model and H0
ylist.H1 <- ylist
ylist.H1$all <- ydf$weight
boxplot(ylist.H1, col=0, border=0, ylab="weight",
  xlim=c(0.75, 4.4), ylim=c(2.9, 6.1))
points(as.numeric(ydf$type) + jit, ydf$weight,
  pch=19, col=ydf$type, cex=0.75)
abline(v=3.6, lwd=2, lty=2)

## visualizing the Gaussian fits (actually Student-t)
ygrid <- seq(2.75, 5.75, length=1000)
for(j in 1:m) {
  points(j, ybarj[j], pch=8, col=j, cex=1.25)
  lines(j + 0.4*dnorm(ygrid, ybarj[j], sqrt(s2.pool)), ygrid, col=j)
}
text(2.1, 6, "unrestricted mean, pooled var")

## similar calculations under H0
points(rep(m + 1, n) + jit, ydf$weight, col="gray", cex=0.75)
points(m + 1, ybar, pch=8, col="gray", cex=1.25)
```

```
lines(m + 1 + 0.4*dnorm(ygrid, ybar, sqrt(s2)), ygrid, col="gray")
text(4.05, 6, "null")
```

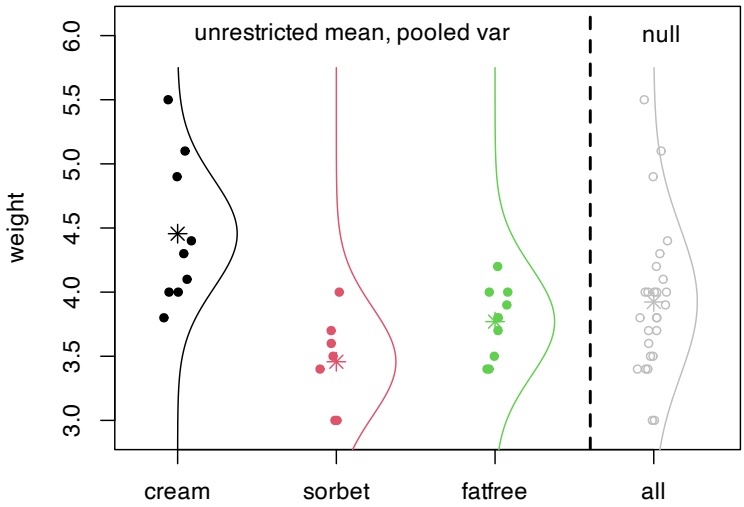

FIGURE 6.7: Illustrating ANOVA fit(s), overlaying Figure 6.6 with Gaussian fits under $\mathcal{H}_0$.

Observe how fits with disparate means, to the left of the vertical dashed line, all have the same spread. This is because they use all s^2_{pool}. (The rightmost fit, under $\mathcal{H}_0$, has wider spread via s^2.) We know already, from our analysis earlier in §6.2, that the first two groups likely have the same variance. We could also test the other two pairs. I'll spoil it for you: one rejects, and the other accepts. So it may be that a pooled variance model is not appropriate for these groups. This is a downside of the typical ANOVA setup. We've already seen, in §6.2, that the math is much easier with a pooled variance. The same is true here, too. We'll look at relaxing this later, beginning with a homework exercise in §6.5.

The statistic preferred in an ANOVA doesn't involve any assumptions about pooled variances; however, its distribution does. In fact, at first blush it would appear, rather, to involve separately estimated variances. Remember, the testing statistic is technically your choice. I'm showing you what's canonical, which is usually based on a compromise between what works (what has good power, §A.2) and what has a distribution available in closed form. With MC, we're less beholden to closed forms, but it's important to make connections to established procedures.

Alright, here we go. Above, I said that the ANOVA test statistic relates the quality of unconstrained ($\mu_j \neq \mu_k$) fits under $\mathcal{H}_1$ via $\bar{y}_{\cdot j}$ to a constrained one ($\mu \equiv \mu_1 = \mu_2 \cdots = \mu_m$) under $\mathcal{H}_0$ via $\bar{y}$. A neat identity that makes this possible is provided below.

$$\sum_{j=1}^{m}\sum_{j=1}^{n_j}(y_{ij} - \bar{y})^2 = \sum_{j=1}^{m}\sum_{i=1}^{n_j}(\bar{y}_{\cdot j} - \bar{y})^2 + \sum_{j=1}^{m}\sum_{i=1}^{n_j}(y_{ij} - \bar{y}_{\cdot j})^2 \tag{6.9}$$

$$\text{SST} = \text{SSR} + \text{SSE}$$

Showing this is true is left as a homework exercise. You may have done one like this before in §3.5 #5. The second line above defines some acronyms for the components of the identity.

SS stands for "Sum of Squares". The third letter – E, R or T – is supposed to help you remember which part is what, though I still get it confused.

- E is for "Error", which is a strange choice. All of these SSs are measuring (squared) errors of a sort; here it's the error in predicting y_{ij} with $\bar{y}_{\cdot j}$. Each component in the sum for SSE is a scaled within-group variance estimate: SSE $= \sum_{j=1}^{m}(n_j - 1)s_j^2$. The more useful the grouping variable, the smaller the SSE.

```
sse <- sum((nj - 1)*s2j)
```

- R is for "Regression", which is awkward because we haven't talked about what regression is yet. That comes in Chapter 7. Anyways, this one is measuring how different each within-group average $\bar{y}_{\cdot j}$ is from the grand average $\bar{y}$. The more useful the grouping variable, the larger the SSR.

```
ssr <- sum(nj*(ybarj - ybar)^2)
```

- T is for "Total". This one is easy to remember. SST $= (n-1)s^2$ under $\mathcal{H}_0$, measuring quality of fit for all of the data around the grand average $\bar{y}$.

```
sst <- ssr + sse
```

Well there you have it. Again, please don't shoot the messenger. I bet the machine learning community (if it cared about ANOVA) would have come up with better shorthands. I'll show you actual `sse`, `ssr`, and `sst` values from data later. They're not germane to our discussion at this time.

Since the whole (SST) equals the sum of its parts (SSR and SSE), any of the parts over the whole is a proportion, and you can always get the third one from the other two. Compactly:

$$0 < r^2 \equiv \frac{\text{SSR}}{\text{SST}} = 1 - \frac{\text{SSE}}{\text{SST}} < 1. \tag{6.10}$$

Ratio $r^2 = \text{SSR}/\text{SST}$, verbalized as "r-squared", is also known as the coefficient of determination[10]. The reason for the lettering r^2 has to do with a correlation and linear regression connection coming later in Chapters 7 and 12. The reason for that strange name is that statisticians are bad at naming things. Quantity r^2 is always positive because squares are positive.

```
r2 <- ssr/sst
r2
```

```
## [1] 0.5086
```

An r^2 value may be interpreted in percent terms. The grouping, in this case by frozen treat type, explains 51% of the total variability in the data.

Quick digression on the inequalities in Eq. (6.10). These should technically not be strict – both should be $\leq$ – but something is very wrong if you get an r^2 that's zero or one. You

[10] https://en.wikipedia.org/wiki/Coefficient_of_determination

either have $n_j = 1$ for some j, or $n_j > 1$ but with duplicated values, and/or $m = n$. Those are degenerate cases with more estimated quantities than (unique) data values. You have a DoF problem, or a bug in your code.

Alright, back to ANOVA. The important thing to take away from all this is that a grouping is valuable, compared to not grouping at all (i.e., under $\mathcal{H}_0$), if it explains most of the variability. This is where the V in ANOVA comes from. If r^2 is large, close to 1, then the grouping in question is valuable. As usual, "large" must be measured statistically, against its sampling distribution under $\mathcal{H}_0$.

Conversely, if SSE is small, then within-group averages $\bar{y}_{\cdot j}$ are close to their y_{ij}-values, providing a good fit. The closer they are the smaller the aggregate squared distances, SSE, and therefore the larger amount left over in SSR. Coming full circle, SSR is large when $\bar{y}_{\cdot j}$ values are far from one another, and from the grand average $\bar{y}$.

Summarizing: the grouping is useful when between-group variability, SSR, is large relative to within group variability, SSE. Turning that into a test involves inspecting those quantities under $\mathcal{H}_0$, and forming a contradiction.

I suggest reading over the previous three paragraphs a few times to make sense of things. Refer back to Figure 6.7, Eq. (6.9) and subsequent bullets, and ponder. If you're still struggling, don't worry. I got it all wrong twice when writing those passages, because things were all twisted up in my head, before getting it right. Quantities like r^2, and the three SSs are just statistics.

It bears repeating that what matters is how a statistic looks under a sampling distribution implied by $\mathcal{H}_0$. That's actually pretty easy and comes next. The idea behind ANOVA is that if some $\mu_j \neq \mu_k$, under the alternative hypothesis $\mathcal{H}_1$, then these quantities will look "out-of-whack" relative to one another under the sampling distribution from $\mathcal{H}_0$. Since they're related to one another as a sum of parts (6.9), a ratio like r^2 provides a nice, interpretable scalar statistic for inspection.

It turns out that a different, but related, ratio has some advantages mathematically. I'm going to quote it here, and talk more about it when it comes to closed-form calculations later. When testing via MC, we have more flexibility when it comes to the choice of test statistic.

$$f = \frac{\mathrm{SSR}/(m-1)}{\mathrm{SSE}/(n-m)} \equiv \frac{\mathrm{MSR}}{\mathrm{MSE}} \tag{6.11}$$

```
msr <- ssr/(m - 1)
mse <- sse/(n - m)
f <-  msr/mse
```

You guessed it, the random F that you get by replacing y_{ij} with Y_{ij}, etc., has an F distribution. But I'm getting ahead of myself. The M in MSE and MSR stands for "Mean", because we're dividing by the number of DoF involved in the sums SSE and SSR, respectfully. The reason for putting this here, as opposed to later when I get to the math version of the test, is so that I can draw a comparison between r^2 and f values via MC.

ANOVA test by MC

That was complicated; I apologize. This is going to be a lot easier. Why? Because it's like a two-sample (pooled variance) t-test from §5.2, but with m samples. And at the end we

calculate R^2 or F, thinking now of them as random quantities. Finally, a comparison is drawn to observed versions: r^2 or f.

As with other examples involving a common mean μ and/or variance σ^2 under $\mathcal{H}_0$, it doesn't matter what values are used in the simulation. Your instinct may be to iterate over the m groups with a `for` loop, and there's nothing wrong with that. However, a `list` representation makes things tidier with `lapply` and `sapply` and is more faithful to data structures for, and manipulation of, the observed data and its statistics. See comments below for how a `for`-loop variation might work instead.

```
N <- 1000000
mu <- sigma2 <- 1               ## any mu and sigma2 values work
R2s <- Fs <- rep(NA, N)         ## storage for both stats
for(i in 1:N) {

  ## draw under null hypothesis
  ## essentially a "for" loop {
  Ys <- lapply(nj, rnorm, mean=ybar, sd=sqrt(s2))

  ## within-group calculations
  Ybarjs <- sapply(Ys, mean)
  S2js <- sapply(Ys, var)
  ## } where the "for" loop would end

  ## grand average
  Ybars <- sum(nj*Ybarjs)/sum(nj)

  ## sums of squares
  SSEs <- sum((nj - 1)*S2js)
  SSRs <- sum(nj*(Ybarjs - Ybars)^2)

  ## sampled R^2 statistic
  SSTs <- SSEs + SSRs
  R2s[i] <- SSRs/SSTs

  ## sampled F statistic
  MSRs <- SSRs/(m - 1)
  MSEs <- SSEs/(n - m)

  ## f statistic
  Fs[i] <- MSRs/MSEs
}
```

Figure 6.8 provides views of both sampled statistics, R^2 on the left and F on the right, with observed values overlayed as red vertical lines. ANOVA is always done as a one-tailed test. It could be done two-tailed, in the same way as other two-tailed tests (with a reflected statistic, etc.), but we're not going to do it that way here so that everything will line up with library output, coming later. There's also a good reason for doing things one-tailed, and I'll get back to that later in §6.4 after I've covered some of the math.

```
par(mfrow=c(1, 2))
hist(R2s, main="")
abline(v=r2, col=2, lwd=2)
hist(Fs, main="")
abline(v=f, col=2, lwd=2)
legend("topright", "observed stat", lty=1, lwd=2, col=2,  bty="n")
```

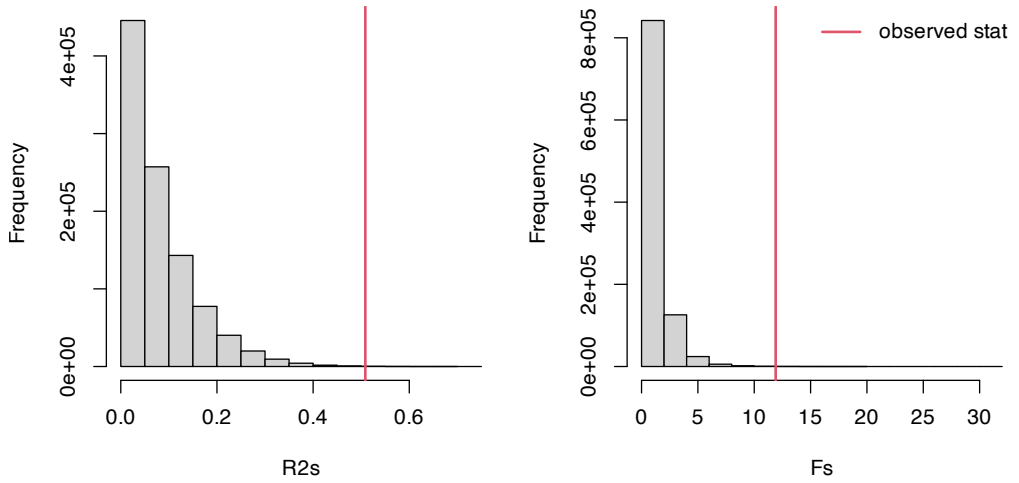

FIGURE 6.8: Histogram of sampled R^2 (left) and F-values (right) for the scoop weight data under the null hypothesis, with observed statistics overlayed.

Looking at either of those histograms, it is clear that the observed statistic is extreme compared to its sampling distribution(s). This should be an easy reject of the null hypothesis, concluding that at least one pair of means differs. Code provided below calculates p-values in the usual way, under both stats/distributions.

```
pval.mc <- c(r2.mc=mean(R2s > r2), f.mc=mean(Fs > f))
pval.mc
```

```
##    r2.mc     f.mc
## 0.000274 0.000274
```

In situations where both right tails are light, p-values under R^2 or F are often the same or very close. R^2 values are always between 0 and 1, and so its distribution doesn't really have tails in quite the same way. This can impact how an r^2 tail probability compares to an f one. However, in this particular instance, both distributions are "light" at their extremes, as shown in the figure. Both p-value calculations are also based on exactly the same pseudo-random numbers. So the two numbers are identical.

I prefer r^2 because it's both simpler, and has nice interpretation. Historically, f is preferred because of how the math turns out.

ANOVA test by math

Let's begin by repurposing the analysis of §6.2 around Eqs. (6.2) and (6.3). That was really an adaptation of Eq. (3.16) from §3.2, where I introduced the t-test. Sorry about using

all those equation references all at once. Bear with me. Referring back to Eq. (6.9) in the context of Chapter 3, we have

$$\text{MSR} = \frac{\text{SSR}}{m-1} \sim \sigma^2 \chi^2_{m-1} \qquad \text{and} \qquad \text{MSE} = \frac{\text{SSE}}{n-m} \sim \sigma^2 \chi^2_{n-m}. \tag{6.12}$$

As in Chapter 3, I'll wave my hands at this a little and offer the following justification that involves counting DoF in sums of squares. SST involves estimating one quantity, $\bar{y}$, and summing up n squares from y_{ij}, and therefore has a χ^2_{n-1} distribution. SSE involves estimating m quantities and then summing up n squares. Since SST = SSR + SSE, we may deduce that SSR involves $n-m$ DoF. Compactly, $n-1 = n-m+m-1$.

A key feature of Eq. (6.12) is that σ^2 is involved in the distribution of both MSR and MSE. That way σ^2 cancels out in the f ratio (6.11). This is where a pooled variance assumption (6.5) is pivotal. Without that, there's no cancellation. In fact, it's not clear at all what you'd get otherwise. An ordinary, unpooled two-sample t-test requires approximation, e.g., via the Welch test. Or, worse still, you fall deep into a rabbit hole and develop a whole new form of inference. (See discussion in §5.2.) That hole is m-fold deeper in an ANOVA context when you don't pool variance across all m samples.

But when you do pool variance, everything falls out nicely. Combining Eqs. (6.3) and (6.4), the f statistic (6.11) – when viewed as a random variable for random Y_{ij}, $\bar{Y}_{\cdot j}$ and $\bar{Y}$ – has the following distribution.

$$F \sim F_{m-1,n-m}$$

A p-value for observed f under this distribution may be calculated by integrating into the right tail as follows.

```
pval.math <- pf(f, m-1, n-m, lower.tail=FALSE)
c(f.math=pval.math, pval.mc)
```

```
##    f.math     r2.mc      f.mc
## 0.0002829 0.0002740 0.0002740
```

That's pretty close to our MC variation. With data arranged into a `data.frame`, a library call could look like this.

```
fit <- aov(weight ~ type, data=ydf)
anova(fit)$Pr[1]
```

```
## [1] 0.0002829
```

That'll take a little unpacking. The function `aov` fits the ANOVA model (6.5). If you want, you can inspect the intermediate `fit` object. I find that not very enlightening because of the way things are stored/presented. I'll have to circle back to that later after I introduce multiple linear regression in Chapter 13, for which ANOVA is a special case.

The first argument to `aov` is a `formula` that explains how to interpret the second, `data` argument. Here, the `formula` explains that `weight` is y and `type` is the grouping. In this context, `type` is sometimes called an explanatory or independent variable[11], and `weight` is

[11] https://simple.wikipedia.org/wiki/Dependent_and_independent_variables

TABLE 6.1: ANOVA table for scooping weights.

	DoF	SS	MS	fstat	pval
Regress	2	4.306	2.1529	11.9	3e-04
Error	23	4.160	0.1809		
Total	25	8.466			

the dependent or response variable, or outcome. More on these vocabulary words is provided near Table 7.1 in Chapter 7.

Calling `anova` on the `fit` provides the actual analysis. I've extracted the p-value, above, for quick comparison to the other calculations. The full output is possibly of interest, so it's provided below.

```
anova(fit)
```

```
## Analysis of Variance Table
##
## Response: weight
##           Df Sum Sq Mean Sq F value  Pr(>F)
## type       2   4.31   2.153    11.9 0.00028 ***
## Residuals 23   4.16   0.181
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

This is what is known as an "ANOVA table", summarizing the outcome of intermediate steps via key quantities. We have all of the ingredients from our own calculations, and can thus hand-build our own ANOVA table as follows. See Table 6.1.

```
tab <- cbind(c(m - 1, n - m), c(ssr, sse), c(msr, mse),
  c(f, NA), c(pval.math, NA))
tab <- rbind(tab, c(sum(tab[,1]), sum(tab[,2]), rep(NA, 3)))
rownames(tab) <- c("Regress", "Error", "Total")
colnames(tab) <- c("DoF", "SS", "MS", "fstat", "pval")
kable(round(tab, 5), caption="ANOVA table for scooping weights.")
```

Notice I've renamed things a little bit. The biggest change is in the row labels. The "E" in SSE is for "Error", but R calls this "Residuals" which starts with an R, clashing with SSR and potentially causing confusion. So I use "Regress" and "Error". That way things line up better with notation introduced earlier, which is fairly standard. I've also added an extra "Total" row. This doesn't appear in the R output, but is also typical for ANOVA tables and is relatively self-explanatory in light of Eq. (6.5).

ANOVA tables were helpful back in the day to check your bookwork, especially in a classroom, homework, or exam context where you're doing a lot of it by hand. It's so easy to make a little mistake and throw everything off. Neatly arranging intermediate output allows you, or a grader, to trace back through your calculations to see where the problem might be. (Always show your work!) ANOVA tables are less useful in a modern context, although perhaps you could still be asked to fill one out on an exam.

Just in case you encounter it – I've got a good one for you in §6.5 – it's important to know how the rows and columns work together. Observe that the third column is the ratio of the first two, and the fourth one is the ratio of the two entries of the previous one. (I've deliberately been a little terse in that explanation as a nudge to you to investigate what, exactly, I might mean.)

6.4 Multiple testing hazard

I've said there's a connection between ANOVA and a two-sample, pooled-variance t-test. When $m = 2$, the models (6.5) are identical. They use different test statistics (f and t, respectively), which are measured against different sampling distributions. Are they different tests? Let's find out.

But wait, watch me kill two birds with one stone while pivoting to a new topic. We've determined, with the ANOVA analysis above, that at least one of these three things is not like another. One way to get to the bottom of that is to explore groups pairwise. R code below loops over all pairs, first performing t-tests, then performing $m = 2$ ANOVA tests on pairs. These are stored in opposing triangles of a symmetric matrix for easy viewing, provided Table 6.2.

```
## matrix storing p-values for all pairs
pval.pairs <- matrix(NA, nrow=m, ncol=m)
colnames(pval.pairs) <- rownames(pval.pairs) <- names(ylist)

## loop over all pairs
for(i in 1:(length(ylist) - 1)) {
  for(j in (i + 1):length(ylist)) {

    ## pooled-variance t-test
    pval.pairs[i,j] <- t.test(ylist[[i]], ylist[[j]], var.equal=TRUE)$p.val

    ## anova test
    r <- ydf$type %in% c(i,j)
    pval.pairs[j,i] <- anova(aov(weight ~ type, data=ydf[r,]))$Pr[1]
  }
}

## pretty table
kable(pval.pairs,
  caption="$p$-values from ANOVA (rows) and
  pooled-var $t$-tests (columns).")
```

Two key takeaways from the table are as follows:

1. The p-values in the $(i, j)^{\text{th}}$ cell match those in the $(j, i)^{\text{th}}$ one, so the tests have the same outcome. Ask the same question, get the same answer. That's a relief. This explains, at least in part, why ANOVA is a one-tailed test, because that's how you match the result of a two-tailed, pooled variance t-test. In a way, ANOVA is already two-tailed since

TABLE 6.2: p-values from ANOVA (rows) and pooled-var t-tests (columns).

	cream	sorbet	fatfree
cream		0.0014	0.0038
sorbet	0.0014		0.0598
fatfree	0.0038	0.0598	

it looks at squared deviations from averages. Squares treat under- and over-prediction symmetrically.

2. Regular, full-fat ice-cream seems to be heavier than both sorbet and fat-free ice cream. But there's no detectable weight difference between sorbet and fat free ice cream.

You might be wondering about what the value of ANOVA is, when you can just do a bunch of paired t-tests. The answer is hidden in the name of this section. Simply put, the more questions you ask, the more chances you have of messing it up. If what you want to know is whether (at least) one thing is not like another, you're more likely to get it wrong if you ask $\binom{m}{2} = m(m-1)/2$ questions, which is 3 when $m = 2$, than if you just ask one question.

Our hypothesis tests are set up to reject $\mathcal{H}_0$ when the p-value is below $\alpha = 0.05 = 1/20$. So $1/20$ is our chance incorrectly rejecting the null hypotheses (a so-called Type-I error), when there's a close call (p near α). Those chances compound each time we look at a p-value and compare it to α. When $m \geq 7$ we have $m(m-1)/2 > 20$, possibly much bigger than 20. So if you proceed pairwise, with 20+ paired tests, you're essentially guaranteed to get something wrong. It's much safer, though not easier, to do an m-fold ANOVA, which asks just one question. If and when you reject the m-fold ANOVA null, you might find it handy to drill down with smaller tests.

I'm describing what's known as the multiple testing problem[12], but I prefer the word hazard over problem. There are some good ways to work around the hazard, and I encourage you to read more. There are many contexts where you might be looking for a needle in a haystack, and thus feel compelled to do thousands of tests. Great examples come from genomics and genetic association studies where comparisons (mini-experiments) are drawn on tens of thousands of genes. That isn't going to work out well without some adjustment.

Ignoring the multiple testing hazard is a common mistake known as p-hacking[13] or data dredging. "Hacking" shouldn't have a negative connotation. There are plenty of noble hackers. Nonetheless, you don't want to be caught p-hacking. Instead, remember to make a Bonferroni correction[14], or make some other similar adjustment, to account for compounding Type-I error.

[12] https://en.wikipedia.org/wiki/Multiple_comparisons_problem
[13] https://en.wikipedia.org/wiki/Data_dredging
[14] https://en.wikipedia.org/wiki/Bonferroni_correction

6.5 Homework exercises

These exercises help gain experience with one- and two-sample variance tests, and ANOVA. When the question involves performing a test for data, I suggest doing it both ways: via MC and with math. Check with your instructor for details.

Do these problems with your own code, or code from this book. I encourage you to check your work with library functions like `var.test` *and* `anova`*. Check with your instructor first to see what's allowed. Unless stated explicitly otherwise, avoid use of libraries in your solutions.*

#1: MC CI for one sample

Write code that calculates a CI for σ^2 in a Gaussian sample via MC (6.1). Use your code to calculate the interval for ice cream scoops in §6.1, and compare your result to values stored in `CI.math`.

#2: `monty.var` for one- and two-samples

Write a function called `monty.var` that automates testing and CI calculation (see #1 above) for one- and two-sample variance tests. Follow specifications similar to `monty.bern` and `monty.t` one- and two-sample tests from earlier chapters, including optional visuals. I don't need to rehash all that here. Just make sure you cover both MC and math-based procedures.

Hint: many of the following questions are easier after this one is squared away.

#3: Postal scales

A budget supplier of postage scales claims that its scales are accurate for letters weighing less than 10 oz because the standard deviation of measurements, across weights and scales, is 0.1 oz (i.e., $\sigma^2 = 0.01$). Twenty-five scales were tested with a single letter weighing 2 oz and the sample variance was $s^2 = 0.02$. Do you have enough evidence to refute the supplier's claim?

#4: Pitching machine

A pitching machine[15], used by baseball players to practice hitting, is designed to serve up baseballs at a desired speed, e.g., 80 mph, with a standard deviation of 1 mph. Radar, accurate to 1/100 mph, was used to measure the speeds of ten pitches with the machine set to 80 mph.

```
mph <- c(80.05, 79.82, 80.13, 80.18, 79.81, 79.1, 80.03, 80.41,
  80.30, 79.71)
```

What do you think about the actual variance of balls flung by this pitching machine?

#5: Nitrogen absorption from fertilizer

Two kinds of fertilizer (A and B) are tested in a laboratory, where an experiment was performed to measure the nitrogen absorption rate for a corn crop. Seven plots of corn with fertilizer A yielded a sample absorption rate with $s_A^2 = 2.1$ (in units of %-squared). Nine

[15] https://en.wikipedia.org/wiki/Pitching_machine

plots of corn with fertilizer B yielded $s_B^2 = 0.8$. Is the variability of nitrogen absorption similar for the two fertilizers?

#6: Homemade energy drinks

A budget- and health-conscious triathlete has been experimenting with a new energy drink recipe that contains 1 liter of water, 80 grams of sugar, 2 ml of salt, and 5 tablespoons of lime juice. She has two methods of combining these ingredients. Method A involves following the recipe on a per-bottle basis, filling one at a time in this way. (Note: 1 liter bottles are used, so the recipe includes slightly less water than is prescribed, so all the other ingredients can fit.) Method B involves combining all the ingredients (approximately 18-fold) into a five-gallon jug, mixing, and the dispensing into 18 one-liter bottles. She wants to know if there is any difference in the variance of sugar content, measured in g/L, between the two methods. Some measurements are provided below.

```
A <- c(85.1, 85.0, 85.1, 84.8, 85.1, 84.8, 83.5, 84.9, 84.2, 85.0)
B <- c(76.3, 75.8, 77.2, 75.7, 76.7, 74.9, 75.1, 79.8, 76.9, 75.4,
  75.3, 75.5, 75.3, 75.8, 75.7, 75.8, 75.7, 78.1)
```

What do you think?

Follow-on challenge question: there is one data point in the `B` set which, if you remove it from the data set the outcome of the test changes at the 5% level. Which one do you think it is and why? (Confirm your intuition.)

#7: Return on investment

Refer back to `roi$company` and `roi$industry` from §2.5, via roi.RData[16] on the book webpage.

```
load("roi.RData")
```

Does there seem to be any difference in ROI variance between companies that use the IT product in question and the industry at large?

#8: ANOVA SS identity

Show the sum-of-squares (SS) identity quoted in Eq. (6.9). *See the hint provided for §3.5 #5. Be careful, in general* $(a+b)^2 \neq a^2 + b^2$. *As always when "showing" an equality: start on the left-hand side, and manipulate the expression with one (equal) modification at a time, ending eventually with the expression on the right-hand side.*

#9: `monty.ANOVA`

Write a function called `monty.ANOVA` that automates ANOVA testing by MC and using closed-form, mathematical calculations, including optional visuals. You know the drill by now. Assume a `list` representation, like `ylist` above in §6.3. In addition to explicitly proving a `$pval` entry in your output `list` containing the p-value, provide `$tab` output that contains ANOVA table information like in Table 6.1.

Hint: many of the following questions are easier after this one is squared away.

[16] https://bobby.gramacy.com/hipp0/roi.RData

TABLE 6.3: Partial ANOVA table.

	DoF	SS	MS	fstat	pval
Regress	4				
Error		964			
Total	53	1123			

#10: Partial ANOVA table

The ANOVA summarized in Table 6.3 is only partially completed. Complete the table and use it to answer the following prompts.

- How many groups were in the study?
- How many total observations were in the study?
- What is the outcome of the hypothesis test?

This is a classic exam-type question because you don't need to do any serious coding, just arithmetic and a `pf` *calculation.*

#11: Puppy chow

A study was performed in partnership with a dog breeder to compare five different premium dog food brands. Weened, but otherwise "newborn" puppies were fed fixed servings of a single brand of dog food for six months. Their weights were recorded at the beginning and end of the study, and that difference (their weight gain in pounds) was recorded. Those values are provided below.

```
wtgain <- list(
  A=c(44, 49, 50, 50, 49),
  B=c(51, 52, 52, 52, 50, 49),
  C=c(55, 52, 53, 54, 53, 50, 51),
  D=c(51, 50, 52, 49, 51),
  E=c(51, 54, 56, 53, 55, 56))
```

Is there any difference between these dog foods as regards weight gain?

#12: Fisher's iris

Possibly the most famous ANOVA-style data set is Fisher's iris data, available in R as a built-in `data.frame` called `iris`. These data were collected by Anderson (1935), and Fisher's analysis was published a year later in a – now highly controversial outlet – on eugenics[17] (Fisher, 1936). A similar data set was included in the S language[18], a precursor to R (Becker et al., 1988). A quick peak at some of the columns are provided below.

```
summary(iris[,2:5])
```

```
##   Sepal.Width    Petal.Length   Petal.Width          Species
##  Min.   :2.00   Min.   :1.00   Min.   :0.1   setosa    :50
```

[17] https://en.wikipedia.org/wiki/Eugenics
[18] https://en.wikipedia.org/wiki/S_(programming_language)

```
##  1st Qu.:2.80   1st Qu.:1.60   1st Qu.:0.3   versicolor:50
##  Median :3.00   Median :4.35   Median :1.3   virginica :50
##  Mean   :3.06   Mean   :3.76   Mean   :1.2
##  3rd Qu.:3.30   3rd Qu.:5.10   3rd Qu.:1.8
##  Max.   :4.40   Max.   :6.90   Max.   :2.5
```

The grouping variable is iris `Species`: Setosa, Versicolor, and Virginica. These comprise just three of 300+ species of iris[19]. There are four measured responses available for analysis, and which might vary by species. Those are sepal length and width, and petal length and width. I cut sepal width (`iris[,1]`) out of the summary above to save space. Don't exclude it from your analysis.

What do you think? Do iris sepal and petal measurements vary by species? *Hint: one convenient way to turn two vectors, one with real-valued measurements and another* `factor` *grouping variable, into a* `list` *is with* `split` *in R.*

#13: Warm-blooded T-rex?

Data supporting this question come from the `Sleuth3` library (Ramsey et al., 2024), which you will need to install. See `ex0523`.

```
library(Sleuth3)    ## first do install.packages("Sleuth3")
summary(ex0523)
```

```
##      Oxygen          Bone
##  Min.   :10.6   Bone10 : 6
##  1st Qu.:11.3   Bone11 : 5
##  Median :11.5   Bone12 : 5
##  Mean   :11.6   Bone4  : 5
##  3rd Qu.:11.9   Bone5  : 5
##  Max.   :12.4   Bone7  : 5
##                 (Other):21
```

(*See* `split` *hint above in #12.*)

These quantities, originally from Barrick and Showers (1994) and repurposed for a popular textbook (Ramsey and Schafer, 2013), consist of several measurements of the oxygen isotopic composition of bone phosphate, which is related to body temperature at formation, in each of twelve bone specimens from a single Tyrannosaurus Rex skeleton. Differences in means at different bone sites would indicate non-constant temperatures throughout the body. This would be expected in warm-blooded animals, but not cold-blooded ones. Do these data comprise evidence that T-rex was warm blooded?

#14: Non-pooled ANOVA variance

This one is a challenge problem, but totally doable. Also, I can think of a lot of different ways to do it, so have fun!

How would you modify a MC-based ANOVA test in order to accommodate a non-pooled variance setup? That is, the model is as follows

$$Y_{ij} \overset{\text{iid}}{\sim} \mathcal{N}(\mu_j, \sigma_j^2) \qquad i = 1, \ldots, n_j \;\text{ and }\; j = 1, \ldots, m,$$

[19] https://en.wikipedia.org/wiki/Iris_(plant)

but everything else, including hypotheses (6.6), is just as in §6.3.

You might draw inspiration from two-sample t-tests (with unpooled variance) from §5.2. Aim for a hybrid between that setup and ordinary ANOVA.

Go back through the earlier ANOVA data sets from §6.3 and exercises #11–13 above. How does relaxing a pooled variance assumption change, or not, the outcome of the tests?

7

Correlation and linear regression

You might think I'm lumping together two topics that don't belong again, like in Chapter 6. But these two go hand in hand. On the Wikipedia page for correlation[1] it says, "Although in the broadest sense, correlation may indicate any type of association, in statistics it usually refers to the degree to which a pair of variables are linearly related." So look, both linear and correlation in the same sentence. I'll explain this in more detail later.

Correlation and linear regression both involve measurements on two sets of variables. Call them X and Y as in previous chapters. In this chapter we shall exclusively study samples from these populations in pairs.

$$(x_1, y_1), \ldots, (x_n, y_n)$$

We presume they're paired this way because each of x_i and y_i were observed under similar conditions. Earlier, like with the paired t-test of §5.3, we inspected samples from these populations and evaluated the extent to which they were the same (or different). Now we're going to take for granted that they're different but entertain that one may have useful "association" information about the other, and vice versa. So correlation and (linear) regression are about association, like in that Wikipedia quote.

Correlation involves a study of the extent (or not), or strength of that association. Regression is where you lean on that association in order to make a prediction, or to investigate the mechanism behind that association. If X and Y are associated, you might be able to leverage the nature of that relationship in order to determine Y given a value of $X = x$. In other words, you could lean on the conditional probability $\mathbb{P}(Y \mid X = x)$.

Quick digression to refresh you on conditional probability and independence. If $\mathbb{P}(Y \mid X = x)$ changes with x, then X and Y are not independent of one another. Recall the definition of conditional probability: $\mathbb{P}(Y \mid X) = \mathbb{P}(Y, X)/\mathbb{P}(X)$, abbreviating things a little. If these two events are independent, then $\mathbb{P}(Y, X) = \mathbb{P}(Y)\mathbb{P}(X)$. In that case $\mathbb{P}(Y|X) = \mathbb{P}(Y)$, i.e., Y doesn't change with X. So correlation and regression have something to do with independence, or lack thereof. These aren't exactly the same concept. Two random variables can be dependent (not independent) but not (linearly) correlated. Let me get back to that later. However, the opposite does work: if X and Y are correlated, they are definitely not independent. This means they have an association which can be quantified and that can be exploited for prediction.

Alright, back from the refresher. Let me introduce some data to use as a running example, beginning with an illustration of broad themes. Suppose a crop of fraternity pledges at ΣΣΣ (the ultimate stats frat!), were timed at a 100 yard dash both before (X) and after (Y) chugging a 40 (oz of beer). Figure 7.1 offers a visual of these data. Going forward, I shall refer to these values as the frat dash data.

[1] https://en.wikipedia.org/wiki/Correlation

```
x <- c(12.3, 16.1, 18.6, 13.5, 16.8, 14.1, 18.1, 17.8, 15.2, 13.1)
y <- c(13.6, 17.1, 19.3, 13.9, 16.3, 16.9, 17.9, 15.9, 16.1, 16.1)
plot(x, y, xlab="time before beer", ylab="time after beer")
```

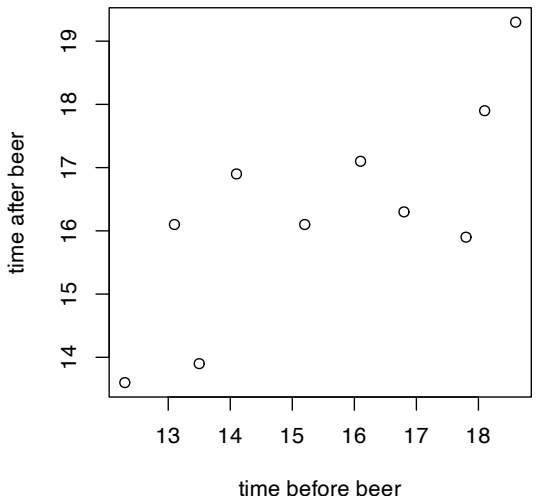

FIGURE 7.1: Frat dash data: times (seconds) to run 100 yards before (x) and after (y) a beer.

In any correlation or linear regression analysis, visualizing (x_i, y_i) pairs as a scatterplot is definitely the way to go before jumping in with calculations. Judging by these times, pledges might not have been taking the exercise too seriously. (They're a little on the slow side.) It may also have been that the time keepers were a few 40s in too. (I didn't interface much with Greek life in college, so what would I know?)

As to whether there's an association, to my eye maybe so, but perhaps not a strong one. It looks like pledges that were faster before were also faster after, and likewise with slower times. The relationship isn't perfect, but there's definitely an upward trend from left to right. That association is also plausibly linear, by which I mean that the trend seems to go up and to the right along a (noisy) line. It doesn't turn around and go back down or anything like that.

What if a time of $x = 14.5$ seconds was recorded for one recruit – call him Brandon (my middle name) – before a beer, but somehow a second time was not recorded later. Maybe Brandon decided to sleep it off. Could we use this association to make a prediction $Y(x)$? Could we perhaps also couple that with a range of times summarizing uncertainty? If I had to use my eyeballs, I'd go with $\hat{y}(x) = 15.5$ with a range of $[14, 17]$. Could those guestimates be made precise, with probabilities attached? How can I determine if there's an association at all? That's where statistics comes in.

7.1 Correlation reminders

Covariance[2] is the expectation of the product of differences between two random variables and their respective means.

$$\mathbb{C}\text{ov}(Y, X) = \mathbb{E}((Y - \mathbb{E}\{Y\})(X - \mathbb{E}\{X\})) \tag{7.1}$$

Covariance is a property of the joint distribution[3] of Y and X, capturing both scale (note $\mathbb{V}\text{ar}\{Y\} = \mathbb{C}\text{ov}(Y, Y)$) and association. In this form (7.1), covariance is a theoretical construct – a definition for a quantity that can only be calculated if you can perform those expectations, which are either integrals or sums. Notice that, unlike variance, $\mathbb{C}\text{ov}(Y, X)$ can be negative, indicating a negative association. If Y tends to be above its mean while X tends to be blow, or vice versa, then covariance will be negative (as a product of negative and positive values). However, if both tend to be below their respective means, then a product of two negatives is positive.

Quick digression to address my choice of order in notation. Often you'll see X listed before Y, as $\mathbb{C}\text{ov}(X, Y)$, because that puts the letters in order alphabetically. But this is arbitrary, since $\mathbb{C}\text{ov}(X, Y) = \mathbb{C}\text{ov}(Y, X)$. I often, but not always, do it the other way around. You may have noticed me doing that back as early as Chapter 5, when introducing two-sample tests. I do this to privilege Y as the main object of interest, even though often it isn't. In many situations both Y and X may be of interest. However, in the absence of a second sample, I strongly prefer Y (for the only sample), not X. This preference isn't universal in statistics, but it is common.

When analyzing association, or in a two-sample test of means, X and Y are on equal footing. However, in regression we usually write Y as the "response" – the main random quantity of interest – that we wish to predict, given X. So X is on the right-hand side of the conditioning bar: $\mathbb{P}(Y \mid X)$. Therefore, I try to keep it on the right as much as possible. One exception, however, is when listing data as in $(x_1, y_1), \ldots, (x_n, y_n)$ and plotting them as `plot(x, y)`. If you flip them, you get a different plot! So don't do that. However, when you're looking at the plot, from left to right, you see the y-axis before the x-axis, so it works that way in a sense. I guess just be nimble, and be aware of when order matters and when it doesn't. In most cases it doesn't, and you get identical (or at least similarly interpretable) results either way around.

Alright, back to covariance. Given data, we may calculate their sample covariance, an estimate of Eq. (7.1), as follows.

$$s_{yx} = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x}) \tag{7.2}$$

Notice that $s^2 \equiv s_y^2 = s_{yy}$, so sample covariance nests sample variance the same way that covariance nests variance. And like covariance, sample covariance provides an estimate of both scale and association.

Figure 7.2 offers an illustration on the frat dash data. Averages are shown as horizontal (for $\bar{y}$) and vertical (for $\bar{x}$) dashed lines. Covariance tends to be larger when the scale is large,

[2] https://en.wikipedia.org/wiki/Covariance
[3] https://en.wikipedia.org/wiki/Joint_probability_distribution

meaning big ranges on the axes, and when y_i tends to be above $\bar{y}$ when x_i is above $\bar{x}$. We say that (x_i, y_i) is *concordant* with $(\bar{x}, \bar{y})$ when the points land in the area shaded in green. Any points in the white quadrant are *discordant*. For now, those definitions are for your entertainment. We'll use them more seriously later in Chapter 12.

```
plot(x, y, xlab="time before beer", ylab="time after beer")
ybar <- mean(y)
abline(h=ybar, lty=2, lwd=2)
xbar <- mean(x)
abline(v=xbar, lty=2, lwd=2)
inf <- 1000
gx <- c(-inf, xbar, xbar, inf, inf, xbar, xbar, -inf)
gy <- c(ybar, ybar, inf, inf, ybar, ybar, -inf, -inf)
polygon(gx, gy, col=3, density=10, angle=45)
legend("topleft", c("xbar or ybar", "concordant"), col=c(1, 3),
  lty=2:1, lwd=2:1, bty="n")
```

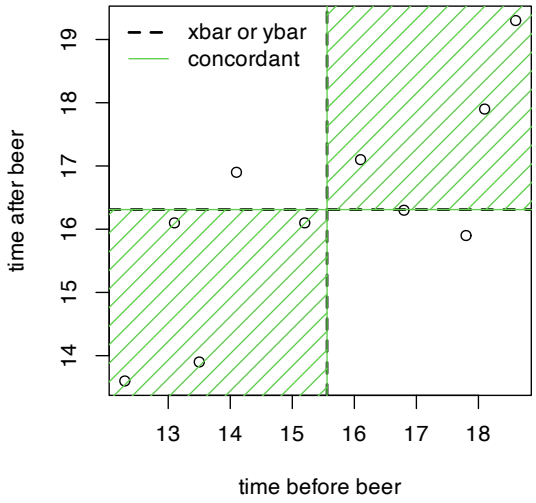

FIGURE 7.2: Illustrating covariance on frat dash data.

In R, covariance can be calculated with `cov`.

```
cov(y, x)
```

```
## [1] 2.906
```

Observe that this particular sample covariance is positive. That happened, intuitively speaking, because there are many more pairs in the data that are concordant (in the green region) with their averages than discordant (in the white), as shown in Figure 7.2. Estimated covariance would be negative if there were more pairs discordant with their averages.

It can be hard to separate out scale and association and make sense of a number like this. To focus just on association, divide by the (square root) of the scales estimated for Y and X to get correlation[4].

[4] https://en.wikipedia.org/wiki/Correlation

$$\rho_{yx} = \mathbb{C}\text{or}(Y, X) = \frac{\mathbb{C}\text{ov}(Y, X)}{\sqrt{\mathbb{V}\text{ar}\{Y\}\mathbb{V}\text{ar}\{X\}}} \tag{7.3}$$

This is known as Pearson's product-moment coefficient[5], or more simply "Pearson's ρ". It is named for Karl Pearson[6], yet another notable British statistician who, like Fisher, has a bunch of other stuff named after him. Apparently, Pearson and Fisher had a bit of a rivalry[7] going, despite sharing similar (now widely considered) racist beliefs.

Note that $\rho = \rho_{yx} = \rho_{xy}$, by virtue of similar symmetry in covariance. Since covariance can be positive or negative, and we are dividing by only positive values, correlation can also be positive and negative. Normalization gives that ρ is at most one, in absolute value: $-1 \leq \rho \leq 1$. Closer to one, in absolute value, means stronger association. Random variables that have $\rho = 0$ are uncorrelated, and thus have no (linear) association.

Estimating ρ is simply a matter of following the formula (7.3) in sample analogs.

$$r \equiv r_{yx} = \frac{s_{yx}}{s_y s_x} \tag{7.4}$$

Like ρ, we have that $-1 \leq r \leq 1$, measuring the strength and direction of association. In R:

```
r <- cor(y, x)
r
```

```
## [1] 0.7608
```

So the estimate calculated on the frat dash data indicates a positive association. This makes sense. Although pledges seem to slow down after a beer, their time before is indicative of their time after. If they tended to be among the slowest before a beer, then they were among the slowest afterwards. The same goes for the middle and fastest runners.

Some discussion

Can you think of any examples where correlation would be negative? We'll see some later in the homework exercises of §7.8. But in the meantime, here are some complicated ones to get your wheels spinning. I'm not going to provide any formal references. You'll find many by Googling.

- People who are wealthier/more affluent tend to live longer and have children who are taller than they are. There's two positive associations. I bet if you collect data on the height difference (or difference in lifespan) between fathers and sons, or mothers and daughters (that's your Y), and parent's income (X) or some other measure of their socioeconomic status (SES)[8], you'll find a positive association. This is certainly true in my family, many-fold over generations.
- But … controlling for income/SES (e.g., holding it fixed), I bet you'll find that shorter people live longer than taller people. Women tend to be shorter than men and also live longer. The association is negative. My uncle is shorter than my dad by almost a foot, and lived 15 years longer. RIP Dad and Uncle Bill. That's anecdotal, and I'm sure there

[5] https://en.wikipedia.org/wiki/Correlation#Pearson's_product-moment_coefficient
[6] https://en.wikipedia.org/wiki/Karl_Pearson
[7] https://njoselson.github.io/Fisher-Pearson/
[8] https://en.wikipedia.org/wiki/Socioeconomic_status

are many exceptions. On average, I suspect lifespan (Y) is negatively associated with height (X), all else being equal.

- Another weird one is US voting preferences in the 20th century. (Things have changed on this recently, though not as much as you might think.) Income was/is an excellent predictor of propensity to vote Republican on an individual level. More money (X) better chance of voting Red (Y).
- But … it's different on the state level. I've got a homework question for you on this in §12.5. Wealthier states (think CA, NY) tend to vote Blue (Democrat, not Republican), whereas poorer ones (think MS, AL) tend to vote Red. The association is negative. You can even see this over time. Virginia, where I live, used to be a poorer state. But as Northern VA amassed wealth in the late-20th century, attracting affluence with proximity to DC and government work, VA flipped from Red to Blue. These days it's considered a Purple state.

Both pairs of examples owe their quirkiness to something called Simpson's paradox[9], which is a fancy way of saying "it depends on how you look at it". Things sometimes look different zoomed out, with wider perspective, than if you look really closely. So just be careful. Resolution matters. It can go either way.

Alight, one last thing before we actually start doing statistics. Why is it called *linear* correlation? For that, let's look at some examples. Figure 7.3 depicts six sets of (x, y) pairs. Never mind how they were generated. Putting all that code here would be a distraction.

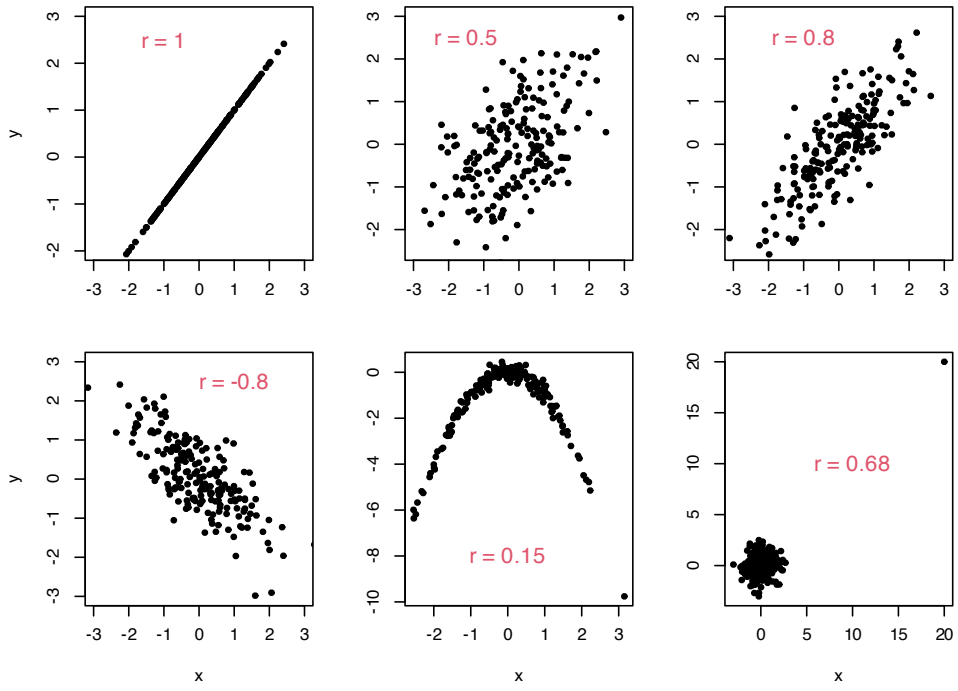

FIGURE 7.3: Some example correlation calculations.

Consider the panels in turn, clockwise from the top-left. If pairs perfectly follow a line, then $r = \pm 1$. It doesn't matter what the slope is as long as it's nonzero. A non-noisy association

[9] https://en.wikipedia.org/wiki/Simpson's_paradox

that follows a straight line is perfectly correlated. So that's one reason to call it "linear" correlation. If data are noisy, like in the other five panels, correlation is less-than-perfect: $-1 < r < 1$. The next pair of panels depict a weaker and somewhat stronger relationship, respectively. The bottom-left panel is the reverse of the top-right, showing a similarly strong negative association.

The final two panels in Figure 7.3 are the most interesting. The bottom-middle one shows that you can fool correlation if the association is nonlinear. Visually, the relationship between Y and X is stronger than in any other panel except the first one, yet here r is smallest of all. Correlation is not a good measure of association – at least not in its most basic, linear form – when that association is nonlinear. The final panel represents a caution of a different sort, which we shall address later in Chapter 12. Whenever there are extreme outliers[10], they can exert undue influence on a correlation. You can't trust estimates in this case.

The reason for this last phenomenon is that the mathematical construct, ρ in Eq. (7.3), is making an implicit Gaussian assumption, which we'll get to momentarily. You can calculate ρ for any joint distribution, and estimate r for any pairwise sample, whether Gaussian or not. Deciding what an estimate means requires additional modeling scaffolding. You can't just look at an estimate, r, and judge whether or not it's significant (different from zero) in a vacuum. How r compares to a hypothesis about ρ depends on modeling assumptions which (usually) involve distributions. If those assumptions are egregiously at odds with reality, as they are on the bottom-right Figure 7.3, then all bets are off.

7.2 Pearson's ρ

Suppose we make a marginally Gaussian assumption for X and Y in addition to $\mathbb{C}\text{or}(Y, X) = \rho$. Copying from Eq. (5.5), that means:

$$Y_1, \ldots, Y_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu_y, \sigma_y^2) \qquad \text{and} \qquad X_1, \ldots, X_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu_x, \sigma_x^2).$$

If you want to be fancy, you could say that X and Y have a bivariate Gaussian (or normal) distribution[11].

$$\text{Model:} \quad \begin{bmatrix} Y_i \\ X_i \end{bmatrix} \overset{\text{iid}}{\sim} \mathcal{N}_2 \left(\begin{bmatrix} \mu_y \\ \mu_x \end{bmatrix}, \begin{bmatrix} \sigma_y^2 & \rho\sigma_y\sigma_x \\ \rho\sigma_x\sigma_y & \sigma_x^2 \end{bmatrix} \right), \quad \text{for} \quad i = 1, \ldots, n \tag{7.5}$$

The bivariate Gaussian is a special case of the multivariate normal (MVN)[12], often notated as $\mathcal{N}_p(\mu, \Sigma)$. Above, $p = 2$, $\mu = [\mu_y, \mu_x]^\top$ and Σ is the matrix in the second argument to $\mathcal{N}_2(\cdot, \cdot)$. It is worth clarifying, briefly, that iid here doesn't mean that Y_i is independent of X_i. They are, in fact, quite dependent through ρ. That's the whole point of this chapter! Here, as previously, iid is interpreted over $i = 1, \ldots, n$, meaning that (Y_i, X_i) is independent of (Y_j, X_j) for $i \neq j$.

Don't let matrices and vectors frighten you. When $\rho = 0$, e.g., when testing under the null hypothesis of no linear correlation (the most common case), we won't need them at all.

[10] https://en.wikipedia.org/wiki/Outlier
[11] https://mathworld.wolfram.com/BivariateNormalDistribution.html
[12] https://en.wikipedia.org/wiki/Multivariate_normal_distribution

It's only when generating under a nonzero null correlation that the bivariate Gaussian is important. We'll lean on a library-based pseudo-random number generator in that case.

In my first draft of this chapter, I totally forgot to discuss options for estimating ρ, along with all other parameters in the model (7.5): $\mu_y, \mu_x, \sigma_y^2, \sigma_x^2$. I guess it seemed automatic that $\hat{\rho} = r$ would work, along with usual suspects from earlier chapters: $\bar{y}, \bar{x}, s_y^2, s_x^2$. Those are definitely sensible choices, but are they good? What are the best settings of those parameters? For example, what parameter settings maximize the likelihood (MLE)?

Perhaps I'm being lazy, but I've decided to leave that to you as a homework exercise. To figure that out, you'll have to work with a bivarate Gaussian density, which you can find in the links above, and take logs and derivatives. Some other interesting considerations include the following. Is the estimate you find for ρ unbiased? What is its variance? (That might require asymptotic analysis.)

Going forward, my narrative will focus on inference for ρ via observed r. Remember that you can estimate a parameter however you'd like, and your test statistic need not be a parameter estimate at all. It's totally up to you. Some things are easier when you stick to MLEs, especially when the math benefits from asymptotic analysis. You're a lot freer to choose otherwise with Monte Carlo (MC).

Before jumping into that, here is a formal characterization of competing hypotheses. For some ρ_0, which is usually zero, we wish to test as follows.

$$\begin{aligned} \mathcal{H}_0 &: \rho = \rho_0 \qquad \text{null hypothesis ("what if")} \\ \mathcal{H}_1 &: \rho \neq \rho_0 \qquad \text{alternative hypothesis} \end{aligned} \tag{7.6}$$

As previously, I'll be a little cavalier with my notation, often write ρ_0 as ρ with the understanding that this is the value specified by $\mathcal{H}_0$. Or, I'll just substitute in the particular numerical value, usually $\rho = 0$ but not always. The test statistic is r. How to run the test? As usual, that depends on whether you like simulation or math, and I'll show you both in turn.

Pearson's ρ-test via MC

Here I shall utilize a, by now, familiar sampling framework for Gaussian populations. An estimated variance has a χ^2 distribution, e.g.,

$$\sigma_y^2 \sim (n-1)s_y^2/\chi_{n-1}^2 \quad \text{and} \quad \sigma_x^2 \sim (n-1)s_x^2/\chi_{n-1}^2.$$

Those random variables, along with the value of ρ specified by $\mathcal{H}_0$, pin down a bivariate Gaussian (7.5) covariance structure. Since parameter ρ is a (co-) variance quantity, and we know from Chapter 5 that estimated variances of Gaussians are independent of means, μ_y and μ_x specifications won't affect testing. We can use whatever values we want. Below I use data averages, but I encourage you to try other settings.

```
n <- length(y)        ## same as length(x)
s2y <- var(y)
s2x <- var(x)
```

Looking back at the frat dash example, the main "what if" (7.6) of interest centers around $\rho = 0$. We want to know if the times are correlated before and after an intervention (beer).

This $\mathcal{H}_0$ simplifies simulation. When $\rho = 0$ we don't really have a bivariate Gaussian, but rather two completely independent univariate ones. Those off-diagonal terms in Σ are zero. So we're ready to go.

```
N <- 100000
Rs <- rep(NA, N)
for(i in 1:N) {
  sigma2ys <- (n - 1)*s2y/rchisq(1, n - 1)
  Ys <- rnorm(n, ybar, sqrt(sigma2ys))
  sigma2xs <- (n - 1)*s2x/rchisq(1, n - 1)
  Xs <- rnorm(n, xbar, sqrt(sigma2xs))
  Rs[i] <- cor(Xs, Ys)
}
```

Figure 7.4 shows these sampled correlations in our usual histogram format, summarizing an empirical sampling distribution for r, with the data-estimated r-value and its reflection overlayed.

```
hist(Rs, xlim=c(-1.15, 1.15), main="")
abline(v=c(r, -r), lty=1:2, col=2, lwd=2)
legend("topright", c("r", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
```

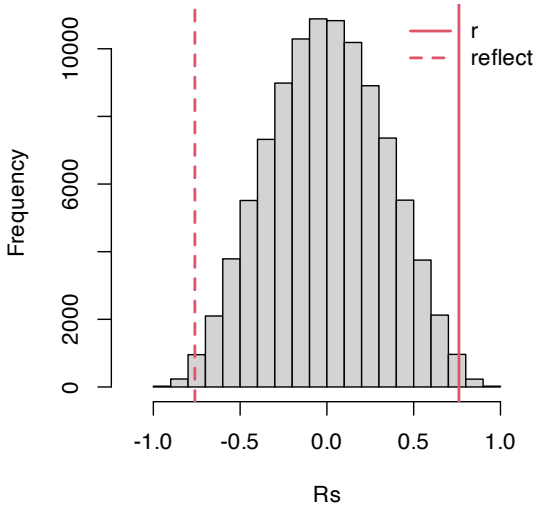

FIGURE 7.4: Histogram of sampled R for testing $\rho = 0$ in the frat dash experiment, with estimated r and it's reflection overlayed.

This looks like a reject of the null, but a p-value calculation can help add precision.

```
pval.mc <- 2*mean(Rs > r)
pval.mc
```

```
## [1] 0.01042
```

Indeed, this is less than our default 5% threshold. It's a close call, but it turns out that there *is* indeed an association between dashing times before and after chugging a beer.

How about a confidence interval (CI) for ρ? Just go back through the simulation, but instead of using $\rho = 0$, use the estimated value of $\hat{\rho} = r$. That's easier said than done in this case, however. Since $r \neq 0$ we must draw from a bivariate Gaussian. After sampling marginal variances σ_y^2 and σ_x^2, the code below builds a 2×2 matrix Σ using those values and $\hat{\rho}$ following Eq. (7.5). Then, I draw from a $p = 2$-variate MVN using a function from the `mtvnorm` library (Genz and Bretz, 2009).

```
Rs <- rep(NA, N)
library(mvtnorm)                                    ## for rmvnorm
for(i in 1:N) {
  sig2ys <- (n - 1)*s2y/rchisq(1, n - 1)
  sig2xs <- (n - 1)*s2x/rchisq(1, n - 1)
  syxs <- sqrt(sig2ys*sig2xs)                       ## product of sds
  Sigs <- rbind(c(sig2ys, r*syxs), c(r*syxs, sig2xs))  ## Build 2x2 Sigma
  XYs <- rmvnorm(n, c(ybar, xbar), Sigs)            ## library MVN RNG
  Rs[i] <- cor(XYs)[1, 2]                           ## diag always 1
}
CI.mc <- quantile(Rs, c(0.025, 0.975))
CI.mc
```

```
##   2.5%  97.5%
## 0.3105 0.9446
```

The code above would be useful, with a minor tweak, in order to test for $\rho = \rho_0$ for $\rho_0 \neq 0$. Remember that for your homework.

One thing you'll notice is that this code is lots slower than the chunk earlier, for $\rho = 0$. Drawing from an MVN involves a Cholesky decomposition[13] of Σ, which is a lot of work computationally. There are work-arounds in the bivariate case, but the code is cleaner the way it is above.

Pearson's ρ-test via math/asymptotics

Ok, here is what in my opinion will be the first big let-down in the book. Maybe it's not the first for you. Correlation is among the most fundamental ideas in statistics. Yet the test for significance and associated CI for ρ is an inscrutable mess. Why? Because the math is hard. Not only that, there are two different procedures depending on the situation. That means more to remember, a burden of choice, and potentially consternation if you try both and they don't agree.

This is a shame for several reasons. One is that I'm going to quote results with intuition rather than through derivation. Another is that a related idea, of slopes in a linear model coming next, is so simple by comparison and gives you pretty much the same information. (It's not exactly the same because lines involve both slopes and intercepts, two unknown parameters compared to correlation's lone ρ. But I haven't, in my whole career, come across a problem where the outcome of a slope-based test differed from a ρ one.) Finally, it's particularly a shame when the MC is so easy – no fancy math. I'm hoping that's a running theme in this book that resonates with you.

[13] https://en.wikipedia.org/wiki/Cholesky_decomposition

Both math-based testing approaches for ρ involve a transformation of r, similar to how we transform r^2 into f for an F-test in §6.3. In fact, r here and r^2 there have another thing in common, but that'll have to wait until §7.6, and we won't fully appreciate the connection until we get to multiple linear regression (MLR)[14] in Chapter 13. I don't want to spoil it here.

The first testing approach for ρ is an asymptotic approximation, based on a CLT-like argument so that you can use a Gaussian. The second one is exact, but can only be used for testing $\rho = 0$, and consequently does not yield a CI. Often, software libraries mix the two in some way, which is not always made obvious in the output.

Ok, here goes. Let R be the random version of $r \equiv r_{yx}$. It can be shown that an inverse hyperbolic tangent[15] transformation of R, i.e., $\operatorname{artanh}(R)$, is Gaussian asymptotically.

$$W = \frac{1}{2}\log\frac{1+R}{1-R} \sim \mathcal{N}(\mu_w, \sigma_w^2) \quad \text{as } n \to \infty, \tag{7.7}$$
$$\text{where} \quad \mu_w = \frac{1}{2}\log\frac{1+\rho}{1-\rho} \quad \text{and} \quad \sigma_w^2 = \frac{1}{n-3}$$

This is also known as the Fisher z-transformation[16]. Now we can follow the usual z-test approach with $z = (w - \mu_w)/\sigma_w$ and calculate a p-value as $\psi = 2\Phi(-|z|)$ for a two-sided test. At least that part is pretty easy. This test can be used for any ρ, but in our frat dash example we're interested in $\rho = 0$.

```
w <- 0.5*log((1 + r)/(1 - r))           ## also atanh(r)
rho <- 0
muw <- 0.5*log((1 + rho)/(1 - rho))     ## also atanh(rho)
s2w <- 1/(n - 3)
z <- (w - muw)/sqrt(s2w)
pval.clt <- 2*pnorm(-abs(z))
c(mc=pval.mc, clt=pval.clt)
```

```
##       mc      clt
## 0.010420 0.008279
```

This agrees with the MC version when rounded to the second decimal place. Of course, $n = 10 \ll \infty$, so a Gaussian approximation may not be accurate.

Similarly, a CI may be built by inverting $z \pm q_{\alpha/2}$ via Eq. (3.19).

$$\begin{aligned} & w \pm q_{\alpha/2}\sigma_w && \text{is a CI for } \mu_w \\ \text{so} \quad & \frac{\exp\left\{2(w \pm q_{\alpha/2}\sigma_w)\right\} - 1}{\exp\left\{2(w \pm q_{\alpha/2}\sigma_w)\right\} + 1} && \text{is a CI for } \rho \end{aligned}$$

Check it.

[14] https://en.wikiversity.org/wiki/Multiple_linear_regression
[15] https://en.wikipedia.org/wiki/Inverse_hyperbolic_functions
[16] https://en.wikipedia.org/wiki/Fisher_transformation

```
q <- qnorm(0.975)
mu.ci <- w + c(-1, 1)*q*sqrt(s2w)
emu.ci <- exp(2*mu.ci)
CI.clt <- (emu.ci - 1)/(emu.ci + 1)
rbind(mc=CI.mc, clt=CI.clt)
```

```
##       2.5%  97.5%
## mc  0.3105 0.9446
## clt 0.2517 0.9401
```

This CLT-based CI is similar to the MC one, but shifted a little bit lower toward zero. It's important to remember which is the gold standard here. Although both are approximations, you can always make N bigger in the MC. I encourage you to try that, but spoiler alert: it doesn't change much in this instance. With n and the CLT, you've got what you've got and it is what it is, and n is small in this instance.

When $\rho = 0$, it may be shown that

$$U = \frac{R\sqrt{n-2}}{\sqrt{1-R^2}} \sim t_{n-2}, \quad \text{exactly.}$$

That is, this is not an approximation. You can actually work out what the distribution is for R in generality (i.e., for any ρ) in closed form, but it's not pretty to look at and it isn't a standard distribution that has a density function (like `pt` or `pnorm`) to go with it. That makes it more of an intellectual curiosity as far as testing goes. The special case of $\rho = 0$ reduces to a standard density, making it more useful in practice.

```
u = r*sqrt(n - 2)/sqrt(1 - r^2)
pval.t <- 2*pt(-abs(u), df=n - 2)
lib <- cor.test(y, x)                                    ## see below
c(mc=pval.mc, clt=pval.clt, math=pval.t, lib=lib$p.value)
```

```
##       mc      clt     math      lib
## 0.010420 0.008279 0.010615 0.010615
```

This one is much closer to our MC calculation, identical up to two digits. Limiting to $\rho = 0$ means no CI, so libraries tend to mix CLT and exact versions. In fact, the `cor.test` function built into R doesn't allow you to test anything but $\rho = 0$, yet it still mixes CLT calculations for the CI with the other, exact variation for testing. Above, observe that the library output matches `pval.t` exactly, which is close to the MC calculation. Below, notice that the CI matches the CLT alternative.

```
rbind(mc=CI.mc, clt=CI.clt, lib=lib$conf.int)
```

```
##       2.5%  97.5%
## mc  0.3105 0.9446
## clt 0.2517 0.9401
## lib 0.2517 0.9401
```

But again, I trust the MC answer more because (a) I understand what's going on, and (b) I can always refine it with larger N.

7.3 Lines refresher

You probably remember from middle school that there are three ways to describe a line[17]: point–point, point–slope, and slope–intercept. There are others, but those are the main ones; the last one is actually a special case of the second one. You can go back and forth between them because they each *uniquely* describe a line. That means there's only one line that goes through two distinct points, and only one with a particular slope and intercept. The opposite isn't true: there are many ways to describe the same line with pairs of points, since lines are made up (in a certain sense) of infinitely many points.

But I digress. Here I'll primarily work with the slope–intercept form. Later, in Chapter 12 we shall mostly use point–point. Each can be handy in its own way. You're probably familiar with the following mathematical depiction of the slope–intercept form: $y = mx + y_0$. The slope is m, and the intercept is y_0. The intercept tells you where the line crosses the y-axis. Put another way, it's giving you that point $(0, y_0)$ on the line, when $x = 0$. The slope is telling you how far to go up (or down for negative m) in y as x increases by one. You can think of that as giving you the point $(1, y_0 + m)$ on the graph, but actually it's giving you $(x, y_0 + mx)$ for any x.

Conversely, if someone gives you two points (x_1, y_1) and (x_2, y_2), then you can calculate slope by following the rise-over-run formula

$$m = \frac{y_2 - y_1}{x_2 - x_1}.$$

Once you have m, you can solve for y_0 by plugging in either point and solving:

$$y_0 = y_1 - mx_1.$$

I sure hope that was enough of a review so we can move on to some stats. Actually, before I jump in, I want to change the notation a little. In linear regression the intercept and slope are parameters of interest, and statisticians like Greek letters for unknowns. So the linear equation is usually notated as:

$$y = \beta_0 + \beta_1 x,$$

so that the intercept is β_0 and the slope is β_1. I know these Greek letters and subscripts seem cumbersome, but you'll have to take my word for it that it's well thought out and makes sense. You'll find some references that use α and β for intercept and slope, avoiding subscripts. Our investment in subscripts will pay off when we get to MLR in §13.3.

Our interest is in choosing values for these parameters to compactly describe (x, y) relationships in data. For example, what settings of (β_0, β_1) would give a good fit to the frat dash data in Figure 7.1? What "good fit" means is something we'll have to unpack a little later. First, I want to suggest one fit that connects to correlation.

Suppose we were to assume that the conditional distribution $\mathbb{P}(Y \mid X)$ was such that it follows a linear relationship. Notice that I'm leaving much to the imagination here. I'm not saying what the distribution for Y is – or for X – but just how the two are related to one

[17] https://en.wikipedia.org/wiki/Linear_equation#Equation_of_a_line

another in their conditional distribution. If you wanna get fancy, I'm telling you something about the copula[18] of the joint distribution, but not its margins. Don't worry, I'm not going to pull us into a copula rabbit hole. I just wanted you to know a nerdy stats word.

That was a long-winded way of saying: suppose $Y = \beta_0 + \beta_1 X$. Now, what happens if we try to calculate covariance between Y and X under this relationship? The development below uses a property for covariance of linear combinations[19], similar to ones I used in Chapters 2 and 4 for expectations.

$$\mathbb{C}\text{ov}(Y, X) = \mathbb{C}\text{ov}(\beta_0 + \beta_1 X, X) = \mathbb{C}\text{ov}(\beta_1 X, X) = b_1 \mathbb{V}\text{ar}\{X\}.$$

$$\text{Then solve:} \quad \beta_1 = \frac{\mathbb{C}\text{ov}(Y, X)}{\mathbb{V}\text{ar}\{X\}}.$$

Next replace those theoretical covariance and variance quantities with their sample analog which can be calculated from data.

$$b_1 = \frac{s_{yx}}{s_x^2} \tag{7.8}$$

Notice how I switched from Greek to Roman lettering, because we can actually calculate b_1 from data. I want you to think of it as an estimate. I'm not saying it's an MLE or anything like that, which is why I'm not using $\hat{\beta}_1$.

Now suppose we do the same thing for correlation, which is easy because that just means divide both sides by the standard deviation of Y and X.

$$\mathbb{C}\text{orr}(Y, X) = \beta_1 \frac{\sqrt{\mathbb{V}\text{ar}\{X\}}}{\sqrt{\mathbb{V}\text{ar}\{Y\}}} \quad \text{or in sample terms} \quad r_{yx} = b_1 \frac{s_x}{s_y}$$

Solving for β_1 or b_1 gives

$$b_1 = r_{yx} \frac{s_y}{s_x} \equiv \text{corr} \times \frac{\text{rise}}{\text{run}}. \tag{7.9}$$

This could be an "Ah-ha!" moment. It was for me when I first saw it. Apparently, slope from example data pairs $(x_1, y_1), \ldots, (x_n, y_n)$ is correlation times rise over run, where rise and run are measured in standard deviations. Using the frat dash data in R:

```
b1 <- r*sd(y)/sd(x)   ## or sqrt(s2y)/sqrt(s2x)
b1
```

```
## [1] 0.572
```

This number is telling us that for every one second slower a frat boy is before a beer, he is 0.57 seconds *even* slower afterward. That's complicated to say, because not having an intercept makes it so that I have to express everything on relative terms. I can't be concrete and make a prediction.

How about that intercept? We've got slope, so all we need now is a point. How about insisting that the line "go through the middle" of the data? In other words, what if I insist that the point $(\bar{x}, \bar{y})$ is on the line. Then solve to get …

[18] https://en.wikipedia.org/wiki/Copula_(statistics)
[19] https://en.wikipedia.org/wiki/Covariance#Properties

$$b_0 = \bar{y} - \bar{x}b_1. \tag{7.10}$$

I call this the "correlation and means" method for dialing in a line from a cloud of points. In R:

```
b0 <- ybar - b1*xbar
b0
```

```
## [1] 7.41
```

This number isn't super meaningful in and of itself, but it allows the line to be plotted. See Figure 7.5. I jammed a few other things into the figure, because it had been a while, so it'll take a bit to narrate.

```
plot(x, y, xlab="time before beer", ylab="time after beer")
abline(b0, b1, col=2, lwd=2)            ## adding the estimated line
points(xbar, ybar, col=2, pch=19)       ## indicating intercept
xs <- c(13, 14, 14)                     ## indicating slope (next 4 lines)
ys <- c(b0 + xs[1]*b1, b0 + xs[1]*b1, b0 + xs[3]*b1)
lines(xs, ys, col=2, lty=2)
text(xs[2], (ys[2] + ys[3])/2, "b1", pos=4, col=2)
legend("topleft", c("cor and means line", "slope"),
  lty=1:2, col=c(2, 2), lwd=2:1, bty="n")
legend("bottomright", "(xbar, ybar)", pch=18, col=2, bty="n")
```

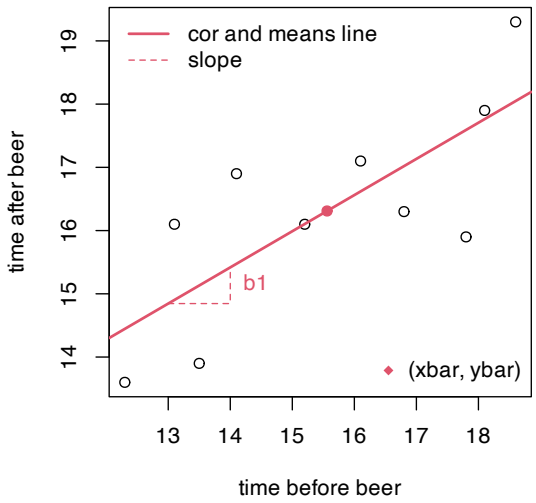

FIGURE 7.5: Line fit to frat dash data via "correlation and means".

Check out the new way I'm using the `abline` function. Actually, `a` is for intercept and `b` for slope in `abline`. Pretty good line, right? I tried to indicate estimated slope (b_1) and intercept (b_0), but somewhat indirectly. Slope is the change in y for one unit of change in x, so that's shown by the vertical travel (of about 0.57) in the dashed red line near $x = [13, 14]$. There is nothing special about those coordinates. I could have indicated slope anywhere along the line. I chose a spot a little out-of-the-way to reduce clutter. The intercept isn't on the plot,

because it lies outside the plotting window near $(0, 7.41)$. We pinned that down through $(\bar{x}, \bar{y}) = (15.56, 16.31)$, and those coordinates are shown on the plot as a red dot.

Now we can make a prediction that is quantitatively precise. Recall I said a time of $x = 14.5$ seconds was recorded for one recruit, Brandon. Here's what I would predict for Brandon, using the line, after he chugged a beer.

```
xp <- 14.5          ## p for predict
yp <- b0 + b1*xp
yp
```

```
## [1] 15.7
```

This is not too far off my guess from earlier. Using the line has some benefits. It's automatic, precise, and objective. But it has some downsides too. That line looked good, but how do we know it *is* good? How do we know there isn't a better line that'll give more accurate predictions? Acknowledging that our estimate of the line is from a small sample of data, how can we assess the uncertainty of that estimation, and how does that uncertainty filter through to predictions?

7.4 Ordinary least squares

First, I'll talk about getting the best line. Then, I'll bring in some probabilities so that we can have a full statistical model and understand its sampling distribution. There's absolutely nothing statistical or probabilistic about what's in this section. It's pure optimization[20]. In particular, there are no Greek letters. They'll be back.

Generically, regression means tending to revert back to something, like how observations in a sample tend to regress to the mean[21]. I said I wasn't going to do statistics here, but bear with me. On a cold summer day, you could guess that tomorrow's temperature will be warmer because you believe that temperature will regress to whatever the historical average has been on that particular day over the years. Linear regression[22] extends that idea, which is inherently pointwise in nature, to lines where the notion of average can vary, linearly, as a function of x.

One way to do linear regression is to posit a line, say $y = b_0 + b_1 x$ and ask, what values of b_0 and b_1 make the line as close as possible to some data $(x_1, y_1), \ldots, (x_n, y_n)$. Then that line would represent the average (y as a function of x, or equally for x as a function of y) that those values tend to revert to. Linear regression, therefore, is all about summarizing $2n$ quantities, or n data pairs, with two quantities, b_0 and b_1. This is information extraction, the essence of learning: simplifying many observations into a compact rule that can be used to generalize, i.e., to make predictions. Obviously, we're interested in optimal information extraction.

What does it mean to be "as close as possible"? What's a good line and how to find the best one by that criterion? There are many good choices here, but only one that has nice properties and is easy to work with. I'll explain why as we go. First, I need some definitions.

[20] https://en.wikipedia.org/wiki/Mathematical_optimization
[21] https://en.wikipedia.org/wiki/Regression_toward_the_mean
[22] https://en.wikipedia.org/wiki/Linear_regression

Fix some intercept b_0 and slope b_1; any values will do. If it helps, imagine $b_0 = 0$ and $b_1 = 1$. You don't have to imagine good or optimal values for a particular data set, though in what follows I shall use the ones we estimated with "correlation and means" for the frat dash example. Now, for a particular data set let the *fitted values* be what the line predicts for y_i given x_i.

$$\text{Fitted values:} \quad \hat{y}_i = b_0 + b_1 x_i, \quad \text{for} \quad i = 1, \ldots, n \tag{7.11}$$

Note that $\hat{y}_i \neq y_i$; fitted values are not the same as observed values. Technically, actually, that's possible. For example, if you only had $n = 2$ points, we could use the two-point approach to dial in a slope and intercept that would give a "perfect fit" line to those two points. However, for $n > 3$ and distinct (x_i, y_i), that's not possible. And in all interesting situations it's not possible, so let's presume that's the case going forward. Using values calculated for the frat dash example in §7.3, we may calculate these as follows.

```
yhat <- b0 + b1*x
yhat
```

```
##  [1] 14.45 16.62 18.05 15.13 17.02 15.47 17.76 17.59 16.10 14.90
```

Since fitted and observed values aren't equal, we can measure the discrepancy between them. That's called a *residual*.

$$\text{Residuals:} \quad e_i = y_i - \hat{y}_i, \quad \text{for} \quad i = 1, \ldots, n \tag{7.12}$$

In R:

```
e <- y - yhat
e
```

```
##  [1] -0.845289  0.481122  1.251129 -1.231686 -0.719277  1.425116
##  [7]  0.137127 -1.691274 -0.004081  1.197113
```

Let's take a look at fitted values and residuals for frat dash. See Figure 7.6. Notice that fitted values are parts of the line that correspond to observed x_i locations, whereas residuals "connect" fitted values $\hat{y}_i$ to observed y_i. Residuals comprise the length and direction (i.e., sign indicating above/positive or below/negative) of the green lines, not their position in (x, y) space.

```
plot(x, y, xlab="time before beer", ylab="time after beer")
points(x, yhat, col=2, pch=19)                    ## fitted values
arrows(x, yhat, x, y, col=3, length=0)            ## residuals
legend("topleft", "fitted values", pch=19, col=2, bty="n")
legend("bottomright", "residuals", lty=1, col=3, bty="n")
```

It makes sense, intuitively, to describe a good (fitting) line as one which has small residuals. If residuals are small, then the line is close to the points. But not all residuals can be small. There's a trade-off. Although it's possible to make residuals zero for any pair of points (because that's one way to define a line), unless all n points already line up on a line, perfectly, then you're faced with a balancing act. You can move the line (adjust b_0 and b_1) closer to some points, but only at the expense of moving it farther from others.

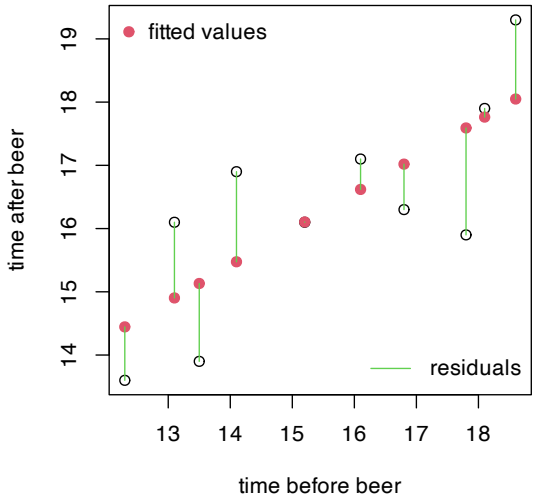

FIGURE 7.6: Fitted values and residuals for the frat dash example. It looks like there's no residual for that middle point, but it's just very small because the line is very close to the observed y_i value.

Presuming it doesn't matter how often the line under-predicts ($e_i > 0$) compared to over-predicting ($e_i < 0$), a sensible way to quantify total distance, for all fitted and observed values in search of balance, is to aggregate squares: $\sum_{i=1}^{n} e_i^2$. This is sometimes called the residual sum of squares (RSS)[23]. There are lots of other reasonable choices here. For example, we could replace the square with an absolute value. This is known as absolute deviation[24]. RSS is special because of what I'm about to do next.

The best line, according to the least squares[25] criteria, solves the following mathematical program, known as ordinary least squares (OLS)[26].

$$\min \sum_{i=1}^{n} e_i^2 = \min \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \tag{7.13}$$

The second "equals" on the right is intended to serve as a reminder that b_0 and b_1 are embedded in each e_i. Sometimes people say OLS to signify that they are using LS in the simplest, most ordinary way, with raw (x_i, y_i) values. If you compare the contents of those two Wikipedia links, you'll see what I mean. There are many non-ordinary applications of LS, but I don't want to get us off-track here. And I don't mind if you say LS when you mean OLS, and vice versa. It's an arbitrary distinction.

How do you solve that mathematical program (7.13)? Well, it's an optimization, and we know all about those, right? Derivative calculus! "Oh no!", you say. Check this out, it's not that bad. Just do b_0 and b_1 one at a time, and when you're boasting to your colleagues, say "gradient" to sound fancy. Starting with b_0, where the first step uses sum, chain and power rules all at once …

[23] https://en.wikipedia.org/wiki/Residual_sum_of_squares
[24] https://en.wikipedia.org/wiki/Average_absolute_deviation
[25] https://en.wikipedia.org/wiki/Least_squares
[26] https://en.wikipedia.org/wiki/Ordinary_least_squares

$$
\begin{aligned}
0 &\overset{\text{set}}{=} \frac{\partial}{\partial b_0} \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \\
0 &= \sum_{i=1}^{n} 2(y_i - b_0 - b_1 x_i)(-1) \\
0 &= \sum_{i=1}^{n} y_i - n b_0 - b_1 \sum_{i=1}^{n} x_i \\
\text{solving} \quad b_0 &= \frac{1}{n} \sum_{i=1}^{n} y_i - b_1 \frac{1}{n} \sum_{i=1}^{n} x_i \\
b_0 &= \bar{y} - b_1 \bar{x}
\end{aligned}
$$

Does this look familiar? It's the same as Eq. (7.10). Coincidence? What do you think is going to happen with slope?

$$
\begin{aligned}
0 &\overset{\text{set}}{=} \frac{\partial}{\partial b_1} \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2 \\
0 &= \sum_{i=1}^{n} 2(y_i - b_0 - b_1 x_i)(-x_i) \\
0 &= \sum_{i=1}^{n} y_i x_i - b_0 x_i - b_1 x_i^2 \\
0 &= \sum_{i=1}^{n} y_i x_i - (\bar{y} - b_1 \bar{x}) x_i - b_1 x_i^2 \qquad \text{sub} \quad b_0 = \bar{y} - b_1 \bar{x} \\
0 &= \left(\sum_{i=1}^{n} y_i x_i - \bar{y} x_i \right) - b_1 \left(\sum_{i=1}^{n} x_i \bar{x} + x_i^2 \right) \\
\text{so} \quad b_1 &= \frac{\sum_{i=1}^{n} y_i x_i - \bar{y} x_i}{\sum_{i=1}^{n} x_i \bar{x} + x_i^2} = \frac{\sum_{i=1}^{n} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \quad \text{i.e.,} \quad b_1 = \frac{s_{yx}}{s_x^2} = r_{yx} \frac{s_y}{s_x}
\end{aligned}
$$

Ok, this can't be a coincidence if it happens twice. Check Eqs. (7.8) and (7.9). The "correlation and means" line is not just a sensible line, it is the best line by the OLS criteria. No other line is closer to the points in a least-squares sense. I don't need to write new code, and I don't need to re-plot Figure 7.5 because nothing has changed in practical terms. But we have a new interpretation. We know that the line is good. We know it's the best!

Don't celebrate yet. We still don't understand the sensitivity of our estimate to a small sample of data. That's because we've been avoiding probability and statistical reasoning.

7.5 Simple linear regression

I just explained (I hope) a connection between the slope of a line and correlation (7.9); in fact, to the best line! Since correlation is a property, or component, in the joint distribution of Y and X, apparently the best line is somehow connected to that joint distribution. That is correct, and makes for a nice warmup, but also irrelevant for what I'm about to pivot to next. The dirty little secret of a statistical approach linear regression is that the only

TABLE 7.1: Some regression vocabulary.

output $Y(x)$	input x
response variable	explanatory variable
dependent variable	independent variable
covariate	covariate

variable that's random is Y. Yes, we study Y conditional on x, by developing $\mathbb{P}(Y \mid x)$, but we don't concern ourselves with $\mathbb{P}(X = x)$.

This is a double-edged sword. It's good because it makes things a lot simpler. We don't have to juggle two random quantities. It also allows the inputs (x) to an input–output experiment $Y(x)$ varying x to come from a design[27] of planned inputs, like a grid. A grid does not have a probabilistic distribution. This is why I have been privileging Y all along, because the distribution of X doesn't matter for what I'm about to show you. And that stuff is huge: this is like 75%[28] of statistics and machine learning in ten pages. Buckle up.

It's bad because sometimes inputs X have a population, and when they're sampled at random they impart uncertainty on the sampling distribution of estimators (like for the slope). A statistical approach to linear regression, as I'm about to present, does not acknowledge that possibility. So it could be missing something. Ignoring the distribution of X will generally undercut any assessments of uncertainty. This is important to know, but it is what it is. I apologize if you're bummed that we're not going to talk too much more about it. I admire you if you find this unsatisfying. You might want to think about pursuing a graduate degree in statistics. For now, I can point you to errors in variables[29] regression. There are many similarities to what's below, but it's lots more complicated.

We're almost ready to get started with technical detail. First, I want to dispense with some terminology. Output $Y(x)$ for input x is sometimes referred to as the *response* variable $Y(x)$ measured under a *condition* or *explanatory variable* x that is varied in an experiment. Or, it could be *dependent* $Y(x)$ at *independent* x. This latter vocabulary emphasizes a designed experiment where x is controlled. There are others, but I really prefer input–output verbiage, which you might say has a machine learning bias. Stats-specific verbiage forces you to think about how the two quantities covary with one another. Sometimes, $Y(x)$ and x are called *covariates*, although again it's important to recognize that $Y(x)$ varies probabilistically, and whereas we think of x as a level that can be controlled with precision. See Table 7.1 for a grouping of synonyms.

A simple linear regression (SLR)[30] setup can be written as follows. Actually, I'm writing it two different ways that are common. There's a third way, in fact, but I'll delay that until we get to MLR in §13.3.

$$\text{Model:} \quad Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i \qquad \text{where} \quad \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2) \tag{7.14}$$

$$\text{or} \quad Y_i \overset{\text{ind}}{\sim} \mathcal{N}(\beta_0 + \beta_1 x_i, \sigma^2) \qquad i = 1, \ldots, n \tag{7.15}$$

These two versions say the same thing, but in a way that emphasizes a different perspective.

[27] https://en.wikipedia.org/wiki/Design_of_experiments
[28] Ok, I made that number up. But it's a lot.
[29] https://en.wikipedia.org/wiki/Errors-in-variables_model
[30] https://en.wikipedia.org/wiki/Simple_linear_regression

I wanted to show you both ways because you'll often see it one way or another depending on where you look. It can also be helpful to briefly contrast the two versions.

Eq. (7.14) expresses deviations from the line as additive, iid noise variables with a known mean (0) but unknown variance (σ^2). These ε_i are similar to residuals e_i, but they're not exactly the same. The e_i are quantities that can be calculated from data given coefficients b_0 and b_1. This is why they're notated with Roman lettering. Greek ϵ_i can never be known exactly, but we assume a distribution. That distribution can be used to deduce a sampling distribution for all estimated quantities. It can even parlay into a distribution for all e_i values, but that comes a little later in Chapter 13.

Eq. (7.15) instead puts the line inside the mean of the Gaussian. I like to think of $\mu_i = \beta_0 + \beta_1 x_i$ in order to make a connection with Chapter 2 on simple location models (μ without a subscript) and §6.3 on ANOVA where we have a different μ_i for each group i. Now we have μ_i that varies with x_i, which we assume follows a line but with coefficients β_0 and β_1, which are unknown to us. This is why it says "ind" for "independent" above the $\sim$. Each Y_i does *not* have an identical distribution to Y_j, when $i \neq j$ and as long as $x_i \neq x_j$, because they have different means μ_i and μ_j. But they are still independent of one another, conditional on β_1, which determines the mean

$$\beta_1 = \frac{d\,\mathbb{E}\{Y \mid x\}}{dx}.$$

Only errors ε_i are iid.

I hope that was helpful. Now I want to talk about how to estimate unknown parameters: $\theta = (\beta_0, \beta_1, \sigma^2)$. We actually already have a way to estimate β_0 and β_1, and we know it's optimal in a "least-squares" sense, but not in a statistical context. Parameter σ^2, which imparts noise through ε_i, is new to this discussion, but not new to Gaussian modeling at large.

How to infer θ, which here is $p = 3$-dimensional? That's exactly what Alg. 3.1 is for. It tells you what to do when you have a statistical model: figure out what the likelihood is and then maximize it (via the log) with calculus. I know you remember that from earlier chapters, but I wanted to say it out loud again because it's been a while.

The fact that the model involves independent Gaussians is crucial. It means that a lot can be borrowed from Chapter 3. This is where the idea of μ_i (here) rather than μ (there) is really handy. We can just copy things over and add an i subscript. Then just replace $\mu_i = \beta_0 + \beta_1 x_i$. The equation below is Step 3 from Eq. (3.3) relabeled in exactly that way.

$$\ell(\beta_0, \beta_1, \sigma_2) = c - \frac{n}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i)^2 \tag{7.16}$$

Look how similar that is to Eq. (7.13). Really the only thing that's new is $\frac{n}{2}\log\sigma^2$, and everything is negative. So taking derivatives with respect to β_0 and β_0 to maximize the likelihood is the same as minimizing for b_0 and b_1 in OLS. There's nothing to do; we already have the answer.

$$\hat{\beta}_0 = b_0 = \bar{y} - \hat{\beta}_1\bar{x} \qquad \text{and} \qquad \hat{\beta}_1 = b_1 = \frac{s_{yx}}{s_x^2} = r_{yx}\frac{s_y}{s_x} \tag{7.17}$$

All that's left is σ^2. Here we can make another deduction and avoid reinventing the wheel.

Look back at the lead up to Eq. (3.5) in §3.1 for $\hat{\sigma}^2$ in a simple mean setup (μ). Now we have μ_i, but otherwise no difference. Just pop in

$$\hat{\mu}_i \equiv \hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i \quad \text{for} \quad \hat{\mu} = \bar{y}$$

and you're good to go. In other words, differentiating $\ell(\beta_0, \beta_1, \sigma_2)$ with respect to σ^2 in Eq. (7.16), setting equal to zero and solving gives

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2. \tag{7.18}$$

As in Chapter 3 this is a biased estimate. I shall leave it to you as an exercise to show that $\mathbb{E}\{\hat{\sigma}^2\} = (n-2)\sigma^2/n$. Consequently, it's common to adjust the denominator to correct for that bias.

$$s^2 = \frac{1}{n-2} \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2 \tag{7.19}$$

The square-root of this quantity is sometimes called the *residual standard error*, because it is the standard error of residuals $e_i = \hat{\varepsilon}_i = y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i$, for $i = 1, \ldots, n$.

The intuition here, involving degrees of freedom (DoF), is similar to Chapter 3. You're estimating two parameters, β_0 and β_1 before estimating σ^2, so you've lost two DoF. Said another way: two points define a line, so if you only have $n = 2$ then you can't possibly observe any variance around that line. All of your DoF are used up in estimating the line. You need $n \geq 3$ in order to estimate σ^2. Moreover, at least three must be distinct – more than three unique (x_i, y_i) pairs. Note that if your responses truly follow the SLR model (7.14)–(7.15), then there can't be any duplicate y_i-values. You'll never get duplicates when drawing samples from a Gaussian. So all you need is (at least three) distinct x_i values.

Here is a by-hand calculation that uses fitted values calculated earlier.

```
s2 <- sum((y - yhat)^2)/(n - 2)
s2
```

```
## [1] 1.361
```

In R, you can fit a linear model (all parameters at once) with the `lm` command. The syntax is a little funny if you've never used a `formula` construct before. Fortunately, we got a glimpse when using library commands to fit an ANOVA at the end of §6.3. The most important thing to remember is that `y` goes on the left of the `~`, and `x` on the right. Aren't you glad I've been doing that all along?

```
fit <- lm(y ~ x)
rbind(coef(fit), c(b0, b1))
```

```
##      (Intercept)     x
## [1,]        7.41 0.572
## [2,]        7.41 0.572
```

Many other quantities may be extracted from the `fit` object returned by `lm`. For example, you can get fitted values $\hat{y}_i$ as `fit$fitted.values` and residuals e_i as `fit$residuals`. One way to calculate s^2 is through those residuals.

```
sum(fit$residuals^2)/fit$df.residual
```

```
## [1] 1.361
```

I've always found it strange that s^2 is not explicitly saved in the `lm` object. Its square root is available as `summary(fit)$sigma`, but I want to wait to describe the `summary` command for `lm`-objects until later in §7.6.

That covers estimation of unknown parameters in a SLR. Now what do we do with that? Two things. One is prediction. I already gave you an example in §7.4 of what to do when a new x comes along, and you want to know what $Y(x)$ is. Remember Brandon? But we didn't have any uncertainty attached to that. I want to get to that.

An important stepping stone is understanding how sensitive estimated intercept and slope coefficients are to the training data: their sampling distribution. And that's the second thing. So I'll show you that first, which can be interesting in its own right. If the slope is zero ($\beta_1 = 0$), which is one important "what if", then x isn't useful for predicting Y. Ultimately, prediction uncertainty involves a synthesis of parameter uncertainty and underlying population variability, both of which involve σ^2 as estimated by s^2.

Diagnostics

Before doing that, I want to make a connection to correlation – to earlier in the chapter – and make two other quick notes. It's always a good idea to visualize your fit along with a scatterplot of the data, like in Figure 7.5, if for no other reason than as a sanity check. The line should "look good" to you, or something isn't right. What does it mean to "look good"? Well, that's subjective. One way to make that quantitative is to measure the correlation between what you predict (your fitted values) and the values in the data.

```
r.lm <- cor(y, yhat)
c(r, r.lm)
```

```
## [1] 0.7608 0.7608
```

Notice that this is the same as r_{yx}. Interesting, right? How could that be? Well, we already know that the fitted values follow a line, because that's how they're defined: $\hat{y}_i = b_0 + b_1 x_i$. (Notice how I'm using $\widehat{\beta}$ and b interchangeably.) So, they are perfectly correlated with the x-values.

```
cor(yhat, x)
```

```
## [1] 1
```

That makes fitted values interchangeable with inputs x as regards correlation. In linear modeling, one usually looks at the square of this correlation.

```
r2 <- r.lm^2
r2
```

```
## [1] 0.5787
```

This r^2 value is intimately related to the ANOVA one from §6.3, but we'll have to wait §13.5 for more detail. It has similar interpretation: the proportion of variability explained by the model fit. Apparently, our MLE line fit explains 58% of that variability. That seems better than nothing, but is it statistically significant? I bet you know how to answer that question by MC already. Wait for Chapter 13. With all the waiting I'm asking you to do one wonders who is getting impatient, you or me? In the meantime, I'll show you a more direct way to answer that question next in §7.6.

Another way to judge goodness-of-fit[31] is through residuals, rather than through fitted values. This may seem indirect, but there's good insight here from additional perspective, especially visually. Figure 7.7 shows residuals versus fitted values as a scatterplot. (And this is basically the same as residuals versus inputs x because, again, these are perfectly correlated with fitted values. I encourage you to remake the plot with `x` rather than `yhat`.)

```
plot(yhat, e, xlab="fitted values", ylab="residuals")
abline(h=0, col=2, lwd=2)
```

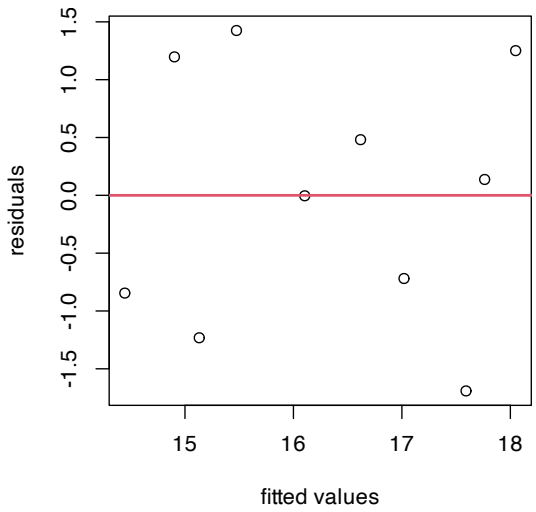

FIGURE 7.7: Residuals versus fitted values.

If your fit is good – no matter how it is obtained (but for us, it was via MLE from a SLR) – there should be no pattern in what's left over. Your residuals, as a function of x or $\hat{y}$, should look like a scatter of noise. That's not guaranteed to happen. Your fit could be crap because your data doesn't follow a linear relationship, or maybe it's not independent Gaussian. But this one is pretty good. It's customary to overlay a horizontal line at zero, as I have done in red, as a visual assist. You should have about the same number of residuals above as below the line anywhere you look from left to right.

Again, that's not guaranteed for all data, but here's something that is.

```
round(c(mean(e), cor(yhat, e)), 10)
```

[31] https://en.wikipedia.org/wiki/Goodness_of_fit

```
## [1] 0 0
```

Every single time you fit a line via OLS/MLE under SLR, the average residual will be zero, and there will be no (linear) correlation between the prediction (or the original inputs x) and your residual. I rounded to ten digits because if you don't do that you get numbers like 10^{-15}, which are numerically nonzero. That happens because computer arithmetic isn't perfect. You should, technically speaking, get exactly zero. Anything less than 10^{-8} is basically zero on a 64-bit machine. Your computer can report numbers smaller than that, but they're not accurate.

Why do we always get zero? Because OLS is "efficient", in a sense. It takes all of the "linearness" out of the data, leaving no residual linearity left over. That means no intercept is left over (`mean(e) = 0`) and no slope (`cor(yhat, e) = 0`). This happens even if the data aren't linear or the responses aren't independent Gaussian. You might see some nonlinear pattern left over in your residuals, but not a linear one.

Perhaps you find that verbal justification unsatisfying, and would have preferred a more technical one. Good, I've got just the exercise for you in §7.8! It's not hard to show that the two statistics provided in the code chunk above are the same as "correlation and means", and therefore the same as an SLR MLE. You can do it; I believe in you.

7.6 SLR inference

My presentation here will privilege testing the slope, β_1, because that's the most interesting "what if". When slope is zero, there's no (linear) relationship between x and y, because it means multiply x by zero, then add something, to get y. But along the way, I shall mention how things work for the intercept, β_0, too. We'll need results for both when it comes to prediction.

Just to be pedantic about it, and so that we're all on the same page, formal hypotheses for testing are provided below.

$$\begin{aligned} &\mathcal{H}_0 : \beta_1 = 0 && \text{null hypothesis ("what if")} \\ &\mathcal{H}_1 : \beta_1 \neq 0 && \text{alternative hypothesis} \end{aligned} \tag{7.20}$$

Feel free to replace zero with anything you want. It's a little awkward to use double-subscripts for an arbitrary β_1-value for the test.

Before getting started, there's an opportunity here for an important reminder about what's said and unsaid in a pair of competing hypotheses. $\mathcal{H}_0$ and $\mathcal{H}_1$ mention β_1, but they don't mention β_0 or σ^2, and what we choose to do with those may impact the analysis. A typical setup is to fix β_0, e.g., at the MLE, and average over $\sigma^2 \sim (n-2)s^2/\chi^2_{n-2}$, following standard means analysis from Chapter 3, or more recently §6.3.

In the past I've commented that it doesn't matter what you fix for unspecified mean parameters when testing. I've been known to put silly values into the MC. This won't work here for linear regression. Estimated slope is negatively correlated with estimated intercept. I'll show you the exact form later (7.27), but for now I shall simply appeal to your intuition. An underestimated intercept ($\hat{\beta}_0$ too low) must be compensated for by a raised slope (higher

$\hat{\beta}_1$) to get the line to go through the points, and vice versa. If slope is too shallow, then you'll have to raise the intercept.

That's all to say that the typical SLR testing apparatus, when focused on one mean parameter (like β_1) at a time, must be both implemented and interpreted conditionally on other mean parameters (like β_0). Our outcome could change if we choose another value for the fixed parameter, e.g., β_0. This isn't a huge deal, although it may seem like one since I'm kinda making a big deal of it, but I just want you to be aware. This will become more important, and I'll revisit this conversation, when we get to MLR in Chapter 13. It'll be a bigger deal then.

As usual, there are two ways to conduct the test: my preferred Monte Carlo (MC), and the old-fashioned way. Actually, of all the old-fashioned testing procedures, I like SLR ones best because they're somewhat intuitive – or, more accurately, they help build intuition because they provide insight – and they're exact, not asymptotic.

Before jumping in, we need a testing statistic. It's natural that a test about β_1 should center around $\hat{\beta}_1$. This is a good choice, and the one I shall run with going forward, but there are other good ones. For example, r^2 is interesting because it captures goodness-of-fit, and has a nice percentage interpretation. But it's less common for SLR. There's a good reason to use r^2 in an MLR, so I'm going to delay demonstrating that choice. I've already pointed to an F-testing connection here with ANOVA from §6.3. You can think of linear regression as defining groups continuously, along the line as $\mu_i = \beta_0 + \beta_1 x_i$ instead of discretely as $\mu_1, \cdots, \mu_m$.

Testing the slope by MC

By now, I hope you're getting the hang of this. Just follow the model (7.15) and null hippopotamus (7.20), but don't forget things you learned in earlier chapters, like about how to sample variances.

```
beta1 <- 0
B1s <- rep(NA, N)
for(i in 1:N) {
  sigma2s <- (n - 2)*s2/rchisq(1, n - 2)          ## Chapter 3 or 6
  Ys <- rnorm(n, b0 + beta1*x, sqrt(sigma2s))     ## Gaussian SLR under H0
  B1s[i] <- cor(Ys, x)*sd(Ys)/sd(x)               ## or coef(lm(Ys ~ x))[2]
}
```

Figure 7.8 shows the empirical sampling distribution of $\hat{\beta}_1$ under $\mathcal{H}_0$. It appears that the estimated statistic is in the tails, and I would guess that we'd reject the null.

```
hist(B1s, main="")
abline(v=c(b1, -b1), lty=1:2, col=2, lwd=2)
legend("topright", c("b1", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
```

A more precise assessment is provided by the p-value.

```
b1.pval.mc <- 2*mean(B1s > b1)
b1.pval.mc
```

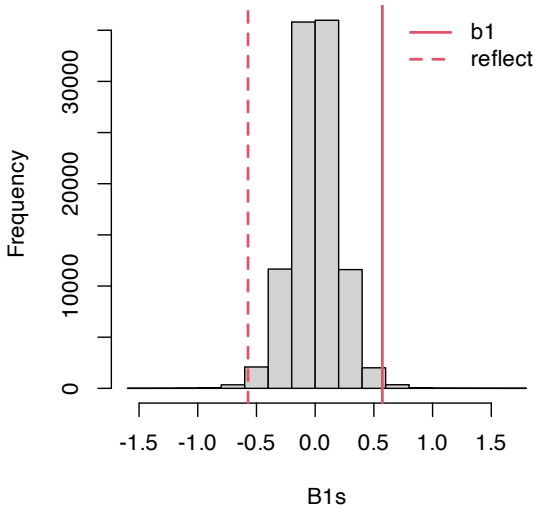

FIGURE 7.8: Histogram of sampled $\hat{\beta}_1$ for testing $\beta_1 = 0$ in the frat dash experiment.

```
## [1] 0.01058
```

Is it a coincidence that this is close to the p-value we got when testing $\rho = 0$? Recall that was $\phi_{\rho=0} = 0.0104$. The two pairs of hypotheses, in Eqs. (7.6) and (7.20), are not asking exactly the same question. Yet it's a similar question, in part because of how the two are intimately linked (7.9).

Running back through a similar simulation in order to test some value for the intercept β_0 is straightforward. I'm not going to bore you with that. In practice, one is rarely interested in testing the intercept. Its meaning in the context of an applied data analysis often isn't immediate. However, it *is* definitely the case that you might want to test a value for β_1 other than zero. Zero has a clear meaning: is the line useful? So do other values depending on context. For example, it is common to test $\beta_0 = 1$. That could shed light on our frat dash analysis. A slope of unity means there's no change running speed before and after a beer: $\mathbb{E}\{Y(x)\} = \beta_0 + x$. That could be interesting to know.

I'm confident you could run back through the code above with a value of `beta1 <- 1`, so I'm not going to hold your hand on that here. Instead, I'd like to run through a CI for β_1. This is pretty easy too, since you can simply use $\beta_1 = \hat{\beta}_1 = b_1$ in that same MC. And, since a CI is the inverse of a test, we can use it to ask any (two-sided) hypothesis testing question implicitly.

```
B1s <- rep(NA, N)
for(i in 1:N) {
  sigma2 <- (n - 2)*s2/rchisq(1, n - 2)
  Ys <- rnorm(n, b0 + b1*x, sqrt(sigma2))        ## only change
  B1s[i] <- cor(Ys, x)*sd(Ys)/sd(x)
}
```

A CI may be extracted from quantiles of that empirical distribution.

```
b1.CI.mc <- quantile(B1s, c(0.025, 0.975))
b1.CI.mc
```

```
##   2.5%  97.5%
## 0.1756 0.9696
```

As you can see, $\beta_1 = 1$ is not included in this interval (barely). Therefore, we would reject $\mathcal{H}_0 : \beta_1 = 1$ in a two-sided test at the 5% level. It does appear that there is indeed a change in speed (a slow-down) after a beer. I'll let you debate whether that's due to inebriation, or the extra weight they're carrying. (Forty ounces of beer is heavy. Just imagine how much slower you'd run if you were carrying an extra 40 oz in a backpack, or your bladder.)

Testing the slope by math

Although the correlation form (7.9) for $\hat{\beta}_1$ is most intuitive, the covariance form (7.8) is easier to work with mathematically, at least in this context.

$$\hat{\beta}_1 = b_1 = \frac{s_{yx}}{s_x^2} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} = \frac{\sum_i (x_i - \bar{x}) y_i}{\sum_i (x_i - \bar{x})^2},$$

where the latter equality uses $\sum_{i=1}^n (x_i - \bar{x}) = \sum_{i=1}^n x_i - n\bar{x} = 0$. This means that the MLE for slope can be written as a linear combination of y_i values.

$$\hat{\beta}_1 = \sum_{i=1}^n w_i y_i \quad \text{where} \quad w_i = \frac{(x_i - \bar{x})}{(n-1)s_x^2}$$

That's convenient for expectations and variances via linearity properties, as we've done several times previously. Here are a few key facts about those weights that are easy to show given the info above.

$$\sum_{i=1}^n w_i = 0, \quad \sum_{i=1}^n w_i x_i = 1, \quad \text{and} \quad \sum_{i=1}^n w_i^2 = \frac{1}{(n-1)s_x^2} \tag{7.21}$$

Also, remember that only Y_i-values are random in this analysis, and $Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$ so really only ε_i are random. Recall from Eq. (7.14) that $\mathbb{E}\{\varepsilon_i\} = 0$ and $\mathbb{V}\text{ar}\{Y_i\} = \mathbb{V}\text{ar}\{\varepsilon_i\} = \sigma^2$. These facts, together with Eq. (7.21), provide lots of potential for algebraic simplification.

Using linearity of means[32],

$$\begin{aligned}
\mathbb{E}\{\hat{\beta}_1\} &= \sum_i w_i \mathbb{E}\{Y_i\} \\
&= \beta_0 \sum_i w_i + \beta_1 \sum_i w_i x_i + \sum_i w_i \mathbb{E}\{\varepsilon_i\} \\
&= 0 + \beta_1 + 0 \\
&= \beta_1.
\end{aligned} \tag{7.22}$$

So the MLE is unbiased for the slope. Using linearity of variances[33] of independent ε_i,

[32] https://en.wikipedia.org/wiki/Expected_value#Properties
[33] https://en.wikipedia.org/wiki/Variance#Linear_combinations

$$\mathbb{V}\text{ar}\{\hat{\beta}_1\} = \sum_{i=1}^{n} w_i^2 \mathbb{V}\text{ar}\{Y_i\} = \sigma^2 \sum_{i=1}^{n} w_i^2 = \frac{\sigma^2}{(n-1)s_x^2}. \tag{7.23}$$

Putting Eqs. (7.22) and (7.23) together with the usual sum of independent Gaussians trick[34] gives

$$\hat{\beta}_1 \sim \mathcal{N}(\beta_1, \sigma_{b1}^2) \quad \text{where} \quad \sigma_{b1}^2 = \frac{\sigma^2}{(n-1)s_x^2}.$$

Notice what affects uncertainty in estimated slope.

- Overall variability in the sample, σ^2. Noisier data makes it harder to learn β_1.
- Training data size, n. This quantity is in the denominator. So bigger n means lower variance, more learning. As $n \to \infty$ we have $\sigma_{b1}^2 \to 0$.
- Finally s_x^2, also in the denominator. Apparently, the more spread you have in your inputs (not just more inputs, bigger n) the faster you learn.

This last item offers insight into experimental design strategies if you have the ability to control at which x-values y's are observed. If you spread them out, then you can make more efficient use of your limited, n-sized sample. The terminology for this in statistical parlance is leverage[35]. Unlike other stats jargon, this one makes sense intuitively. The farther your lever (think line, which looks like a lever) is from a pivot joint (think center of the data), the more force you can exert. You have more leverage.

Since $\sigma^2 \sim (n-2)s^2/\chi^2_{n-2}$ we may use a t statistic, which is more practical, after plugging in s^2 estimating σ^2.

$$t_1 = \frac{\hat{\beta}_1 - \beta_1}{s_{b_1}} \quad \text{with (squared) s.e.} \quad s_{b_1}^2 = \frac{s^2}{(n-1)s_x^2} \tag{7.24}$$

and $T \sim t_{n-2}$ allowing a test of any β_1 value. In R, for the frat dash example, testing $\beta_1 = 0$ ...

```
sb1 <- sqrt(s2/((n - 1)*s2x))
t1 <- b1/sb1
c(sb1, t1)
```

```
## [1] 0.1725 3.3153
```

With $|t| > 2$ we'll reject $\mathcal{H}_0 : \beta_1 = 0$, but it's nice to have a precise p-value.

```
b1.pval.t <- 2*pt(-abs(t1), df=n - 2)
c(mc=b1.pval.mc, math=b1.pval.t)
```

```
##      mc    math
## 0.01058 0.01061
```

[34] https://en.wikipedia.org/wiki/Sum_of_normally_distributed_random_variables
[35] https://en.wikipedia.org/wiki/Leverage_(statistics)

That's pretty close to the MC result, but lots harder. However, it's somewhat satisfying to patiently push through the logic behind a sampling distribution. Everything I showed you is either algebra (of sums), basic probability, and combinations thereof. Contrast that with testing Pearson's ρ. If already you've forgotten all my apologies and hand waving, then I won't remind you. Shoot, I just did.

A CI follows the usual "est $\pm\, q \times$ se" formula.

```
b1.CI.t <- b1 + c(-1, 1)*qt(0.975, n - 2)*sb1
rbind(mc=b1.CI.mc, math=b1.CI.t)
```

```
##        2.5%  97.5%
## mc   0.1756 0.9696
## math 0.1741 0.9699
```

This is again pretty close. I'll leave it to you to check $\beta_1 = 1$ via a t-test, as compared to the MC I left for you as an exercise. (Just replace `b1` with `b1 - 1` in the code.) Everything should check out.

There's a library built into R that can do all this for you. Before we get to that, I want to show you how to work with the intercept β_0. The MC is pretty much the same for intercepts and slopes, but the math is a little different because the formula for the intercept is different. In fact, the formula for the intercept MLE depends on the slope!

Using $\bar{Y} = \frac{1}{n}\sum_{i=1}^{n}(\beta_0 + \beta_1 x_i + \varepsilon_i)$ and following a similar program yields

$$\begin{aligned}
\mathbb{E}\{\hat{\beta}_0\} &= \mathbb{E}\{\bar{Y}\} - \mathbb{E}\{\hat{\beta}_1\}\bar{x} \\
&= \frac{n}{n}\beta_0 + \frac{1}{n}\beta_1 \sum_i x_i + \frac{1}{n}\sum_i \mathbb{E}\{\varepsilon_i\} - \beta_1\bar{x} \qquad \text{since unbiased} \\
&= \beta_0 + \beta_1\bar{x} + 0 - \beta_1\bar{x} \\
&= \beta_0,
\end{aligned}$$

and so also unbiased. Similarly,

$$\begin{aligned}
\mathbb{V}\text{ar}\{\hat{\beta}_0\} &= \frac{1}{n^2}\sum_i \sigma^2 + \bar{x}^2\mathbb{V}\text{ar}\{\hat{\beta}_1\} - 2\bar{x}\mathbb{C}\text{ov}(\bar{Y}, \hat{\beta}_1) \\
&= \frac{\sigma^2}{n} - \bar{x}^2\sigma^2_{b_1} - 0 \\
\text{defining} \quad \sigma^2_{b_0} &= \sigma^2\left(\frac{1}{n} + \frac{\bar{x}^2}{(n-1)s_x^2}\right), \qquad (7.25)
\end{aligned}$$

where $\mathbb{C}\text{ov}(\bar{Y}, \hat{\beta}_1) = 0$ can be shown explicitly, but I'll appeal to your intuition that the average level of the data $\bar{Y}$ has nothing to do with an estimated slope $\hat{\beta}_1$. Move the data up by 100, or something, and it won't change what you estimate for the slope. It will change the intercept though! Interpreting intercept variability (7.25) is similar to slope. More data (bigger n) and greater spread of inputs (s_x^2) balances out spread in outputs (σ^2). Eventually, as $n \to \infty$ you can learn both without error.

As before, a sum of Gaussians is Gaussian, so $\hat{\beta}_0 \sim \mathcal{N}(\beta_0, \sigma^2_{b_0})$, but this result is more practical with estimated s^2, leading to

TABLE 7.2: LM fit: $s = 1.167$ on 8 DoF, $r^2 = 0.579$

	coef	se	tstat	pval
icept	7.410	2.7099	2.734	0.0257
slope	0.572	0.1725	3.315	0.0106

$$t_0 = \frac{\hat{\beta}_0 - \beta_0}{s_{b_0}} \quad \text{with (squared) s.e.} \quad s_{b_0}^2 = s^2 \left(\frac{1}{n} + \frac{\bar{x}^2}{(n-1)s_x^2} \right). \tag{7.26}$$

Just to keep things moving, I'll test $\mathcal{H}_0 : \beta_0 = 0$, even though I don't think that "what if" is particularly interesting for the frat dash example. I want to use the values calculated in R, below, to make a connection to library output summarizing tests for linear models.

```
sb0 <- sqrt(s2*(1/n + xbar^2/((n - 1)*s2x)))
t0 <- b0/sb0
c(sb0, t0)
```

```
## [1] 2.710 2.734
```

This is an easy reject of the null with p-value:

```
b0.pval.t <- 2*pt(-abs(t0), df=n - 2)
b0.pval.t
```

```
## [1] 0.02567
```

Don't worry $s_{b_0}^2$ will come in handy in a less trivial way later in §7.7 when summarizing predictive uncertainty. For now, I want to arrange all of these parameter estimates, standard errors, t statistics, p-values and even r^2 in a table. Everything all in one place, a little like the ANOVA table from §6.3. See Table 7.2.

```
tab <- cbind(c(b0, b1), c(sb0, sb1), c(t0, t1), c(b0.pval.t, b1.pval.t))
rownames(tab) <- c("icept", "slope")
colnames(tab) <- c("coef", "se", "tstat", "pval")
kable(round(tab, 4), caption="(ref:lmtab-beer)")
```

All t stats and p-values are for the null hypothesis that the coefficient in that row is zero. Observe that the third column, with the t stat(s), is the ratio of the first and second columns. You can get similar information in R by calling `summary` on the `fit` object returned by `lm`.

```
summary(fit)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -1.6913 -0.8138  0.0665  1.0181  1.4251
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    7.410      2.710    2.73    0.026 *
## x              0.572      0.173    3.32    0.011 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.17 on 8 degrees of freedom
## Multiple R-squared:  0.579,  Adjusted R-squared:  0.526
## F-statistic:   11 on 1 and 8 DF,  p-value: 0.0106
```

Notice that everything is there, and more, although it may go by a slightly different name. I won't be explaining Adjusted r^2[36], and we'll talk more about that f statistic in Chapter 13, but it's really not that different from the one in §6.3. The summary of residuals at the top may be reproduced as

```
summary(e)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.6913 -0.8138  0.0665  0.0000  1.0181  1.4251
```

Observe how the "`Mean`" entry is missing in the R `summary` because, always being zero, it's pretty boring. Those "`Signif. codes`" (the stars) are meant to provide a quick visual queue on significance. There are more stars to the right of the p-value if it's smaller. No star indicates failure to reject the null that the coefficient in question is zero at the 5% level. Like with ANOVA, a common homework or exam question involves omitting some table entries and asking students to recover them forensically, possibly along with some other helpful information not in the table, like $s_y^2, s_x^2, \bar{y}, \bar{x}$, etc.

Finally, `confint` can be used on an `lm`-class object. By default, it does a 95% interval. I'm just showing you the part that corresponds to β_1, since that's what we calculated by-hand and via MC earlier.

```
rbind(mc=b1.CI.mc, math=b1.CI.t, lib=confint(fit)[2,])
```

```
##        2.5%  97.5%
## mc   0.1756 0.9696
## math 0.1741 0.9699
## lib  0.1741 0.9699
```

Before pivoting to prediction, I want to note that there's a lot more that *could* go here. I've introduced the most important and interesting parts – in my opinion – and left some others for you as homework exercises. What else is there? We glossed over leverage earlier. There's what amounts to a whole section just on that at that Wikipedia link. You can do a whole semester-long class combining SLR and MLR. If you think this stuff is just a little-bit cool, you'll think what's *not* in here – in this chapter or elsewhere in the book – is "totally slay, Ohio-riz and has positive aura". (My daughter said I had to work that into the book

[36] https://en.wikipedia.org/wiki/Coefficient_of_determination#Adjusted_R2

somehow. I'm pretty sure I did it wrong because, as she constantly reminds me, I'm "super cringe".)

7.7 SLR prediction

Suppose you had new x and you wanted to know $Y(x)$. We did this already in §7.3, although not in a statistical modeling context. Remember Brandon with $x = 14.5$? Some things will be the same here. All of the uncertainty will be new. Note that x is an arbitrary input, so whatever we do for one x should work for any x. For all x. We shall see equations and procedures representing our predictions, and their uncertainty, that are a function of x, or would work for any value of `xp` provided in code, including a vector of many `xp`.

In what follows, I'm going to develop predictions for a grid of one hundred x-values in the frat dash example that goes from a little below the smallest input to a little above the largest one.

```
np <- 100
xp <- seq(10, 20, length=np)
```

This helps a lot with plotting, enabling a study of how predictions and uncertainties change as a function of x. There's one x in `xp` that's close enough to Brandon's time to use in lieu of actually plugging in 14.5. You may certainly do that later if you'd like.

If we knew all of the Greek quantities, then the prediction would be $Y(x) = \beta_0 + \beta_1 x + \varepsilon$. But we don't, so we settle for estimates $\widehat{\beta}_0 = b_0$ and $\widehat{\beta}_1 = b_1$. All we'll ever know about ε is that it's Gaussian with mean zero and variance σ^2. Our estimate for σ^2 is s^2.

Using those estimates is straightforward if you follow the simple principles that are, I hope, starting to become second nature. It may seem superficially more complicated, because there are lots of estimated quantities all coming together at once, but with some patience you can get there. We already know how to develop the sampling distribution for each estimated quantity in isolation. All that remains is putting them together for $Y(x)$.

As usual, I think it's easier to begin with MC, and then see how the math matches with cute formulas.

Prediction by MC

The code chunk below starts out the same as the CI code for β_1 from §7.6. The first change is to additionally re-fit the intercept. Notice that I don't save each sample (of coefficients) separately. Rather, I take those values and save the predicted line for `xp`. Finally, that line is used to sample predictive "data" for `xp`.

```
YYmeans <- YYs <- matrix(NA, nrow=N, ncol=np)
for(i in 1:N) {
  sigma2s <- (n - 2)*s2/rchisq(1, n - 2)            ## noise variance
  Ys <- rnorm(n, b0 + b1*x, sqrt(sigma2s))          ## synthetic data
  B1s <- cor(Ys, x)*sd(Ys)/sd(x)                    ## re-fit slope
  B0s <- mean(Ys) - B1s*mean(x)                     ## re-fit intercept
```

```
  YYmeans[i,] <- B0s + B1s*xp                          ## predict line
  YYs[i,] <- rnorm(np, YYmeans[i,], sqrt(sigma2))      ## noise around line
}
```

Figure 7.9 shows a selection of those sampled lines. Showing all $N = 10^5$ of them is overkill, so I took that down by a factor of 100. If I had saved all of `B0s` and `B1s`, I could have instead plotted each line with an `abline` command. That's a perfectly valid choice. However, saving means is somewhat more versatile because it allows an interval to be calculated via `quantile`. But hold that thought for a moment.

```
sel <- seq(1, N, by=100)
matplot(xp, t(YYmeans[sel,]), col="gray", type="l", lty=1,
  xlab="time before beer", ylab="time after beer",)
points(x, y)
legend("bottomright", "selection of YYmeans", col="gray", lty=1, bty="n")
```

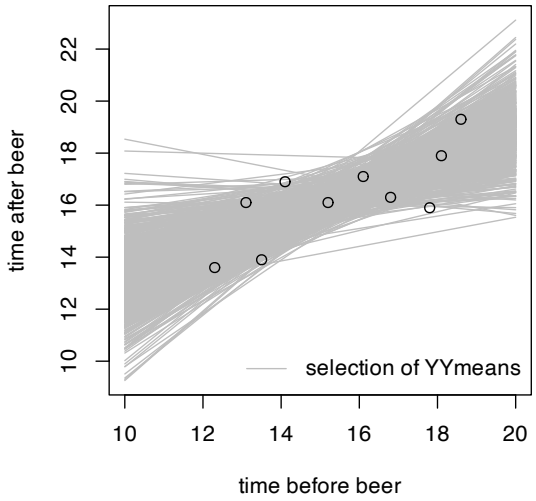

FIGURE 7.9: Empirical sampling distribution of predictive means.

Notice the diversity of lines in the plot. Most go through the data, but not all. Most have positive slope, but not all. Each one of these lines offers a potential explanation for the data, a potential prediction. Their spread derives from the sampling distribution of slopes and intercepts, about which there is considerable uncertainty, largely owing to the noise in this small-n sample.

You can make a similar plot for sampled `YYs`, but since these values have extra iid noise around `YYmeans` it would look lots messier. Check that yourself if curious. `YYs` are more useful to understand the spread of the sample, via σ^2, around the lines. The code provided below extracts quantiles from both samples. The former defines what is known as a *predictive interval* (PI), since it captures all uncertainties. The latter is called a CI because it only captures uncertainty in the mean prediction, like a CI for μ in Chapter 3, but now as a function of x for $\mu(x)$.

```
low.PI.mc <- apply(YYs, 2, quantile, prob=0.025)
high.PI.mc <- apply(YYs, 2, quantile, prob=0.975)
low.CI.mc <- apply(YYmeans, 2, quantile, prob=0.025)
high.CI.mc <- apply(YYmeans, 2, quantile, prob=0.975)
```

A visual is offered in Figure 7.10. This is a classic view of prediction and its decomposition of sources of uncertainty.

```
plot(x, y, xlim=range(xp), ylim=range(c(low.PI.mc, high.PI.mc)))
lines(xp, colMeans(YYmeans), lwd=2)
lines(xp, low.CI.mc, lwd=2, lty=2)
lines(xp, high.CI.mc, lwd=2, lty=2)
lines(xp, low.PI.mc, lwd=2, lty=2, col=2)
lines(xp, high.PI.mc, lwd=2, lty=2, col=2)
legend("bottomright", c("mean", "95% CI", "95% PI"),
  col=c(1, 1, 2),  lty=c(1, 2, 2), lwd=2, bty="n")
```

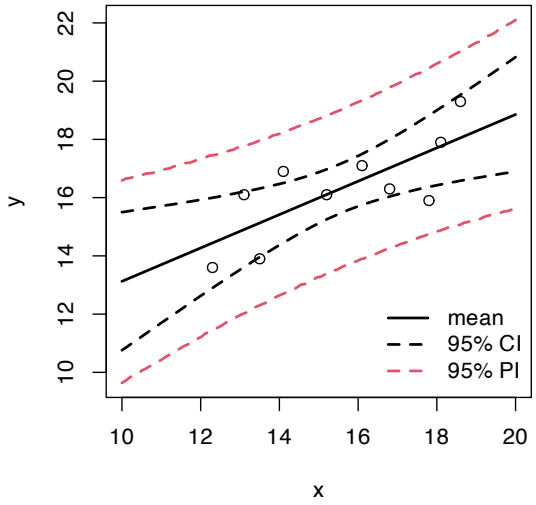

FIGURE 7.10: Full predictive distribution as means (of means) and error bars, via MC.

Notice how both sets of so-called *error bars* take on a hyperbolic[37] shape, more accentuated for the CI but definitely still visible in the PI. Apparently, as you move away from the center of the data, your predictive uncertainty increases. That makes sense intuitively. It also connects back the concept of leverage that I glossed over earlier in §7.6, but here it's the converse. The data have weaker influence on inferences that are farther away from its center of mass – farther from the pivot joint.

Prediction by math

Denote by $\hat{Y}(x) = \hat{\beta}_0 + \hat{\beta}_1 x$ what you would predict given fitted coefficients for new input x. Similarly, let $\tilde{Y}(x) = \hat{\beta}_0 + \hat{\beta}_1 x + \varepsilon$ account for the additional population variance σ^2. The former, $\hat{Y}(x)$ is sometimes called the fitted value (of a prediction), because it's similar to $\hat{y}_i$

[37] https://en.wikipedia.org/wiki/Hyperbolic_functions

for each x_i, but for new x. It's what you would predict for the fit, i.e., the mean line, and it is the basis of a CI summary (connecting back to our MC analysis). The latter, $\tilde{Y}(x)$ is called the predicted value because it has the full population uncertainty, and is the basis of a PI. Note that both have the same mean since $\mathbb{E}\{\varepsilon\} = 0$.

$$\begin{aligned}\mathbb{E}\{\tilde{Y}(x)\} &= \mathbb{E}\{\hat{\beta}_0\} + \mathbb{E}\{\hat{\beta}_1\}x + \mathbb{E}\{\varepsilon\} = \mathbb{E}\{\hat{\beta}_0\} + \mathbb{E}\{\hat{\beta}_1\}x \\ &= \mathbb{E}\{\hat{Y}(x)\} = \beta_0 + \beta_1 x = \mathbb{E}\{Y(x)\}\end{aligned}$$

In other words, predictions are unbiased. Here they are in R.

```
yp <- b0 + b1*xp
```

For variance, consider one of those two options at a time.

$$\begin{aligned}\mathbb{V}\text{ar}\{\hat{Y}(x)\} &= \mathbb{V}\text{ar}\{\hat{\beta}_0\} + \mathbb{V}\text{ar}\{\hat{\beta}_1\}x^2 + 2x\mathbb{C}\text{ov}(\hat{\beta}_0, \hat{\beta}_1) \\ &= \sigma^2_{b_0} + x^2\sigma^2_{b_1} + 2x\sigma_{b_{01}} \quad \text{where} \quad \sigma_{b_{01}} = -\sigma^2\left(\frac{\bar{x}}{(n-1)s_x^2}\right) \\ \text{defining} \quad \sigma^2_{\text{fit}}(x) &= \sigma^2\left(\frac{1}{n} + \frac{(x-\bar{x})^2}{(n-1)s_x^2}\right) \end{aligned} \tag{7.27}$$

The last term involves pulling $\sigma^2_{b_0}$ and $\sigma^2_{b_1}$ from Eqs. (7.25) and (7.23), but $\sigma_{b_{01}}$ is a bit of a doozy. You'll have to take my word for that one until we get to Chapter 13. We talked earlier about how slope is negatively correlated with intercept: move one up and the other must go down, etc. It's not impossible to derive, but it takes quite a bit of patience and would be a distraction here. (It's not a great homework problem either, so I'll spare you.)

Perhaps I can distract you by turning your attention to $\sigma^2_{\text{fit}}(x)$. Notice that as $n \to \infty$ we get a "perfect fit", as it were, out of our prediction of the mean. This is because we have the same property for estimated coefficients (7.23) and (7.25). For finite n there will always be some uncertainty.

Again, sums of Gaussians … and so we have $\hat{Y}(x) \sim \mathcal{N}(\mathbb{E}\{Y(x)\}, \sigma^2_{\text{fit}}(x))$, and making that practical involves substituting in an estimate for σ^2 and upgrading to t_{n-2}.

$$t_{\text{fit}}(x) = \frac{\hat{y}(x) - \mathbb{E}\{Y(x)\}}{s_{\text{fit}}(x)} \quad \text{with (squared) s.e.} \quad s^2_{\text{fit}}(x) = s^2\left(\frac{1}{n} + \frac{(x-\bar{x})^2}{(n-1)s_x^2}\right). \tag{7.28}$$

Note that $\hat{y}(x) = b_0 + b_1 x$ is the actual prediction you calculated with estimated coefficients ($b_0 = \hat{\beta}_0, b_1 = \hat{\beta}_1$), and stored in `yp`. Eq. (7.28) is most useful as a CI: $\hat{Y}(x) \pm q \times s^2_{\text{fit}}(x)$.

```
s2fit <- s2*(1/n + (xp - xbar)^2/((n - 1)*s2x))
sfit <- sqrt(s2fit)
low.CI.t <- yp + qt(0.025, n - 2)*sfit
high.CI.t <- yp + qt(0.975, n - 2)*sfit
```

I'll show you how this measures up to the CI in Figure 7.10 in a moment. First, I have to give you the equations behind the PI, which is shown on the same plot. The hard work for that is done already.

$$\begin{aligned} \mathbb{V}\text{ar}\{\tilde{Y}(x)\} &= \mathbb{V}\text{ar}\{\hat{Y}(x)\} + \mathbb{V}\text{ar}\{\varepsilon\} \\ \text{so} \quad \sigma^2_{\text{pred}}(x) &= \sigma^2_{\text{fit}}(x) + \sigma^2 \end{aligned} \tag{7.29}$$

Notice that $\sigma^2_{\text{pred}}(x) \to \sigma^2$ as $n \to \infty$. No matter how much data you have, you'll never be able to predict any better than the underlying population variance. As long as $n < \infty$ we have that $\sigma^2_{\text{pred}}(x) > \sigma^2$.

It may be a little pedantic, but for completeness ...

$$t_{\text{pred}}(x) = \frac{\tilde{y}(x) - \mathbb{E}\{Y(x)\}}{s_{\text{pred}}(x)} \quad \text{with (s.e.)}^2 \quad s^2_{\text{pred}}(x) = s^2\left(1 + \frac{1}{n} + \frac{(x-\bar{x})^2}{(n-1)s_x^2}\right). \tag{7.30}$$

Again, note $\tilde{y}(x) \equiv \hat{y}(x) = b_0 + b_1 x$ when you actually plug in estimated coefficients. In R ...

```
spred <- sqrt(s2fit + s2)
low.PI.t <- yp + qt(0.025, n - 2)*spred
high.PI.t <- yp + qt(0.975, n - 2)*spred
```

It's silly to make this plot, but check out Figure 7.11. Here the CI is given by Eq. (7.28) and the PI by (7.30). The lines in this plot are, at least to my eye, visually indistinguishable from MC-based versions from Figure 7.10,

```
plot(x, y, xlim=range(xp), ylim=range(c(low.PI.mc, high.PI.mc)))
lines(xp, yp, lwd=2)
lines(xp, low.CI.t, lwd=2, lty=2)
lines(xp, high.CI.t, lwd=2, lty=2)
lines(xp, low.PI.t, lwd=2, lty=2, col=2)
lines(xp, high.PI.t, lwd=2, lty=2, col=2)
legend("bottomright", c("mean", "95% CI", "95% PI"),
  col=c(1, 1, 2), lty=c(1, 2, 2), lwd=2, bty="n")
```

They are different though, a little. Below I calculate a measure of discrepancy called root mean squared error (RMSE)[38], one point at a time for each end of the interval, but separately for each CI and PI. Smaller RMSE is better, meaning the two things being compared – in this case, MC versus exact calculation – are more similar.

```
CI.rmse <- sqrt(mean((c(low.CI.t, high.CI.t) - c(low.CI.mc, high.CI.mc))^2))
PI.rmse <- sqrt(mean((c(low.PI.t, high.PI.t) - c(low.PI.mc, high.PI.mc))^2))
c(CI.rmse, PI.rmse)
```

```
## [1] 0.005647 0.105978
```

Notice that the agreement is more accurate for the CI than the PI. This makes sense because a MC PI calculation involves more random numbers, meaning more MC error (§A.3) for fixed effort, N.

In R you can make predictions using the `predict` command on the `lm` object `fit`. Depending

[38] https://en.wikipedia.org/wiki/Root_mean_square_deviation

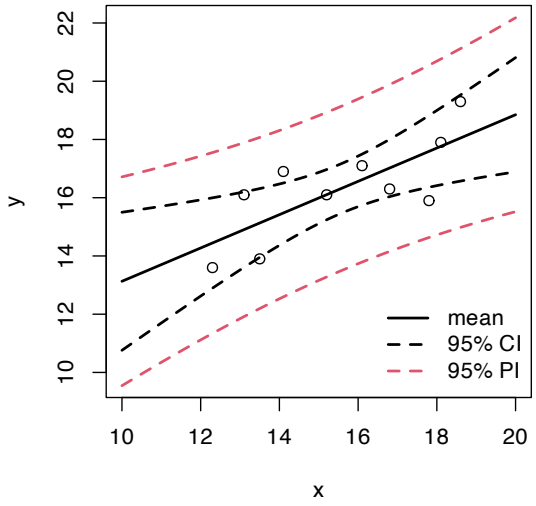

FIGURE 7.11: Full predictive distribution as means (of means) and error bars, via math.

on whether you want a CI or PI, you have to give a different `interval` argument. If you additionally want $s_{\text{fit}}(x)$, you must specify yet another argument. The quantity $s_{\text{pred}}(x)$ is not provided, but you can always follow Eq. (7.29), or with the code immediately below: `spred <- sqrt(s2fit + s2)`.

```
xp.df <- data.frame(x=xp)
pc <- predict(fit, newdata=xp.df, interval="confidence")
pp <- predict(fit, newdata=xp.df, interval="prediction")
sfit.lib <- predict(fit, newdata=xp.df, se.fit=TRUE)$se.fit
```

It's a little cumbersome to specify `xp`, since it must be a `data.frame` whose column names line up with names residing in the original `x` from your `fit`. The reason for this will become more clear in §13.3. If you plotted these, they'd be right on top of the ones we calculated by hand.

```
CI.rmse2 <- sqrt(mean((c(low.CI.t, high.CI.t) - c(pc[,2], pc[,3]))^2))
PI.rmse2 <- sqrt(mean((c(low.PI.t, high.PI.t) - c(pp[,2], pp[,3]))^2))
sfit.rmse <- sqrt(mean((sfit - sfit.lib)^2))
round(c(CI.rmse2, PI.rmse2, sfit.rmse), 10)
```

```
## [1] 0 0 0
```

All numerically zero. R's version matches our handiwork.

Well that's it. This chapter ends somewhat abruptly, but actually there's a lot more (as I already said) to regression, and also to correlation. First, in Chapter 12, we'll revisit both topics from a nonparametric perspective. What's that, you ask? It's coming up next. It'll blow your mind. I'll do some simpler nonparametrics before we get to correlation and regression. Then, at the end of the book, I'll circle back to parametric land with a regression reprise (ultimately MLR).

Regression is so important because it has so many applications, but also because it uses so many fundamentals all at once. Understand this stuff, and you've got it made. I almost can't wait. I'm so excited about the new skills you're developing!

7.8 Homework exercises

These exercises help gain experience with correlation and regression via tests and CIs. Whenever possible, I suggest doing the calculations both ways: via MC and with math. Questions that introduce data could be entertained as correlation or regression analysis. Verbiage below is suggestive of one or the other, but it would be good practice to try both if you feel it's appropriate. I don't want to set expectations too high, but there's some really fun ones toward the end.

Do these problems with your own code, or code from this book. I encourage you to check your work with library functions like `cor.test`*,* `lm`*,* `summary` *and* `predict`*. Check with your instructor first to see what's allowed. Unless stated explicitly otherwise, avoid use of libraries in your solutions.*

#1: Bivariate Gaussian MLE

Derive MLEs for unknown quantities $\theta = (\mu_y, \mu_x, \sigma_y^2, \sigma_x^2, \rho)$, but especially ρ, in the bivariate Gaussian model (7.5). Derive the asymptotic sampling distribution for ρ. *Hint: use Fisher information.*

#2: Library function `monty.cor`

Write a function called `monty.cor` that automates testing and CI calculation for ρ under a bivariate Gaussian model (7.5). Follow specifications similar to earlier `monty.*` implementations. Make sure you cover both MC and math-based procedures. When testing via math, use sensible defaults depending on whether the null is $\rho = 0$ or not. Your output should indicate which testing procedure was performed.

Hint: many of the following questions are easier after this one is squared away.

#3: Mother–daughter golf

A mother and daughter play golf together a handful of times over a summer at different courses near where they live. Here are their scores.

```
golf <- data.frame(
  mother=c(107, 108, 111, 102, 113, 109, 116, 109, 105, 115, 105, 114),
  daughter=c(102, 108, 95, 103, 99, 100, 98, 107, 104, 103, 106, 101))
```

Is there nonzero correlation between them? Additionally, provide a confidence interval for ρ.

#4: Blood pressure

The file bloodpres.csv[39], linked from the course webpage, records a patient's pulse (beats per minute) and their systolic and diastolic blood pressures (millimeters of mercury/mmHg). These data are originally from a Kaggle competition[40].

```
bloodpres <- read.csv("bloodpres.csv")
x <- bloodpres$pulse;    y <- bloodpres$systolic     ## pair 1
x <- bloodpres$pulse;    y <- bloodpres$diastolic    ## pair 2
x <- bloodpres$systolic; y <- bloodpres$diastolic    ## pair 3
```

Is there any correlation between any of these pairs of measurements?

#5: Crocodile size

Data below include measurements, rounded to the nearest centimeter, of the length of Indian crocodile bodies and heads (from the same crocodile). These data may be found, along with other measurements, on the Data and Story Library (DASL)[41], which is a wonderful resource.

```
croc <- data.frame(
  body=c(177, 264, 311, 382, 385, 475, 548, 343, 386, 338, 319, 224, 202,
    218, 183, 203, 209, 161, 179, 226, 259, 287, 300, 342, 333, 406, 459,
    376, 380, 265, 349, 264),
  head=c(35, 49, 56, 64, 61, 71, 83, 51, 62, 52, 52, 40, 38, 37, 32,
    36, 38, 24, 24, 32, 35, 40, 40, 46, 48, 52, 60, 51, 54, 38, 46, 38))
```

It has been claimed that the correlation is 90%. Do these data support or refute this claim?

#6: Non-Gaussian correlation

This one is asking you to think outside the box a little. Consider (x, y) pairs in the following `data.frame`.

```
nonG <- data.frame(
  x=c(0, 2, 1, 1, 1, 0, 1, 1, 2, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1,
   3, 1, 0, 0, 0, 0, 2, 4),
  y=c(2, 5, 4, 2, 5, 2, 5, 8, 4, 3, 1, 6, 5, 3, 4, 3, 5, 1, 5, 2, 1, 3,
   10, 2, 2, 1, 3, 3, 6, 10))
```

Is it reasonable to use a bivariate Gaussian model (7.5) for these data? If not, what model(s) might be better for the margins, i.e., x and y separately. What do you estimate for the parameters of those distributions? Develop a MC-based procedure that uses those margins to test $\rho = 0$, and provide a CI for ρ. Can you think of any downsides to your procedure?

[39] https://bobby.gramacy.com/hipp0/bloodpres.csv
[40] https://www.kaggle.com/datasets/spdsou/predict-systolic-and-diastolic-ratemedical-data?resource=download
[41] https://dasl.datadescription.com/datafile/crocodile-lengths/

TABLE 7.3: LM fit: $s = 0.676$ on 18 DoF, $r^2 = 0.31$

	coef	se	tstat	pval
icept		0.0433	0.069	
slope	0.1892	0.0657		

#7: Residual estimation

The following pair of prompts asks you to estimate coefficients b_0 and b_1 for data $(x_1, y_1), \ldots, (x_n, y_n)$ under $y_i = b_0 + b_1 x_i + e_i$, for $i = 1, \ldots, n$.

a. Fix a slope b_1. Suppose $\bar{e} = \frac{1}{n}\sum_{i=1}^{n} e_i = 0$ and solve for b_0.
b. Using b_0 you found above, suppose that $r_{ex} = 0$ and solve for b_1.

What do you think?

#8: Biased MLE $\hat{\sigma}^2$

Show $\mathbb{E}\{\hat{\sigma}^2\} = \frac{n-2}{n}\sigma^2$, with $\hat{\sigma}^2$ given in Eq. (7.18). You may use that $\hat{\beta}_0$ and $\hat{\beta}_1$ are unbiased for β_0 and β_1, and any other results from the chapter (besides the actual result in question). *Hint: start on the left-hand side. You may wish to look back at your solution to #5 from §3.5.*

#9: Forensic SLR inference

Here is the first of three forensic analyses where you get to explore how well you understand quantities involved in an SLR.

Table 7.3 summarizes an SLR for data $(x_1, y_1), \ldots, (x_n, y_n)$, but it is only partially completed. Use (some of) the information that is provided to fill in the information that's missing. What do you think: is x useful for predicting y?

#10: Forensic SLR prediction

This one is a little more challenging. Suppose we have the following summary information for a SLR: $n = 15$, $\bar{y} = 3$, $\bar{x} = 5$, $s_x^2 = 4$, $\hat{\beta}_1 = 3$ and $s_{b_1} = 5$. What would you predict for $Y(x)$ at a new input $x = 2$? Give a 95% interval around your prediction. (You can give a CI or PI, but you must say which you are doing.)

#11: Telemarking (more forensics)

Here's a variation with R output instead. Details of an SLR fit are provided below, via excerpts from the `summary` command in R, from data on number of calls `y` placed by a telemarketer who has been on the job `x` months, in the span of one hour. Some information in this output is deliberately obscured.

```
> tele1 <- lm(y ~ x)
> summary(tele1)

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 13.67077    1.42697   (???)    (???)
x            0.74351    0.06666   (???)    (???)
```

```
Residual standard error: 1.787 on 18 degrees of freedom
Multiple R-squared:  0.8736
```

Is seniority an important indicator of productivity? Put another way, do employees who have been on the job longer make more calls per hour? Provide a CI for the number of additional calls placed per additional month on the job.

#12: Library functions `monty.slr` and `predict.monty.slr`

Write a function called `monty.slr` that automates testing and CI calculation for coefficients β_0 and β_1 under the SLR model (7.14). Then make `predict.monty.slr` that uses the object output by `month.slr` to make predictions at new `xp` locations. Follow specifications similar to earlier `monty.*` implementations. Make sure you cover both MC and math-based procedures. You'll need two different MCs for each coefficient, but you can share one for confidence intervals. If you save all `B0s` and `B1s` from the CI calculation you can use these later in `predict.monty.slr`. Same for the math way; be sure to save enough to make predictions later.

Hint: many of the following questions are easier after this one is squared away. If you want to get really fancy, ask your professor (or the internet) to help you use S3 object-oriented features via the generic `predict` *method.*

#13: Pay and labor

Greenberg and Kosters (1970) studied 39 demographic groupings of 6000 households where the primary breadwinner made less than $15,000 annually in 1966. One aspect of their study involved estimating the relationship between pay and labor supply, towards influencing social policy decisions and the debate on a guaranteed national wage. Here we shall use their data, provided in wages.csv[42] on the course webpage, to entertain a linear relationship summarizing the effect of pay on labor supply for the working poor.

```
wages <- read.csv("wages.csv")
y <- wages$HRS                    ## number of hours worked/year
x <- wages$RATE                   ## pay per hour ($ in the 1960s)
```

Answer the following questions based on your SLR fit.

a. Does pay rate have an association with number of hours worked?
b. For every extra dollar-per-hour, how many more hours/year are worked? Provide an estimate and a 95% CI.
c. For someone making $2/hr in the 1960s, provide a 95% PI summarizing the number of hours the fitted model says they'd work.
d. Based on your fit, and in particular your estimated intercept $\hat{\beta}_0$, and what you understand about the nature of labor, are there any pitfalls to this analysis?

#14: Tractor maintenance

The `data.frame` below contains ages (in years) along with six-monthly maintenance costs (dollars) for seventeen tractors.

[42] http://bobby.gramacy.com/hipp0/wages.csv

```
tractor <- data.frame(
  age=c(4.5, 2.5, 2.5, 4, 4, 4, 5, 5, 5.5, 5, 0.5, 0.5, 6, 6, 1, 1, 1),
  cost=c(619, 1049, 1033, 495, 723, 681, 890, 1522, 987, 1194, 163, 182,
   764, 1373, 978, 466, 549))
```

Based on an SLR fit with `cost` as the response and `age` as the predictor, answer the following.

a. What proportion of variability is explained by the fit?
b. Suppose you were considering buying a tractor that's 4 years old. What would you expect your six-monthly maintenance costs to be? Provide uncertainties along with your estimates and explain what they mean.
c. Provide a visual of your fit, and overlay a prediction with CI and PI for tractors ranging from brand new to 10 years old.

#15: Rent

The data file rent.csv[43], linked from the book webpage, summarizes records of rental properties advertised around Chicagoland. Here, we will be primarily interested in how monthly rent (in thousands of dollars) relates to size (in thousands of square feet) for smaller units.

```
rent <- read.csv("rent.csv")
rent <- rent[rent$SqFt < 20, ]
y <- rent$Rent
x <- rent$SqFt
```

Based on an SLR fit, answer the following.

a. What proportion of variability is explained by the fit?
b. Is square-feet a useful predictor of rent?
c. How much more does an additional 1,000 square feet cost? Provide a CI.
d. Provide a visual of your fit, and overlay a prediction with CI and PI for dwellings ranging from zero to 20,000 square feet.

#16: Capital asset pricing model (CAPM)

One of the most important applications of linear regression in finance involves the so-called capital asset pricing model (CAPM)[44]. CAPM says, based on some theory, that asset returns (like from stocks) are linearly related to market returns (like from the Dow Jones Industrial Average/DJIA).

What is a return? It's a normalized difference between prices, e.g., how much it costs to buy a share of the stock. Let p_t denote the price at time t, say this month, and p_{t-1} be the price last month. Then, monthly returns may be calculated as follows.

$$\text{return} \quad r_t = \frac{p_t - p_{t-1}}{p_{t-1}}$$

Here it is in R.

[43] https://bobby.gramacy.com/hipp0/rent.csv
[44] https://en.wikipedia.org/wiki/Capital_asset_pricing_model

```
returns <- function(p)
  {
    p <- as.numeric(p)
    r <- (p[-1] - p[-length(p)])/p[-length(p)]
    return(r[-length(r)])
  }
```

Suppose we downloaded some prices from the internet. I've chosen Google, representing the asset, and an index tracking the DJIA.

```
library(quantmod)
getSymbols(c("GOOG", "DIA"), periodicity="monthly")
```

This defines variables called `GOOG` and `DIA` in your current environment. Each is a `data.frame` with lots of information about prices in the span of many years (rows). You can adjust many aspects of what's downloaded if you want, but I'm trying to keep it simple. Code below uses the `returns` function above to calculate returns based on one of the columns of that `data.frame`.

```
r <- data.frame(DIA=returns(DIA$DIA.Adjusted),
  GOOG=returns(GOOG$GOOG.Adjusted))
```

Treat `r$DIA` as x and `r$GOOG` as y and answer the following.

a. Do they look linearly related? (Is CAPM reasonable? If so, then this means that the essense of the relationship between hundreds of pairs of measurements can be captured by just two coefficients, plus s^2 measuring variability. That's pretty incredible!)
b. Fit a model and interpret the estimated coefficients. In CAPM people talk about an asset's α and β where $\alpha \equiv \beta_0$ is the intercept and $\beta \equiv \beta_1$.

Here's some context for part #b, above. If $\alpha \neq 0$, then investing in the asset in question is either consistently better ($\alpha > 0$) or worse ($\alpha < 0$) than the average (i.e., the market as whole). If $\beta = 1$, then the asset's returns move about the same amount as the market. If the market goes up, the asset goes up by a similar amount. The same goes with down. But if $\beta \neq 1$ then the asset either amplifies the market ($\beta > 0$) or dampens it ($\beta < 0$). What do you think?

See if you can find some asset (i.e., besides "`GOOG`") where the outcome is notably distinct from this analysis.

8

Bootstrap and permutation

This chapter serves as a warm-up for the next several to come. I want to take you on a little journey before circling back to linear regression (previous chapter) for the grand finale at the end of the book. Broadly, the topic is nonparametric statistics (non-P). I'll say more about what that is in a moment. This is an unconventional topic for an introductory stats textbook. Sometimes you get a dabble here and there, but I'm giving you five chapters worth starting here. I didn't encounter non-P until I was in graduate school. Even then I didn't fully appreciate what was going on. It wasn't until I taught a class on non-P at VT in 2016, more 10 years after finishing my PhD, that I developed an appreciation.

Larry Wasserman[1], an eminent Carnegie Mellon[2] statistician, and my academic grandfather (my PhD advisor's advisor), famously wrote a book called *All of Statistics*[3] (Wasserman, 2004). It's 450 pages long and doesn't have any non-P in it. Don't worry, he wrote another book called, you guessed it, *All of Nonparametric Statistics*[4] (Wasserman, 2006). Both books are available to download for free at those links. They're targeted to graduate students, and are highly technical. I'm not saying you should read them, at least not yet. They help me make a fun anecdote. By the way, Larry's advisor was Rob Tibshirani[5]. Rob is the most cited author of articles in "data science" according to Google Scholar[6]. He has the second-most cited author beat by more than 100,000 citations at the time of writing. That's my academic great-great grandpa! So proud. (We've never met, but I have been to dinner with his (biological) son, Ryan. Aren't you glad you bought my book?)

But I digress. The glossing over of non-P in undergraduate curricula is a shame because it's a beautiful subject. That's only part of the reason it features so prominently here. The main reason is computational. Presentation of non-P methods can be obscured by difficult math and asymptotics which – and this is my opinion – undermine their uptake and use in practice. For researchers working on non-P, the goal was to develop statistical methods that could be better trusted because they involved fewer untestable or unrealistic assumptions.

If you're ever an expert witness involved in a statistical analysis in a courtroom and you use a paired t-test, say, be prepared to defend your choice of Gaussian distributions for your two populations. With non-P you're in the clear on that. (More soon.) In spite of that enhanced robustness to scrutiny, there's a haze of smoke around non-P because people don't understand what's going on behind the scenes. It looks like voodoo, and I want to fix that.

Cheap computation is a game-changer. As with material earlier in the book, short algorithms coded in a modern language add a degree of transparency. You can "see" what's happening in the code. Calculations can be highly accurate with big enough N. No $n \to \infty$ asymptotics.

[1] https://en.wikipedia.org/wiki/Larry_A._Wasserman
[2] https://en.wikipedia.org/wiki/Carnegie_Mellon_University
[3] https://egrcc.github.io/docs/math/all-of-statistics.pdf
[4] https://www.stat.cmu.edu/~brian/valerie/617-2022/0%20-%20books/2006%20-%20Wasserman%20All%20Of%20Nonparametric%20Statistics.pdf
[5] https://en.wikipedia.org/wiki/Robert_Tibshirani
[6] https://scholar.google.com/citations?view_op=search_authors&hl=en&mauthors=label:data_science

Finally, and especially for me and for this book – and maybe this is the same as my second reason – non-P methods showcase the power of thinking through a statistical analysis from first principles via virtualization.

In this chapter I cover two somewhat more modern non-P methods, the bootstrap (Efron, 1979) and permutation tests (Dwass, 1957). These are, I think, the easiest to understand and so they make for a good warm-up. The bootstrap is particularly powerful, and is perhaps one of the most important computer-based methods in statistics in the last 100-years. So it's perfect for Monty. And it's just so simple, which is what makes it beautiful. Bootstrap resampling and permutation tests are quite generic in their approach to inference, and this is a double-edged sword. In later chapters I'll cover some classical non-P procedures, ones more tailored to particular testing and inference scenarios, lending statistical power (§A.2).

8.1 Non-P

I've always thought that nonparametric statistics[7] is a bit of a misnomer. If you read the definition on that Wikipedia page, it doesn't mention parameters. How can you say what something is not without addressing what's not not? How can you explain what it means to be disgruntled without addressing what it means to be gruntled?

The main thing that's "non" about nonparametric statistics is the lack of probability models for the data-generating mechanism. In other words, there are no distributions assumed for the populations that samples come from. For example, no $Y_1, \ldots, Y_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$. Since we're not using Gaussians in this way, or Bernoulli, Poisson, exponential, or otherwise, then we also don't need their parameters. I think this is where "nonparametric" comes from, but as you can see it's buried a bit.

I'm going to say "non-P" going forward. That way you'll think both about the lack of parameters, and the lack of probability distributions for the populations. Look how there's three p's (not) there: parameter, probability, and population. This doesn't mean there aren't populations at all, nor does it mean we don't use probabilities or estimate quantities that could easily be called parameters. However, we don't use those things all together at once to make assumptions about the data-generating mechanism.

Non-P methods still use probabilistic reasoning, like expectation, independence, identically distributed, or the notion of a distribution. But they just don't commit to a specific distribution dialed in by a small handful of p parameters θ to describe sampled quantities. When p is just one or two, as it's for many common distributions, that's a heavy burden. All n observations, which could be in the hundreds, are boiled down to two numbers. This is powerful when it works, but is the real world ever that simple?

Instead non-P does a delicate dance around assumptions. One great way to characterize non-P is that the observational data *is* the distribution. Either it defines an empirical distribution directly (§A.1), like bootstrap resampling and permutation tests of this chapter, or it is defined through some transformation of the data, often their ranks. I don't want to get too ahead of myself though. I'll address ranks in Chapter 9.

In this way, non-P methods are more flexible than parametric ones (or just "P") because they have n degrees of freedom (DoF) instead of $p \ll n$. Since we don't need to estimate

[7] https://en.wikipedia.org/wiki/Nonparametric_statistics

any of p parameters, we don't lose any DoF for downstream inference. On the other hand though, estimating parameters simplifies things. Distilling n quantities down to p provides neat summaries that are easy to keep track of. Ordinary P methods have many closed-form solutions, whereas many non-P ones don't or must rely on approximation.

In fact, we've already seen something like that: the central limit theorem (CLT) from §4.1. Recall that no distribution for the population was assumed. However, samples must be iid and have common mean and variance. No parameters, no tidy distribution, but yes to probabilistic ideas like iid, expectation, etc. Means and variances are labeled μ and σ^2 for convenience, but they're not really parameters in the same sense because no distributional form is assumed. Previously, however, our use of the CLT was always in a P context. After committing to $Y_i \overset{\text{iid}}{\sim} \text{Pois}(\theta)$, say, that nailed down $\mu = \sigma^2 = \theta$, leading to substantial simplification.

The real problem with the CLT is that it only helps if you want to estimate, and understand the sampling distribution of, a sum or average. Although many statistics can be characterized as sums or averages, some cannot like the median. I'll say more about medians momentarily. Even when the statistic is essentially a sum, like s^2 estimating variance (3.6), it's often non-trivial to deduce from that the distribution of more readily interpretable variations, like those based on standard deviation. I say "the real problem" but, in fact, many non-P methods use the CLT as a subroutine in some way, but again hold that thought. It's time for an example.

Jackknife filet-o-fish

Consider the data values below, which come from a survey of measurements of the length of fish, in inches, caught during a competition hosted in the Chesapeake Bay.

```
y <- c(9.0, 46.2, 9.0, 13.8, 11.5, 18.3, 15.2, 21.0, 11.2, 30.9, 7.1,
  15.2, 18.1, 25.3, 42.6, 32.8, 11.9, 14.3, 8.3, 11.0, 10.5, 10.7,
  36.4, 11.6, 21.7, 26.5, 25.6, 18.8, 55.7, 24.8, 18.5)
n <- length(y)
n
```

```
## [1] 31
```

My father-in-law, who fishes in the Chesapeake and likes to tell me about stuff, says they're some real "whoppers" in there. Two are nearly 5 feet in length, which is enormous. Most are about a foot in size, and none are smaller than six inches, perhaps due to the equipment being used. We'll visualize the sample momentarily, but I bet you can tell already that the distribution is highly skewed.

Now consider some estimates from the sample that might be of interest to fish-and-game personnel.

```
ymed <- median(y)
s <- sd(y)
c(ymed, s)
```

```
## [1] 18.10 12.06
```

As statisticians, we usually want to know more than mere point estimates. What is the sampling distribution for the median? Can a CI be provided? What is the sampling distribution

for the sample standard deviation S? I'm sure you can think of others, but this is plenty for now. How can we answer those questions without assuming some distributional form?

Quick digression on medians, since this is the first time I'm working with them statistically. The notation used for the median of a sample is $y^{(n/2)}$, or the $n/2^{\text{nd}}$ order statistic[8]. The median is the value that separates the top "half" of values from the bottom, when sorted $y^{(1)}, y^{(2)}, \dots, y^{(n/2)}, \dots, y^{(n-1)}, y^{(n)}$. The min $y^{(1)}$ and max $y^{(n)}$ are other important order statistics.

If you have a vector of y-values in R, like `y`, you can find the i^{th} largest by sorting the vector and returning the i^{th} in order. Note that wouldn't be the most efficient way, computationally speaking. For more on that, check out Quickselect[9]. The median is technically the $(n+1)/2^{\text{nd}}$ one if n is odd, or the average of the $n/2$ and $n/2+1$ sorted values when n is even.

```
c(median(y), sort(y)[(n + 1)/2])
```

```
## [1] 18.1 18.1
```

It's customary to keep it simple and write $y^{(n/2)}$ regardless. The random analog of the median is $Y^{(n/2)}$, or similar for any other order statistic. A strange thing about the median, or any other order statistic, is that it's one of the values (or an average of two) in a sample, not somehow an aggregate of all of the values like many other statistics.

Alright, back to the task at hand. Check this out. Let $y_{(-i)}$ denote the data set without y_i. In other words,

$$y_{(-i)} = \{y_1, \dots, y_n\} \setminus \{y_i\},$$

so that $y_{(-i)}$ has $n-1$ observations. There are n unique such leave-one-out data sets, for $i = 1, \dots, n$.

Suppose each $y_{(-i)}$ is used to estimate any quantities of interest, like $y_{(-i)}^{(n/2)}$ and $s_{(-i)}$ for median and standard deviation, respectively. There would be n of each of those. Now treat the whole collection of estimates, over $i = 1, \dots, n$, as a sample from an empirical distribution for any estimators of interest. Each represents a quantity that you would have estimated with a different data set sampled from the population – one with one fewer data point, observed n different ways.

I know it sounds nuts, but we just boarded the crazy train. You're in for a helluva ride. This is called the jackknife[10], actually an apt name. John Tukey[11] coined the term by observing that, like an actual jackknife (a folding, pocket knife), it's a simple tool that can be used to improvise rough-and-ready solutions on the spot, often more efficiently than requisitioning a purpose-built solution. The jackknife is similar to leave-one-out cross-validation (CV)[12], where the word "fold", like a folding knife, is used to describe each sub-data set made from holding out elements of the original data. I just mention that in case you've come across CV elsewhere. CV is very popular in machine learning, and the jackknife is its mommy.

Here is the jackknife in code for medians and standard deviations.

[8] https://en.wikipedia.org/wiki/Order_statistic
[9] https://en.wikipedia.org/wiki/Quickselect
[10] https://en.wikipedia.org/wiki/Jackknife_resampling
[11] https://en.wikipedia.org/wiki/John_Tukey
[12] https://en.wikipedia.org/wiki/Cross-validation_(statistics)

```
Ymeds.jk <- Ss.jk <- rep(NA, n)
for(i in 1:n) {
  ## so easy with y[-i], removing the ith element of y
  Ymeds.jk[i] <- median(y[-i])
  Ss.jk[i] <- sd(y[-i])
}
```

Now consider each estimator in turn. We have $n = 31$ sample medians, $y_{(1)}^{(n/2)}, \ldots, y_{(n)}^{(n/2)}$. One option is to use those directly, and build a CI from their empirical quantiles. This is usually my go-to, but with only 31 samples, it's hard to be precise about 2.5% and 97.5%. The closest you can get is 3.2% and 93.5%. Another option is to use the samples to calculate moments (means and variances) first, and then deploy a CLT-like approximation with those moments to deduce Gaussian quantiles. Both are approximations which improve with n. I've decided to go the Gaussian approximation route here.

```
q <- qnorm(0.975)
CI.med.jk <- mean(Ymeds.jk) + c(-1, 1)*q*sd(Ymeds.jk)
CI.med.jk
```

```
## [1] 15.87 18.94
```

Figure 8.1 offers a visual of these quantities alongside a scatter of the original sample for perspective. The interval is plotted as red brackets. Both the original median (`ymed`, filled red diamond), and the average of jackknife sampled medians (open red diamond) are shown.

```
plot(y, rep(0, length(y)), ylim=c(-0.5, 2.25),
  xlab="fish length (in)", ylab="", yaxt="n", bty="n")
points(mean(Ymeds.jk), 0, pch=5, col=2, cex=1.5)
points(ymed, 0, pch=18, col=2, cex=1.5)
text(CI.med.jk, rep(0, 2), c("[", "]"), col=2, cex=1.5, pos=3, offset=0)
legend("topright", c("raw median", "avg median"), col=2, pch=c(18, 5),
  pt.cex=1.5, bty="n")
legend("topleft", "[ ] median fishlen CI", text.col=2, bty="n")
```

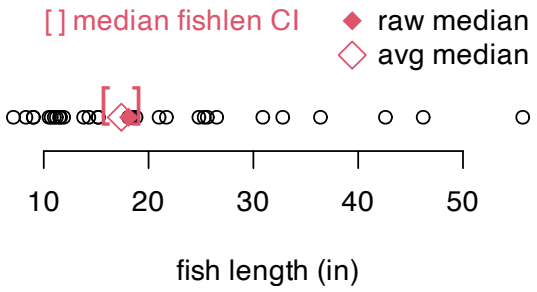

FIGURE 8.1: Jackknife median (open red diamond) and 95% CI (brackets) for the fish length data (open black circles). The raw median on the original data is indicated as a filled red diamond.

We have nothing to compare this to, but the visual is fairly compelling. For my taste, the

interval is a bit on the narrow side. This is just based on instinct, and some awareness about jackknife's limitations. There are operational similarities between the jackknife and MC from earlier chapters. One big difference, however, is that the jackknife is not stochastic. That means it doesn't involve randomness. Instead of generating synthetic data from a known distribution with fixed parameterization (via the null), the data itself (with one observation removed) is used. Not very many such "simulations" are entertained; only $n \equiv N$. Consequently, the level of diversity of values, compared to MC, is low.

```
table(Ymeds.jk)
```

```
## Ymeds.jk
## 16.65 16.75  18.2
##    15     1    15
```

Having three unique values is better than one, which is what you get without a jackknife. I'll leave it to you to convince yourself that this isn't just bad luck. If you only remove one point, there can be at most three unique median values in $y_{(-i)}$ over all i for any $n > 4$.

Now for standard deviation; similar options are available, and I'll stick with a Gaussian approximation.

```
CI.s.jk <- mean(Ss.jk) + c(-1, 1)*q*sd(Ss.jk)
CI.s.jk
```

```
## [1] 11.29 12.82
```

Again, there's little (yet) to compare this to. I trust this calculation a bit more since it is based on 30 unique values, as opposed to three in the case of the median.

```
length(unique(Ss.jk))
```

```
## [1] 29
```

Ultimately, I'm setting up the jackknife as a straw man[13]. It embodies the spirit of non-P, but has downsides. You don't get sufficient "good looks" at a diversity of (sub-) samples with just n examples, or even fewer in the case of the median. A jackknife is a handy tool to have in your pocket, but it's not a Sawzall[14].

As I said earlier, many non-P methods, but not all, lean on the CLT – or something like it – in some fashion. Those that don't involve some pretty fancy mathematics, landing technical details well outside of the scope of this book. Computation with MC is easy by contrast. There will be many instances where I simply quote a mathy result, with a quick pointer to its origin or intuition, and then we get on with drawing a comparison to MC provided in full.

Many applications of non-P involve small data sets – small n. In those settings, it can be hard to characterize how far an asymptotic calculation is from its exact ideal. I much prefer MC in such cases, but you know that already. The jackknife is an extreme example

[13] https://en.wikipedia.org/wiki/Straw_man
[14] https://en.wikipedia.org/wiki/Reciprocating_saw

of this. Its performance is tightly coupled to n, and there's no MC alternative. This makes it somewhat limiting, except as a warm-up.

When the CLT is not involved, one would refer to tables (often published in the backs of textbooks) to obtain percentiles and quantiles for p-value and confidence interval (CI) calculations in a non-P setting. Producing such tables required hefty computation in their own right. These endeavors comprised statisticians' early forays into supercomputing. (Think 1960s, vacuum tubes, punch cards, and mainframes occupying entire buildings, and hundreds of hours of compute time and energy.) For some of those, there are now library-based automations, but not for all. I'll show you a few, but mostly focus on how to do-it-yourself (DIY) with MC and a few handful of lines of R or Python, all in a matter of seconds.

Bootstrap resampling and permutation tests are not only a good warm-up. These are *omnibus* non-P methods that can be adapted to many contexts. The bootstrap is a direct update to the jackknife, as its creator Efron (1979) explains in the title to that paper: "Bootstrap Methods: Another Look at the Jackknife". Only n-many "samples" are not many – not really an empirical sampling distribution. We're used to getting $N \geq 10{,}000$ via MC. The bootstrap fixes this.

I think of the bootstrap as a tool for summarizing uncertainty in a way that's perfect for calculating CIs. Don't forget you can always invert a CI to set up a test. Permutation tests are more specific, targeting tests of differences in populations, like a non-P alternative to a two-sample t-test (§5.2). In subsequent chapters, we'll get even more specific in order to target specialized hypotheses. Enhanced specificity means improved statistical power (§A.2). The more you tailor your assumptions and testing apparatus to the question of interest, the more discerning you can be, statistically speaking.

8.2 Bootstrap

In describing the jackknife above, I've all but told you what the bootstrap[15] is. Here it is in one sentence: Instead of deleting each data element, forming n data sets, sample with replacement and have (almost) as many as you want. It'll help to be a little more specific. Also, in what follows, I'm going to start from scratch so you needn't have read about the jackknife, although I shall revisit the fish example to draw a comparison.

Choose a statistic $s_n = s(y_1, \ldots, y_n)$ of interest, like median or standard deviation. In what follows, I'll drop the n subscript on s_n and simply refer to $s(\cdot)$. I want to use subscripts for something else, and the formalism for arbitary statistics is clumsy anyways. A particular hazard here is estimated standard deviation, which also uses the letter $s = \sqrt{s^2}$, following Eq. (3.6). I'll try to make any distinctions clear when necessary, but mostly it should be automatic from context.

Now, instead of calculating the statistic(s) directly on the original data $y \equiv (y_1, \ldots, y_n)$, resample those data values $Y^\star = \text{resample}(y)$ with replacement and then calculate the statistic(s) on that, virtual data $S = s(Y^\star)$. Another way to write that is

$$Y^\star \overset{\text{iid}}{\sim} \text{Unif}\{y_1, \ldots, y_n\} \quad \rightarrow \quad S = s(Y^\star).$$

This is even easier in code. Here is an example of what I mean with the median statistic.

[15] https://en.wikipedia.org/wiki/Bootstrapping_(statistics)

```
Ystar <- sample(y, n, replace=TRUE)    ## always take n = length(y) samples
Ymed <- median(Ystar)                  ## this is s(Ystar)
c(Ymed, ymed)
```

```
## [1] 15.2 18.1
```

Notice how the calculation turned out different than when using the original data directly. Why is that? Because resampled $Y^\star$ is different from the original data vector y. In particular, $Y^\star$ has some entries missing,

```
setdiff(y, Ystar)
```

```
##  [1] 18.3 21.0 18.1 32.8 11.9 11.0 11.6 26.5 18.8 24.8 18.5
```

leading to 18 unique values, rather than $n = 31$. Since the sampled version $Y^\star$ still has n entries, then it must have some of the original data values represented more than once.

```
table(round(Ystar, 2))
```

```
##
##  7.1  8.3    9 10.5 10.7 11.2 11.5 13.8 14.3 15.2 21.7 25.3 25.6 30.9
##    1    1    2    3    1    2    3    1    1    2    2    3    1    3
## 36.4 42.6 46.2 55.7
##    1    1    2    1
```

See that some are even in triplicate. It'll be yet again different when you try it on your own machine as `sample` is random. In drawing this so-called *bootstrap sample* $Y^\star$, we implicitly treat the original data $y_1, \ldots, y_n$ as if it were the entire population. Doing that over and over again is not unlike MC from earlier chapters. Rather than using an assumed distribution under a null hypothesis, we now treat the observed data as the distribution. It's a finite, discrete distribution with n, not necessarily unique, values.

The "with replacement" caveat is crucial. Resampling n items from n items *without* replacement is the same as randomly permuting them, without otherwise altering its composition. This changes nothing in practical terms if we're working under an iid assumption, where order doesn't matter. Technically, we're making an exchangeability[16] assumption which is weaker than iid, but that distinction isn't really important here. (I just say these things so my nerdy colleagues don't complain.) Take-home message: don't forget `replace=TRUE`.

Figure 8.2 lays it all out in a diagram. I contemplated using an algorithm environment, but ultimately thought a schematic would look more slick. The total number of bootstrap samples is B. This is like N for MCs in earlier chapters. I could have used N here, too, but I didn't for a few reasons. One is that you see B a lot in the literature. Besides always being quick to jump on bandwagons, I think it's helpful to make a distinction between ordinary MC, and a bootstrap MC. Recall that all MC means is that you're using random sampling in your calculation. Using B reminds you it's sampling observed data with replacement, instead of generating from a distributional family under $\mathcal{H}_0$.

Finally, B can't be as big as you want, like N can. The largest reasonable B is linked to n. Consider the simple but silly case of $n = 2$. It's easy to count that there are only three

[16] https://en.wikipedia.org/wiki/Exchangeable_random_variables

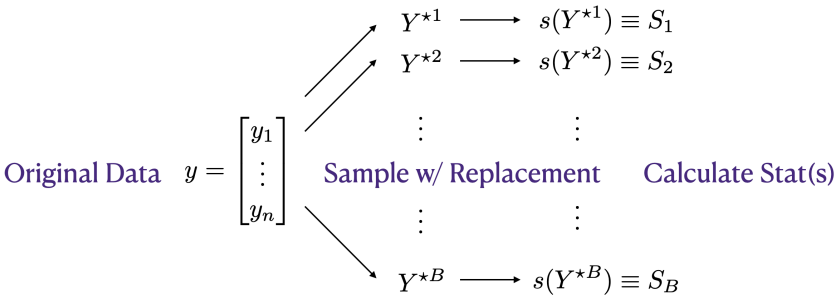

FIGURE 8.2: Bootstrap resampling diagram, yielding samples $S_1, \ldots, S_B$ comprising an empirical distribution for $s(Y) \equiv s(Y_1, \ldots, Y_n)$.

unique possible $Y^\star$s up to permutation: $(y_1, y_1), (y_2, y_2), (y_1, y_2)$. It doesn't make a whole lot of sense to ask for $B = 10{,}000$ in that case. But three bootstrap samples is already more than the jackknife which would only have two, each being of size just one. Obviously $n = 2$ is an extreme example. I've left counting the number of unique bootstrap samples, in greater generality, to you as an exercise in §8.5. Spoiler alert: it's really big even for smallish n.

It's worth remarking that Figure 8.2 applies to the jackknife as well. Just take $B = n$ and don't sample with replacement, but take $Y^{i\star} = y_{(-i)}$. Since a jackknife implementation is just a `for` loop, so is the bootstrap. Easy code, but the result is surprisingly rich, at least compared to a jackknife. The something-for-nearly-nothing aspect is what led Brad Efron[17] to call it the bootstrap.

Efron, like Tukey, worked at the interface between statistics and machine learning, so he was better at naming things than most classical statisticians. "Bootstrap" combines two uses of the term from elsewhere. One, in the vernacular but also common in the business world, refers to the impossibility of "pulling yourself up by your bootstraps". The other is common in technical usage, like in computer engineering, where a self-sustaining process is started with very little external input. A bootloader[18], short for bootstrap loader, is a program that starts up, or "boots" your computer. In the case of bootstrap resampling, and indeed of the jackknife, that "very little external input" is $y_1, \ldots, y_n$. Nothing else except an iid assumption, and not even that in all cases, is required. At some point it helps to choose a statistic, $s(\cdot)$, but even that can be delayed 'till later. Let me show you.

Fish length via bootstrap

Here's the bootstrapping loop. Notice how I'm saving each sample, and not calculating a statistic. This is the "Sample w/ Replacement" step in Figure 8.2.

```
B <- 10000
Ystar <- matrix(NA, nrow=B, ncol=n)
for(b in 1:B) {
  Ystar[b,] <- sample(y, n, replace=TRUE)
}
```

[17] https://en.wikipedia.org/wiki/Bradley_Efron
[18] https://en.wikipedia.org/wiki/Bootloader

Those samples may be used to calculate a distribution for whatever statistic I want. This is the "Calculate Stat(s)" step in the figure. Let's begin with the median.

```
Ymeds <- apply(Ystar, 1, median)
```

Since the median is one of $n = 31$ values from the original data, it's worth checking how many unique ones, and with what frequency, ended up in the bootstrap collection.

```
table(Ymeds)
```

```
## Ymeds
## 10.7   11 11.2 11.5 11.6 11.9 13.8 14.3 15.2 18.1 18.3 18.5 18.8   21
##    1    2   18   37  109  302  531  786 2453 1468 1341 1154  816  516
## 21.7 24.8 25.3 25.6 26.5
##  304  109   39   12    2
```

This is many more than the paltry three that we got with a jackknife. This distribution is depicted in Figure 8.3. Still, these 19 values are not $B = 10^4$ values. Using quantiles directly for CI calculation could be fraught with inaccuracy. So again I opt for a Gaussian approximation.

```
CI.med <- mean(Ymeds) + c(-1, 1)*q*sd(Ymeds)
```

That interval is overlayed as purple brackets on the histogram in Figure 8.3. To reiterate, that histogram is not the original sample, but rather bootstrap sample of medians, `Ymeds`. However, the range of the x-axis is the same as in Figure 8.1 in order to keep perspective. Compared to the jackknife, the bootstrap interval is much wider, owing to a greater diversity of samples. Just on instinctual grounds, I like this new interval a lot better.

```
hist(Ymeds, xlim=range(y), main="", xlab="fish median length")
points(mean(Ymeds), 0, pch=5, col="purple", cex=1.5)
points(ymed, 0, pch=18, col=2, cex=1.5)
text(CI.med.jk, rep(0, 2), c("[", "]"), col=2, cex=2, pos=3, offset=0)
text(CI.med, rep(0, 2), c("[", "]"), col="purple", cex=2, pos=3, offset=0)
legend("topright", c("raw median", "boot avg median"),
  col=c("red", "purple"), pch=c(18, 5), pt.cex=1.5, bty="n")
legend("right", "[ ] new boot CI", text.col="purple", bty="n")
```

Original median $y^{(n/2)}$ is plotted as a red diamond. This is the same as in Figure 8.1, along with those red jackknife brackets. The purple circle, however, is calculated as an aggregate of bootstrap samples. Bootstrap aggregating is sometimes referred to as bagging[19] and, again owing to the diversity of samples from the empirical distribution of the original data, lends a degree of robustness to any point estimate.

I'd like to spend a brief moment on testing. Suppose some pundit asserted that the median catch size for competitive anglers fishing in the Chesapeake was 20 inches. We would not be able to reject that hippopotamus with these data because 20 is well within the 95% interval.

[19] https://en.wikipedia.org/wiki/Bootstrap_aggregating

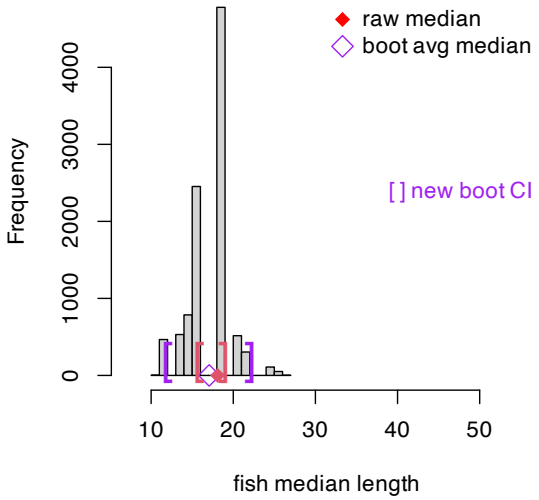

FIGURE 8.3: Bootstrap sampling distribution of medians (purple), compared to jackknife (red).

And if someone quibbled that they don't trust our modeling assumptions, we can chirp back that our analysis was non-P. All we assumed is that the data are iid.

Now we move on to standard deviation. This is easy with bootstrap resamples already in hand. Storing all of those samples could be memory-intensive if B and n are both big, but that pays off in compute time (and in code) if you have more than one statistic in mind.

```
Ss <- apply(Ystar, 1, sd)
```

Since estimated standard deviation, s, is derived from a squared average of all observations in the data – or in this case in a bootstrap resample – we obtain many more unique values.

```
length(unique(Ss))
```

```
## [1] 9992
```

Observe that this is close to, but less than, $B = 10^4$, the total number of samples. It must have been the case, on

```
B - length(unique(Ss))
```

```
## [1] 8
```

occasions, that one bootstrap resample $Y^{\star i}$ was identical to another $Y^{\star j}$ up to permutation. That can happen. It's even possible to get $Y^{\star i} \equiv y$, up to a reshuffling, in which case $S_i = s(y_1, \ldots, y_n)$. A homework exercise explores this further.

Since there are so many unique values, there's little harm in calculating a CI via quantiles, as opposed to using the CLT.

```
CI.s <- quantile(Ss, c(0.025, 0.975))
```

See Figure 8.4, which additionally shows the full bootstrap sampling distribution of estimated standard deviation S. Again, this new bootstrap interval is much wider than the jackknife analog.

```
hist(Ss, main="", xlab="fish length sd")
points(mean(Ss), 0, col="purple", pch=5, cex=1.5)
points(s, 0, pch=18, col="purple", cex=1.5)
text(CI.s.jk, rep(0, 2), c("[", "]"), col=2, cex=2, pos=3, offset=0)
text(CI.s, rep(0, 2), c("[", "]"), col="purple", cex=2, pos=3, offset=0)
legend("topleft", c("raw median", "boot avg sd"),
  col=c("red", "purple"), pch=c(18, 5), pt.cex=1.5, bty="n")
legend("topright", "[ ] new boot CI", text.col="purple", bty="n")
```

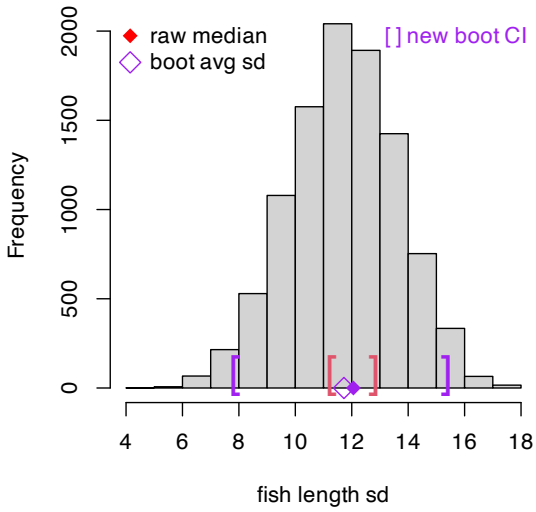

FIGURE 8.4: Bootstrap sampling distribution of standard deviations (purple), compared to jackknife (red).

I encourage you to contrast those quantile-based intervals with a CLT version. I'd guess they'd be quite similar.

Return on investment

Recall the return on investment (ROI) example from §2.5. Consider ROI for individual companies who use the particular IT product first. So as not to write over `y` and `n` – I want to re-use those later – I'll use `yc` and `yi`. Industry values, comprising `yi` will come later.

```
load("roi.RData")
yc <- roi$company
```

Recall from the histogram of these quantities in Figure 2.3 that the distribution appears

heavy-tailed. Consequently, an analysis based on Gaussians in the homework exercise of §2.6, if you did that one, and later in §5.4, is potentially flawed.

Suppose there is interest in performing an analysis based on the sampling distribution of averages. I've coded that up below. This time I'm not saving each individual bootstrap resample, just the sample average statistic. You get to decide what you want to do as a software developer (or book author).

```
Ycbars <- rep(NA, B)
for(b in 1:B) {
  Ycbars[b] <- mean(sample(yc, length(yc), replace=TRUE))
}
CI.c <- quantile(Ycbars, c(0.025, 0.975))
CI.c
```

```
##   2.5%  97.5%
##  7.071 18.104
```

Figure 8.5 provides a visual of this CI, with a histogram summarizing the original data overlayed. According to this view, individual company ROI is positive. We would reject the hypothesis that ROI is zero, because zero falls well outside the CI.

```
hist(yc, main="", xlab="ROI")
text(CI.c, rep(0, 2), c("[", "]"), col=2, cex=2, pos=3, offset=0)
legend("left", "mean ROI CI", text.col=2, bty="n")
```

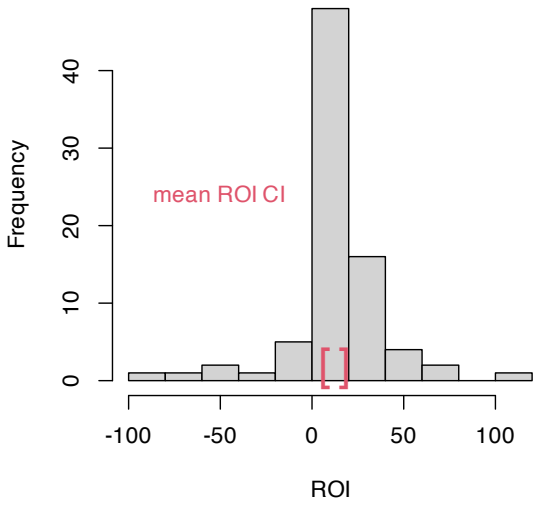

FIGURE 8.5: Company average ROI CI via bootstrap resampling, with original data histogram overlayed for reference.

In the homework question of §2.6 you were asked to test if $\mu = 15$, representing the industry average. We would not be able to reject that either. But that's not the right way to conduct a two-sample analysis. There's uncertainty in both estimates. One option is another bootstrap.

```
yi <- roi$industry
Yibars <- rep(NA, B)
for(b in 1:B) {
  Yibars[b] <- mean(sample(yi, length(yi), replace=TRUE))
}
```

Next, form the distribution of their difference, just like in a two-sample t-test from §5.2, but without assuming Gaussianity.

```
Ybds <- Yibars - Ycbars                          ## bd is short for "bar diff"
CI.diff <- quantile(Ybds, c(0.025, 0.975))
CI.diff
```

```
##   2.5%  97.5%
## -3.536  8.260
```

Zero is in that interval, so we wouldn't be able to reject that the two samples have the same mean. Evidence is mounting that the "professional" analysis behind the negative ad campaign was suspect. From every which angle you look, it seems like companies that use that IT product have the same mean ROI as does the industry at large.

You might have noticed, in both examples above, that I avoided explicit p-value labeling and nomenclature. You certainly *can* be probabilistically precise about the visual assessments in the figure above. For example, here's the probability that that average ROI is bigger than zero.

```
mean(Ycbars < 0)
```

```
## [1] 0
```

That's pretty strong evidence. You could multiply that by 2 to make it "two-sided"; same thing with the two-sample version based on differences.

```
2*mean(Ybds < 0)
```

```
## [1] 0.4356
```

But neither is strictly a p-value, by definition, because we didn't sample under a null hypothesis. Rather, we used that CI as the inverse of a hypothesis test. Some may quibble with this. I'm fine with it, but as you can tell, I'm a bit cavalier about these things – about most things, except hard work and no excuses. Those are important. You could argue that, implicitly, our $\mathcal{H}_0$ was that the two populations were independent. But again, this is the opposite of the typical setup. A classical null would specify that the two hypotheses are the same, and seek to reject that in favor of evidence otherwise.

One might conclude that bootstraps are limited in this sense, if all tests must invert CIs. Perhaps. That's not a completely accurate portrayal, but it would take us into a rabbit hole to correct. I'd rather focus on how its most basic form is flexible in other, possibly even more useful ways. You can do so much with simple `for` loop and `sample`, basically without modification, in so many settings. For example, you can bootstrap ordinary least squares (OLS)

from §7.4, and conduct inference for linear models without placing a Gaussian assumption on responses. Just treat each training data pair as a single unit, $z_i = (x_i, y_i)$, bootstrap resample over $z_1, \ldots, z_n$, save estimates of slope and intercept and make histograms. You can do it with predictions too. I've left these to you to explore in your homework.

Strangely enough, there's also something called the parametric Bootstrap[20], and other ways to use similar resampling ideas toward other ends. This comes in handy when you're ok with P, but you find resampling more convenient than the pure, null hypothesis MC. There are good reasons to proceed in this fashion, but I'll leave it to you to track those down. Some pointers are provided in §8.4.

8.3 Permutation

A permutation test[21] specifically targets two-sample testing. Whereas a bootstrap can be appropriated to test two samples, it can be somewhat underpowered (§A.2), as nothing about it is targeted to that application specifically. It's just two independent bootstraps stitched together. Whereas it's hard (but not impossible) to place a bootstrap analysis within a conventional hypothesis testing framework, this is rather more natural for the permutation test.

It goes like this. Suppose $Y_1, \ldots, Y_{n_y} \overset{\text{iid}}{\sim} F_Y$ for some generic, unspecified distribution F_Y, and $X_1, \ldots, X_{n_x} \overset{\text{iid}}{\sim} F_X$ similarly.

$$\begin{aligned} &\mathcal{H}_0 : F_Y = F_X && \text{null hypothesis (“what if”)} \\ &\mathcal{H}_1 : F_Y \neq F_X && \text{alternative hypothesis} \end{aligned} \tag{8.1}$$

In some sense this setup is ridiculous, since it's overly generic. Every sample is a little different than another one for some reason or another. The real question is: what differences can be detected with limited data? As with any statistical test, that hinges on your choice of statistic, and how you study its sampling distribution. That's your wedge.

With permutation tests, it's common to develop a statistic that can be applied to the combined sample

$$s(y, x) = s(y_1, \ldots, y_{n_y}, x_1, \ldots, x_{n_x}), \tag{8.2}$$

but, ultimately makes contrasts between y and x vectors, of length n_y and n_x respectively, through simpler statistics applied separately to the two groups of data. For example,

$$s(y, x) = \bar{y} - \bar{x}, \quad \text{sd}(y) - \text{sd}(x) \quad \text{or} \quad \text{sd}(y)/\text{sd}(x). \tag{8.3}$$

It may look as if I've created some overly cumbersome notation for a simple thing. It wouldn't be the first time. The important thing about Eq. (8.2) is that although the first n_y are labeled as y_i and the latter n_x as x_i, when we actually use $s(\cdot, \cdot)$ the x's and y's will be jumbled up. You'll see. In that case, say for $s(y, x) = \bar{y} - \bar{x}$, what happens is that the

[20] https://en.wikipedia.org/wiki/Bootstrapping_(statistics)#Types_of_bootstrap_scheme
[21] https://en.wikipedia.org/wiki/Permutation_test

first n_y values, no matter what they are, shall be averaged and likewise the last n_x. Then their difference is returned.

Why jumble things up? Because the null hypothesis (8.1) says that the distributions of Y and X are the same ($F_Y = F_X$). So if we were to shuffle the full deck of data before feeding into $s(\cdot, \cdot)$, e.g., obtaining $s(x_{12}, x_7, y_3, x_8, y_9, y_2, \dots)$, in effect treating some y's as x's and vice-versa when calculating $s(y, x)$, that shouldn't change anything *on average*, because X and Y have the same distribution. Or do they? That's what the permutation test helps us determine.

Let $n = n_y + n_x$ be the size of the combined sample. Consider *all* $n!$ shufflings, or permutations, of the combined data and calculate $s(\cdot, \cdot)$ for each one, providing $s_1, \dots, s_{n!}$. This is your sampling distribution for the statistics, which is a bit of a misnomer since the procedure (iterating over all permutations) is deterministic. While the statistics will indeed distribute themselves, meaning they will be spread out and you can compare that spread to the observed/unpermuted one, the distribution is not stochastic.

This is all, of course, nonsense since you can't iterate over $n!$ anything for n of any reasonable size. When $n = 60$, combining two size thirty samples, say, you get $n! > 10^{81}$ which is near in number to what many scientists estimate to be the total count of all atoms in the universe. Good luck enumerating all of those, and storing a statistic for each!

Instead, most applications of permutation tests sample at random from available permutations, which is the same as saying sample without replacement from the combined n-sized collection. Choose a number of permutations P as large as computation or practical limitations allow (like N or B), and calculate $S_1, \dots, S_P$. These are random, and have a probabilistic distribution, as long as each permutation is sampled at random.

In other words, it's like the bootstrap as diagrammed in Figure 8.2, but (a) with two samples pasted together; (b) use `replace=FALSE`; and (c) use a statistic $s(\cdot)$ that targets contrasts between the two groups (8.3). It's bizarre that the opening line to the Wikipedia page reads "A permutation test … is an exact statistical hypothesis test."[22] It's not in most applications. You can't believe everything you read on the internet. Also like with bootstrap resampling, some of the sampled statistics may be identical to others if they involve the same first n_y and last n_x values.

More fish

Suppose that the fish-length survey was repeated on another day, and lengths were recorded below.

```
x <- c(23.4, 18.1, 37.8, 32.5, 26.1, 22.0, 15.0, 32.2, 37.5, 19.2, 15.9,
  31.4, 13.0, 14.8, 24.1, 21.6, 34.6, 28.0, 27.1, 25.2, 26.1, 37.6,
  33.1, 14.7, 25.0)
```

Let's first test by averages, which may be calculated as follows. I find it helpful to combine the two samples, straight away in order to calculate the test statistic. That way I can test things out before using a similar calculation later in the random permutation loop.

```
nx <- length(x)
ny <- length(y)     ## formerly n, but now n is for combined sample
```

[22] Maybe that'll change before this book goes to print.

```
yx <- c(y, x)
n <- length(yx)                                  ## same as nx + ny
bdiff <- mean(yx[1:ny]) - mean(yx[(ny + 1):n])   ## this is s(.)
c(bdiff, mean(y) - mean(x))
```

```
## [1] -5.005 -5.005
```

I chose the variable name `bdiff`, short for "bar difference", or difference in $\bar{y}$ and $\bar{x}$. Writing the statistic somewhat generically for the combined sample, and checking that it works as desired, is helpful for the next step.

```
P <- 10000
Bds <- rep(NA, P)
for(p in 1:P) {
  YXs <- sample(yx, n)                           ## rand permutation
  Bds[p] <- mean(YXs[1:ny]) - mean(YXs[(ny + 1):n])  ## s(YXs)
}
```

Figure 8.6 shows the distribution of sampled statistics along with the observed one, and it's reflection, overlayed. This is a nice looking histogram, and one that seems to favor $\mathcal{H}_0$. A p-value calculation adds precision.

```
pval.bd <- 2*mean(Bds <= bdiff)
pval.bd
```

```
## [1] 0.0822
```

```
hist(Bds, main="", xlab="differences in means", xlim=c(-11, 13))
abline(v=bdiff, col=2, lwd=2)
abline(v=-bdiff, col=2, lty=2, lwd=2)
legend("topright", c("obs", "reflect"), lty=1:2, lwd=2, col=2, bty="n")
```

Although the $P = 10^4$ samples is but a small fraction of $n! > 10^{75}$, I'm reasonably confident that the calculation is accurate. Indeed, a substantial proportion (around 79%) of those sampled permutations led to identical statistics, because they amounted to reshuffled y's to the first n_y locations and x's to the last n_x. Any doubts can be put at ease with larger P.

```
(P - length(unique(Bds)))/P
```

```
## [1] 0.7873
```

So we weren't able to reject that the distributions were the same by targeting differences in averages. How about differences in estimated standard deviation? You could try either of the options in Eq. (8.3). One advantage to going with a difference, as opposed to a ratio, stems from familiarity with two-tailed testing. Ratios of positive statistics are always positive; recall that we preferred one-tailed tests of two sample variances in §6.2.

As with the bootstrap, you could potentially save the random permutations and re-use

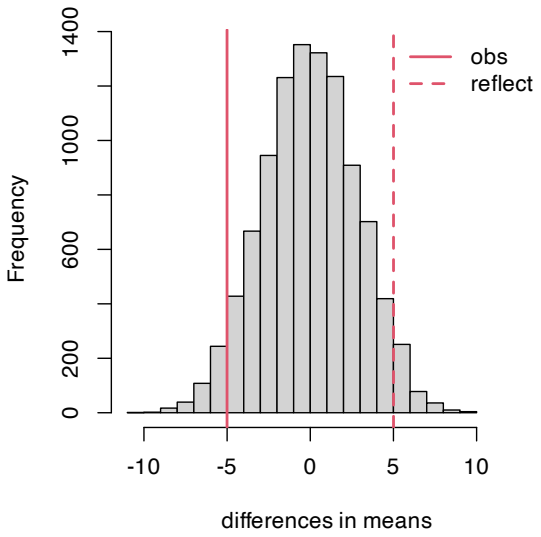

FIGURE 8.6: Permutation distribution for differences in mean fish length.

them with another statistic. The fancy word for that is caching[23]. Alas, I didn't think ahead. Cut-and-paste it is …

```
sdiff <- sd(yx[1:ny]) - sd(yx[(ny + 1):n])
Sds <- rep(NA, P)
for(p in 1:P) {
  YXs <- sample(yx, n)
  Sds[p] <- sd(YXs[1:ny]) - sd(YXs[(ny + 1):n])
}
```

Figure 8.7 shows the sampling distribution of differences in estimated standard deviation obtained from sampled permutations. This is a closer call than many recent examples. The p-value says …

```
pval.sd <- 2*mean(Sds >= sdiff)
pval.sd
```

```
## [1] 0.0336
```

… reject $\mathcal{H}_0$ at the 5% level, and determine that the samples likely came from a different distribution ($F_Y \neq F_X$) because they differ in their standard deviation.

```
hist(Sds, main="", xlab="differences in stdevs", xlim=c(-7,10))
abline(v=sdiff, col=2, lwd=2)
abline(v=-sdiff, col=2, lty=2, lwd=2)
legend("topright", c("obs", "reflect"), lty=1:2, lwd=2, col=2, bty="n")
```

A word of caution though: if you keep testing the same hypotheses, over and over again

[23] https://en.wikipedia.org/wiki/Cache_(computing)

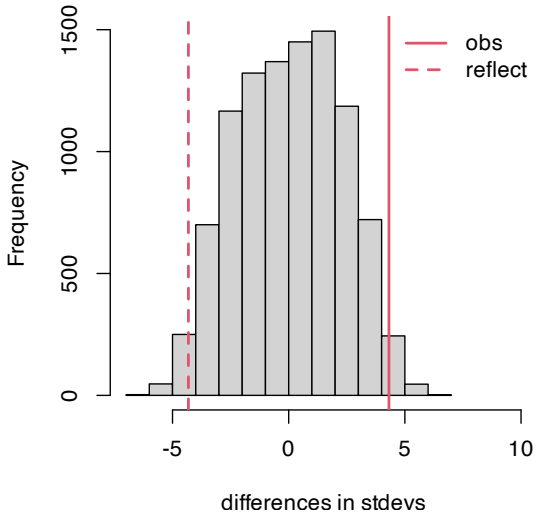

FIGURE 8.7: Permutation distribution for differences in standard deviation of fish lengths.

with different statistics, you're bound to be snagged by a multiple testing hazard. Recall our discussion in §6.4. Twenty borderline tests at the 5% level yield one false positive on average. We ran the test (only) twice targeting very different aspects, so I'm not that concerned about it. But if you keep hunting around for something, you'll probably find it. In that sense, every null hypothesis is doomed.

More ROI

Now consider a similar pair of tests for ROI. First, set up the combined sample and calculate observed test statistics.

```
yx <- c(roi$company, roi$industry)
ny <- length(roi$company)
nx <- length(roi$industry)
n <- ny + nx
bdiff <- mean(yx[1:ny]) - mean(yx[(ny + 1):n])
sdiff <- sd(yx[1:ny]) - sd(yx[(ny + 1):n])
```

This time, I shall combine average- and s-sampling into a single `for` loop.

```
Bds <- Sds <- rep(NA, P)
for(p in 1:P) {
  YXs <- sample(yx, n)
  Sds[p] <- sd(YXs[1:ny]) - sd(YXs[(ny + 1):n])
  Bds[p] <- mean(YXs[1:ny]) - mean(YXs[(ny + 1):n])
}
```

I'll skip the histogram visuals and jump straight to p-values. Be sure to get those inequalities the right way around. Don't be "that guy" who reports a p-value bigger than one.

```
pvals <- c(avg=2*mean(Bds <= bdiff), sd=2*mean(Sds >= sdiff))
pvals
```

```
##    avg     sd
## 0.6256 0.0000
```

Careful here: zero doesn't really mean zero. It means $\psi < 1/P = 10^{-4}$, which is pretty small. This result is consistent with earlier analysis. Figure 2.3 clearly indicates distributions with different variances, and this makes sense considering that numbers in `roi$industry` are aggregates. So they clearly come from different distributions $F_Y \neq F_X$. Those differences are not in location, but rather in scale.

8.4 Wrapping up

I normally mention software along with the method in order to make a direct comparison, but here I'm doing it at the end of the chapter. One reason is that, at least for bootstrap (and jackknife) and permutation tests, I'm not a big fan of the library options. It's not that the libraries aren't good, but rather the nature of the beast means that they offer lower value compared to DIY MC. These methods are just simple `for` loops with re-sampling alongside simple statistical summaries. That's pretty easy. By the time you read all the documentation for the library, you could have DIY'd.

Take `boot` (Canty and Ripley, 2024) built into R, for example. The main function, also called `boot`, is clearly intended for more ambitious applications than the introductory ones I presented in this chapter. It seems straightforward enough to provide `boot(data, statistic, replicates)`, for example as `boot(y, mean, B)` using our setup for ROI from §8.2. But that doesn't work because the second, `statistic` argument must itself take two arguments. The contents of those arguments depend on other `boot` specifications, etc., determining exactly how resampling is to be deployed. There are many good reasons to do things a little differently than I've described in this chapter, but that's for another course. There are whole books on this stuff (Davison and Hinkley, 1997).

I eventually figured out that I needed the following if using the default of `stype="i"`, which utilizes indices in the observed data for resampling.

```
avgstat <- function(data, index)
  {
    mean(data[index])
  }
```

The reason for this is complicated, but ultimately sensible. No doubt the authors landed on this setup after much deliberation. Rather than resampling data values directly, the `boot` function samples indices from 1 to n, like `I <- sample(1:n, n, replace=TRUE)`, and then applies those indices to the data, like `y[I]`. This is a little more flexible, allowing your data to be a `matrix` or `data.frame` where you may wish to bootstrap over rows. That can be handy if you're bootstrapping OLS, like in the homework exercises. The data object can even be more complex, like a `list`. But for our simple setup, this extra scaffolding represents an additional layer of complexity.

```
library(boot)
bout <- boot(roi$company, avgstat, B)
CI.boot <- rbind(byhand=CI.c,
  lib=quantile(bout$t, c(0.025, 0.975)))
CI.boot
```

```
##         2.5% 97.5%
## byhand 7.071 18.10
## lib    7.059 18.16
```

That's pretty much the same result, but possibly more work. One really nice thing about `boot` is that it automates parallelization, over multiple cores on your machine. That can be important with really big B and/or with a statistic that's computationally demanding to compute. There are some other nice features which come in handy as you entertain other forms of bootstrap, like weight-based, parametric, residual bootstrap, and so on. I won't go over those. Davison and Hinkley (1997) is an excellent reference, and you can find others in the `boot` documentation (`?boot`). The package also provides an implementation of the jackknife.

Permutation tests are provided by the `perm` library (Fay and Shaw, 2010), but it's somewhat limited in options. As far as I can tell, only a test based on differences in averages is provided, at least as pertains to our modeling setup.

```
library(perm)
pval.perm <- permTS(roi$company, roi$industry)$p.value
pvals <- c(pvals[1], lib.avg=pval.perm)
pvals
```

```
##     avg lib.avg
##  0.6256  0.6210
```

That's basically the same result. It wouldn't be hard to write your own `monty.perm` library that worked more like `boot`, allowing for specification of an arbitrary statistic as an argument to the function in order to automate aspects of permutation tests. Speaking of good homework problems …

8.5 Homework exercises

These exercises help gain experience with bootstrap and permutation-based procedures. Although some questions are mathematical in nature, when it comes to application, the procedures are purely numerical. Some problems refer to questions performed via parametric procedures in earlier chapters. These are just a selection. Pretty much any of the earlier questions could be entertained with non-P. Likewise, many questions targeting non-P procedures could be entertained with P instead, so long as you choose an appropriate distribution for modeling.

Do these problems with your own code, or code from this book. Unless stated explicitly otherwise, avoid use of libraries in your solutions.

#1: Counting resamples

Let your training data comprise of n unique values, $y_1, \ldots, y_n$. That means no duplicates. Answer the following questions about resampling which are relevant for the bootstrap in an iid modeling situation.

a. How many distinct resamples of size n are possible? Since we are in an iid modeling context, two samples are not distinct if one is a permutation of the other.
b. What is the probability that a random resample matches a permutation of the original data?
c. Among the distinct bootstrap samples from #a, what is the probability of observing one with at least one repeated value?

#2: Library function `monty.boot`

Write a function called `monty.boot` that automates bootstrap tests and intervals with an arbitrary statistic. Take inspiration from `boot`, as described in §8.4 above, but yours will be much simpler. The function prototype should look like `monty.boot(y, stat, B, s0=NULL)` with sensible defaults like `stat=mean`, assuming `y` is a vector, and `B=10000`. A user should additionally be invited to provide some other function for `stat` that could be calculated for `y`. Return a CI for the statistic and calculate an a p-value, via inversion, if the user optionally provides a `s0` to compare to.

As a bonus, embed your new `monty.boot` within `monty.t` from §3.5 and `monty.var` from §6.5, providing the option of a non-P one-sample CI and test for means and variances, respectively. In this application it's fine to hard-code `stat=mean` or `stat=var` as appropriate. *Don't just cut-and-paste code. Call your new function.*

Hint: many of the following questions are easier after this one is squared away.

#3: Filling potato sacks

A sack of Idaho potatoes typically weighs 110 lbs. A new potato-sack filling machine was purchased and tested on twelve bags. Their weights, in lbs, are provided below.

```
y <- c(110.6, 109.9, 110.8, 111.0, 110.7, 110.8, 111.1, 110.8, 111.0,
  110.8, 110.9, 109.5)
```

Use bootstrap resampling to answer the following.

a. Does the average fill weight agree with the nominal specification of 110 lbs?
b. Provide a 95% CI for the average fill weight.

#4: Revisit heights

Revisit the heights data from Chapter 2, whose analysis was based on a Gaussian distribution. In the exercises of §2.6, you were asked to entertain a Gamma distribution. Now use the bootstrap in order perform a non-P test. Are women in my class of average height? Additionally provide a CI for their heights.

#5: Revisit TV lifespans

Question #2 from §3.5 involved data on times-to-failure of a certain new smart TV. Use a bootstrap to provide a CI for the lower quartile of failure times, in years. *Lower quartile*

is sometimes notated as $y^{(n/4)}$. It separates the bottom 25% from the top 75%, in the same way that the median separates the bottom half from the top half.

#6: Revisit re-insurer claims

Question #3 from §3.5 involved data on insurance claims. You were asked to use a quirky distribution I called the scale-1 Pareto. Repeat parts #c and #d from that exercise with non-P via the bootstrap.

#7: Revisit rent OLS inference

Question #15 from §7.8 considered data on rent as predicted by size. You may find the data in rent.csv[24], linked from the book webpage.

```
rent <- read.csv("rent.csv")
rent <- rent[rent$SqFt < 20, ]
y <- rent$Rent
x <- rent$SqFt
```

Revisit parts #a–#c via bootstrapped OLS. In particular, provide sampling distributions for the proportion of variability explained by the fit, and any regression coefficients that allow you to say if square-feet is a useful predictor. Finally, provide a CI for the additional cost per 1,000 square feet.

I suggest going back to the formulas for b_0 and b_1, but you could opt to use `coef(lm(...))`. Note, however, you do not need to use any of the standard errors, etc., that `lm` objects provide. The idea of this question is that you're not making a Gaussian assumption, because a bootstrap is non-P. If you're assuming a Gaussian, you can't use any of the results that are based on it. Finally, it will help to use the `sample` function to generate indices so that you can resample (x_i, y_i) pairs together, as a single unit. Optionally, you could figure out how to use `boot` to automate things, but actually I think this is harder.

#8: Revisit rent OLS prediction

This question carries on from #7 above, completing question #15d from §7.8 via a non-P bootstrap. Use your bootstrap samples to overlay errorbars derived from a CI on prediction.

Hint: each bootstrap sample gives you a line, and so the collection of all B lines can be used to form that CI, tracing out errorbars in the scatterplot. Taking the additional uncertainty into account via a PI is harder, because that involves an additional distribution assumption. I'm not suggesting you do that here.

#9: Library function `monty.perm`

Write a function called `monty.perm` that automates permutation tests with an arbitrary statistic. The function prototype should look like `monty.perm(y, x, stat, P)`. Use sensible defaults like `stat=mean` with `y` and `x` vectors, and `P=10000`. Your user should additionally be able to provide some other, possibly custom-built, function for `stat` that could be calculated for `y` and `x`. Automate the formation of the combined `yx`, and its randomly permuted `YXs`, saving the `stat` as differences over `P` iterations. Return an appropriate p-value.

As a bonus, embed `monty.perm` within `monty.t` (§5.4) and `monty.var` (§6.5), providing the

[24] https://bobby.gramacy.com/hipp0/rent.csv

option of a non-P two-sample test. In this application it's fine to hard-code `stat=mean` or `stat=var` as appropriate. *Don't just cut-and-paste code. Call your new function.*

Hint: many of the following questions are easier after this one is squared away.

#10: Cloud seeding

Simpson et al. (1975) report on an experiment that was conducted to determine if seeding clouds with silver iodine increases rainfall. Fifty-two clouds were seeded in this way, and fifty-two other (but otherwise similar) clouds were not seeded as controls, and the amount of rain in acre-feet was saved. These data may be found on DASL[25], or as clouds.csv[26] on the book webpage.

```
clouds <- read.csv("clouds.csv")
```

a. Treat the rows of the data as paired and conduct a non-P test via the bootstrap to determine if there is a difference in average (i.) or median (ii.) rainfall for seeded and unseeded (control) clouds.
b. Now treat the rows as unpaired and conduct a permutation test with the same statistics in part #a.

What do you conclude?

#11: Revisit heights again

Revisit the two-sample heights experiment from §5.2 which combined women's heights from my class (Chapter 2) and new ones from the VT women's basketball team. Perform a non-P permutation test to determine if these women's heights come from the same distribution.

#12: Memory-enhancing supplements

Solomon et al. (2002) report on a study where elderly adults were randomly assigned to take Gingko[27], via an over-the-counter supplement, and others to take a placebo. After four weeks the subjects had their memory evaluated by several objective neuropsychological tests and subjective ratings, which were summarized into a single score. Those scores are provided in memory.RData[28] linked from the book webpage and may additionally be found on DASL[29].

```
load("memory.RData")    ## defines a list called "memory" in your env
```

Use a permutation test, with any statistics you decide, in order to determine if Ginkgo supplements have any effect on memory.

[25] https://dasl.datadescription.com/datafile/cloud-seeding/
[26] https://bobby.gramacy.com/hipp0/clouds.csv
[27] https://en.wikipedia.org/wiki/Ginkgo_biloba
[28] https://bobby.gramacy.com/hipp0/memory.RData
[29] https://dasl.datadescription.com/datafile/memory/

9

Non-P location

This chapter is the non-P version of Chapters 2 and 5, combining one-sample and two-sample (unpaired and paired) location tests, respectively. But I'm going to do them in reverse, because in this presentation the one-sample version is a special case of the two-sample one.

All three tests derive from a paper by Wilcoxon (1945), although there are some other players involved as I'll explain. Frank Wilcoxon[1] was a chemist by training and trade, who discovered statistics through a study of Fisher's papers in the course of his own research for the chemical companies that he worked for. Wilcoxon's main insight, statistically, stemmed from a study of the distribution of ranks, which is what now underpins much non-P analysis.

Converting raw data into ranks represents clever insight, in my opinion, and really opened things up for me as I was learning about non-P. There are some beautiful mathematics in there, but we'll only scratch the surface on that. As usual, MC with ranks is super simple, in part because ranks are permutations of positive integers – something we already know lots about from the previous chapter.

Sometimes, the procedures presented in this chapter are referred to as Wilcoxon t-tests. I don't like that. Although Wilcoxon's procedures target the same setting as parametric t-tests, for one and two samples (and paired), there's no t statistic, nor t_ν distribution. Sometimes statistical texts generically use the letter t for "test statistic". I think that's confusing.

I want to acknowledge a book by Conover (1999), which is where I first learned the material here, and in the next few chapters, while teaching a non-P class at Virginia Tech (VT). That book was a great resource, but also one that feels antique when set in a contemporary (computing) landscape. The tables of quantiles and percentiles in the back of the book are now obsolete given `q*` and `p*` functions in R. In fact, it was a desire to modernize that material that led, eventually, to the idea of my Monty. First, I figured out how to avoid the tables (the "math" way), and then I figured out how to do the MC, which I now prefer.

While writing these passages, and on scale-based tests in Chapter 11, I stumbled across an R package called `ANSM5` (Spencer, 2024). That package supports a book (Smeeton et al., 2025) which, in turn, was also a new discovery for me. The spirit of that book, and its accompanying software, is quite similar to my Monty. They even do MC. I'll admit that at first that bummed me out. Their book is more broad on non-P. Mine focuses a bit more on statistical foundations.

I realized in particular that much of the software in that package could be used as a benchmark, especially for routines that are not available in base R. I have not done that here, because I already had library-based alternatives in place. `ANSM5` will come in handy later in Chapter 11. I wanted to give them a shout now because they deserve it, and in case the reference is helpful to you.

[1] https://en.wikipedia.org/wiki/Frank_Wilcoxon

9.1 Ranks

Many classical non-P procedures involve ranks[2]. Ranks are intimately related to order statistics, which were first introduced as a digression in §8.1. In a way, ranks are the inverse of an order statistic. The rank of a member $y_i \in \{y_1, \ldots, y_n\}$ is its index (i) in sorted order $y^{(1)}, \ldots, y^{(n)}$.

Ranks are easier to describe with a formula if there are no duplicated values in $\{y_1, \ldots, y_n\}$, i.e., no ties in the ranks. Let's consider that case first. The rank $r_{n,i}$ of y_i is

$$r_{n,i} = \sum_{j=1}^{n} 1_{\{y_j \leq y_i\}} \quad \text{for unique} \quad \{y_1, \ldots, y_n\} \tag{9.1}$$

where $1_{\{\cdot\}}$ returns 1 when the expression in $\{\cdot\}$ is true and 0 otherwise. So the rank of a data point is the count of the number of other data points it is greater than, plus itself. Subscript n in $r_{n,i}$ is intended to indicate a relationship to the other $y_1, \ldots, y_n$ values. Often I simplify to $r_i \equiv r_{i,n}$ when that connection is explicit from context.

If you found that description cumbersome, here are some examples in R to help. Suppose we had some data values, like those below. I just made them up. Again, no ties for now.

```
y <- c(73, 14, 6, 9, 85, 3, 12, 37, 36, 23, 11)
```

What is the rank of $y_4 = 9$? The formula says …

```
sum(y <= y[4])
```

```
## [1] 3
```

Or, by inspection, we can see that 9 is the third largest because 6 and 3 are the only ones that are smaller.

Eq. (9.1) would be a cumbersome way to calculate all n ranks. A better algorithm might go something like this. First sort the data, but rather than return the sorted values just return the sorted order. This is what the `order` function does in R. (These aren't the ranks, yet. Just the first step.)

```
o <- order(y)
o
```

```
##  [1]  6  3  4 11  7  2 10  9  8  1  5
```

In other words, to sort `y`, take index 6 first, then 3, and so on. Checking …

```
y[o]
```

```
##  [1]  3  6  9 11 12 14 23 36 37 73 85
```

[2] https://en.wikipedia.org/wiki/Ranking_(statistics)

Increasing – perfect. According to `o`, take index 4 in position 3. And so that's the rank $r_4 = 3$ of y_4. Said another way, a rank tallies when a particular data value would be taken in sorted order. So a rank is the order of the order.

```
r <- order(o)
rbind(oo=r, r=rank(y))
```

```
##    [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
## oo   10    6    2    3   11    1    5    9    8     7     4
## r    10    6    2    3   11    1    5    9    8     7     4
```

If you'd like, you can check any of those via Eq. (9.1) above. Now we have all the ranks for all n elements. See what I mean by ranks being the inverse of an order statistic? Note that an order statistic isn't the same as the `order` function, though they're related. R's `rank` automates `order` of the `order`. That's what we'll use going forward, but this wasn't waste because now you know how it works!

The main reason to use `rank(y)` rather than `order(order(y))` is how the former automates handling ties. The `rank` function has several options for this, and we're going to use the default which is `ties.method="average"`.

It goes like this: take the `order` of the `order`, as above. Call these the OO-ranks. Any ties will receive consecutive ranks by the OO-method. That's not good, because it would arbitrarily favor otherwise equivalent measurements with lower or higher ranks, depending on the number of ties. So, replace the OO-ranks of those points (the ones with ties) with their average OO-rank.

Here's a new y vector to experiment with, augmenting the other one with a few ties.

```
y <- c(y, y[2], y[7], y[7])       ## adding some ties
```

First find OO-ranks.

```
oor <- order(order(y))
```

Then, for the duplicated ones, replace with an average.

```
dups <- unique(y[duplicated(y)])
r <- oor
for(d in dups) {
  inds <- which(y == d)
  r[inds] <- mean(r[inds])
}
```

I'm not sure this implementation is the most efficient, but it's fairly transparent. Fortunately, the library provides a nice automation, and we can check that the two agree.

```
rbind(r=r, lib=rank(y))
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
## r      13  8.5    2    3   14    1    6   12   11    10     4   8.5
## lib    13  8.5    2    3   14    1    6   12   11    10     4   8.5
##      [,13] [,14]
## r        6     6
## lib      6     6
```

Rank r_2 has a 1/2 fractional part because y_2 is duplicated, causing two consecutive ranks to be averaged. Rank r_7 is not fractional, but rather shares the rank 6 as the average of three consecutive whole numbers: 5, 6, 7.

Why are ranks helpful? One practical thing is that they make the sample unitless. Whether y is measured in miles-per-gallon, or kilometers-per-liter, ranks r of y go from $1, \ldots, n$. Ranks preserve the order of data values, while homogenizing the gaps between them. You can have a heavy tail in one or both directions, or multiple modes[3] causing data values to clump and spread and clump and spread again, etc. Rank them and they'll be evenly spaced. There's always a gap of one between one rank and the next, modulo ties.

Finally, and perhaps most importantly for statistics, ranks impose an implicit empirical distribution (§A.1) after dividing by n. I encourage you to read more about that in the Appendix. What it means for us here is that the uniform distribution[4], Unif$(0, 1]$, is sensible for modeling r/n. Fixating just on the n particular rank values, a discrete uniform[5] will do. So when working with ranks in a non-P context, you boil an unknown distribution down to the very simplest one: uniform. What a trip!

If you don't completely follow, which would be understandable because I just distilled a highly technical argument occupying whole chapters of other textbooks down into a paragraph, don't worry. I just wanted to give you some context. We're not going to need a handle on all of that to get busy with testing. Just appreciate that ranks offer a transformation of observed data values, from whatever values they had originally down to integers between 1 and n, inclusive.

9.2 Wilcoxon/Mann–Whitney

The Wilcoxon rank sum test[6] is a test for differences in means for samples from two populations. It's sometimes also known as the Mann–Whitney test, as it was developed independently around the same time (Mann and Whitney, 1947). The setup is the same as for permutation tests. Suppose $Y_1, \ldots, Y_{n_y} \overset{\text{iid}}{\sim} F_Y$, and $X_1, \ldots, X_{n_x} \overset{\text{iid}}{\sim} F_X$ similarly. Implicitly, the test is about determining if $F_Y = F_X$ or not, but unlike the permutation test which is somewhat generic through a choice of testing statistic, a Wilcoxon test specifically targets means.

$$\begin{aligned} \mathcal{H}_0 &: \mathbb{E}\{Y\} = \mathbb{E}\{X\} && \text{null hypothesis (“what if”)} \\ \mathcal{H}_1 &: \mathbb{E}\{Y\} \neq \mathbb{E}\{X\} && \text{alternative hypothesis} \end{aligned} \tag{9.2}$$

[3] https://en.wikipedia.org/wiki/Multimodal_distribution
[4] https://en.wikipedia.org/wiki/Continuous_uniform_distribution
[5] https://en.wikipedia.org/wiki/Discrete_uniform_distribution
[6] https://en.wikipedia.org/wiki/Mann–Whitney_U_test

There are variations targeting other aspects, such as variance, but those are coming later in Chapter 11. Also like the permutation test, the Wilcoxon test works on the combined sample of size $n = n_y + n_x$. Let $r_i^{(y)}$ and $r_j^{(x)}$ denote the rank assigned to y_i and x_j in the combined sample, respectively, for all $i = 1, \ldots, n_y$ and $j = 1, \ldots, n_x$. If there are ties, then assign an average rank as described in §9.1. Although it helps to keep track of them separately, as $r_i^{(y)}$ and $r_j^{(x)}$, don't forget that ranks are assigned on the combined sample.

It'll help to introduce a running example to make everything concrete. Data values below come from a survey of Washington Metro[7] riders at various stations in Virginia (y) and Washington DC (x) during a particular commute hour. These numbers tally how many riders said that they used a prepaid paper fare card. These were introduced when Metro opened and which are now being phased out[8] in favor of the newer plastic, credit-card-like SmarTrip[9] cards.

```
y <- c(80, 76, 56, 67, 73, 58, 51, 65, 68, 61)
x <- c(83, 66, 71, 82, 81, 89, 97, 59, 74)
```

First, form the combined sample and then rank.

```
yx <- c(y, x)
ryx <- rank(yx)
```

The portion of those ranks corresponding to y and x, $r(y)$ and $r(x)$, respectively, may be extracted as follows.

```
ny <- length(y)
ry <- ryx[1:ny]
nx <- length(x)
n <- ny + nx
rx <- ryx[(ny + 1):n]
```

Several statistics get used with Wilcoxon rank sum tests, depending on context and software, but all of them are based around the sum of the ranks assigned to the first, y, group.

$$\bar{r} \equiv \bar{r}^{(y)} = \sum_{i=1}^{n_y} r_i^{(y)} \tag{9.3}$$

Actually, it's equivalent to use the sum of the second, x, group, which is why I drop the (y) superscript. It doesn't matter which you pick. Just keep track of what you're doing and be consistent. The test comes out the same, either way. (I encourage you to verify.)

```
rbar <- sum(ry)
rbar
```

```
## [1] 72
```

[7] https://www.wmata.com
[8] https://www.wmata.com/fares/paperless.cfm
[9] https://en.wikipedia.org/wiki/SmarTrip

Note that I'm using a bar over r, as $\bar{r}$, which is a notation usually reserved for an average rather than a sum. We could take the average instead, i.e., divide by n_y. For the MC method I take you through momentarily, it doesn't matter. For the math way, we'd have to multiply $\bar{r}$ by n_y before using it if we prefer to use an average. I've chosen not to do that because this is known as the Wilcoxon "rank sum" test, not the "rank average" test, and I don't want to be too confusing. I'll admit that my choice of $\bar{r}$ as the label for the statistic is unconventional. Many simply write $t = \sum_{i=1}^{n_y} r_i^{(y)}$, and I've already explained why I don't want to use the letter t.

People always find this statistic (9.3) curious. You have two samples, but the test only uses the ranks of one of them, say $r^{(y)}$. I saved `rx` above, but actually I don't need it for anything. When you think about it, though, the x sample is "felt" through `ry`, because those values were calculated from the combined sample. Ranks $r^{(x)}$ are there implicitly, in the in-between spaces of the ranks of $r^{(y)}$. So the distribution of $R^{(x)}$ is felt through $R^{(y)}$.

In a moment I'll take you through the MC, and then through the math. In both cases, the idea is as follows. If $F_Y = F_X$, under the null[10], then ranks of Y will look like ranks of X in the combined sample, $R^{(y)}$ and $R^{(x)}$. You won't be able to tell the difference, because they came from the same population. Therefore, any summary of those ranks, like their sum or average, will also look the same, on average. Once we understand the sampling distribution of $\bar{R}$, we may compare that to $\bar{r}$ from observed data.

One last note is relevant for both ways. Under the null hypothesis

$$\mathbb{E}\{\bar{R}\} = \frac{n_y(n_y + n_x + 1)}{2} = \frac{n_y(n+1)}{2}. \tag{9.4}$$

This is pretty easy to show. The sum of all n ranks, assuming they're distinct (i.e., no ties) is $\sum_{i=1}^{n} i = n(n+1)/2$. I suspect you've seen sums of finite series[11] before, but if not that's fine. The particular one in question here is known as a triangular number[12]. Instead of summing up all n ranks, $\bar{R}$ sums just n_y of them. It doesn't matter which ones when it's uniformly at random. By similar logic, the caveat "no ties" isn't really needed since ties are replaced with average ranks.

Wilcoxon test by MC

Sampling ranks is as easy as randomly permuting $\{1, \ldots, n\}$.

```
N <- 1000000
Rbars <- rep(NA, N)
for(i in 1:N) {
  Ryxs <- sample(1:n, n)
  Rys <- Ryxs[1:ny]
  Rbars[i] <- sum(Rys)
}
```

According to Eq. (9.4), the mean of this distribution may be calculated as follows.

[10] In fact, the null (9.2) says $\mathbb{E}\{Y\} = \mathbb{E}\{X\}$ which is weaker, because the Wilcoxon rank sum test "targets" means. But it's ok to strengthen the null. It is certainly the case that if $F_Y = F_X$ then $\mathbb{E}\{Y\} = \mathbb{E}\{X\}$.

[11] https://en.wikipedia.org/wiki/Series_(mathematics)

[12] https://en.wikipedia.org/wiki/Triangular_number

```
Rmean <- ny*(n + 1)/2
c(Rmean, mean(Rbars))
```

```
## [1] 100 100
```

You can see that the theoretical value and MC calculation agree. $\mathbb{E}\{\bar{R}\}$ is important here in order to reflect the observed statistic over the mean for our favorite visual. Use either the theoretical value, or its MC estimate, and the visuals come out about the same. Graphics aren't rocket science, but that's not the main reason for showing you. I'm building trust for later when we really need Eq. (9.4) for the math way. In the meantime, Figure 9.1 provides a histogram summary of MC samples of $\bar{R}$ with observed and reflected statistics overlayed.

```
hist(Rbars, main="")
abline(v=c(rbar, 2*Rmean - rbar), col=2, lty=1:2, lwd=2)
legend("topright", c("obs", "reflect"), lty=1:2, lwd=2, col=2, bty="n")
```

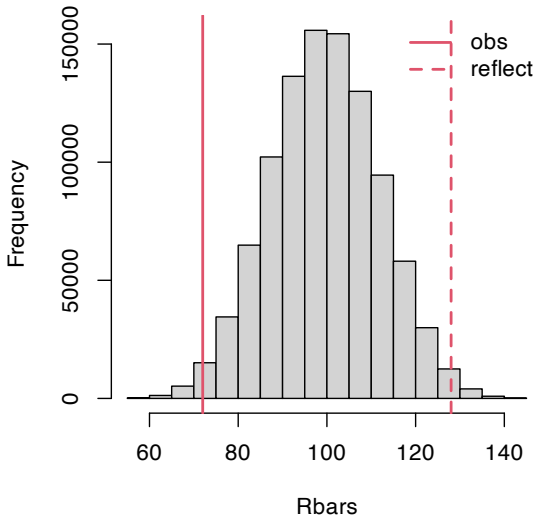

FIGURE 9.1: Sampling distribution of $\bar{R}$ with observed $\bar{r}$ for the Metro fare card data, and its reflection overlayed.

This is a close call. Looks like a reject to me, but let's be sure with a p-value calculation.

```
pval.mc <- 2*mean(Rbars <= rbar)
pval.mc
```

```
## [1] 0.02222
```

Indeed.

Ok, now I want to pivot slightly and talk about ties in the combined sample resulting in averaged ranks. Here is a new data set to work with. These values come from a follow-on study with an otherwise similar setup: counting paper fare card rides on Metro during a particular commute hour at stops in DC and Maryland (MD). So as not to overwrite the

earlier VA (y) and DC (x) values – I want to use them later – this time I'll work with MD as y2 for y and DC as x2 for x.

```
y2 <- c(67, 94, 83, 98, 35, 73, 29, 36, 60, 105, 34, 84, 89, 76, 79, 92,
  49, 97, 46, 32, 104, 60, 61, 59, 98, 101)
x2 <- c(107, 114, 87, 102, 85, 94, 109, 91, 99, 59, 97, 92, 103, 90, 43,
  108, 61, 111, 87, 112, 109, 115, 89, 105, 109, 83, 35, 114)
```

It's easy to see some duplicated entries. A pair of 87s jumps out at me. There are many more than that, and I'll be more precise later. What I want to do first is carry out the analysis as if this whole ties thing was no big deal. Perhaps it is no big deal. Tracing steps from above, with suitable changes in symbols, …

```
yx2 <- c(y2, x2)
ryx2 <- rank(yx2)
ny2 <- length(y2)
ry2 <- ryx2[1:ny2]
nx2 <- length(x2)
n2 <- ny2 + nx2
```

Next calculate $\bar{r}$ and run the MC loop for $\bar{R}$. This time I'm allowing some variables from the earlier VA/DC analysis to be overwritten because I don't need them later.

```
Rbars <- rep(NA, N)
for(i in 1:N) {
  Ryxs <- sample(1:n2, n2)
  Rys <- Ryxs[1:ny2]
  Rbars[i] <- sum(Rys)
}
```

I'll skip the histogram visual and jump right to the p-value.

```
rbar2 <- sum(ry2)
pval2.mc <- 2*mean(Rbars <= rbar2)
pval2.mc
```

```
## [1] 0.000422
```

Ok, so easy reject. However, this virtualization was not faithful to the observational data. We know that there *are* ties in the ranks of the observed values, but there are no ties (by construction) in any of the virtual ranks sampled within the MC loop. Each is a random permutation $1, \ldots, n \equiv$ n2 without any averaged values. What to do?

There's actually no playbook for this, at least when it comes to MC. I'm not the first person to use MC for statistical testing. But as far as I'm aware, I'm on the vanguard of its application to the Wilcoxon rank sum test, and therefore so are you. I'm sorta making it up as I go. So feel free to expand or modify things and see what happens. People have thought about it some when it comes to the math way I'll present next, but I'm hoping you're going to find that unsatisfying. What I'm about to show you is better, or at least more flexible.

The idea here, as with any MC, is to tailor a virtualization to what you know about the data-generating mechanism. Mimic what little you know, or additionally wish to assume, based on what you observe. The goal is to explore variability of the (rank sum) statistic within the confines of the null hypothesis (9.2) and the nature of observational data. So begin with assessing the situation. How far off are things from the rough-and-ready no-ties solution above?

Let's count ties. I used `duplicated` in §9.1 on raw data values. This time, I think `table` will give a more compact summary, and I've done it on the ranks themselves. There are many good options long as you're careful.

```
tab <- table(ryx2)
table(tab[tab > 1])
```

```
##
##  2  3
## 13  1
```

Thirteen ranks are duplicated, and one appears in triplicate. What to do about that? How about randomly generating a similar number of ties through averaging, before randomly permuting those ranks? This is a little tricky to implement because it only makes sense to average consecutive ranks. The code file rank_ties.R[13], available on the book webpage, contains a `rank.ties` function that randomly inserts rank-averages. For more on this, see §B.2. Here's a little demo of how we shall use it for the current analysis.

First, generate 13 degree-2 ties in raw ranks ranging from 1 to the combined sample size `n2` $\equiv n_y + n_x = n$.

```
source("rank_ties.R")
rd2 <- rank.ties(1:n2, 13, 2)                  ## 13 degree-2 ties
rd2
```

```
## $ranks
##  [1]  1.0  2.0  3.5  3.5  5.5  5.5  7.0  8.5  8.5 10.0 11.0 12.0 13.5
## [14] 13.5 15.5 15.5 17.0 18.5 18.5 20.0 21.5 21.5 23.0 24.0 25.0 26.0
## [27] 27.0 28.0 29.0 30.5 30.5 32.0 33.0 34.0 35.0 36.0 37.5 37.5 39.0
## [40] 40.5 40.5 42.0 43.5 43.5 45.0 46.0 47.5 47.5 49.0 50.0 51.5 51.5
## [53] 53.0 54.0
##
## $uniq
##  [1]  1  2  7 10 11 12 17 20 23 24 25 26 27 28 29 32 33 34 35 36 39 42
## [23] 45 46 49 50 53 54
```

A quick scan of consecutive ranks reveals fractional values that come from averaging. The choice of which pairs for averaging is random, so if you run this on your own machine, you'll likely get a different outcome.

The `rank.ties` function is designed to be daisy-chained. So you can take ranks from one call and pass them into another as follows. The key ingredient is keeping track of any remaining unique, non-average ranks in `rd2$uniq`.

[13] https://bobby.gramacy.com/hipp0/rank_ties.R

```
rd3 <- rank.ties(rd2$ranks, 1, 3, rd2$uniq)    ## with 1 degree-3 tie
tab <- table(rd3$ranks)
table(tab[tab > 1])
```

```
##
##  2  3
## 13  1
```

Now we have 13 averages from duplicates and 1 from a triplicate. A note of caution, though, when daisy-chaining: do the bigger ties first. That is, first do degree 3, and then degree 2. The larger the degree, the harder it is to find runs of that length. If some ranks are eliminated as runs of degree three because they're already averaged in a run of degree two, you could end up in a spot where the code can't find the ties you're looking for. I didn't do that above, but I will below.

So first, randomly generate ties, then randomly permute them. Otherwise, the setup is the same as earlier, where we ignored ties. It must be said that this new MC is much less efficient, computationally speaking, than the earlier one. My `rank.ties` implementation uses a `while` loop (so we have nested loops here) in order to iterate over tie possibilities. This incurs substantial overhead. It's possible that could mitigated with a more clever implementation. To assess the damage, I've added some timing code.

```
tic <- proc.time()[3]                              ## stopwatch start
Rbars <- rep(NA, N)
ranks <- 1:n2                                      ## ranks without ties
for(i in 1:N) {
  rd3 <- rank.ties(ranks, 1, 3)                    ## one degree-3 tie
  rd2 <- rank.ties(rd3$ranks, 13, 2, rd3$uniq)     ## plus 13 degree-2 ties
  Ryxs <- sample(rd2$ranks, n2)                    ## randomize over ranks
  Rys <- Ryxs[1:ny2]                               ## rest as usual
  Rbars[i] <- sum(Rys)
}
toc <- proc.time()[3]                              ## stopwatch stop
```

First, a p-value calculation.

```
pval2.mc.ties <- 2*mean(Rbars <= rbar2)
c(ignore=pval2.mc, ack=pval2.mc.ties)
```

```
##   ignore      ack
## 0.000422 0.000472
```

That's not much different than we had before. It could be that having more ties, as a proportion of the total ranks, would lead to a more dramatic effect. I draw comfort from exploring both options, and having a variation that uses stochasticity (i.e., random ranks) in a way that's faithful to my observational data.

Having the number of ties in each MC sample mimic patterns in the data makes sense. The way I did it above is only one of potentially many possibilities. You could, for example, posit that the maximal tying degree has a Poisson distribution, and similarly the number of ties

at each degree could be Poisson. I'll leave that to you as an exercise in §9.5. Again, that's just one idea.

Considering the extra computational effort,

```
(toc - tic)/60    ## converting from seconds to minutes
```

```
## elapsed
##   253.3
```

which is substantial, one wonders if such small differences are worth the hassle. What's nice about this is that it's relatively easy to entertain, in terms of human effort. You don't have to be a clever mathematician. One only needs to come up with a way to virtualize, and randomize over, a feature observed in data. Any way you wish to do that is legitimate, so long as you explain/justify relative to other reasonable alternatives.

Wilcoxon test by math

If Y_i and X_j are iid from the same distribution (9.2), and there are no ties, then their ranks on the combined sample should resemble a random permutation of integers $\{1, \dots, n\}$, for $n = n_y + n_x$. The n_y ranks assigned to the first group, should look like a random selection of those integers. Therefore, the distribution of the rank sum statistic (9.3), $\bar{R}$, may be obtained by considering the distribution of the sum of n_y integers, selected without replacement from $\{1, \dots, n\}$.

The number of ways of forming a sum of n_y integers from n is $\binom{n}{n_y}$. If each way is equally likely, then $\mathbb{P}(\bar{R} = k)$, for some positive integer $k \leq \sum_{i=1}^{n} i = n(n+1)/2$, may be found by counting the number of different subsets of size n_y, from integers $\{1, \dots, n\}$, that add up to k, and dividing by $\binom{n}{n_y}$.

$$\mathbb{P}(\bar{R} = k) = \frac{\text{SSP}(k, n_y, \{1, \dots, n\})}{\binom{n}{n_y}}$$

Subset sum problems[14] (SSPs) are, in general for any set (we're using $\{1, \dots, n\}$), NP-hard[15]. NP-hard means that there's no known polynomial-time algorithm in n for all n_y and k. That's not quite right. What it actually means is that if you *could* solve the SSP in polynomial time, then you would also be able to solve a bunch of other intractable problems fast, because those problems belong to an equivalence class (of NP-hard problems). That is a subtle distinction.

Quick digression on SSPs, which I find fascinating. There are lots of seemingly "toy" problems out there like SSP that involve counting how many ways you can do this or that, or other. They seem like games without any real applications, but eventually one comes around. It sure is handy when a solution is ready off-the-shelf. There's a connection between SSP and a Hollywood movie called The Man Who Knew Infinity[16], based on a book[17] by

[14] https://en.wikipedia.org/wiki/Subset_sum_problem
[15] https://en.wikipedia.org/wiki/NP-hardness
[16] https://en.wikipedia.org/wiki/The_Man_Who_Knew_Infinity
[17] https://en.wikipedia.org/wiki/The_Man_Who_Knew_Infinity_(book)

Robert Kanigel[18]. Both are a biography of Indian mathematician Srinivasa Ramanujan[19] who was interested in counting things.

I won't spoil it; but if you like the genre, it's up there along with: The Imitation Game[20], about the famous/first computer scientist Alan Turing[21] and code-breaking during World War II; The Theory of Everything[22] about Stephen Hawking[23] and the origins of the universe; and A Beautiful Mind[24] about John Nash[25] and game theory[26].

Alright, back to SSP and computing. There are some clever approximations and solvers for special cases. In ranks.R[27], available on the book webpage, I provide mass and distribution functions, called `dranks` (implementing $\mathbb{P}(\bar{R} = k)$) and `pranks` (for $\sum_{\ell=1}^{k} \mathbb{P}(\bar{R} = \ell)$), respectively, which use a SSP solver library in R. Check this out …

```
source("ranks.R")  ## note: these are not vectorized
c(d=dranks(rbar, ny, nx), p=pranks(rbar, ny, nx))    ## using k or l = rbar
```

```
##        d        p
## 0.002403 0.011009
```

Here, I'm following the convention in R that density (or mass) functions start as `d*` and distribution functions as `p*`, like `dnorm` and `pnorm`. For more details on these functions and their implementation, see §B.3. Figure 9.2 provides the mass function for $\bar{R}$ corresponding to $n_y = 10$ and $n_x = 9$ from the first Metro VA/DC example. Overlayed are the observed and reflected $\bar{r}$-values calculated earlier.

```
rgrid <- 50:150
dr <- rep(NA, length(rgrid))
for(k in 1:length(rgrid)) dr[k] <- dranks(rgrid[k], ny, nx)
sfun <- stepfun(rgrid[-1], dr)
plot(sfun, xlim=c(50, 165), xlab="rbar",
  ylab="mass dranks(rbar, ny, nx)", main="", lwd=2)
abline(v=c(rbar, 2*Rmean - rbar), col=2, lty=1:2, lwd=2)
legend("topright", c("rbar", "reflect"), lty=1:2, col=2, lwd=2, bty="n")
```

This view is quite similar to Figure 9.1. The observed statistic is in the left tail, so a p-value may be calculated as follows.

```
pval <- 2*pranks(rbar, ny, nx)
c(mc=pval.mc, exact=pval)
```

```
##      mc   exact
## 0.02222 0.02202
```

[18] https://en.wikipedia.org/wiki/Robert_Kanigel
[19] https://en.wikipedia.org/wiki/Srinivasa_Ramanujan
[20] https://en.wikipedia.org/wiki/The_Imitation_Game
[21] https://en.wikipedia.org/wiki/Alan_Turing
[22] https://en.wikipedia.org/wiki/The_Theory_of_Everything_(2014_film)
[23] https://en.wikipedia.org/wiki/Stephen_Hawking
[24] https://en.wikipedia.org/wiki/A_Beautiful_Mind_(film)
[25] https://en.wikipedia.org/wiki/John_Forbes_Nash_Jr.
[26] https://en.wikipedia.org/wiki/Game_theory
[27] https://bobby.gramacy.com/hipp0/ranks.R

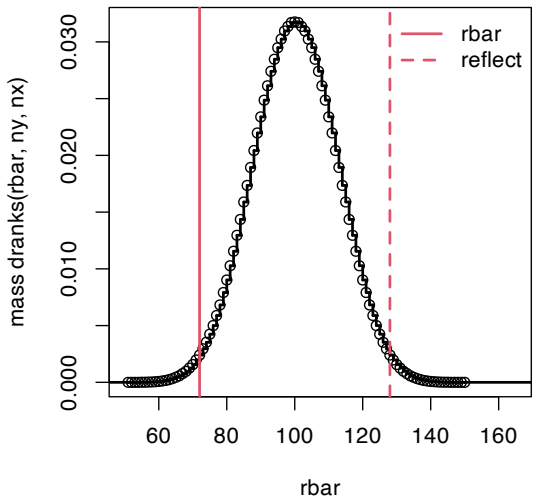

FIGURE 9.2: Exact rank sum sampling distribution for Metro VA/DC example without ties.

It looks like our MC is highly accurate. If observed $\bar{r}$ were in the right tail, we would usually provide `lower.tail=FALSE` to a `p*` function. But this is not implemented by `pranks`. I was being lazy. Imagine we had observed the reflected value of the statistic instead, which is in the right tail. In that case, provide …

```
reflect <- 2*Rmean - rbar
2*(1 - pranks(reflect - 1, ny, nx))
```

```
## [1] 0.02202
```

In other words, take away the integral from left.

In R, there's a distribution function called `pwilcox` (and similarly `dwilcox`, `qwilcox` and `rwilcox`), but it works with a shifted statistic:

$$\bar{r}' = \bar{r} - \frac{n_y(n_y+1)}{2} \quad \text{so that} \quad \mathbb{E}\{\bar{R}'\} = \frac{n_y n_x}{2}.$$

I'm not sure what the reason for this shift is, but I did warn you that several similar statistics were used with Wilcoxon tests.

```
2*pwilcox(rbar - ny*(ny + 1)/2, ny, nx)
```

```
## [1] 0.02202
```

So `pwilcox` does the same thing as my by-hand `pranks` function. It has the advantage of offering a `lower.tail=FALSE` option, but rather less transparency in the underlying calculation.

The name of this function, which uses "wilcox" rather than "wilcoxon", was a source of confusion to me for some time. I would say "Wilcox test" (wrong) rather than "Wilcoxon test" (right). Just be careful. The dude's name is Wilcoxon, not Wilcox. Adding another

layer to confusion, there's a `wilcox.test` library function automating things. But it does not perform a "Wilcox test", because that's not a thing. It does a Wilcoxon test.

```
pval.lib <- wilcox.test(y, x)$p.value
c(mc=pval.mc, exact=pval, lilb=pval.lib)
```

```
##      mc   exact    lilb
## 0.02222 0.02202 0.02202
```

Now that was for the case where there were no ties. All of that discussion about randomly permuting ranks and solving SSPs relies on having ranks filling $\{1, \ldots, n\}$. Ties lead to fractional, duplicated and skipped ranks. There's no simple way to count things in this context, because those ties, etc., can be anywhere. MC, as $N \to \infty$ offers the only exact way to work with the sampling distribution in this case. We saw that things didn't change much, and the same is true here.

Consider my second MD/DC Metro fare card example which had 13 duplicates and one triplicate. What if we just use `pranks`, `pwilcox` or the library function directly, ignoring the ties?

```
pval2.mc.ties <- 2*mean(Rbars <= rbar2)    ## Rbars still for to MD/DC
pval2.exact <- 2*pranks(rbar2, ny2, nx2)
pval2.pwil <- 2*pwilcox(rbar2 - ny2*(ny2 + 1)/2, ny2, nx2)
c(mc=pval2.mc, mc.ties=pval2.mc.ties, exact=pval2.exact, pwil=pval2.pwil)
```

```
##        mc   mc.ties     exact      pwil
## 0.0004220 0.0004720 0.0004403 0.0004403
```

These new "exact" calculations are in agreement with one another, and in close agreement with previously calculated MC p-values. Yet the library function gives a different number along with a warning.

```
wilcox.test(y2, x2)
```

```
## Warning in wilcox.test.default(y2, x2): cannot compute exact p-value
## with ties
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  y2 and x2
## W = 166, p-value = 6e-04
## alternative hypothesis: true location shift is not equal to 0
```

A p-value of $\psi = 6 \times 10^{-4}$ is pretty close to those other estimates, but the warning together with the message about "continuity correction" suggests something new is afoot. What's happening?

This is the central limit theorem (CLT; §4.1) in action. Subtracting off the mean (9.4) is the easy part, but dividing by the standard error of $\bar{R}$ is rather more involved. I'm just going to quote the result here so we can see what the software is doing.

$$Z = \frac{\bar{R} - \frac{n_y(n+1)}{2}}{\sqrt{\frac{n_y n_x}{n(n-1)} \sum_{i=1}^{n} r_i^2 - \frac{n_y n_x (n+1)^2}{4(n-1)}}} \sim \mathcal{N}(0,1) \quad \text{as } n \to \infty$$

That standard error is intense! Someone worked hard. It's not at all clear that, for fixed n, this is any better than using the exact result in the presence of ties. Both represent an approximation in this case.

```
r2sum <- sum(ryx2^2)
np1 <- n2 + 1
nm1 <- n2 - 1
z.numer <- rbar2 - ny2*np1/2
v <- (ny2*nx2/(n2*nm1))*r2sum - ny2*nx2*np1^2/(4*nm1)
z <- z.numer/sqrt(v)
pval2.clt <- 2*pnorm(-abs(z))
pval2.clt
```

```
## [1] 0.0006258
```

This matches what the library provides with `exact=FALSE` (meaning use CLT) and `correct=FALSE` (meaning don't make a continuity correction).

```
c(clt=pval2.clt,
  lib.cF=wilcox.test(y2, x2, exact=FALSE, correct=FALSE)$p.val,
  lib.cT=wilcox.test(y2, x2, exact=FALSE)$p.val)
```

```
##       clt    lib.cF    lib.cT
## 0.0006258 0.0006258 0.0006460
```

You get a slightly different result with the default of `correct=TRUE`. Continuity correction is common when approximating a distribution for discrete quantities with a continuous one (i.e., Gaussian). It leads to a slightly more accurate result, which in this case isn't much different.

```
z.cT <- (z.numer - sign(z.numer)*0.5)/sqrt(v)
2*pnorm(-abs(z.cT))
```

```
## [1] 0.000646
```

I'm just showing you for full transparency. It's pretty in the weeds, splitting hairs, and much ado about nothing when it comes to a p-value that leads with three zeros. If I'm being honest, I wasn't exactly sure about the continuity correction until I downloaded the source code for R[28], and searched the archive for "`wilcox.test`".[29] This is the beauty of an open-source project, like R. No smoke and mirrors. Back to Wilcoxon tests … When it comes to which statistic, and how to approximate with ties, I'm pretty open to good judgment.

On a pen-and-paper exam, a CLT version is doable especially if the exam provides $\sum_{i=1}^{n} r_i^2$. You can't exactly do a MC, or solve SSPs problems, without a computer. I think this is the

[28] https://cran.r-project.org/

[29] If you're curious, `wilcox.test` resides in `src/library/stats/R`.

main reason why textbooks provide this approximation, historically speaking, not because it's actually better than something else. In a modern context, the CLT can still come in handy when n is large. This is when it's approximation is most accurate, and solving SSPs can be computationally fraught. Remember, SSPs are NP-hard, which means things get bad as n gets big.

Any of the four or five methods, depending on how you count them, are probably fine if the p-value is far from 0.05. Perhaps worry more about it when you have lots more ties, as a proportion of the total, and your p-values differ substantially from one method to the next (near 0.05). In that situation, I'd caution against rejecting the null.

9.3 Signed ranks

If the Wilcoxon rank sum test is the non-P analog of a two-sample t-test, then the Wilcoxon signed ranks test[30] is the non-P paired version.[31] Hypotheses (9.2) and data-generating mechanism are the same (F_Y and F_X, respectively), but we presume that the data come as $(x_1, y_1), \ldots, (x_{n'}, y_{n'})$, where each pair (x_i, y_i) is observed under similar circumstances, or are otherwise linked. Notice I'm using n' not n to count the total number of pairs. The reason for that will be apparent soon.

Just like with a paired t-test, define $d_i = y_i - x_i$, or vice-versa for $i = 1, \ldots, n'$. The signed ranks test makes an additional assumption that the distribution of these distances D_i, as random variables, is symmetric, which means their mean and median are the same, and that they are iid.

Now, discard all $d_i = 0$, so that n nonzero distances remain, and relabel these as $d_1, \ldots, d_n$. There are a couple of reasons for doing this. One is a practical matter when ranking, which we shall do momentarily. Any zero distances would all have rank 1. Another is statistical: zero distances don't provide any evidence against $\mathcal{H}_0$. Throwing them away, and effectively reducing the sample size, can't affect statistical power.

Let $r_i^{\pm}$ denote the *signed rank* of each d_i. That is, first rank absolute distances $|d_1|, \ldots, |d_n|$, yielding $r_1, \ldots, r_n$. Then put the sign of each d_i back on it's rank.

$$r_i^{\pm} = \begin{cases} r_i & \text{if } y_i > x_i \\ -r_i & \text{otherwise} \end{cases} \qquad \text{for} \quad i = 1, \ldots, n \tag{9.5}$$

The idea with these signed ranks is to work within the discrete, order-based notion that ranks provide, while retaining sign information. There are several statistics that are common in this setting, and most are based on a sum of positive ranks.

$$\bar{r}^{+} = \sum_{i=1} r_i^{\pm} 1_{\{d_i > 0\}} \tag{9.6}$$

When there are no ties, the null distribution of $\bar{R}^{+}$ is trivial via MC, and involves SSP calculations in closed form. When there are ties, the MC can be adjusted (or we can ignore

[30] https://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test

[31] It is technically known as the signed rank test. I've made it plural because I like it better. No rank makes sense in a vacuum, i.e., without other ranks.

them), or a CLT analysis can be performed. I'll get to these in turn, momentarily. It is easy to show that

$$\mathbb{E}\{\bar{R}^+\} = \frac{n(n+1)}{4}. \tag{9.7}$$

This makes sense intuitively since that's halfway between zero and the maximal sum of $n(n+1)/2$ in the case of all positive signed ranks. I'll verify via MC.

Like earlier, I shall introduce two examples: one with ties and one without. The first example involves a study on 12 mice, who were given a drug to improve their blood circulation. Circulation was measured, in milliliters per minute, before (x) administering the drug and again an hour later (y).

```
circ <- data.frame(
  after=c(0.0541, 0.0196, 0.0234, 0.0217, 0,0165, 0.0305, 0.0351, 0.0155,
    0.0272, 0.02425, 0.02605, 0.0513),
  before=c(0.0335, 0.0240, 0,0375, 0.0133, 0.0305, 0.0173, 0.0138, 0.0259,
  0.0134, 0.0365, 0.0205, 0.0352))
y <- circ$after
x <- circ$before
```

Here are the calculations needed to develop $\bar{r}^+$.

```
nprime <- length(y)
d <- y - x
d <- d[d != 0]                ## remove any zeros
n <- length(d)
r <- rank(abs(d))
sr <- r*sign(d)               ## sr for r^{+/-}
rbarp <- sum(sr[sr > 0])
c(nprime=nprime, n=n, u=length(unique(r)), rbarp=rbarp)
```

```
## nprime      n      u  rbarp
##     13     13     13     64
```

The numbers above indicate no zeros removed ($n = n'$), no tied ranks, and $\bar{r}^+ = 64$. Based on this latter number, I can tell already that we won't be able to reject $\mathcal{H}_0$. Why? Because the estimated statistic is slightly closer to its mean (9.7) of

```
Rbarp.mean = n*(n + 1)/4
Rbarp.mean
```

```
## [1] 45.5
```

than its largest possible value of $n(n+1)/2 = 91$. But just how close is close depends on the spread of the sampling distribution, or its standard error.

To span more possibilities, the next example involves data with zeros and ties. These values come from a study on the impact of physical fatigue on mental acuity. Volunteers were recruited to solve a series of mental puzzles both before (x) and after (y) participating in a strenuous exercise regimen. Times to complete the puzzles were recorded in minutes.

As earlier, I use `y2` and `x2` to keep track of these alongside values from the earlier blood circulation experiment.

```
y2 <- c(134, 120, 86, 221, 75, 84, 87, 210, 122, 218, 68, 143, 86)
x2 <- c(78, 61, 70, 93, 75, 89, 100, 60, 93, 98, 81, 87, 84)
```

Here are the calculations needed to develop the $\bar{r}^+$ statistic.

```
nprime2 <- length(y2)
d2 <- y2 - x2
d2 <- d2[d2 != 0]             ## remove any zeros
n2 <- length(d2)
r2 <- rank(abs(d2))
sr2 <- r2*sign(d2)            ## sr for r^{+/-}
rbarp2 <- sum(sr2[sr2 > 0])
c(nprime=nprime2, n=n2, u=length(unique(r2)), rbarp=rbarp2)
```

```
## nprime      n      u  rbarp
##     13     12     10     69
```

As you can see, one zero-distance pair was removed, and there's one tie in the ranks.

Signed ranks test by MC

Let's begin with the no-ties blood circulation data (`y` and `x`). My setup here isn't much different than the (no-ties) MC for the rank sum test in §9.2. After generating a random permutation on $\{1, \ldots, n\}$, simply randomly assign signs. If means are the same (9.2), then the chances that $y_i > x_i$ is the same as $y_i < x_i$. I found that, in this example, it helps to have a larger N, as indicated by the commented out code below. You may wish to try that, but I decided to keep the same N as above for consistency in this chapter.

```
## N <- 10000000                              ## opt. for higher accuracy
Rbarps <- rep(NA, length=N)
for(i in 1:N) {
  Rs <- sample(1:n, n)                       ## using Rs <- 1:n works too
  Signs <- sample(c(-1 ,1), n, replace=TRUE)  ## because signs are random
  SRs <- Rs*Signs
  Rbarps[i] <- sum(SRs[Signs > 0])
}
```

There's an interesting side note about the latter two commented code lines above. Technically, it's sufficient to put random signs on ranks $\{1, \ldots, n\}$ without first randomly permuting them. Signs end up allocated to ranks, uniformly at random, no matter what. Still, I like the extra `sample` step because it's conceptually faithful the overall virtualization enterprise.

I'm figuring that you're tired, by now, of looking at histograms of sampling distributions with observed and reflected statistics overlayed. So I'll give you a break from those for a while. They're helpful to check intuition, and to see which tail an observed statistic is in, so that you get the p-value calculation the right way around. For now, I'll cut to the chase. (Shape-wise, the histogram looks similar to the exact one coming momentarily in Figure 9.3.)

```
pval.mc <- 2*mean(Rbarps >= rbarp)
pval.mc
```

```
## [1] 0.2155
```

My intuition from earlier was correct. The observed statistic is not unlikely under the sampling distribution of signed ranks. Therefore, we cannot reject the null hypotheses that the samples have the same means. It seems that the drug is ineffective at improving/changing blood circulation.

As a quick check of Eq. (9.7), consider the following.

```
c(mc=mean(Rbarps), exact=Rbarp.mean)
```

```
##    mc exact
##  45.5  45.5
```

This is pretty darn close!

Now let's move on to the second example. Dropping a single zero distance doesn't change much downstream in our code, except that one pair of observations was discarded, redefining n. Having a tie could potentially change how our MC works. Last time (§9.2) we ran two MCs, one ignoring the tie(s) and another simulating them. This time, how about doing both at once with a single `for` loop? (Like before, this takes a bit longer due to `rank.ties`. Not much longer because we're only looking for one tie, not 13.)

```
Rubarps <- Rtbarps <- rep(NA, N)
ranks <- 1:n2                                   ## ranks without ties
for(i in 1:N) {

  ## deal with ties
  rd2 <- rank.ties(ranks, 1, 2)                 ## 1 degree-2 tie

  ## permutations
  perm <- sample(ranks)                         ## shared permutation (opt)
  Rts <- rd2$ranks[perm]                        ## permute tied ranks
  Rus <- ranks[perm]                            ## permute unique ranks

  ## signs
  Signs <- sample(c(-1, 1), n2, replace=TRUE)   ## shared random signs
  SRts <- Rts*Signs
  SRus <- Rus*Signs

  ## final statistic
  Rtbarps[i] <- sum(SRts[Signs > 0])            ## rest as usual
  Rubarps[i] <- sum(SRus[Signs > 0])
}
```

The two p-values are calculated below, ignoring or acknowledging a single tie in the ranks.

```
pval2.mc <- 2*mean(Rubarps >= rbarp2)
pval2.mc.ties <- 2*mean(Rtbarps >= rbarp2)
c(ignore=pval2.mc, ack=pval2.mc.ties)
```

```
##  ignore     ack
## 0.01614 0.01553
```

Comparatively speaking, this result is similar to earlier with the rank sum test – those p-values aren't too different. I draw comfort from looking at things multiple ways. In any case, this is a clear reject of $\mathcal{H}_0$. It appears that physical exertion can have an effect on mental acuity.

Signed rank test by math

The sampling distribution for $\bar{R}^+$ under the null may be calculated in closed form in the same way as for the rank sum test. Just count how many positive ranks in a partition of positive and negative ranks sum up to a particular value, like $\bar{r}^+$, and divide by the total number of all partitions. That's just a hint, since I've left the details to you as a homework exercise. Check out the implementation of `pranks` in §B.3, and appropriate its SSP solver to aid in your counting.

You can check your work against `dsignrank` and `psignrank` in R. Take the first example without ties. The former can be useful for visualizing the sampling distribution, as in Figure 9.3. It's clear from this view that there's not enough evidence to reject the null.

```
srgrid <- 1:90
dsr <- rep(NA, length(srgrid))
dsr <- dsignrank(srgrid, n)
sfun <- stepfun(srgrid[-1], dsr)
plot(sfun, xlab="rbar-plus",
  ylab="mass dsignrank(rbar-plus, ny, nx)", main="", lwd=2)
abline(v=c(rbarp, 2*Rbarp.mean - rbarp), col=2, lty=1:2, lwd=2)
legend("topright", c("rbar", "reflect"), lty=1:2, col=2, lwd=2, bty="n")
```

The latter is used for p-value calculation. Since observed $\bar{r}^+$ is in the right tail, one must both provide `lower.tail=FALSE` *and* subtract one from the statistic, because of the discrete nature of the distribution. Alternatively, you may use the reflected statistic (unaltered) and work in the left tail. Check your work with `wilcox.test(..., paired=TRUE)`, which is set up like `t.test()`.

```
pval.exact <- 2*psignrank(rbarp - 1, n, lower.tail=FALSE)
pval.lib <- wilcox.test(y, x, paired=TRUE)$p.value
c(mc=pval.mc, exact=pval.exact,
  reflect=2*psignrank(2*Rbarp.mean-rbarp, n), lib=pval.lib)
```

```
##      mc   exact reflect     lib
##  0.2155  0.2163  0.2163  0.2163
```

Now on to the second example involving mental puzzles under physical fatigue. When there are ties, as there are in these data – and/or n is too large for solving SSPs (or for MC simulation) – the CLT may be used. For reasons that are not completely clear to me, it is

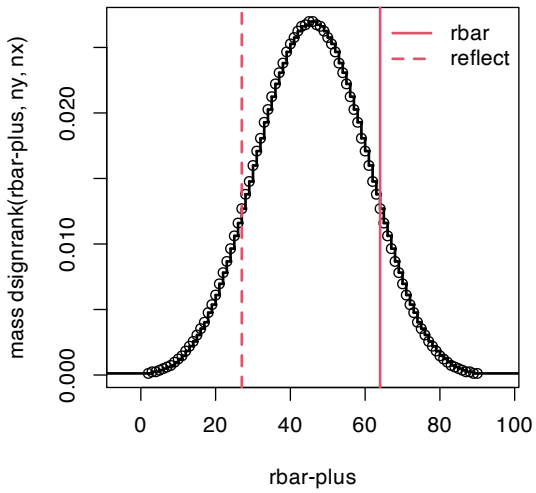

FIGURE 9.3: Exact $\bar{R}^+$ sampling distribution for blood circulation example without ties.

common to use a different statistic with the CLT. It seems to me that it would be just as easy to work with positive signed ranks (9.6), but instead it's preferred to use the sum of all signed ranks (9.3) with the CLT. (Note that Eq. (9.3) was developed in the context of unsigned ranks on a combined sample, but here I'm reinterpreting it for signed ranks on differences, d_i, following Eq. (9.5).)

Clearly $\mathbb{E}\{\bar{R}\} = 0$ since each rank is equally likely to be positive or negative under the null (9.2). Working out the variance isn't much harder.

$$\mathbb{V}\text{ar}\{\bar{R}\} = \sum_{i=1}^{n} \mathbb{V}\text{ar}\{R_i\} = \sum_{i=1}^{n} \mathbb{E}\{R_i^2\} - \mathbb{E}\{R_i\}^2 = \sum_{i=1}^{n} R_i^2$$

The last step follows since $\mathbb{E}\{R_i\} = 0$. Reducing from $\mathbb{E}\{R_i^2\}$ to R_i^2 is a bit of a slight-of-hand. The square obliterates the sign of the signed rank since $(-1)^2 = 1$. Since every rank is possible, but only one rank can be held by each of $i = 1, \ldots, n$ at a time, it must be that all ranks are present when summing what to "expect" over all i. The end result isn't that helpful, though, since R_i are capitalized random variables. The best we can do is slot in observed r_i. If there are no ties, then $\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6$, a so-called pyramidal number[32]. That could be helpful if your only calculating tool is pen-and-paper, like on an exam, where summing up lots of squares isn't practical. (Our example has ties.)

Summarizing, we have

$$Z = \frac{\bar{R}}{\sqrt{\sum_{i=1}^{n} r_i^2}} \sim \mathcal{N}(0,1) \quad \text{as} \quad n \to \infty.$$

Therefore, the following calculations provide an approximate p-value for fixed n.

```
z <- sum(sr2)/sqrt(sum(sr2^2))
pval2.clt <- 2*pnorm(-abs(z))
```

[32] https://en.wikipedia.org/wiki/Square_pyramidal_number

```
c(mc=pval2.mc, mc.ties=pval2.mc.ties, clt=pval2.clt,
  lib=wilcox.test(y2, x2, paired=TRUE, exact=FALSE, correct=FALSE)$p.value)
```

```
##      mc mc.ties     clt     lib
## 0.01614 0.01553 0.01851 0.01851
```

This is definitely in the ballpark. Keep in mind everything that in the printout above is an approximation. None of these numbers leads to a different conclusion; not even close. All evidence points to fatigue impacting mental performance.

The calculations above omit a continuity correction (`correct=FALSE`). If you wish to compare to `correct=TRUE`, make the following adjustment.

```
z.numer <- sum(sr2)
z <- (z.numer - sign(z.numer))/sqrt(sum(sr2^2))
c(2*pnorm(-abs(z)), wilcox.test(y2, x2, paired=TRUE, exact=FALSE)$p.value)
```

```
## [1] 0.02056 0.02056
```

It's not clear to me why adding ± 1 is required this time, whereas it was $\pm\frac{1}{2}$ previously. I know that, as the author of this book, you're looking to me to be an authority. But these things are the worst part about statistics, in my opinion. Off-by-a-half (or by one) issues can be a huge distraction to understanding the big picture in any subject. If something so small substantively changes the calculation in a statistical analysis, and ultimately affects big downstream decision-making, then the whole enterprise is flawed. Go back to the drawing board. Do a new/bigger experiment, and get less lost in the weeds of a Gaussian approximation. Or, use MC.

9.4 One sample non-P location

A signed ranks test can also be applied for a single sample, to test for a mean (or median, since the symmetry assumption makes those the same). Assume $Y_1, \ldots, Y_{n'} \overset{\text{iid}}{\sim} F_Y$, for some symmetric F_Y. Suppose you want to know if $\mathbb{E}\{Y_i\} = \mu$ for some value of interest μ. Formally, the hypotheses are as in Eq. (9.2), except replace $\mathbb{E}\{X\} = \mu$ everywhere. There is no X sample or F_X.

The testing procedure is as in §9.3 except everywhere there's an X_i, replace with μ instead. Just treat the data as $(\mu, y_1), \ldots, (\mu, y_{n'})$ and push on. If any of the $d_i = y_i - \mu = 0$, then discard that data point, and let n be the number of nonzero $d_1, \ldots, d_n$ that remain. There could still be ties in the ranks, etc., so everything else still applies.

Since this is pretty straightforward, I'm not going to cover all of the cases with examples. One should do. So here we go. Intelligence Quotient (IQ)[33] tests are designed to produce scores that average around 100 with a standard deviation of 15, meaning that 95% of scores range from 70 to 130. Since IQ scores are whole numbers, a Gaussian assumption may not be appropriate although assuming a symmetric distribution may be reasonable. Scores quoted

[33] https://en.wikipedia.org/wiki/IQ_classification

below come from teenagers in a certain small college town with affluent parents who might be overzealous micro-managers of their children's education.

```
y <- c(95, 101, 100, 115, 105, 99, 101, 107, 120, 100, 98, 77, 89, 121,
  102, 97, 106, 111, 101, 107, 102, 88, 112, 128)
```

More than 70% of these scores are above 100, so you might be inclined to think that kids in this town are exceptionally smart based on this sample. That's a statistic, but it's not statistical inference. Can we reject the hippopotamus that these data were generated from a distribution with $\mathbb{E}\{Y\} = 100$?

Rather than re-code everything, I prefer to simply assign `x <- mu` and then cut-and-paste.

```
x <- 100              ## E(Y) = mu = 100
nprime <- length(y)
d <- y - x
d <- d[d != 0]
n <- length(d)
r <- rank(abs(d))
sr <- r*sign(d)
rbarp <- sum(sr[sr > 0])
c(nprime=nprime, n=n, u=length(unique(r)), rbarp=rbarp)
```

```
## nprime      n      u  rbarp
##     24     22     13    175
```

It looks like two zeros were removed, and there are lots of ties in the ranks.

```
tab <- table(r)
table(tab[tab > 1])
```

```
##
## 2 3 4
## 4 1 1
```

Here's a single MC that entertains a sampling distribution for $\bar{R}^+$ in two situations: ignoring ties, and mimicking them stochastically.

```
Rubarps <- Rtbarps <- rep(NA, N)
ranks <- 1:n                                        ## ranks without ties
for(i in 1:N) {

  ## deal with ties
  rd4 <- rank.ties(ranks, 1, 4)                     ## 1 degree-4 tie
  rd3 <- rank.ties(rd4$ranks, 1, 3, rd4$uniq)       ## 1 degree-3 tie
  rd2 <- rank.ties(rd3$ranks, 4, 2, rd3$uniq)       ## 4 degree-2 ties

  ## permute
  perm <- sample(ranks)                             ## shared permutation
  Rts <- rd2$ranks[perm]
```

```
  Rus <- ranks[perm]                                ## permute unique ranks

  ## signs
  signs <- sample(c(-1, 1), n, replace=TRUE)        ## shared random signs
  SRts <- Rts*signs
  SRus <- Rus*signs

  ## final statistic
  Rtbarps[i] <- sum(SRts[signs > 0])                ## rest as usual
  Rubarps[i] <- sum(SRus[signs > 0])

}
```

As you can see below, the p-values aren't much different. There is not enough evidence to reject the null hypothesis. Over-parented children in this small town aren't exceptionally smart.

```
c(ignore=2*mean(Rubarps >= rbarp), ack=2*mean(Rtbarps >= rbarp))
```

```
## ignore    ack
## 0.1191 0.1169
```

Here are calculations doing things the math way. First, ignoring the presence of ties, and performing calculations using closed-form calculations via SSP. (There seems to be no way to force the library function `wilcox.test` to use the exact calculation in the presence of ties. If there are ties, then it spits out a warning and uses the CLT (`exact=FALSE`) anyways.)

```
c(exact=2*psignrank(rbarp - 1, n, lower.tail=FALSE))
```

```
##  exact
## 0.1207
```

Finally, here is a version using a CLT approximation with continuity correction, first by hand and then with the library.

```
z.numer <- sum(sr)
z <- (z.numer - sign(z.numer))/sqrt(sum(sr^2))
c(clt=2*pnorm(-abs(z)), lib=wilcox.test(y, mu=100, exact=FALSE)$p.value)
```

```
##    clt    lib
## 0.1187 0.1187
```

It's pretty much the same outcome any way you slice it. And that's lots of cutting-and-pasting. Perhaps it would make sense to write our own library function before jumping into your own analyses. Guess what's coming next?

9.5 Homework exercises

These exercises help gain experience with Wilcoxon rank sum and signed rank tests. There are no math questions this time, all coding and/or data analysis. Most involve new data/problems, however many questions on t-tests from previous chapters (e.g., §2.6 and §5.4) offer good non-P practice. Don't forget to try each test multiple ways, e.g., via MC or math as appropriate.

Do these problems with your own code, or code from this book. Unless stated explicitly otherwise, avoid use of libraries in your solutions.

#1: Subset sum for signed ranks

Follow the template provided for `dranks` and `pranks` in rank_ties.R[34] (via §B.3), which is for rank sums, to provide a similar capability for signed ranks. That is, appropriate the SSP solver from those subroutines for working with signed ranks instead. You may check your work against `dsignrank` and `psignrank`. Maybe call your version `dsranks` and `psranks`.

#2: More stochasticity in ties

This one is a bit of a challenge problem. The (possibly) daisy-chained `rank.ties` calls from this chapter mimic tie-degrees that are present in observed ranks. Add flexibility to this by instead allowing the tying degree, and the number of ties for each degree, to follow Poisson distributions. Specifically, take the maximal tie degree as $D \sim \text{Pois}(\lambda_d)$. Given $D = d$, take the number of ties as $T_1, \ldots, T_d \overset{\text{iid}}{\sim} \text{Pois}(\lambda_t)$. Generate new D and $\{T_j\}_{j=1}^{d}$ in each MC iteration.

You'll still need to use `rank.ties` to find these ties for each degree, and don't forget to daisy chain from largest to smallest. Be careful not to ask for too many, and of too-high a degree, for small n, or it'll break. That means small λ_d and λ_t, like $\lambda_d = \lambda_t \leq 2$.

#3: `monty.wilcoxon` for one- and two-samples

Write a library function, inspired by `monty.t` from §2.6 and §5.4, that covers all variations of Wilcoxon tests. This includes two-sample, paired and unpaired, and one-sample tests; MC, closed-form exact and CLT versions; special considerations for ties (e.g., using `rank.ties` and optionally your solution to #2), and warnings as needed when your user makes ill-advised choices. You may use `pwilcox` or `pranks` and `psignrank` as desired, and `pnorm` as usual. Support optional visuals.

Hint: many of the following questions are easier after this one is squared away. (You should be getting the hang of this by now.)

#4: Your brain on artificial intelligence

A team of neuroscientists recruited thirty participants for a study on the effects of artificial intelligence (AI) on memory. Each participant was asked to write an essay on a topic of interest to them. Before writing these essays, participants were randomly assigned to one of two groups. One, the control group, was asked to use a computer terminal (connected to the internet), but with all AI functionality locked out. The other, the AI group, had access

[34] https://bobby.gramacy.com/hipp0/rank_ties.R

to an ordinary terminal without restrictions and were encouraged to use AI. Later, each participant was asked to recall passages about their essays. Their accuracy in recollection exercises was assigned a score from 0 to 100. Those are provided below.

```
control <- c(72, 94, 72, 93, 72, 91, 63, 52, 84, 59, 70, 66, 98, 91, 71)
AI <- c(53, 52, 83, 81, 63, 71, 50, 86, 43, 77, 49, 63, 57, 41, 55)
```

What do you think? Does AI have an effect on the average recall score?

#5: Zoom learning

Two sections of the same statistics class, led by the same instructor, were offered at VT one semester. Both classes were completely in sync: same lectures, homework, exams, etc. The only differences were the students and the official mode of instruction. One mode was "Zoom", and the other was "classic" in-person delivery. Students were even invited to attend classes, and take exams, outside their registered modality (`zoom` or `classic`), although anecdotally students rarely swapped modes. Final class percentages, combining all homeworks and exams, are provided by classmode.RData[35] on the book webpage.

```
load("classmode.RData")
```

Is there a difference in mean scores for students across these two modalities?

#6: Non-P ROI

Refer back to `roi$company` and `roi$industry` from §2.5. Recall that these data are available as roi.RData[36] on the book webpage.

```
load("roi.RData")
```

Based on a non-P analysis, does there seem to be any difference in mean ROI between companies that use the IT product in question and the industry at large? *You may have studied this parametrically in §5.4, and via bootstrap and permutation tests in Chapter 8.*

#7: Identical twin multivitamin study

Twin studies[37] are popular as a means of mitigating subject variability, among other (especially) genetic factors, in treatment–control trials (a.k.a., A/B tests). Twenty pairs of twins were recruited for a study of the effectiveness of an over-the-counter multivitamin on the immune system. In each pair of twins, one was randomly selected to take the multivitamin, and the other a placebo. After 30 days all participants had their blood drawn in order to measure T-cell counts, in cells per cubic millimeter of blood. These numbers are recorded below.

```
placebo <- c(1002, 517, 1100, 790, 1333, 787, 803, 720, 1243, 850, 696,
  713, 769, 556, 873, 563, 922, 660, 1401, 810)
```

[35] https://bobby.gramacy.com/hipp0/classmode.RData
[36] https://bobby.gramacy.com/hipp0/roi.RData
[37] https://en.wikipedia.org/wiki/Twin_study

```
multivit <- c(956, 693, 1044, 920, 1258, 873, 883, 1033, 1476, 940, 947,
  463, 678, 717, 775, 723, 1033, 715, 1245, 798)
```

What do you think? Does this particular multivitamin boost the immune system?

#8: Non-P paired ROI

Return to roip.csv[38] on the book webage, which pairs company and industry ROI row-wise.

```
roip <- read.csv("roip.csv")
```

Perform a non-P analysis to determine if `roip$company` and `roip$industry` have different means. *You may have studied this parametrically in §5.4.*

#9: Benzene conversions

Nandi et al. (2002) studied benzene conversions, measured in mole percent, for twenty-four different benzenehydroxylation reactions.

```
convs <- c(80.8, 59.4, 19, 30.1, 30.1, 81.2, 28.8, 40, 67.8, 26.8, 63, 80.8,
  40.0, 30.3, 41, 38.1, 81.2, 52.3, 36.2, 36.2, 33.4, 19, 9.1, 30.1)
```

Industrial application requires that mean conversions be 30 mole percent, or greater. What do you think?

[38] https://bobby.gramacy.com/hipp0/roip.csv

10

Pearson χ^2-tests

I know what you're thinking. We did χ^2-tests back in Chapter 6. You're right. The ones in this chapter aren't all that different. Chapter 6's focus was on a particular parameter, σ^2 in a Gaussian model, or σ_y^2 and σ_x^2 for two samples. That was back in the land of parametric (P) statistics. This chapter's χ^2-tests involve a similar statistic, but they're non-P and they're not (usually) about variance.

Remember, testing statistics are your choice, and they need not be neatly matched to an estimated quantity. It's convenient that residual sums of squares (RSSs) have a χ^2 distribution. We've used that fact a number of times, across several chapters. I won't list them all here. Recall our discussion around Eq. (7.18) where I explained that you could estimate, and thus test, variance under a simple linear regression model by borrowing from a simple means model (3.5), writing $\mu_i = \beta_0 + \beta_i x_i$. Here that idea is appropriated for another sort of μ_i. Both of those were Gaussian models, but it works even in a non-Gaussian setting because of the central limit theorem (CLT; §4.1). It works for any distribution, even ones without "parameters". RSSs are highly portable.

In this chapter I'll show you how to use normalized RSSs to test other/many aspect(s) of a distribution, in both one- and multi-sample contexts. Emphasis will be on non-P settings, although I'll show you an important application to P testing. Closed-form versions of the tests are exclusively asymptotic, due to their connection to the CLT. With small sample sizes, n, it can be hard to know whether those calculations are accurate. With Monte Carlo (MC), one can ensure that accuracy is only a question of computation effort.

10.1 Goodness-of-fit

So far, our tests have targeted specific, yet sometimes relative, features of a distribution of random quantities. Now I want to pivot to something that's simultaneously more specific and more all-encompassing. Here focus will be on a single population having a certain distribution. Rather than scrutinize a particular aspect, like a mean, median or standard deviation, a goodness-of-fit (GoF)[1] test looks at the entire distribution. Such a bold approach must be reigned in a little bit. Usually χ^2-tests are on discrete and finite distributions. They're also applicable to real-valued ones through binning. I'll show you.

Suppose our data consist of n independent observations of a random variable Y grouped into c classes. Grouping implies that sufficient information about such data would comprise the number of observations that are sorted into each class. Let o_j, for $j = 1, \ldots, c$, denote those observed counts. Each must be a non-negative integer, and $\sum_{j=0}^{c} o_j = n$. These

[1] https://en.wikipedia.org/wiki/Goodness_of_fit

TABLE 10.1: GoF contingency table.

	class 1	class 2	⋯	class c	total
observed	o_1	o_2	⋯	o_c	$n = \sum_j o_j$

quantities are often presented in a $1 \times c$ contingency table (CT)[2]. See Table 10.1. CTs with one-dimensional rows or columns represent overkill, since you can just list observed counts in-line like $o_1, o_2, \ldots, o_c$, but we'll see more interesting ones later, and it makes sense to start simple.

These data are compared against a distribution, specified as probabilities by the null hypothesis: $p_1, \ldots, p_c$, where $p_j \geq 0$ for $j = 1, \ldots, c$ and $\sum_{j=1}^{c} p_j = 1$. If $c = 2$, then those probabilities, along with n, prescribe a binomial distribution like in Chapter 1 for coin flips. In that context o_1 and o_2 tally the number of heads and tails, respectively, in n flips. When $c > 2$ we have what is known as a multinomial distribution[3], of which the binomial is a special case.

You don't really need to know many multinomial facts to understand the basics, except moments which I shall provide momentarily. There are some shortcuts in code if you're willing to work with a multinomial random-number generator. I'll show it to you both ways later and other technical nuggets as needed. For now, just think of a c-sided die that you roll n times instead of flipping a two-sided coin.

Let $p = p_1, \ldots, p_c$ denote the c-vector of probabilities, and likewise let $o = o_1, \ldots, o_c$ denote observations. Then, technically speaking, the model is

$$\text{Model:} \qquad O \sim \text{Multinom}(n, p).$$

Don't let the presence of O on the left of $\sim$ trip you up. True, I almost exclusively have Y in that position, and sometimes X, elsewhere in the book. Here, Y determines O through some classification or tally, as in Table 10.1. O is a compact way to look at Y: $o_j = \sum_{i=1}^{n} 1_{\{y_i = \text{class } j\}}$.

Multinomial means and variances are upgrades of results from the binomial:

$$\mathbb{E}\{O_j\} = np_j \quad \text{and} \quad \mathbb{V}\text{ar}\{O_j\} = np_j(1 - p_j) \quad \text{for} \quad j = 1, \ldots, c. \tag{10.1}$$

Quick digression on notation. Throughout, I've been careful to use Greek letters for parameters, but here I've used p instead of θ, like for the binomial in Chapter 1. I could – maybe I should – but I'm sticking with p here for a number of reasons. One is that the literature prefers p for GoF and similar tests, and that's related to my second reason. GoF tests are thought of as non-P, where the notion of "parameter" is complicated to say the least. How could a test that's non-P involve a c-vector of unknown parameters θ? Good question.

In a GoF test, and also for tests of independence and homogeneity coming soon in §10.3, probabilities p serve as an intermediary. They're derived from some other distribution or relationship of interest. That distribution might involve other, genuine parameters θ, or it might not. I'll table that for a later discussion, beginning in §10.2, so we don't get derailed

[2] https://en.wikipedia.org/wiki/Contingency_table
[3] https://en.wikipedia.org/wiki/Multinomial_distribution

here. For now, just be aware of it and know that I know that I'm not following my own rules. I've given it some thought and have decided that the presentation here works best with p, not θ which will come back later.

Alright, back to inference. The formal hypotheses involved in a GoF test question whether a particular p could have generated observations o. Let p_0 be a specific c-vector of probabilities.

$$\begin{aligned} \mathcal{H}_0 &: p = p_0 && \text{null hypothesis ("what if")} \\ \mathcal{H}_1 &: p \neq p_0 && \text{alternative hypothesis} \end{aligned} \tag{10.2}$$

Above, $p \neq p_0$ means that any of the c probabilities could differ. Since they must sum to one, changing one probability really means changing two or more. Going forward, I shall drop the cumbersome subscript in p_0 and simply refer to p under the null hypothesis. I have a particular p in mind for some reason or another and I want to know if it could have generated what was observed, o. This involves studying the sampling distribution for some statistic that can be calculated for o, and sampled for O, under $\mathcal{H}_0$.

Following Eq. (10.1), let $e_j = np_j$, denote expectations for $j = 1, \ldots, c$. Again, such quantities might have been assigned Greek letters elsewhere in the book, but I'm sticking with convention here and I think it will make sense from context as we get to fancier examples. Note that $\sum_{j=1}^{c} e_j = n$.

If the p_j values specified by $\mathcal{H}_0$ are correct, then o_j should be close to e_j, on average. So a sensible statistic comparing the two might aggregate their discrepancies: $\sum_j (o_j - e_j)^2$. Pearson's χ^2 statistic[4] normalizes each discrepancy before aggregating.

$$x^2_{nc} = \sum_{j=1}^{c} \frac{(o_j - e_j)^2}{e_j} \tag{10.3}$$

Quick digression on notation and nomenclature. You'll see how χ^2 distributions are involved when we get more into some of the math. I've put a complicated nc subscript, but this is not common, and I will generally drop all subscripts to streamline things. I did that above to remind you that two types of counts are involved: the total number of observations, or rolls of the dice n, and the number of classes, or die faces c. Although I've titled this section with "GoF", a multinomial setting and Pearson's χ^2 statistic (10.3) are quite general. Whether to call something a GoF test, multinomial test or otherwise, depends on its application. It doesn't really matter, as will make more sense once we see some examples. The first one is coming soon.

Alright, back to analysis. Pearson's χ^2 statistic has a few advantages over other aggregates, measuring discrepancies between observed and expected quantities under the null. Normalizing by expected counts e_i lends some interpretability since it may be shown that

$$\mathbb{E}\{X^2_{nc}\} = c - 1. \tag{10.4}$$

I'm leaving that to you as a homework exercise in §10.4. Just follow your nose and use Eq. (10.1) liberally. So if you have an observed x^2 that is much larger than $c-1$ you'll probably reject. Small x^2 aren't interesting since those support the null: observed is close to expected.

[4] https://en.wikipedia.org/wiki/Pearson's_chi-squared_test

Only the right-hand tail matters. Other advantages to Eq. (10.3) involve even more math, and we'll get to those after MC.

Here's a simple example to get us going: the dice upgrade of coin flips from Chapter 1. Suppose a six-sided die was tossed 600 times, and a count of the number of times each face (with $1, 2, \ldots, 6$ dots) came up was recorded.

```
o <- c(119, 90, 86, 116, 103, 86)
c <- length(o)
n <- sum(o)
c(n=n, c=c)
```

```
##   n   c
## 600   6
```

That's $c = 6$ faces or classes, $n = 600$ flips, and observed counts stored in o. We wish to know if the die is fair, i.e., $p = (1/6, \ldots, 1/6)$. Is the assumption of a perfectly balanced six-sided die a "good fit" to these data? Although rolling dice is an inherently multinomial enterprise, the idea of comparing observed to expected (under any model) via categories is generic. I think this is where the idea of "GoF" comes from.

Six-hundred rolls was chosen to make it easy to calculate `e` by hand, which would be 100 if the die is fair. But since we have R …

```
p <- rep(1/6)     ## null hypothesis
e <- n*p
e
```

```
## [1] 100
```

Now, the test statistic.

```
x2 <- sum((o - e)^2/e)
x2
```

```
## [1] 11.18
```

Since `x2` is two times larger than the distance between its mean (10.4), which is $c - 1 = 5$, and the smallest possible x_2 of zero, my money is on rejecting the null (barely). No need to guess, let's see what the sampling distribution says.

GoF test by MC

As with coin flips, the program here involves virtual dice rolling. Under $\mathcal{H}_0$, each roll is a uniform sample of the face values $\{1, \ldots, 6\}$. For example …

```
sample(1:c, 1, prob=rep(1/c, c))
```

```
## [1] 3
```

You don't actually need a `prob` argument when p is uniform, but I wanted to make sure you knew what to do for other p. A total of n such rolls may be obtained via replacement

with simple modification. Rather than record each of n tosses individually, it's simpler (and sufficient) to have a summary of how many times each face came up. The `table` command can do that. This time without the redundant `prob` argument …

```
table(sample(1:c, n, replace=TRUE))
```

```
##
##   1   2   3   4   5   6
## 105  95  90 102 110  98
```

See how we get a c-vector, just like `o`. All that's left is to put that into a big MC loop. This code is a little on the slow side because `sample` and `table` functions incur substantial overhead. I'll show you a faster, and more streamlined way to do it momentarily.

```
N <- 100000
X2s <- rep(NA, N)
for(i in 1:N) {
  Os <- table(sample(1:c, size=n, replace=TRUE))
  X2s[i] <- sum((Os - e)^2/e)
}
```

Figure 10.1 shows the empirical sampling distribution for X^2 along with observed x_2. No reflection this time. GoF tests are right-tailed since only large distances between observed and expected offer evidence against the null.

```
hist(X2s, main="")
abline(v=x2, col=2, lwd=2)
legend("topright", "obs", lwd=2, col=2, bty="n")
```

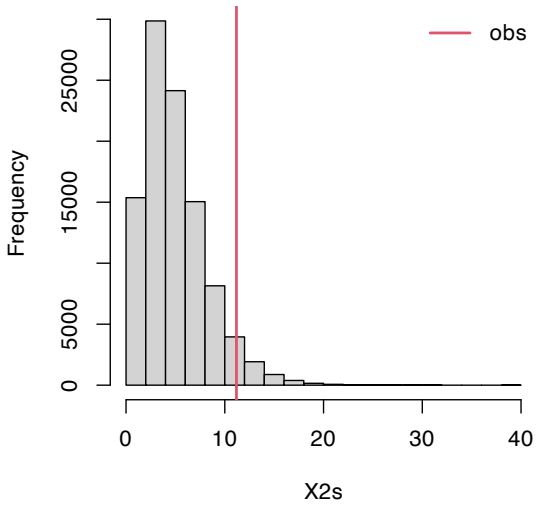

FIGURE 10.1: Empirical sampling distribution of X^2 for a GoF test on dice rolls.

It's a close call. Here is the p-value …

```
pval.mc <- mean(X2s > x2)
pval.mc
```

```
## [1] 0.04779
```

… which falls on the "reject" side (this is not a fair die) of 5%. If you're concerned you can do an even larger N. I tried $10\times$ bigger below but still saw $\psi < 0.05$. So it's a close call. Decisively, a reject "by the book", but I wouldn't bet the farm on it.

In the example above, I was deliberate in avoiding direct use or reference to a multinomial distribution. Understanding and implementing a GoF test via MC doesn't require anything fancy, just like with coin flips in Chapter 1. Yet even in Chapter 1 it helped to recognize that counts of heads in many coin flips follows a binomial distribution. If nothing else, this streamlined MC simulation. You can play the same game here.

```
drop(rmultinom(1, size=n, prob=rep(1/c, c)))
```

```
## [1] 101  88  92 111 106 102
```

This time the `prob` argument *is* required. It's not the same output as `table(sample(...))` above because random numbers are involved. But it has the right format out-of-the-box. It's also faster, computationally speaking. You're probably wondering about that `drop` command. This isn't strictly necessary, and I don't use it in the MC loop below. What you get from `rmultinom` is a `matrix`. Since I only asked for one sample, that matrix is $c \times 1$ so it takes up a lot of vertical space when printed to the screen, or to pages of this book. The `drop` function will reduce the dimension of the object in its argument if possible, in this case from a `matrix` to a `vector`.

Here's the new MC. We don't really need to do it again, but I like it because it lets us explore MC error a little via $10\times$ larger N.

```
X2s10 <- rep(NA, 10*N)
for(i in 1:(10*N)) {
  Os <- rmultinom(1, size=n, prob=rep(1/c, c))
  X2s10[i] <- sum((Os - e)^2/e)
}
mean(X2s10 > x2)
```

```
## [1] 0.04773
```

Observe how this agrees with previous results up to at least the first significant digit, and is still less than 5% when rounded. Another reason `rmultinom` is better than `table(sample(...))` is that there's some chance, especially if n is large relative to c, that a simulation won't include a representative count from all categories in $1, \ldots, c$. When that happens, `table` omits a count for those categories (which should be zero), providing a vector on output that has fewer than c entries. The whole simulation will break if that happens, so be aware. Using `rmultinom` is more robust. It includes zero counts, so you always get a c-vector on output.

In the homework exercises of §1.5 I asked you to do a one-liner for the binomial test. You can do that here too, but it's neater as a two-liner: one to calculate all $N \times c$ samples of O, arranged in a matrix, and another to `colSum` the statistic. Give it a try in §10.4.

GoF test by math

It can be shown that

$$X^2_{nc} \sim \chi^2_{c-1} \quad \text{as} \quad n \to \infty. \tag{10.5}$$

Now you know how a χ^2 distribution is involved in Pearson's statistic. I warned you that I was just going to quote some results when it came to the math. But let me argue that this makes sense via circumstance. You already know the mean (10.4) matches exactly, and that particular result isn't asymptotic. Variances match pretty closely too, which may be verified via MC.

```
c(mc=var(X2s), chi2=2*(c - 1))    ## see Wikipedia for chi2 var formula
```

```
##    mc  chi2
## 10.05 10.00
```

It makes sense that there would be $c-1$ degrees of freedom (DoF). When c probabilities are involved, and which must sum to 1, there are really only $c-1$ free quantities. The last one may be determined by the others: $p_c = 1 - \sum_{j=1}^{c-1} p_j$. Finally, we know that normalized Gaussian RSSs are χ^2, e.g., Eq. (3.15). Non-Gaussian ones enjoy similar asymptotic results via CLT. However, the CLT is for sums (or means) of n quantities not c, and the limiting distribution is Gaussian not χ^2.

None of that is proof of anything, but the technical details are way outside the scope of this book. In fact, you won't find it even in more technical books. Results like those in Eq. (10.5) come up lots, especially in non-P contexts. Measuring distance between observed and expected quantities via normalized RSS (10.3) makes sense, and those are asymptotically χ^2. Think of it like a CLT for normalized RSSs. Don't go Googling "CLT for normalized RSSs". You won't find anything because I made that up, but I still think it's a good way to remember it.

Figure 10.2 illustrates how to use that result (10.3) on the dice example, though I hope this is becoming second nature. The view is similar to the empirical sampling distribution provided by Figure 10.1.

```
x2grid <- seq(0, 8*(c - 1), length=1000)
plot(x2grid, dchisq(x2grid, c - 1), type="l", lwd=2)
abline(v=x2, col=2, lwd=2)
legend("topright", "obs", lwd=2, col=2, bty="n")
```

A p-value calculation involves integrating into the right tail, and is in close agreement with our MC sum from earlier.

```
pval.asym <- pchisq(x2, c - 1, lower.tail=FALSE)
c(mc=pval.mc, asym=pval.asym)
```

```
##      mc    asym
## 0.04779 0.04793
```

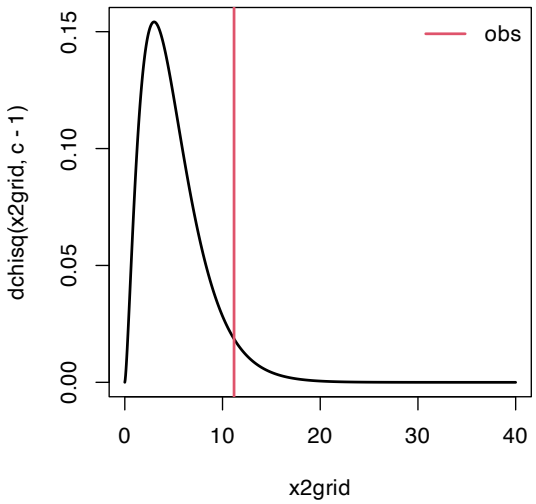

FIGURE 10.2: Asymptotic sampling distribution of X^2 for GoF test on dice rolls.

The agreement tightens with larger N in the MC. Since $n = 600$ is large, the asymptotic approximation is probably pretty accurate, although you never can be sure.

The library function `chisq.test` built-into R provides an automation. All that's required is a vector of observed counts `o`. Everything it needs is derived from `o` including `n = sum(o)` and `c = length(o)`.

```
pval.lib <- chisq.test(o)$p.value
pval.lib
```

```
## [1] 0.04793
```

That one matches the asymptotic result I calculated by hand above. Testing non-uniform `p` requires specifying a `p=` argument, but that's not absolutely necessary otherwise since `p=rep(1/length(o), length(o))` by default.

Interestingly, `chisq.test` has a `simulate.p.value` option that, according to the documentation, provides a p-value following a MC test due to Hope (1968). As far as I know, this is the only built-in library function for testing simple hypotheses in R that supports MC. The actual code for that simulation comes from Patefield (1981) and is in Fortran. Both were way ahead of their time. Simulation was not popular in top statistics journals even as late as the 1990s. Computing was a different beast in 1968.

That's it. Now you know how to perform a GoF test, or a Pearson χ^2 test. As I've hinted, one (GoF) is technically a special case of the other (Pearson χ^2), but it's a subtle distinction depending on application. You'll see what I mean as I take you through more examples, beginning in the next section.

Operationally speaking, both are multinomial tests. Much of our analysis – especially the MC – circumvented multinomials except to make a conceptual connection. I've left a homework exercise to help convince you that you know how to work with multinomials. A multi-

nomial connection makes everything seem parametric (P), but I'm presenting this in the non-P part of the book. It's a matter of perspective. Let me show you.

10.2 P or non-P?

Rolling dice is pretty basic. Here are two more involved examples that I think will help demonstrate the capabilities of a GoF test, and also clarify the way in which it's non-P, even though it may be applied in a P context. Larry Wasserman put GoF tests in his *All of Statistics* book, not his *All of Non-parametric Statistics* book. So it's definitely in a gray area. Inspecting the probabilities of a few categories via expectations is not really non-P, is it? I think it depends on how few a few is, and where those probabilities come from.

Is it Gaussian?

Like, is it cake?[5]

Data in nels_math.csv[6] on the book webpage comes from a 2002 (national) Educational Longitudinal Study (ELS) involving a survey of students' math scores from a sample of schools across the United States. I became aware of this data while studying Peter Hoff's[7] book titled *A First Course in Bayesian Statistical Methods*[8] (Hoff, 2009). In that book Hoff provides several variations on the analysis of these data, and all of them are based on Gaussian distributions. Is that reasonable? Let's take a look.

```
nels <- read.csv("nels_math.csv")
y <- nels[nels[,1] == 1, 2]          ## math scores in first school only
n <- length(y)
n
```

```
## [1] 31
```

We won't handle the entire data set here, just the scores for students in the first school. There are 100 schools in total, so there's a lot of data to play with. Each row corresponds to a student, and each entry/column records that student's school, math score, and socio-economic status (SES), a measure of affluence for the family that student belongs to. Ignore that last column for now, too. However, an interesting subject of inquiry might be to see if there's any association between SES and score (Chapters 7 and 12).

When entertaining a model based on a certain distribution, like a Gaussian, it helps to be specific about which Gaussian. What settings for μ and σ^2? MLEs, or bias-corrected analogs, are nice and automatic and have good properties.

```
ybar <- mean(y)
s2 <- var(y)
```

Now the question is more specific: is *this* Gaussian a good fit?

[5] https://en.wikipedia.org/wiki/Is_It__Cake?
[6] https://bobby.gramacy.com/hipp0/nels_math.csv
[7] https://pdhoff.github.io
[8] https://pdhoff.github.io/book/

Quick digression on scope. You can do this with other distributions (with any parameters θ), not only Gaussian. Call it a *reference distribution* $\mathcal{F}$ if you want to be generic. Gaussian is a good place to start.

You may be familiar with other tools designed to assess GoF. Two visual options are provided in Figure 10.3, based on a simple histogram (left) and quantile-quantile (QQ) plot[9]. Both offer a comparison between empirical and theoretical distributions. If you're not really sure what a QQ-plot is, that's fine. I'm not going to totally derail us to explain that one here. Visual tools are powerful. But here my focus is on providing a statistical test, something quantitative and methodical. From a certain perspective, statistical tests and other procedures may be seen as merely automating, and lending precision and mathematical/scientific heft to, visual checks and other judgment calls from data.

```
par(mfrow=c(1, 2))
ygrid <- seq(ybar - 3*sqrt(s2), ybar + 3*sqrt(s2), length=1000)
hist(y, freq=FALSE, main="", xlim=range(ygrid))
lines(ygrid, dnorm(ygrid, ybar, sqrt(s2)), col=2, lty=2, lwd=2)
legend("topright", "MLE Gauss", col=2, lty=2, lwd=2, bty="n")
qqnorm(y, main="")
qqline(y, col=2, lwd=2)
summary(dnorm(ygrid, ybar, sqrt(s2)))
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## 0.000394 0.002817 0.011487 0.014760 0.026722 0.035461
```

```
legend("bottomright", "1:1 line", col=2, lwd=2, bty="n")
```

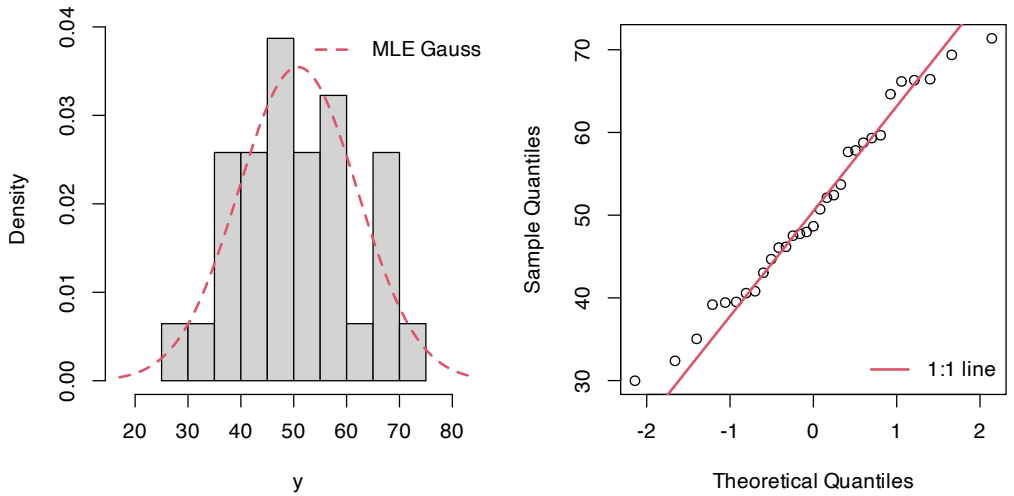

FIGURE 10.3: Visual checks of GoF of a Gaussian to the (N)ELS math data for school one: histogram (left) and QQ-plot (right).

From the figure, it's a close call. The histogram of the raw data (left panel) is in the same "ballpark" as the estimated Gaussian – that's the job of the MLE! – but the tails look

[9] https://en.wikipedia.org/wiki/Q–Q_plot

"light" to my eye. There's that strange dip in the 60-65 range. The QQ-plot shows a similar divergence at the tails, though it indicates less concern elsewhere. Both judgments, while rooted in visuals that have clear statistical justification, are ultimately qualitative.

A GoF test can add precision, procedure and formalism to a determination that a particular distribution offers a good fit to observed data. If the fit is good, any downstream assessments based on that distribution, such as confidence intervals (CIs) or t-tests, etc., which leverage that particular distributional form can be trusted. If the fit is bad, then any conclusions based on the assumed distribution are suspect. Either choose a different distribution, or go fully non-P.

Alright, back to testing. We have a hammer: a test based on categories. And so we need to turn our problem nail: make our data fall into categories that can be matched with probabilities and expectations. There are many ways to do that. Here's one I came up with, which is probably not very different from other things you'd find elsewhere.

Create ten evenly spaced bins from the smallest to the largest value in the training data.

```
c <- 10
bins <- seq(min(y), max(y), length=c + 1)  ## c + 1 endpoints give c bins
mids <- bins[-11] + diff(bins)/2           ## for visuals later
```

My choice of ten is somewhat arbitrary. It's desirable to have several observations in each bin, so you can't have too many bins if you only have 31 data points, as with math scores observed for one school. If you have more data you can have more bins, loosely speaking. Having the bins be equally spaced also makes sense, but can be tinkered with if desired, as I'm about to do next.

Gaussians have real-valued support $(-\infty, \infty)$, so it will help to change the outermost bins to reach out to the edge of the universe.

```
bins[1] <- -Inf
bins[c + 1] <- Inf
```

Next, calculate the probability of a data point falling in each bin according to the reference distribution, which is Gaussian in our example. The probability for a bin with ends (a, b), say p_{ab}, could be calculated as a difference in (cumulative) distribution via the cdf:

$$p_{ab} = \Phi\left(\frac{b-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right) \qquad \text{for Gaussians,}$$

$$\text{or} \quad p_{ab} = \mathcal{F}(b) - \mathcal{F}(a) \qquad \text{generically.}$$

Here's that in code for all (a, b) provided by `bins`. At the same time, the loop tallies a count of the number of observed points in each bin from `y`.

```
o <- p <- rep(NA, c)
for(i in 1:c) {
  p[i] <- pnorm(bins[i + 1], ybar, sqrt(s2))
  p[i] <- p[i] - pnorm(bins[i], ybar, sqrt(s2))
  o[i] <- sum(y < bins[i + 1] & y >= bins[i])
}
```

Figure 10.4 offers a visual of those bins, probabilities and observations. Bin probabilities look a little wonky at the tails because (a) these data don't fit the distribution well out there; and (b) from stretching to $\pm\infty$. Recall that the bins were determined by the observed data extremes. Try not to be bothered by the `4.25*` in the code. Probabilities (always in $[0, 1]$) and densities (always non-negative) are not on the same scale, so I re-scaled the Gaussian density for my visual.

```
sfun <- stepfun(c(bins[-c(1,c + 1)]), p)
plot(sfun, xlab="y", ylab="bins & truth", main="",
  xlim=range(ygrid), ylim=c(0, 0.16), lwd=2)
lines(ygrid, 4.25*dnorm(ygrid, ybar, sqrt(s2)), col=2, lty=2, lwd=2)
text(mids, rep(0.02, length(mids)), o)
text(23, 0.02, "o:")
legend("topleft", "p", lwd=2, bty="n")
legend("topright", "MLE Gauss", col=2, lty=2, lwd=2, bty="n")
```

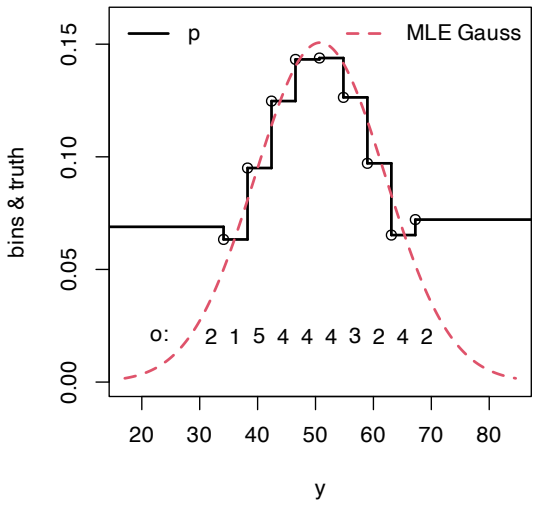

FIGURE 10.4: Visualizing classes categorized by binning an estimated Gaussian for NELS math scores.

In spite of wonky-looking probabilities in the "tail bins", there's reasonably good agreement between observed bin counts, our `o`, and those probabilities `p`. These may be readily converted into expected counts `e`. Calculating Pearson's statistic and subsequent testing may commence.

```
e <- p*n
x2 <- sum((o - e)^2/e)
```

Below, find our MC from much earlier (§10.1).

```
for(i in 1:N) {
  Os <- rmultinom(1, size=n, prob=p)
  X2s[i] <- sum((Os - e)^2/e)
}
```

Next comes p-value calculations, including by MC, math and a library subroutine. For some reason the library gives a warning if you simply call `chisq.test(o)`. I'm not sure why that is. Perhaps with small n and c there's worry that a χ^2 approximation (10.5) is inaccurate. You already know that I worry about that all the time! This is exactly what MC simulation is for, and so I decided to give `simulate.p.value=TRUE` a try.

```
c(mc=mean(X2s > x2), math=pchisq(x2, c - 1, lower.tail=FALSE),
  lib.mc=chisq.test(o, p=p, simulate.p=TRUE)$p.val)
```

```
##     mc   math lib.mc
## 0.8832 0.8740 0.8841
```

In any case, everyone agrees. I cannot reject the null that a Gaussian is reasonable for these data. Peter Hoff is breathing a sigh of relief – as if! By the way, his is a wonderful entry-level book on Bayesian stats – way ahead of its time blending code with serious methodological exposition, and an inspiration for many of my tutorials and books, like this one.

What *really* just happened? A very parametric fit – a Gaussian, the most vanilla of all ice creams – was sliced and diced with a mandoline[10] and investigated under a multinomial microscope. Both are very "P" procedures. Parameters were estimated for one model, $\theta = (\mu, \sigma^2)$ for the Gaussian, and used to derive parameters p for the other. But I want you to think about it as non-P because of the mandoline.

The bigger the training data size n, the finer the mandoline can be, yielding larger numbers of classes, c. Any modeling apparatus whose DoF, which in this case is c, can grow and thereby organically refine its resolution on the data-generating mechanism, is considered non-P. You could say GoF tests are non-P because they leverage large DoF, if that helps you remember. Some setups have this in the extreme, like with ranks whose DoF grows identically with n. Neural networks[11], especially deep[12] ones, and kernel-based methods[13] utilize DoF that grow faster than n. Crazy, right? Some of the most advanced methods in use today are non-P.

While on the topic of DoF, notice that my GoF procedure on the Gaussian uses the data twice: once to estimate parameters $\widehat{\theta} = (\hat{\mu}, s^2)$, and then again to calculate the test statistic (10.3). Double-dipping like that usually costs you some DoF in a χ^2 sampling distribution. See discussions in Chapters 3, 5, 6, 7. This is becoming old hat. If the dimension of θ is p, then $X^2 \sim \chi^2_{c-p-1}$ by deduction.

It turns out this is mostly correct. On the one hand, it's really splitting hairs. The χ^2 sampling distribution (10.5) is already an asymptotic result that is only accurate for large n. In all likelihood, once n is "large enough" subtracting off a few extra DoF won't change things much. If you want to get really technical, Chernoff and Lehmann (1954) show that the actual asymptotic distribution lies "somewhere in between" a χ^2 with $c-p-1$ and $c-1$ DoF. That's not super helpful, except as justification to do whatever you feel like.

[10] https://en.wikipedia.org/wiki/Mandoline
[11] https://en.wikipedia.org/wiki/Neural_network_(machine_learning)
[12] https://en.wikipedia.org/wiki/Deep_learning
[13] https://en.wikipedia.org/wiki/Kernel_method

Hardy–Weinberg equilibrium

Here's one of my favorite examples of all time, even though I know little about genetics. Others like it too; it's a classic! Many of the initial applications of Pearson's statistic are from genetics, coinciding with a boom of untested theories around the time of the original paper (Pearson, 1900) at the turn of the 20th century. More on that with exercises in §10.4.

Consider alleles "A" and "a" with population frequencies $1-\theta$ and θ respectively. Under random mating, the Hardy–Weinberg (HW) principle[14] provides that, in equilibrium (i.e., no external forces such as natural selection or mutation) expected genotype frequencies are $(1-\theta)^2$ and θ^2 for homozygotes "AA" and "aa" and $2\theta(1-\theta)$ for heterozygotes "Aa".

In a sample from Hong Kong in 1937, blood types occurred with the following frequencies.

```
o <- c(M=342, MN=500, N=187)      ## read as M=AA, MN=Aa, N=a
c <- length(o)
n <- sum(o)
c(c, n)
```

```
## [1]    3 1029
```

Does this follow the HW law?

Before jumping in, let's unpack that a little. HW are basically handing us a multinomial with $c = 3$ classes, but one whose probabilities are determined by an unknown parameter θ. The question asks if these blood type data "fit" or follow a HW law. That question may be answered statistically by determining if a multinomial is a good fit for some parameter θ.

What parameter θ? The best one, of course! So this example is a bit of a hybrid between the previous two. With dice, "nature" provides c classes and our "want to know" directly furnishes probabilities. There are no unknown parameters. With math scores, a "mandoline" provides c classes, and the Gaussian provides a probability model and unknown parameters $\theta = (\mu, \sigma^2)$ to estimate. Here, with HW, everything is coupled together. "Nature" provides $c = 3$ classes, HW provides multinomial probabilities $p = ((1-\theta)^2, 2\theta(1-\theta), \theta^2)$, but they're determined up to an unknown parameter θ. Rather than maximize the likelihood for θ offline, like with the Gaussian, it must be done inline with the multinomial.

Maximizing the likelihood for θ requires a multinomial mass function, which I've been avoiding until this point. If $O \sim \text{Multinom}(n, p)$, then

$$\begin{aligned} \mathbb{P}(O_1 = o_1, O_2 = o_2, \dots, O_c = o_c) &= f(o_1, o_2, \dots, o_c; p) \\ &= \binom{n}{o_1\, o_2\, \cdots\, o_c} p_1^{o_1} p_2^{o_2} \cdots p_c^{o_c}. \end{aligned} \tag{10.6}$$

Notice how that generalizes the binomial (1.3), although at that time in Chapter 1 I didn't use the word "binomial", *and* I dropped the ugly normalizing combinatorial coefficient:

$$\binom{n}{o_1\, o_2\, \cdots\, o_c} = \frac{n!}{o_1! o_2! \cdots o_c!}.$$

You're more experienced now, so I can explain that this constant is technically necessary to ensure that probability masses for o add up to 1. Showing you involves writing out c nested

[14] https://en.wikipedia.org/wiki/Hardy–Weinberg_principle

sums, so I'll skip that here. As in Eq. (1.3), a fully-normalized mass (10.6) is often written proportionally as

$$f(o_1, o_2, \ldots, o_c; p) \propto \prod_{j=1}^{c} p_j^{o_j}.$$

This is legit because the combinatorial coefficient is constant with respect to probabilities $p = (p_1, \ldots, p_c)$, and those are the main object of interest. Multiplicative constants become additive when logged, and then they vanish in the derivative, which is the next step in maximizing the likelihood (Alg. 3.1). So that coefficient wasn't necessary in the $c = 2$ binomial case in §3.1, and nor is it here. Now I get to provide you another look at that calculation, but for $c = 3$ and $p = ((1-\theta)^2, 2\theta(1-\theta), \theta^2)$. Let's go!

Below, $\propto$ means there's a dropped multiplicative constant in the case of the likelihood, and a dropped additive one in the case of its logarithm. That happens a couple of times, so be careful!

$$\begin{aligned}
L(\theta) &\propto ((1-\theta)^2)^{o_1} (2\theta(1-\theta))^{o_2} (\theta^2)^{o_3} \\
\ell(\theta) = \log L(\theta) &\propto 2o_1 \log(1-\theta) + o_2 \log\theta + o_2 \log(1-\theta)) + 2o_3 \log\theta \\
&= (2o_1 + o_2) \log(1-\theta) + (o_2 + 2o_3) \log\theta. \\
0 \stackrel{\text{set}}{=} \ell'(\theta) &= -\frac{2o_1 + o_2}{1-\theta} + \frac{o_2 + 2o_3}{\theta} \\
\theta(2o_1 + 2o_2 + 2o_3) &= o_2 + 2o_3 \\
\hat{\theta} &= \frac{o_2 + 2o_3}{2(o_1 + o_2 + o_3)}
\end{aligned}$$

Using `o` in R.

```
that <- as.numeric((o[2] + 2*o[3])/(2*sum(o)))
that
```

```
## [1] 0.4247
```

The `as.numeric(...)` part above isn't essential. I like it because it strips away the names ("M", "NM", and "M") I assigned to the components of `o`, and which I shall put back momentarily. Plugging in gives $\hat{p} = ((1-\hat{\theta})^2, 2\hat{\theta}(1-\hat{\theta}), \hat{\theta}^2)$.

```
phat <- c(M=(1 - that)^2, MN=2*that*(1 - that), N=that^2)
phat
```

```
##      M     MN      N
## 0.3310 0.4887 0.1804
```

That's everything required for the GoF test. Cut-and-paste from here on out … I hope you don't mind if I lump everything into one big code block. (It's really not much code.)

```
e <- phat*n
x2 <- sum((o - e)^2/e)
for(i in 1:N) {
```

```
  Os <- rmultinom(1, size=n, prob=phat)
  X2s[i] <- sum((Os - e)^2/e)
}
c(mc=mean(X2s > x2), math=pchisq(x2, c - 1, lower.tail=FALSE),
  lib=chisq.test(o, p=phat)$p.val)
```

```
##     mc   math    lib
## 0.9832 0.9839 0.9839
```

Those are all basically the same and a resounding failure to reject the null. It seems that the HW principle is good to go for blood types in Hong Kong.

10.3 Homogeneity and independence

Pearson χ^2-tests are generic. I keep repeating that, in part to apologize for not showing you all variations and applications. (I left you one in the homework.) Anywhere you can compare observed to expected, broken down into a discrete and finite number classes, a χ^2 statistic offers you a non-P test based on a multinomial.

Here I'm going to cover two very similar tests, known as the (Pearson χ^2) test for independence[15] and homogeneity[16], respectively. They're *almost* the same test. In the classical/math way they are operationally identical, meaning you do exactly the same procedure but interpret the situation and results differently. This is a bit of a fudge, but any closed-form analysis is approximate via asymptotic results anyways. When using MC you can faithfully virtualize a distinction between the two setups. In spite of that, they still tend to come out about the same.

I plan to take some liberties with the presentation here in an attempt to cover two, similar things at once without being overly pedantic. Basically, I'm going to be even less formal than usual, and I hope you won't mind. Both tests involve data that may be summarized in an $r \times c$ CT.

Pearson's test for homogeneity can be thought of as an r-fold upgrade to the multinomial/GoF procedure at the start of the chapter. See Table 10.2. Instead of one population sorted into c classes or categories, there are r populations. Rather than comparing classification(s) to a particular distribution, as specified by $\mathcal{H}_0$, they are compared to one another on relative terms.

Think of it as a classification-based ANOVA (6.3). The "want to know" is if each population (rows of the table) has the same classification probabilities as the others, or not. Specifically $\mathcal{H}_0 : O_{ij} \sim p_j$ for all $i = 1, \dots, r$, $j = 1, \dots, c$. $\mathcal{H}_1$ that there's some $k \neq i$ such that $O_{ij} \sim p_{ij}$ and $O_{kj} \sim p_{kj}$ and $p_{ij} \neq p_{kj}$. In other words, the null specifies that classification probabilities are the same across the rows; alternative says some are not.

Just like in a GoF test, the number of samples from each population, n_i for $i = 1, \dots, r$, is fixed by design. I mean, by the design of the experiment. The number that falls in each class O_{ij}, and the total for each class C_j for $j = 1, \dots, c$ are random (with particular values

[15] https://en.wikipedia.org/wiki/Pearson's_chi-squared_test#Testing_for_statistical_independence
[16] https://online.stat.psu.edu/stat415/lesson/17/17.1

TABLE 10.2: Homogeneity contingency table.

	class 1	class 2	$\cdots$	class c	total
pop 1	o_{11}	o_{12}	$\cdots$	o_{1c}	$n_1 = \sum_j o_{1j}$
pop 2	o_{21}	o_{22}	$\cdots$	o_{2c}	$n_2 = \sum_j o_{2j}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
pop r	o_{r1}	o_{r2}	$\cdots$	o_{rc}	$n_r = \sum_j o_{rj}$
total	$c_1 = \sum_i o_{i1}$	$c_2 = \sum_i o_{i2}$	$\cdots$	$c_c = \sum_i o_{ic}$	$n = \sum_{ij} o_{ij}$

TABLE 10.3: Independence contingency table.

	col 1	col 2	$\cdots$	col c	total
row 1			$\cdots$		$r_1 = \sum_j o_{1j}$
row 2			$\cdots$		$r_2 = \sum_j o_{2j}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
row r					$r_r = \sum_j o_{rj}$
total	$c_1 = \sum_i o_{i1}$	$c_2 = \sum_i o_{i2}$	$\cdots$	$c_c = \sum_i o_{ic}$	$n = \sum_{ij} o_{ij}$

observed, and represented in lowercase, in the data). Observe that $n = \sum_{j=1}^{c} c_j = \sum_{i=1}^{r} n_i$. A tally of the last row and column totals converges on the same number.

A test for independence is different, but it has nearly the same table. See Table 10.3. The innards of the table, involving each o_{ij}, are the same and to draw attention to that I've left those entries blank. Look at Table 10.2 for those. The final, "total" row of the table(s) is the same too, but I duplicated that part for symmetry. The thing that's different – and it's subtle! – is in the row totals. Those are now notated as r_i rather than n_i. Also notice that I've renamed the rows and columns as simply "row i" and "col j". What's the deal?

Tests for independence involve a single population where each individual sampled from that population is sorted by two categorizations. The CT generically notes these as "row" and "col", respectively, of which there are r and c classes. A crucial distinction here is that row totals R_i are *not* fixed by the experimental design; they're random. Although particular values r_i are observed for particular data, we would not presume to know in advance how things would be sorted by either categorization (both C_j and R_i are random).

The "want to know" is if these two classifications are dependent on one another. (The null is always the simpler option, that they are independent.) It's not that they have the same probabilities, but rather: does knowing a row class determine what the col class is, or vice versa? The null assumes that it does *not*.

This is super subtle, and sometimes it's not clear which situation is appropriate. As a practitioner, you must decide if you have separate populations across the rows, and a single classification, or one population and two classifications. I'll give you two examples momentarily, but first we need to talk about estimation.

The "model" behind both tests is multinomial: $O \sim \text{Multinom}(n, p)$. I put model in quotes because this is really just a choice of convenience. Most would regard the procedure as non-P because the number of DoF – the number of probabilities p in the multinomial – is large. There's one for each o_{ij} entry in the table, so the dimension of p is $r \times c$. In that sense

there isn't really a data-generating mechanism at all, just some probabilities that may or may not depend on the row/col distinction under the null hypothesis.

Although the tables and null hypotheses for the two tests are different, the probabilities in play, and how those translate into expected counts and compare to observed counts via the test statistic, are the same. Under both nulls, rows are distinct from columns, where "distinct" either means different or independent, respectively. Consequently, both sets of probabilities for O_{ij} use p_{ij} via the following formula. Under the null …

$$p_{ij} = \frac{n_i}{n} \times \frac{c_i}{n} \quad \text{or} \quad p_{ij} = \frac{r_i}{n} \times \frac{c_i}{n}, \quad \text{for} \quad i = 1, \ldots, r \text{ and } j = 1, \ldots, c, \tag{10.7}$$

… depending on the test. Those formulas are really the same since $r_i \equiv n_i$; it's just a difference of labeling. Eq. (10.7) encodes that there's no interaction[17] between rows and columns beyond multiplying their marginal[18] probabilities together, as observed in the data. In this way, there aren't actually $r \times c$ free parameters under the null hypothesis. There are $r + c$, since those column and row marginals determine all of the probabilities through their product. Recall that the definition of independence[19] is that the joint factorizes as a product of marginals.

Combining Eqs. (10.7) and (10.1) gives that

$$\mathbb{E}\{O_{ij}\} = np_{ij} = \frac{n_i c_j}{n} \quad \text{or} \quad \mathbb{E}\{O_{ij}\} = \frac{r_i c_j}{n}, \tag{10.8}$$

for all i and j as usual. Again, these two options are operationally identical. Finally, the test statistic is simply a double-sum upgrade of Person's χ^2 from earlier (10.3):

$$x^2 = \sum_{i=1}^{r} \sum_{j=1}^{c} \frac{(o_{ij} - e_{ij})^2}{e_{ij}}. \tag{10.9}$$

To help make these and other aspects concrete, I shall introduce two running examples, one for each test. You can probably make a case for either test being appropriate from a certain perspective, but I'll try to make a particular argument in favor of each one.

The first came from a survey that a company did of its employees' job satisfaction. The population is stratified by seniority from fewer than 10 years in service, to more than 40, in spans of ten. A Likert scale[20] was used to record satisfaction with categories "miserable", "discontent", "content", "happy" and "smitten". Responses are summarized by observed count data provided below.

```
o <- rbind(
    c(2, 6, 20, 12, 6),
    c(4, 6, 20, 14, 7),
    c(2,  12,  28, 30, 11),
    c(1, 2, 18, 22, 12),
    c(2, 3, 4, 6, 8))
colnames(o) <- c("miser", "discon", "content", "happy", "smitten")
rownames(o) <- c("<10y", "10-20y", "20-30y", "30-40y", ">40y")
```

[17] https://en.wikipedia.org/wiki/Interaction_(statistics)
[18] https://en.wikipedia.org/wiki/Marginal_distribution
[19] https://en.wikipedia.org/wiki/Independence_(probability_theory)
[20] https://en.wikipedia.org/wiki/Likert_scale

TABLE 10.4: Job satisfaction contingency table.

	miser	discon	content	happy	smitten	n
<10y	2	6	20	12	6	46
10-20y	4	6	20	14	7	51
20-30y	2	12	28	30	11	83
30-40y	1	2	18	22	12	55
>40y	2	3	4	6	8	23
c	11	29	90	84	44	258

The way I've described it, a test for homogeneity is appropriate since there are five populations (along the rows) and five classes (columns). Those populations are not random since they'd be the same no matter what question was asked, e.g., if they were asked instead what they chose to eat for lunch. Here are the marginal totals.

```
n <- rowSums(o)
c <- colSums(o)
ntot <- sum(n)
```

Purely for aesthetics, I like to build a table in R that looks like Table 10.2. This puts the totals calculated above in an additional row and column, and displays them in a pretty format with a caption. See Table 10.4.

```
tab <- rbind(o, c)
tab <- cbind(tab, c(n, ntot))
colnames(tab) <- c(colnames(o), "n")
rownames(tab) <- c(rownames(o), "c")
kable(tab, caption="Job satisfaction contingency table.")
```

The test statistic (10.9) may be calculated through expectations (10.8). Your instinct might be to deal with double-indexing via nested `for` loops like this.

```
e <- matrix(NA, nrow(o), ncol(o))
for(i in 1:nrow(o)) {
  for(j in 1:ncol(o)) {
    e[i,j] <- n[i]*c[j]/ntot
  }
}
```

There's nothing wrong with that from a correctness perspective. I don't like it because it's a clunky chunk of code, and it's slow. Any time in R that you need `for` loops for simple arithmetic, it'll be much slower than a vectorized alternative. Fortunately, R has an automation for exactly this kind of calculation, which is known as an outer product[21], whose binary operator is $\otimes$. The $r \times c$ matrix of expectations may be abstracted as $\mathbb{E}(O) = n \otimes c/n$ or $\mathbb{E}(O) = n \otimes r/n$ depending on the test/notation. Technically, an outer product is linear

[21] https://en.wikipedia.org/wiki/Outer_product

algebra[22], but all that really means is "abstracted sums of products", and those aren't that scary.

```
efast <- outer(n, c/ntot)
round(sum((e - efast)^2), 10)
```

```
## [1] 0
```

Since the sum of squared differences is numerically zero, these must be the same quantities. It doesn't matter which of `e` or `efast` is used below, except in a MC where a faster and simpler code has obvious advantages. Code below uses these to set up the test statistic. Even though a matrix `e` is stored, a single vectorized `sum` command works transparently over both dimensions.

```
x2 <- sum((o - e)^2/e)
x2
```

```
## [1] 20.21
```

The second example is based on data that could potentially generate a whole suite of examples, and you'll have the opportunity to play with others in the exercises of §10.4. The file assault.csv[23] contains county records on more than 5,000 assaults over a span of several years. The spreadsheet has several columns that would be interesting to analyze with the methods in this section.

```
assault <- read.csv("assault.csv")
```

Consider a table built from the location of the reported incident, and the income of the victim.

```
o2 <- table(assault$locale, assault$income)
o2 <- o2[,c(1,3,4,2)]   ## change column order for increasing income
o2
```

```
##
##             <50K 50-75K 75-150K >150K
##   rural      367    196      76    41
##   suburban   905    839     480   463
##   urban     1131    597     221   187
```

My thinking here is that assault events are random, and so their location and whom they are perpetrated on (and whether or not they choose to report them) is also random. You may quibble with this, but if you allow me to press ahead, this means a test for independence is appropriate. We wish to know if income is independent of locale, and vice versa.

As earlier, I need to collect the relevant statistics and may additionally wish to construct the full table for visualization. See Table 10.5.

[22] https://en.wikipedia.org/wiki/Linear_algebra
[23] https://bobby.gramacy.com/hipp0/assault.csv

TABLE 10.5: Assault victims contingency table.

	<50K	50-75K	75-150K	>150K	r
rural	367	196	76	41	680
suburban	905	839	480	463	2687
urban	1131	597	221	187	2136
c	2403	1632	777	691	5503

```
r2 <- rowSums(o2)
c2 <- colSums(o2)
ntot2 <- sum(r2)
tab2 <- rbind(o2, c2)
tab2 <- cbind(tab2, c(r2, ntot2))
colnames(tab2) <- c(colnames(o2), "r")
rownames(tab2) <- c(rownames(o2), "c")
kable(tab2, caption="Assault victims contingency table.")
```

Expectations (10.8) and test statistic via outer product (10.9) wrap up the calculations here before turning to their sampling distribution.

```
e2 <- outer(r2, c2/ntot2)
x22 <- sum((o2 - e2)^2/e2)
x22
```

```
## [1] 271.3
```

This time I'm going to do things in the opposite order: first math then MC. Why? Because we already know how to do it the math way, and so that'll be really quick. MC involves a slightly bigger code than usual and so will take a little more effort, and require additional discussion.

Homogeneity/Independence via Math

We use the Pearson χ^2 statistic, the same asymptotic sampling distribution, but a different DoF. Multinomial probabilities (10.7) are calculated as an outer product of $r-1$ (row) and $c-1$ (column) freely estimated quantities. Recall that the "last" row and column probability is not "freely estimated" since it's always fixed at one minus the sum of all the others. Although the table of e_{ij}-values is filled with $r \times c$ numbers, they're based on $(r-1) \times (c-1)$ probabilities. Therefore,

$$X^2_{nrc} \sim \chi^2_{(r-1)(c-1)} \quad \text{as} \quad n \to \infty. \tag{10.10}$$

I know this is a little hand-wavy, but it's intuitive as an extension of Eq. (10.5), and the math is absolutely bonkers. A p-value may be calculated by hand, or with the same `chisq.test` library in R as follows. First for job satisfaction …

```
pval.asym <- pchisq(x2, (nrow(o) - 1)*(ncol(o) - 1), lower.tail=FALSE)
```

```
pval.lib <- suppressWarnings(chisq.test(o)$p.value)
c(asym=pval.asym, lib=pval.lib)
```

```
##  asym   lib
## 0.211 0.211
```

I put `suppressWarnings(...)` around the library command, because otherwise it warns that the "Chi-squared approximation may be incorrect". Of course, this is always true. But the reason for the warning in this instance is that at least one of the expected counts is less than five, a threshold built into the software.

```
e[e <= 5]
```

```
## [1] 1.9612 2.1744 3.5388 2.3450 0.9806 2.5853 3.9225
```

Using `simulate.p.value=TRUE` is another option, and one directly comparable to my own MC coming momentarily.

```
pval.libmc <- chisq.test(o, simulate.p.value=TRUE)$p.value
pval.libmc
```

```
## [1] 0.2109
```

This is not much different. It seems that there's not enough evidence to reject the null. This was a homogeneity test, so the conclusion is that job satisfaction does not seem to vary by seniority with the company.

Now, for assaults… This time there are no warnings that require suppression.

```
pval2.asym <- pchisq(x22, (nrow(o2) - 1)*(ncol(o2) - 1), lower.tail=FALSE)
pval2.lib <- chisq.test(o2)$p.value
round(c(asym=pval2.asym, lib=pval2.lib), 10)
```

```
## asym  lib
##    0    0
```

This is an easy reject. Apparently, the distribution of assaults is not independent across locale and income.

Homogeneity/Independence via MC

Although mutinomials are clearly involved, and might be helpful in virtualizing random generation of $r \times c$ tables, I prefer to begin by thinking about how raw data might be generated. By that, I mean n pairs of samples. In a test for homogeneity, row sums are fixed under the null. This means that the first of each of those n pairs is fixed. R code below generates such values for the job satisfaction example by pasting n_1 ones, n_2 twos, etc., up to n_r copies of r.

```
ns <- rep(1:length(n), n)
```

In an actual data record these could be randomly re-ordered, but that doesn't matter since the model assumes that those n draws are iid. Any order is fine. The other member of each pair comes from a distribution, and our best guess for those probabilities is $(c_1/n, \ldots, c_c/n)$. Here's code that generates n of those according to that distribution.

```
Cs <- sample(1:length(c), ntot, prob=c/ntot, replace=TRUE)
```

I don't want to waste space by printing all those pairs onto the pages. Anyways, I don't need those values directly, just their CT summary. Because of how the `table` command in R works, it's risky to directly calculate `table(ns, Cs)`. If any of the random `Cs` values are missing full representation of all classes, then the resulting table won't have the right dimensions. It could be missing entries that should be recorded as zero. The way around that is to use a `factor` type to expressly provide all class labels via its `levels` argument.

```
ns <-  factor(ns, levels=1:length(n))
Cs <- factor(Cs, levels=1:length(c))
table(ns, Cs)
```

```
##    Cs
## ns   1  2  3  4  5
##   1  2  4 18 18  4
##   2  3  4 16 16 12
##   3  3 13 29 23 15
##   4  0  9 12 26  8
##   5  0  0  8 10  5
```

Omitting this `factor` step may still work for you on the examples here, but it won't work in all settings, and for all random MC samples. I added them here because my code would occasionally break without them.

So that's one random sample from the virtual data-generating mechanism under the null hippopotamus, summarized as a CT. Now I just need to do that N times, calculate expectations (10.8), and evaluate Pearson's statistic (10.9).

```
for(i in 1:N) {

  ## sample under the assumption of homogeneity (null hypothesis)
  Cs <- sample(1:length(c), ntot, prob=c/ntot, replace=TRUE)

  ## build observed sample table
  Cs <- factor(Cs, levels=1:length(c))
  Os <- table(ns, Cs)

  ## build expected sample table
  ## (could also use Es <- e here)
  Es <- outer(rowSums(Os), colSums(Os)/ntot)
```

```
  ## Pearson test statistic
  X2s[i] <- sum((Os - Es)^2/Es)  ## could be NA if any Es = 0
}
```

Notice the parenthetical "`## (could also use Es <- e here)`". The simulation, as written, is faithful to the data-generating mechanism and the procedure used to evaluate the test statistic. However, simulated O values come from the same probabilities used to calculate expectations (10.8). While each individual `Es` will not be identical to `e`, on average they should be the same, and so it doesn't matter which is used in the calculation of the testing statistic via MC. I suggest trying it both ways.

How does the p-value compare to those calculated earlier? One downside to having a random `Es` in the denominator is that this could result in undefined values if any are zero. Hence the `na.rm=TRUE` option.

```
pval.mc <- mean(X2s > x2, na.rm=TRUE)  ## guard against rare Es = 0
c(mc=pval.mc, libmc=pval.libmc, asym=pval.asym, lib=pval.lib)
```

```
##     mc  libmc   asym    lib
## 0.2083 0.2109 0.2110 0.2110
```

All are quite similar. This is one of those comparisons which is hard to narrate since the outcome is highly sensitive to the pseudo-random numbers involved. One explanation may be that `simulate.p.value` only uses `B=2000` samples by default, whereas I have two orders of magnitude more than that for `N`. Moreover, the mechanism that `chisq.test` uses to generate random tables when `simulate.p.value=TRUE` doesn't differentiate between our two different cases: homogeneity and independence. The code is poorly documented (I suggest having a look), but it seems to me that the simulation under the hood favors the independence setting where rows and columns are both random.

Now onto the assault example and independence testing, an ideal segue. As you might imagine, the flow will be similar and so will much of the code. Yet, this time we must acknowledge that row totals are not fixed, they're random. Rather than fixed `ns`, we now require random `Rs`. For example, combining with the column sample …

```
Rs <- sample(1:length(r2), ntot2, prob=r2/ntot2, replace=TRUE)
Cs <- sample(1:length(c2), ntot2, prob=c2/ntot2, replace=TRUE)
Rs <- factor(Rs, levels=1:length(r2))
Cs <- factor(Cs, levels=1:length(c2))
table(Rs, Cs)
```

```
##    Cs
## Rs    1   2   3   4
##   1 276 202 105  81
##   2 1196 782 395 343
##   3 931 638 288 266
```

Repeat that lots of times, saving each sampled test statistic. Since n is several thousands for this example – `ntot2` in the code – execution will be on the slow side. A variation

that shortcuts building all n pairs by going directly to the table via `rmultinom` could pay dividends, computationally speaking.

```
for(i in 1:N) {

  ## sample under the assumption of independence (null hypothesis)
  Rs <- sample(1:length(r2), ntot2, prob=r2/ntot2, replace=TRUE)
  Cs <- sample(1:length(c2), ntot2, prob=c2/ntot2, replace=TRUE)

  ## build observed sample table
  Rs <- factor(Rs, levels=1:length(r2))
  Cs <- factor(Cs, levels=1:length(c2))
  Os <- table(Rs, Cs)

  ## build expected sample table
  ## (could also use Es <- e here)
  Es <- outer(rowSums(Os), colSums(Os)/ntot2)

  ## Pearson test statistic
  X2s[i] <- sum((Os - Es)^2/Es)  ## could be NA if any Es = 0
}
```

Calculating the p-value asymptotically led to numerically zero values. Consequently, and because it's been a while since we've had a visual, I've decided to plot the empirical sampling distribution in Figure 10.5. See how no part of the histogram covers the observed value of the statistic? It's not even close! This will lead to an estimated p-value of exactly zero; however, the real interpretation is that $\psi < 1/N = 10^{-5}$. You can see from the figure that it's likely much smaller than even that.

```
hist(X2s, main="", xlim=range(c(X2s, x22)))
abline(v=x22, col=2, lwd=2)
legend("top", "obs", lwd=2, col=2, bty="n")
```

It is worth remarking that in a test for homogeneity, in the $r > 2$ case, one can always perform pairwise tests if the null is rejected. That could help narrow down which things are unlike others. As with ANOVA-like methods, be careful with multiple testing hazards (§6.4). Pairwise tests don't make any sense in an independence context.

10.4 Homework exercises

These exercises help gain experience with χ^2-tests for GoF, homogeneity and independence. There are just a couple of math question(s) this time; the rest are coding and/or data analysis. Don't forget to try each test multiple ways, e.g., via MC or math as appropriate.

Do these problems with your own code, or code from this book. Unless stated explicitly otherwise, avoid use of libraries in your solutions.

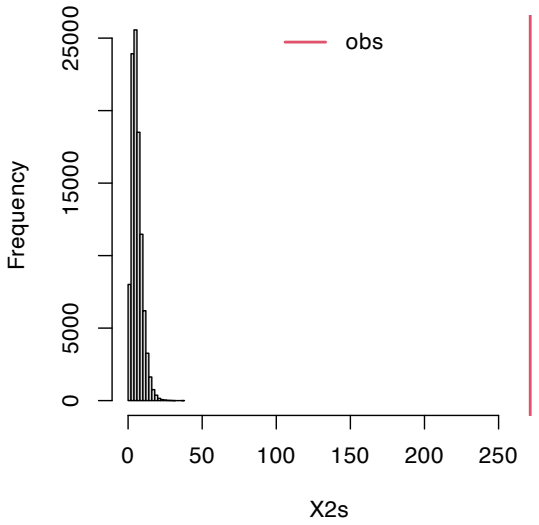

FIGURE 10.5: Empirical sampling distribution of X^2 for an independence test with the assault data.

#1: Expected χ^2

Show (10.4), i.e., that the expected χ^2 statistic exactly matches the mean of its distribution (10.5), no asymptotics required. *Hint: start on the left-hand side, manipulate with linearity of expectations and use facts about multinomial distributions.*

#2: Multinomial MLE

Let $Y \sim \text{Multinom}(n, \theta)$ where θ is a c-vector of probabilities and $\theta_j > 0$ for all $j = 1, \ldots, c$ and $\sum_{j=1}^{c} \theta_j = 1$. *Note that this is the same as* $O \sim \text{Multinom}(n, \theta)$ *from earlier in the chapter, just re-labeled.* Derive the MLE $\hat{\theta}$.

#3: Two-liner GoF test

Given `n` and `x2`, use `rmultinom` to develop code that furnishes N MC samples of χ^2 GoF statistics (§10.1) with one line, and calculates a p-value with a second line.

#4: MC for $r \times c$ tables via multinomial

Rewrite the MC for homogeneity and independence sampling (§10.3) so that it avoids sampling data-pairs (via `sample` for `Cs` and `Rs` followed by `table`) and instead directly creates `Os` via `rmultinom`. *Think of this as an upgrade to the use of* `rmultinom` *for GoF tests (§10.1) for* $1 \times c$ *tables to* $r \times c$ *tables. This is not simple even though the code is short.*

#5: `monty.chisq` for all cases

Write a library function that covers all variations of Pearson χ^2 tests from the chapter. This includes GoF (§10.1), homogeneity and independence tests (§10.1) in both MC and closed-form/asymptotic variations. Use `chisq.test` to check your work, but otherwise develop the implementation yourself. Optionally, incorporate your solutions to #3–4, above.

Hint: many of the following questions are easier after this one is squared away.

#6: Transportation study

Al-Ghamdi (2001) presents a theoretical model for the time elapsed between vehicles on urban roads along with an analysis on observational data collected in Riyadh, the capital city of Saudi Arabia. Those pairs of values, recorded in seconds and categorized into two-second bins, are provided below.

```
veh <- rbind(
  observed=c(18, 28, 14, 7, 11, 11, 10, 8, 30),
  theory=c(23, 18, 16, 13, 11, 9, 20, 8, 19)
)
colnames(veh) <- c("0-2s", "2-4s", "4-6s", "6-8s", "8-10s", "10-12s",
  "12-18s", "18-22s", ">22s")
```

What do you think? Does the theoretical model do a good job of explaining those observed values?

#7: Mendelian inheritance

Gregor Mendel's[24] principles for predicting genetic inheritance experimental tests made validation difficult. One of the earliest applications of Pearson χ^2-tests aimed at a quantitative assessment of the theory in real experiments (Harris, 1912). That (very old) paper[25] reports many excellent experiments that can be used to test your understanding of GoF tests. Here's just one. (Note that the author uses the term "calculated" instead of "expected".)

Crossing a first-generation of pea plants producing yellow, round seeds with pea plants producing green, wrinkled seeds yielded the following observed plants (seeds) in the second generation.

```
o <- c(yr=4926, yw=1656, gr=1621, gw=478)
```

Assuming independent genes, each with a dominant and recessive allele, Mendel's theory predicts a 9:3:3:1 phenotypic ratio. Do these data support this?

#8: Are 911 calls Poisson?

Data below tally the number of 911 emergency calls within the span of an hour, across 50 randomly chosen hours, in a certain jurisdiction. The default model for such data is Poisson.

```
y <- c(5, 4, 0, 0, 6, 1, 3, 4, 1, 1, 0, 1, 2, 6, 4, 3, 0, 3, 5, 3, 2,
  8, 6, 1, 4, 3, 1, 3, 1, 1, 0, 3, 5, 4, 1, 9, 0, 5, 1, 1, 1, 12, 0,
  2, 3, 1, 1, 0, 2, 4)
```

Is it reasonable to use a Poisson to model these data?

Hint: first choose the parameter for the Poisson. Then, `table(y)` *can help you determine bins for* `o`*, however it might be sensible to merge all bins with just one observation.*

[24] https://en.wikipedia.org/wiki/Gregor_Mendel
[25] https://www.journals.uchicago.edu/doi/pdf/10.1086/279323

#9: Math scores for less affluent students

Revisit the (N)ELS math score data, but this time rather than separating out a particular school, look at all math scores at all schools for students in the bottom half of the socioeconomic spectrum.

```
y <- nels$score[nels$ses < 0]
n <- length(y)
n
```

```
## [1] 1021
```

Is it reasonable to study this sample with a Gaussian distribution? *Since this is a much larger sample compared to any single school, consider using a larger/finer mandoline (choose larger c).*

#10: ROI with Gaussian?

Refer back to `roi$company` and `roi$industry` from §2.5. Recall that these data are available as roi.RData[26] on the book webpage.

```
load("roi.RData")
```

Is it reasonable to model either of these samples with Gaussian distributions? In other words, can the P-analyses from the first half of the book be trusted, or should we rely on the non-P ones in the second half? *Be careful not to use bins with too few observations.*

#11: Women's health

Assaf et al. (2016) report on a randomized control trial of almost 50,000 women meeting certain criteria (healthy, age 50-79, etc.), exploring relationships between nutrition and long-term health. Data from that study were summarized and cross-referenced against incidence of coronary heart disease (CHD) for treatment (healthy diet) and control populations.

```
o <- rbind(
  c(1000, 18541),
  c(1549, 27745)
)
colnames(o) <- c("treatment", "control")
rownames(o) <- c("chd.yes", "chd.no")
```

Is there evidence of a difference in CHD incidence for these two groups?

#12: Math and stat grades

CMDA 2006[27] is an integrated topics course taught at Virginia Tech (VT), where I work. Most classes at VT are 3 credits, but this one is 6 credits, because it's basically two classes in one. Although students register once, there are two different professors separately covering math and stats topics, two sets of homeworks, and two sets of midterms. (I have taught

[26] https://bobby.gramacy.com/hipp0/roi.RData
[27] https://catalog.vt.edu/undergraduate/course-descriptions/cmda/

the stats half several times, and much of the material from that class went into this book.) There is a combined final, and ultimately students are assigned just one, combined letter grade. However, it is interesting to see what separated grades *would* be right before the combined final.

Data below record a breakdown of pre-final letter grades on math and stats halves of CMDA 2006 tallied over several semesters/sections. *Category `O` combines letter grades D, F, and any other special cases like withdraw and incomplete.*

```
o <- rbind(
  c(32, 13, 20, 10),
  c(20, 25, 21, 10),
  c(11, 13, 23, 11),
  c(8, 9, 10, 19))
rownames(o) <- c("stats.A", "stats.B", "stats.C", "stats.O")
colnames(o) <- c("math.A", "math.B", "math.C", "math.O")
```

Is performance on the stats half independent of the math half?

#13: Other assault columns

Revisit analysis of the `assault` data from §10.3, but choose two different pairs of columns. For example, look at `police` with `ER` or `year` with `female`. Perform an appropriate (homogeneity or independence) test. Be clear in your explanation of why a particular test was conducted and what the hypotheses and conclusions are.

#14: Median test

Alright, here is perhaps my biggest indulgence in the book. I'm going to lecture to you in a homework exercise. Bear with me, it'll be worth it.

The median test[28] is an application of Pearson's χ^2-test where there are "columns", defined by counts of numbers of samples from populations, arranged in "rows", indicating above and below the median, respectively.[29] Words "columns" and "rows" are in quotes, because actually median test tables are transposed in most presentations. What I mean is, there are actually $c = 2$ rows and r columns for each of r populations. I think that's because a short and fat table looks better on the page than a tall and skinny one.

Let $y^{(n/2)}$ denote the grand median from all $n = n_1 + \cdots + n_r$ samples. Then, let o_{1i} count the number of observations from the i^{th} sample that exceed $y^{(n/2)}$; and let $o_{2i} = n_i - o_{1i}$ be the number that is less than or equal to that value. See the CT depicted in Table 10.6.

The null hypothesis is that all r populations have the same median, versus the alternative that at least two have different ones. Test statistic x^2 is the same as usual (10.3); however, when you consider that $o_{2i} = n_i - o_{1i}$, and that there are only two rows, some simplification arises.

$$x^2 = \frac{n^2}{ab} \sum_{i=1}^{c} \frac{\left(o_{1i} - \frac{n_i a}{N}\right)^2}{n_i} \quad \text{and} \quad X^2 \sim \chi^2_{c-1}.$$

[28] https://en.wikipedia.org/wiki/Median_test

[29] If you read that Wikipedia link, you'll see that the test is under-powered compared to others that I present in the next chapter (or the Wilcoxon test of §9.2 in the case of two populations); making this "question" a perfect segue.

TABLE 10.6: Median test contingency table.

	pop 1	pop 2	$\cdots$	pop r	total
$> y^{(n/2)}$	o_{11}	o_{12}	$\cdots$	o_{1r}	$a = \sum_i o_{1i}$
$\leq y^{(n/2)}$	o_{12}	o_{22}	$\cdots$	o_{2r}	$b = \sum_i o_{2i}$
total	n_1	n_2	$\cdots$	n_r	$n = \sum_i n_i$

As with any other heterogeneity-like test with multiple populations, one can pair them off order to determine which (pairs) are actually different from one another (if the null is rejected). These would be 2×2 tables, for which there are specially built methods like Fisher's exact test[30].

Ready to give it a try?

Shelf height at supermarkets is important for sales. In a study of sales of breakfast cereal, four shelf heights were varied. The list below records sales in thousands of dollars for cereal brands at those heights.

```
sales <- list(
    feet=c(8.1, 6.5, 6.6, 8.1, 7.5, 6.7, 9.1, 7.0),
    knee=c(7.9, 8.1, 7.6, 8.1, 8.5, 7.7, 8.5, 7.6, 8.0),
    waist=c(9.0, 8.8, 9.5, 9.4, 9.1, 9.2, 9.5, 8.9),
    eye=c(8.9, 8.8, 8.6, 8.7, 8.2, 8.3, 8.3, 8.8, 9.3))
```

Do these data indicate that shelf-height affects median cereal sales? Conduct the test via MC and in closed form/asymptotically.

Hint: begin by making Table 10.6 in R.

[30] https://en.wikipedia.org/wiki/Fisher's_exact_test

11

Non-P scale

Scale is to variance as location is to mean. This chapter is the non-P version of Chapter 6's P. It's not common to test variance nonparametrically for just one sample. If that's what you're looking for – a non-P version of §6.1 – there's always the bootstrap (Chapter 8).

Previous chapters covered concepts in pairs, and this one is no different. Here I cover the so-called squared ranks test and Kruskal–Wallis test. Squared ranks targets a comparison of scale, or spread, of samples from two populations – a non-P §6.2. Kruskal–Wallis is like non-P ANOVA. That's an unofficial designation but, like ordinary/P ANOVA (§6.3), juxtaposing residual scales allows locations to be compared across multiple samples.

11.1 Squared ranks

Consider a two-sample setup with $y_1, \dots, y_{n_y}$ and $x_1, \dots, x_{n_x}$ being *mutually independent* but possibly not from the same distribution. Mutually independent is a fancy way of saying that samples are independent from one another, and also (at least in this case) that they're iid within-sample. Interest lies in testing if these samples come from the same distribution, $F_Y = F_X$, i.e., that they are iid across both samples. Competing hippopotami are just like in Eq. (9.2), but with $\mathbb{E}$ swapped for $\mathbb{V}\text{ar}$. In other words, focus here is on variance.

$$\begin{aligned} &\mathcal{H}_0 : \mathbb{V}\text{ar}\{Y\} = \mathbb{V}\text{ar}\{X\} && \text{null hypothesis (“what if”)} \\ &\mathcal{H}_1 : \mathbb{V}\text{ar}\{Y\} \neq \mathbb{V}\text{ar}\{X\} && \text{alternative hypothesis} \end{aligned} \tag{11.1}$$

Writing things out with variances rather than expectations clarifies the mindset, but ultimately if distributions are the same then any aspect is the same. What matters is how a chosen statistic provides a wedge that, under the null hypothesis, reveals a clear probabilistic contradiction in favor of an alternative. Whenever $\mathbb{V}\text{ar}\{Y\} \neq \mathbb{V}\text{ar}\{X\}$ that implies $F_Y \neq F_X$, and sometimes you'll see alternative hypotheses written out that way instead. The preferred statistic involves a study of how far individual data points are from their averages. This isn't exactly “variance”, but it's closely related. Hence hedging the name of the chapter with “scale”.

Convert each y_i and x_j into its absolute deviation from its corresponding sample average. That is, let

$$\begin{aligned} v_i &= |y_i - \bar{y}| \quad \text{for} \quad i = 1, \dots, n_y \\ \text{and} \quad u_j &= |x_j - \bar{x}| \quad \text{for} \quad j = 1, \dots, n_x. \end{aligned}$$

Assign ranks (§9.1) to the combined sample $v_1, \dots, v_{n_y}, u_1, \dots, u_{n_x}$ using average ranks when

there are ties. Let $r_1^{(v)}, \ldots, r_{n_y}^{(v)}$ and $r_1^{(u)}, \ldots, r_{n_x}^{(u)}$ denote those ranks. As with other tests based on ranks there are several common test statistics, and I'll show you a couple of them.

First comes the eponymous squared ranks[1], which are the squared analog of $\bar{r}$ from a Wilcoxon rank sum test (9.3).

$$\bar{r}^2 \equiv \bar{r}^{2(v)} = \sum_{i=1}^{n_y} r_i^{2(v)} \tag{11.2}$$

In words, $\bar{r}^2$ is the sum of squared ranks corresponding to v, which are absolute deviations for y. Ranks for u, via x, are not directly involved, but instead occupy the "negative space". It would be equivalent to using a sum of squared u ranks, being diligent to keep everything else straight downstream. It doesn't matter which sample is labeled with x and which with y. Careful here not to get $\bar{r}^2$ confused with r^2 which is a squared correlation from §7.2. It's unfortunate that both rank, and sample correlation use the letter r.

Following logic similar to Eq. (9.4), we may deduce that

$$\mathbb{E}\{\bar{R}^2\} = \frac{n_y(n+1)(2n+1)}{6} \quad \text{under } \mathcal{H}_0, \tag{11.3}$$

because Eq. (11.2) is tallying n_y (random) parts of a square pyramidal number[2]. You don't have to take my word for it. I'll show how that theoretical expectation compares to a MC one momentarily.

That's pretty much it. Before turning to strategies for assessing the sampling distribution of $\bar{r}^2$, I shall pause to introduce a running example. I hope that will help clarify details of the development so far in code. In keeping with themes from Chapter 9, consider a study tallying the numbers of bicycles on Metro train lines. Data below record counts of bikes observed on Green- (y) and Orange-line (x) trains during randomly chosen commute hours.

```
y <- c(46, 51, 61, 70, 66, 69, 50, 44, 58, 59, 49, 54)
x <- c(74, 72, 90, 51, 65, 63, 100, 57, 82)
ny <- length(y)
nx <- length(x)
```

Here are the relevant calculations in R. First, form absolute deviations from sample averages.

```
v <- abs(y - mean(y))
u <- abs(x - mean(x))
```

Then form the combined ranking.

```
r <- rank(c(v, u))
```

Extract the part of the ranking due to `v` (or `y`) and calculate the test statistic (11.2).

[1] https://en.wikipedia.org/wiki/Squared_ranks_test
[2] https://en.wikipedia.org/wiki/Square_pyramidal_number

```
rv <- r[1:ny]
r2bar <- sum(rv^2)
r2bar
```

```
## [1] 1390
```

As in Chapter 9, it's worth knowing whether or not there are ties in the ranks, since that might affect strategies for determining the sampling distribution.

```
1 - length(unique(r))/length(r)
```

```
## [1] 0
```

No ties, so there's nothing to worry about. In Chapter 9 I had two running examples, one with ties and one without. I did that twice over, for Wilcoxon and signed rank tests, respectively. Things got complicated to keep track of, code wise, and in the end there wasn't much value added except a takeaway that concerns may be overblown. Here, I shall keep it simple with one example, and provide pointers to possible audibles[3] in the presence of ties.

Squared ranks test by MC

Much can be borrowed from the Monte Carlo (MC) for Wilcoxon rank sum tests. Since there are no ties, no special extensions to virtualize random ties in ranks are required. If you do encounter ties, and wish to entertain those, `rank.ties` in rank_ties.R[4] on the book webpage is equally applicable here. Really, the only difference compared to Wilcoxon tests is how the statistic is calculated in the last line of the loop.

```
N <- 1000000
n <- length(r)
R2bars <- rep(NA, N)
for(i in 1:N) {
  Rvus <- sample(1:n, n)
  Rvs <- Rvus[1:ny]
  R2bars[i] <- sum(Rvs^2)
}
```

According to Eq. (11.3), the mean of this distribution may be calculated as follows.

```
R2mean <- ny*(n + 1)*(2*n + 1)/6
c(R2mean, mean(R2bars))
```

```
## [1] 1892 1892
```

Identical, though that's not guaranteed. Either (if they are different) may be used to reflect an observed statistic for visualization purposes. I shall skip that here and go straight to the p-value, but `R2mean` will come back later. For now, note that `r2bar` is less than `R2mean`, so the tail calculation is on the left side.

[3] https://en.wiktionary.org/wiki/call_an_audible
[4] https://bobby.gramacy.com/hipp0/rank_ties.R

```
pval.mc <- 2*mean(R2bars < r2bar)
pval.mc
```

```
## [1] 0.1188
```

There's not enough evidence to reject $\mathcal{H}_0$. For now, it seems that the number of bikes on Green- and Orange-line trains come from the same distribution, at least as regards their scale.

Squared ranks test by math

As with Wilcoxon tests, you can turn evaluation of the mass function of the test statistic under the null into a subset sum problem (SSP). Since ranks are being squared, yielding bigger sums, we have a harder SSP as compared to ordinary ranks. Why? Many more combinations could be formed to reach those larger sums.

Details are left to §B.3. Here I shall just take `dsqranks` and `psqranks` from ranks.R[5] on the book webpage, providing mass and cumulative distribution, respectively.

```
source("ranks.R")
```

Figure 11.1 shows what you get for Metro bikes. Go ahead and peek at it – what a strange distribution! Consecutive integers can have wildly different numbers of ways that squares can sum into them, and thus wildly different masses.

```
r2grid <- 500:3000
dr2 <- rep(NA, length(r2grid))
denom <- dsqranks.denom(ny, nx)  ## faster with denom pre-calculated
for(k in 1:length(r2grid))
  dr2[k] <- dsqranks(r2grid[k], ny, nx, denom=denom)  ## used here
plot(r2grid, dr2, type="l", lwd=2,
  xlab="r2bar", ylab="mass dsqranks(r2bar, ny, nx)")
abline(v=c(r2bar, 2*R2mean - r2bar), col=2, lty=1:2, lwd=2)
legend("topleft", c("r2bar", "reflect"), lty=1:2, col=2, lwd=2, bty="n")
```

At first, I thought I had made a mistake with that visual, but it's correct. Here's how I know. Consider the special case of $n_y = n_x = 2$, representing a very small experiment. That means $n = 4$, and so the squared ranks that are available are as follows.

```
r2 <- (1:4)^2
r2
```

```
## [1]  1  4  9 16
```

You can't make any sum of $n_y = 2$ ranks from those four numbers. There just are six possibilities.

[5] https://bobby.gramacy.com/hipp0/ranks.R

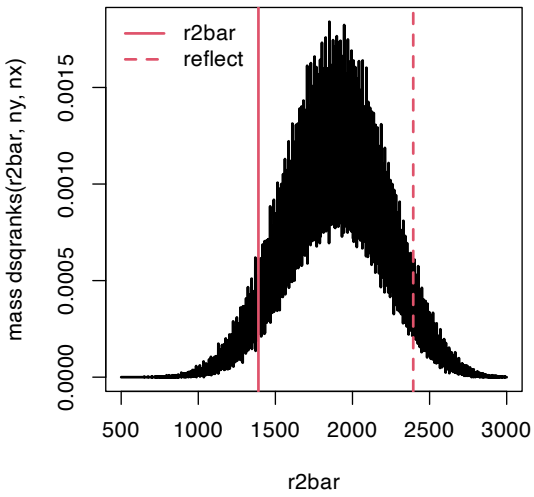

FIGURE 11.1: Exact squared ranks sampling distribution for Metro bikes example.

```
colSums(combn(r2, 2))
```

```
## [1]  5 10 17 13 20 25
```

Anything that's not one of those numbers would have a mass of zero under the sampling distribution. That's a lot of bouncing back and forth, between zero and nonzero values. By the way, you might think you could short-circuit some `dsqranks` calculations by enumerating all nonzero ones via `combn`. You can't, alas, because there are too many for large n and n_y, e.g., when $n = 30$ and $n_y = 15$ you get get an error from R saying "cannot allocate vector of size 17.3 Gb".

Figure 11.1 shows the observed $\bar{r}^2$ and its reflected value. The area under the curve in either tail, determining the p-value, may be calculated as follows. As implemented, `psqranks` is merely a sum of `dsqranks`.

```
pval.exact <- 2*psqranks(r2bar, ny, nx)
c(mc=pval.mc, exact=pval.exact)
```

```
##     mc  exact
## 0.1188 0.1200
```

The MC, which is lots simpler in concept and in code, gives a result that's quite close to the real thing. Technically speaking, the MC sampling distribution is just as crazy as the exact one in Figure 11.1. You wouldn't be able to tell with a histogram. In fact, the histogram looks Gaussian! Its bins are obscuring a nuanced relationship. You'd have to count the number of "hits" for each unique value in `R2bars` and use a `barplot` to see what's really happening. I encourage you to take a look/give that a try.

Speaking of Gaussians, you might not be surprised that the central limit theorem (CLT) is an option. After all, $\bar{r}^2$ is a sum, and we already know its mean (11.3). You might also not be surprised that its variance is a nightmare. I'm just going to give you the z statistic so we can make a quick comparison. Sometimes, this is preferred when there are many ties, and when n is large. I haven't found much difference in my own experience.

$$Z = \frac{\bar{R}^2 - \frac{n_y(n+1)(2n+1)}{6}}{\sqrt{\frac{n_y n_x}{n(n-1)}\sum_i r_i^4 - \frac{n_y n_x}{n^2(n-1)}(\sum_i r_i^2)^2}} \sim \mathcal{N}(0,1) \quad \text{as} \quad n \to \infty$$

Here that is in code. Note that those sums are over all ranks: $\sum_i \equiv \sum_{i=1}^n$.

```
r4sum <- sum(r^4)
r2sum <- sum(r^2)
se <- sqrt(nx*ny/(n*(n - 1))*r4sum - nx*ny/(n^2*(n - 1))*r2sum^2)
z <- (r2bar - R2mean)/se
pval.clt <- 2*pnorm(-abs(z))
pval.clt
```

```
## [1] 0.1153
```

This is very close to the others. All three of these options are automated in the `ANSM5` package on CRAN (Spencer, 2024), where the test is credited to Conover (1999). That citation is to the book I learned it from. However, only the CLT version was provided in that text. Conover only alluded to an exact option in passing, and didn't provide any hint of MC. The library implementation in `ANSM5` is a bit different than mine on all fronts. Its MC calculation is slow and inaccurate. There might be a bug. They make up for it with a better exact version. Smeeton et al. (2025) describe a more purpose-built scheme for enumerating rank partitions, rather than using a general-purpose SSP solver. This makes execution much faster, and seemingly no less accurate, though also an approximation. Finally, their CLT/asymptotic approximation seems a little off.

```
library(ANSM5)
pvals <- rbind(
  c(pval.mc, pval.exact, pval.clt),   ## by hand, calculated above
  c(conover(y, x, do.mc=TRUE)$pval.mc, conover(y, x)$pval.exact,
    conover(y, x, do.asymp=TRUE)$pval.asymp)
)
colnames(pvals) <- c("mc", "exact", "clt")
rownames(pvals) <- c("byhand", "lib")
pvals
```

```
##            mc exact     clt
## byhand 0.1188  0.12 0.11528
## lib    0.5830  0.12 0.09483
```

See what I mean about those `lib` MC and CLT results? Two of these things are not like the others. Although it's possible to increase the number of MC iterations via `nsims.mc`, that didn't seem to change the answer much when I tried it. Another caveat about `conover(...)` is that it may not perform exactly the test you requested. Internal logic overrides exact calculation of p-values for approximate ones in certain cases, such as when there are many ties or when n is large. These rules are sensible, but hard to bypass in situations where you prefer a particular option in spite of accuracy concerns or computational cost, like when you're writing a textbook.

11.2 Kruskal–Wallis

I already said that Kruskal–Wallis (KW) is non-P ANOVA. That means it's a test for heterogeneity in location across m samples. Usually $m \geq 3$ since for $m = 2$ there's always the Wilcoxon test (§9.2). Specifically, suppose our data consist of y_{ij}, for $i = 1, \ldots n_j$ and $j = 1, \ldots, m$. We assume that these samples are mutually independent and wish to test if all $n = \sum_{j=1}^{m} n_j$ of them come from the same distribution. Or, is it that at least one pair of samples (some $j \neq k \in \{1, \ldots, m\}$) come from different distribution(s)?

In other words, the hypotheses are like in Eq. (9.2), but for $m \geq 3$ samples. Alternatively, get rid of the Gaussian in (6.6), and replace μ_j and μ_k, etc., with $\mathbb{E}\{Y_{ij}\}$ and $\mathbb{E}\{Y_{ik}\}$, etc. I'm trying to keep it casual and avoid being too pedantic here. KW is a location-based test, and so the test statistic involves aggregated ranks like a Wilcoxon rank sum test (9.3). Yet KW is also like ANOVA in that it uses sums of squares, via a weighted combination of squared average ranks for each group. Don't worry, I'll be specific, but I wanted to start by giving you something easier to remember.

Let r_{ij} denote ranks of the combined sample of all n observations, with ties yielding averaged ranks as usual. Denote

$$\bar{r}_j = \sum_{i=1}^{n_j} r_{ij}, \qquad \text{for} \qquad j = 1, \ldots, m,$$

as the sum of ranks for each group. Next, combine squares of those rank-aggregates and adjust for their sizes $n_1, \ldots, n_m$.

$$\bar{r}^2 = \sum_{j=1}^{m} \bar{r}_j^2 / n_j \tag{11.4}$$

I'm using the same letter as for a squared ranks statistic (11.2) because this one is also a sum of squared ranks. Eq. (11.4) is a little different since raw data values y_{ij} are ranked with KW (targeting location), whereas a squared ranks test involves absolute deviations v_i and u_j (targeting scale).

Shocker: $\bar{r}^2$ in isn't the only statistic that is used with KW. But it's the simplest one that allows us to get up-and-running quickly. Others are variations on Eq. (11.4) become more important for asymptotic approximations behind closed-form tests. I'll get to those later, and pivot here to introducing an example.

Keeping things familiar, a `list` below collects all Metro fare card data from §9.2.

```
ylist <- list(
  VA=c(80, 76, 56, 67, 73, 58, 51, 65, 68, 61),
  DC=c(83, 66, 71, 82, 81, 89, 97, 59, 74),
  MD=c(67, 94, 83, 98, 35, 73, 29, 36, 60, 105, 34, 84, 89, 76, 79, 92,
    49, 97, 46, 32, 104, 60, 61, 59, 98, 101),
  DC2=c(107, 114, 87, 102, 85, 94, 109, 91, 99, 59, 97, 92, 103, 90, 43,
    108, 61, 111, 87, 112, 109, 115, 89, 105, 109, 83, 35, 114))
```

As with ANOVA, `list` and `data.frame` data structures have advantages when it comes to grouped data. Here, code from §6.3 is borrowed verbatim to extract dimension information, and construct a `data.frame` representation.

```
nj <- sapply(ylist, length)
m <- length(nj)
n <- sum(nj)
ydf <- data.frame(farecards=unlist(ylist), venue=factor(rep(1:m, nj)))
```

If you recall our separate, pairwise analysis of (`VA`, `DC`) and (`MD`, `DC2`) from Chapter 9, you might be able to guess the outcome of the KW test I'm about to take you through here. Figure 11.2 provides a visual of these data. There are certainly differences between each of the samples, but it's not cut-and-dry that they all (don't) have the same mean.

```
boxplot(ylist, border=1:4, ylab="weight")
jit <- rnorm(n, 0, 0.05)
points(as.numeric(ydf$venue) + jit, ydf$farecards, pch=19,
  col=ydf$venue, cex=0.75)
```

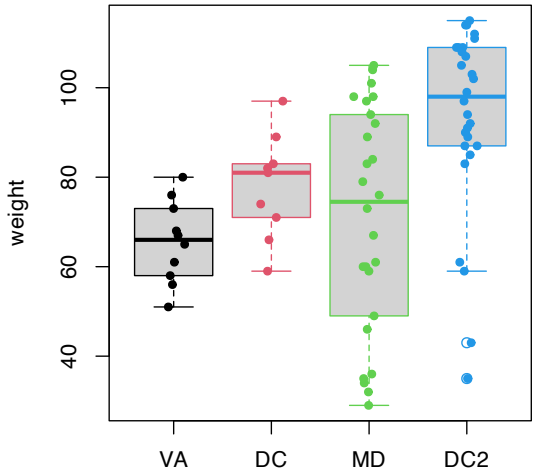

FIGURE 11.2: Visualizing the full Metro fare card data set, with horizontal jitter.

Obtaining ranks for the full data set is easy via `unlist`.

```
r <- rank(unlist(ylist))
c(n, length(unique(r)))
```

```
## [1] 73 50
```

Note that there are a substantial number of ties. Here's a tidy way to extract ranks for each group or sample, regardless of how many there are, and add them up.

```
g <- factor(rep.int(seq_len(m), nj))
```

```
rg <- tapply(r, g, sum)
rg
```

```
##      1      2      3      4
##  212.5  294.0  777.5 1417.0
```

I encourage you to take one step at a time through that code and study what each subroutine is doing. It's not magic. Finally, R code below completes calculation of the test statistic (11.4).

```
r2bar <- sum(rg^2/nj)
r2bar
```

```
## [1] 109080
```

This is an interesting number. Definitely big! Also it's nonsense without something to compare it to.

KW test via MC

As always, just be careful to follow the statistic (11.4) under the null hypothesis, which in this case – as it has been with all methods based on ranks – is based on uniform permutation. Generate random ranks by sampling indices $\{1, \dots, n\}$ without replacement, break them into m groups of size $n_1, \dots, n_m$, sum within each group, and then aggregate their squares weighted by inverse sample size.

```
for(i in 1:N) {
  Rs <- sample(1:n, n, replace=FALSE)       ## random permutation
  Rgs <- tapply(Rs, g, sum)                 ## break into groups and sum
  R2bars[i] <- sum(Rgs^2/nj)                ## calculate statistic
}
```

Without acknowledging ties, this virtualization is not completely faithful to the conditions under which our Metro fare card data was observed. As with squared ranks, you may easily mimic observed ties with `rand.ties`, but I'm skipping that here to keep things moving. Since the statistic is a square, the sampling distribution will have positive support and be heavily right-tailed. See Figure 11.3. In such situations, the test is implicitly one-sided, just like ANOVA. Any small value of $\bar{r}^2$ supports the null.

```
hist(R2bars, main="")
abline(v=r2bar, col=2, lwd=2)
legend("top", "obs", lwd=2, col=2, bty="n")
```

The figure indicates a clear reject, but the additional precision offered by p-value never hurts.

```
pval.mc <- mean(R2bars > r2bar)
pval.mc
```

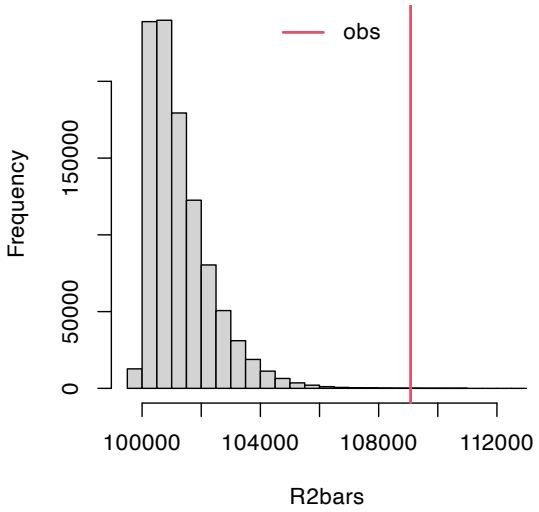

FIGURE 11.3: Empirical sampling distribution of $\bar{R}^2$ for KW on Metro fare card data.

```
## [1] 5.5e-05
```

Since two of the pairs of samples led to rejections of the null in Chapter 9, the outcome here is not surprising: at least two of the samples differ in their mean. As always with an ANOVA-type analysis, one can pair off to find the culprit after a rejection. In this case – because of the order I thought it best to organize material into chapters – we've done that already. Perhaps that makes this example a little boring, but you'll have the opportunity to try others in your homework.

KW test via math

One may, technically speaking, use an SSP-based counting routine to derive mass and distribution for $\bar{r}^2$ exactly. However, considering the magnitudes involved ($\bar{r}^2 = 1.0908 \times 10^5$), that would represent a computationally daunting undertaking. Instead, an asymptotic approach is favored.

It can be shown that

$$X^2 = \frac{\bar{R}^2 - \frac{n(n+1)^2}{4}}{\frac{1}{n-1}\left[\sum_{j=1}^{m}\sum_{i=1}^{n_j} r_{ij}^2 - \frac{n(n+1)^2}{4}\right]} \sim \chi^2_{m-1} \quad \text{as } n \to \infty. \tag{11.5}$$

In the case of no ties, the denominator above reduces to $n(n+1)/12$. On the face of it, that might seem helpful for by-hand calculations. On the other hand, if you can calculate $\bar{r}^2$, then it stands to reason that you can sum up and square all n ranks without issue. So perhaps that cute result is mere trivia. I'm not going to prove any part of Eq. (11.5) to you, but I'll show it to you in action.

```
denom <- (sum(r^2) - n*(n + 1)^2/4)/(n - 1)
x2 <- (r2bar - n*(n + 1)^2/4)/denom
x2
```

```
## [1] 20.32
```

One nice thing about this number, as opposed to $\bar{r}^2$, is that it has a higher degree of interpretability. χ^2_{m-1} distributions have a mean and variance of $m-1$ and $2(m-1)$, respectively. With $m = 4$ that makes $x^2 = 20.32$ point to an easy reject at 5% since that's 3 standard deviations away from the mean. A p-value works too.

```
pval.asym <- pchisq(x2, m - 1, lower.tail=FALSE)
c(mc=pval.mc, asym=pval.asym)
```

```
##        mc      asym
## 0.0000550 0.0001456
```

`ANSM5` on CRAN offers the only packaged option for comparison that I'm aware of. Although its `kruskal.wallis` routine has a `do.exact=TRUE` option, that choice only works for problems where n is small. Here small means $n \leq$ `max.exact.cases=15` by default. Increasing to `max.exact.cases=n` results in a memory allocation error. Consequently, the only (other) library options are MC and asymptotic calculations, just like those above.

```
yflat <- unlist(ylist)  ## so code below will fit horizontally
c(lib.mc=kruskal.wallis(yflat, g, do.mc=TRUE, nsims.mc=100000)$pval.mc,
  lib.asym=kruskal.wallis(yflat, g, do.asym=TRUE)$pval.asymp)
```

```
##    lib.mc  lib.asym
## 0.0000400 0.0001456
```

That's close in the first case and exact in the second. Notice that I upped the number of MC simulations so that the result would more closely resemble those from my bespoke loop above. I wasn't able to do `nsims.mc=N`; that was too big of an ask to return a result in a reasonable amount of time.

Note that the `SuppDists` on CRAN (Wheeler, 2025) provides a `pKruskalWallis` function, promising a cdf evaluation. However, I've been unable to figure out from its documentation how it can be used in a KW test using a setup that was familiar to me. Perhaps I haven't been patient enough with it, since other functions from that library have come in handy; see Chapter 12.

An astute observer will notice that the asymptotic sampling distribution for squared ranks – a test of scale – uses a Gaussian approximation while KW – a test of location – uses χ^2. I find this puzzling, since Gaussians are location models and χ^2 are scale. The KW result is consistent with ANOVA-like procedures, so perhaps it's squared ranks that's the odd duck.

11.3 Homework exercises

These exercises help gain experience with squared ranks and Kruskal–Wallis tests. There are no math questions this time; all are coding and/or data analysis. Don't forget to try each test multiple ways, e.g., via MC and math/asymptotics. If you're looking for more practice, any questions on P-based tests of variance and ANOVA from §6.5 are appropriate as well. Needless to say: my intention here is that you use a non-P test.

Do these problems with your own code, or code from this book. Unless stated explicitly otherwise, avoid use of libraries in your solutions.

#1: Ties in ranks

Repeat the Metro fare card example with KW, except augment your MC to respect ties in the ranks via `rank.ties` in rank_ties.R[6].

#2: `monty.sqrank` and `monty.kw`

Write library functions for all methods in this chapter. That means squared ranks and KW via MC, exact, and asymptotic calculations. Since they are different tests for distinct situations, I suggest two separate functions. Optionally incorporate your `rank.ties` solution from #1, which may additionally include support for KW. Feel free to use any of the code in ranks.R[7].

Hint: many of the following questions are easier after this one is squared away.

#3: Outside temperature

Measurements below provide the highest recorded temperature in degrees Fahrenheit for Blacksburg, VA and Arlington, VA on several randomly sampled days in one summer.

```
bburg <- c(75, 82, 80, 85, 77, 90, 83, 85, 88, 89, 81)
arlington <- c(88, 81, 96, 76, 98, 79, 81, 82, 88, 93, 91, 85, 84, 90)
```

Is variability in temperatures at these VA locations different?

#4: Inside temperature

As part of a survey on personal comfort, participants were asked to report their gender (recorded as male and female), among other information, and their ideal winter-time room temperature in degrees Fahrenheit. A subset of those values are provided below.

```
male <- c(65, 73, 69, 71, 68, 69, 70, 67, 71, 65, 72, 67, 69, 65,
  66, 68, 67)
female <- c(70, 69, 70, 68, 68, 69, 71, 70, 68, 69, 69, 70, 71, 69)
```

Is there a difference in variance for indoor temperature comfort level for these two groups?

#5: Return on investment

Refer back to `roi$company` and `roi$industry` from §2.5. Recall that these data are available as roi.RData[8] on the book webpage.

```
load("roi.RData")
```

Is there any difference in ROI variance between companies that use the IT product in question and the industry at large?

[6] https://bobby.gramacy.com/hipp0/rank_ties.R
[7] https://bobby.gramacy.com/hipp0/ranks.R
[8] https://bobby.gramacy.com/hipp0/roi.RData

#6: TV watching

Student researchers at a liberal arts college asked other randomly selected students questions about their TV-watching habits. They divided their subjects into four groups: male athletes (MA), male non-athletes (MNA), female athletes (FA), and female non-athletes (FNA). These data, recording hours watched per day, may be found on DASL[9], or as tv.csv[10] on the book webpage.

```
tv <- read.csv("tv.csv")
```

What do you think? Are TV-watching habits differentiated by gender and athletics? *Hint, repeated from §6.5: one convenient way to turn two vectors, one with real-valued measurements and another* `factor` *grouping variable, is with* `split` *in R.*

#7: Hand-washing

A student studied four methods for eliminating bacteria via hand-washing: water only, with regular soap, with antibacterial soap (ABS), and spraying hands with an alcohol-based antibacterial spray (AS; containing 65% ethanol as an active ingredient). These data, counting bacteria on participants' hands after an incubation period, may be found on DASL[11], or as bacteria.csv[12] on the book web page.

```
bacteria <- read.csv("bacteria.csv")
```

What do you think? Is the level of effectiveness about the same for all four methods? (*See* `split` *hint above in #6.*)

#8: Light bulbs

Light bulb manufacturers boast longer lifetimes for LEDs compared to classical incandescent and compact-fluorescent (CFL) bulbs. This is generally true in controlled settings like leaving them on continuously. Things are more complicated with everyday home use where they're turned on-and-off several times per day. Data values below record lifetimes in months for three kinds of bulbs from a single manufacturer.

```
bulbs <- list(
  incan=c(73, 69, 66, 72, 66, 70, 85, 73, 79, 92),
  cfl=c(83, 90, 81, 83, 100, 68, 77, 71, 78, 95),
  led=c(84, 102, 79, 98, 75, 102, 115, 74, 58))
```

Is there any difference in their lifetimes?

[9] https://dasl.datadescription.com/datafile/tv-watching/
[10] https://bobby.gramacy.com/hipp0/tv.csv
[11] https://dasl.datadescription.com/datafile/hand-washing/
[12] https://bobby.gramacy.com/hipp0/bacteria.csv

12

Non-P correlation and regression

This is the non-P image of Chapter 7, first with correlation and then with simple linear regression (SLR). I bet you can guess how that goes because there's been a recurring theme. Convert to ranks and then apply a rather more conventional method on those. That's about right here, but there's some nuance to it. Don't worry, it'll be – I'll try to make it – interesting.

I say "more conventional method" because most non-P analyses based on ranks involve averages, or averages of squares depending on the situation. This is easy to say but could be hard to do depending on how you wish to work with the sampling distribution. Monte Carlo (MC) is simple: just randomly draw ranks: shuffle integers from 1 to n (assuming no ties) for sample size n. Closed-form calculations require solving a subset sum problem (SSP). Although there are libraries available (see §B.3), they're not speedy and there's still the question of how to tailor an SSP solver to the statistical inference task at hand.

Correlation and linear regression are inherently linear enterprises. Lines involve slopes and intercepts. So there are still parameters in that sense. But no more Gaussians. By non-P, I mean no parameters defining distributions or controlling randomness.

12.1 Spearman's ρ

This one's pretty easy. Spearman's ρ, which I shall notate as ρ_s, is Pearson's ρ from §7.2 on ranks. The long form of its name is Spearman's rank correlation coefficient[1]. Characterizing ρ_s as "ρ on ranks" is apt but also fast and loose. Pearson's ρ is a parameter in a bivariate Gaussian model (7.5). It's hard to characterize $\hat{\rho}_s$ as parameter in anything. Rather, Spearman offers a framework for working with a statistic on ranks in a stochastic setting. That's a mouthful.

My own feeling is that ρ_s shouldn't be Greek. Rather, the notation should be $r_{yx}^{(s)} \equiv r^{(s)}$ or something similar, because it's an estimate not a parameter. Yet Greek is how you'll find it in the literature. At the very least, ρ_s ought to have a hat on it, like $\hat{\rho}_s$ to indicate that it's an estimate. I'm going to give you a formula momentarily, for completeness, but you're never going to use it because it's so much simpler in code.

Suppose we have paired data $(x_1, y_1), \ldots, (x_n, y_n)$ and wish to know if there is a linear association between them. Convert to ranks separately for x and y and denote these as $r_1^{(y)}, \ldots, r_n^{(y)}$ and $r_1^{(x)}, \ldots, r_n^{(x)}$, with ties as usual. Then, an estimate of Spearman's ρ_s follows ordinary sample correlation (7.4) on those ranks.

Notation gets a little cumbersome since ranks and estimated ρ both use the letter r, thwart-

[1] https://en.wikipedia.org/wiki/Spearman's_rank_correlation_coefficient

ing shorthand like in Eq. (7.4). When there are no ties, both sets of ranks fill out $\{1, \ldots, n\}$ providing some simplification, but the formula is still a hot mess.

$$\hat{\rho}_s = \frac{\sum_{i=1}^{n} r_i^{(y)} r_i^{(x)} - n\left(\frac{n+1}{2}\right)^2}{\left(\sum_{i=1}^{n} r_i^{(y)2} - n\left(\frac{n+1}{2}\right)^2\right)^{1/2} \left(\sum_{i=1}^{n} r_i^{(x)2} - n\left(\frac{n+1}{2}\right)^2\right)^{1/2}} \tag{12.1}$$

Like I said, you won't ever have to code that up, but you could if you'd like to check. To show you what to do instead, I shall introduce a running example earlier in the chapter than usual. Consider the (N)ELS data, first introduced in §10.2, recording high-schoolers' scores on a math test. We shall investigate a linear association between socioeconomic status (SES) and scores for school one.

```
nels <- read.csv("nels_math.csv")
s1 <- which(nels$school == 1)
y <- nels$score[s1]
x <- nels$ses[s1]
n <- length(y)
```

Here a visual is helpful, I think; Figure 12.1 shows a scatterplot. Whereas we were able to determine in Chapter 7 that observed scores alone were (univariate) Gaussian, entertaining a bivariate relationship is not so simple. The observed scatter does not concentrate elliptically the way a bivariate Gaussian[2] would, at least to my eye. It's technically possible to extend a goodness-of-fit test (§10.1) into higher dimension. Yet binning just $n = 31$ values via Cartesian product[3] would result in very few observations in each bin, and many categories c. We'd have a low-resolution quilt of bins.

```
plot(x, y, xlab="SES", ylab="math test score")
```

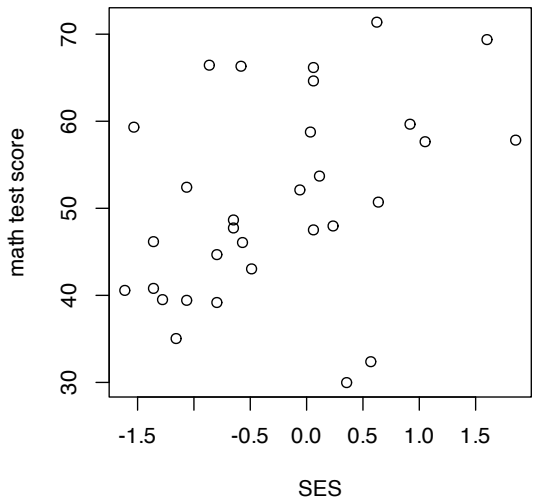

FIGURE 12.1: Math scores versus socioeconomic status (SES) for school one.

[2] https://mathworld.wolfram.com/BivariateNormalDistribution.html
[3] https://en.wikipedia.org/wiki/Cartesian_product

Also note from the figure that there's not an obviously linear relationship between y (score) and x (SES). R code provided below converts raw y- and x-values into ranks and computes correlations on those in several variations. It's worth re-plotting the visual in Figure 12.1 with `rx` and `ry` instead. Give that a try. The presence, or not, of a linear relationship isn't any clearer, but marginal distributions are more uniform – that's exactly what ranks are for.

```
ry <- rank(y)
rx <- rank(x)
rshat <- sum(rx*ry) - n*((n + 1)/2)^2
rshat <- rshat / sqrt(sum(ry^2) - n*((n + 1)/2)^2)
rshat <- rshat / sqrt(sum(rx^2) - n*((n + 1)/2)^2)
c(hand=rshat, pearr=cor(ry, rx), spear=cor(y, x, method="spearman"))
```

```
##   hand  pearr  spear
## 0.3858 0.3858 0.3858
```

These are all the same. Going forward, keep it simple and use the last one. We wouldn't hesitate to use `cor(y, x)` for r_{yx}, the sample analog of Pearson's ρ. And like Pearson's ρ, the hippopotamus-us-es (7.6) are pretty basic. In non-P settings, the null is almost exclusively zero, which is how I've written it out here.

$$\begin{aligned} \mathcal{H}_0 &: \rho_s = 0 \qquad \text{null hypothesis ("what if")} \\ \mathcal{H}_1 &: \rho_s \neq 0 \qquad \text{alternative hypothesis} \end{aligned} \tag{12.2}$$

And so this is a straightforward setup where a sample version is taken as the statistic $\hat{\rho}_s$ in order to investigate a "what if". Straightforward, that is, as long as you don't mind having a "parameter" without a model. I like to think of ρ_s as a reverse-engineered population quantity from a statistic calculated on a sample (12.1). All that remains is to develop its sampling distribution.

Spearman's ρ via MC

By now I hope that sampling ranks is automatic. Here I've gotta do it twice, first for y and then for x. I'm going to ignore the presence of any ties, even though there are some tied scores. See Figure 12.1. If desired, one may always use `rank.ties` from rank_ties.R[4] on the book webpage, as first demonstrated in §9.2 and explained in more detail in §B.2.

```
N <- 1000000
Rs <- rep(NA, N)
for(i in 1:N) {
  RYs <- sample(1:n, n)      ## random Y ranks
  RXs <- sample(1:n, n)      ## random X ranks
  Rs[i] <- cor(RXs, RYs)     ## Pearson correlation on ranks
}
```

Figure 12.2 shows the resulting empirical sampling distribution, indicating a close call. The observed statistic and its reflection are partway into the tails.

[4] https://bobby.gramacy.com/hipp0/rank_ties.R

```
hist(Rs, main="", xlim=c(-0.75, 1))
abline(v=c(rshat, -rshat), lty=1:2, col=2, lwd=2)
legend("topright", c("obs", "reflect"), lty=1:2, lwd=2, col=2, bty="n")
```

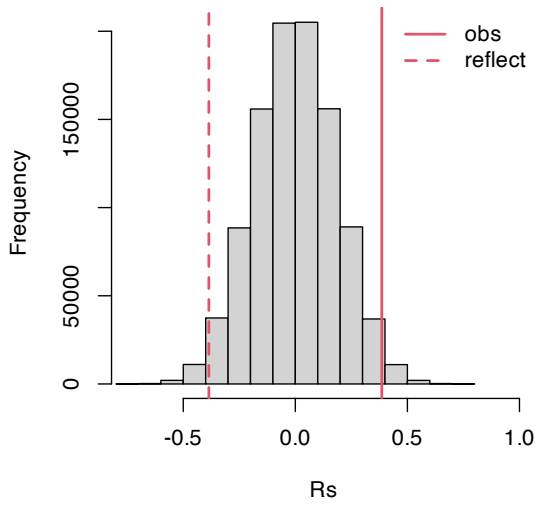

FIGURE 12.2: Empirical sampling distribution of $\hat{\rho}_s$ for association between SES and math test scores for school one.

A p-value calculation reveals a reject of the null at the 5% level.

```
pval.mc <- 2*mean(Rs >= rshat)
pval.mc
```

```
## [1] 0.0328
```

Our conclusion is that SES is linearly related to math scores. I encourage you to check and see how this result compares (or contrasts) with a P-based analysis, following §7.2.

Spearman's ρ via math

An SSP counting argument works here like it does for Wilcoxon and squared ranks tests (§B.3), but things are even more complex because of the cross-term $\sum_i r_i^{(y)} r_i^{(x)}$ in Eq. (12.1). Rather than take us on that wild-goose chase, let me point you Best and Roberts (1975), which is the basis of several library-based methods for evaluating the cdf of a $\rho_s = 0$ sampling distribution. For example, `SuppDists` on CRAN (Wheeler, 2025) provides `pSpearman`, which is pretty easy to use.

```
library(SuppDists)
pval.math <- 2*pSpearman(rshat, n, lower.tail=FALSE)
```

This seems to give the same result as the library method built into R. However, that one warns that it "Cannot compute exact p-value with ties". Hence, `suppressWarnings(...)`.

```
suppressWarnings(pval.lib <- cor.test(y, x, method="spearman")$p.value)
c(mc=pval.mc, math=pval.math, lib=pval.lib)
```

```
##      mc    math     lib
## 0.03280 0.03246 0.03207
```

If you're concerned about the quality of this approximation when there are many ties, a CLT argument works. Look at Eq. (12.1). Things are already centered and normalized: correlations divide covariances by standard deviations (7.4). The denominator, if it is to be interpreted as a standard error, has one too many $1/\sqrt{n-1}$ terms. Since a CLT approximation is only accurate asymptotically, you could equally well say it has one too many $n^{1/2}$ terms, which is a little clearer from the equation. Putting that in the numerator to cancel it out gives

$$Z \equiv \hat{\rho}_s \sqrt{n-1} \sim \mathcal{N}(0,1) \quad \text{as} \quad n \to \infty.$$

In R ...

```
2*pnorm(-abs(rshat*sqrt(n - 1)))
```

```
## [1] 0.03459
```

... is pretty close to the others. My understanding is that this approximation isn't common in practice. Anecdotally, R's built-in library function mentions an "asymptotic t approximations" for `cor.test(..., method="spearman", exact=FALSE)`, but I couldn't find any details on that to relay to you. To add insult to injury, `cor.test` always provides `exact=TRUE` output in both cases, no matter what you ask for. Give it a try. One explanation for this sleight/oversight may be implicit preference for another non-P approach; one which has a different "system" for keeping track of associations; one that doesn't involve ranks. And that's up next.

12.2 Kendall's τ

This one's a trip: a great example of thinking outside the box. I found it fun when I first learned it. If you're bored with ranks, you'll appreciate a change of scenery.

Pearson- and Spearman-based calculations measure tendencies for pairs (x_i, y_i), or their ranks, to be above or below their averages, aggregated over all $i = 1, \ldots, n$ indices. Kendall's method looks directly at the agreement of all pairs of pairs (x_i, y_i) and (x_j, y_j), for $i \neq j$, of which there are many more: $n(n-1)/2$. While it's known as Kendall's rank correlation coefficient[5], I promise there are no ranks involved. That's true in an operational sense: you won't use `rank`. However, Kendall (1948) showed that actually both τ and Spearman's ρ_s are a special case of a third, more general correlation coefficient.

A pair of pairs (x_i, y_i) and (x_j, y_j) are called

[5] https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient

$$\text{concordant if } \frac{y_i - y_j}{x_i - x_j} > 0 \qquad \text{or discordant if } \frac{y_i - y_j}{x_i - x_j} < 0. \tag{12.3}$$

A pair of pairs is considered to be half-concordant and half-discordant if $y_i = y_j$. Any $x_i = x_j$ are eliminated from the sample. Lack of symmetry in treatment of y versus x makes things lopsided and label-dependent in the case of ties. Let me put a pin in that to focus first on core concepts. Recall Figure 7.2 where concepts of concordance and discordance were first introduced. The difference here is that measurements are pairwise, among (x_i, y_i) and (x_j, y_j) pairs, whereas in §7.1 comparisons are drawn to the average pair $(\bar{x}, \bar{y})$.

Figure 12.3 provides a visual upgrade of Figure 7.2. Here, point (x_i, y_i) for $i = 1$ is arbitrarily singled out. Each other $j \neq i$ indexes a point that is either concordant with (x_i, y_i), lying in the green-shaded region, or discordant otherwise. Notice how my code for this figure simply replaces `xbar` with `x[i]` and `ybar` with `y[i]` compared to the code behind Figure 7.2.

```
plot(x, y, xlab="SES", ylab="math test score")
i <- 1
abline(h=y[i], lty=2, lwd=2)
abline(v=x[i], lty=2, lwd=2)
inf <- 1000
gx <- c(-inf, x[i], x[i], inf, inf, x[i], x[i], inf)
gy <- c(y[i], y[i], inf, inf, y[i], y[i], -inf, -inf)
polygon(gx, gy, col=3, density=10, angle=45)
point <- paste0("(x[", i, "], y[", i, "])")
legend(x=0.2, y=45, c(point, "concordant"), col=c(1, 3),
  lty=2:1, lwd=2:1, bty="n")
```

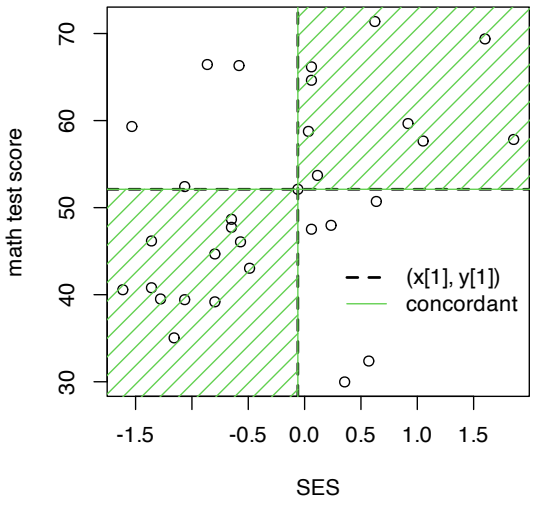

FIGURE 12.3: Illustrating concordant and discordant pairs (of pairs) relative to (x_1, y_1).

Eyeballing points in the plot, there are nine pairs discordant (in the white) with (x_1, y_1), leaving 22 concordant (in the green). You may wish to try another i by adjusting `i` in the code. I'll show you how to automate counting concordant and discordant pairs momentarily. That'll be important for what's coming next.

Let n_c and n_d denote the number of concordant and discordant pairs, respectively, over all $i = 1, \ldots, n-1$ and $j = i+1, \ldots, n$. Note how ranges for i and j are set up so as not to double-count. Kendall's τ is defined as follows.

$$\hat{\tau} = \frac{n_c - n_d}{n_c + n_d} \tag{12.4}$$

As with $\hat{\rho}_s$, I'm putting a hat on the Greek letter when a statistic is being calculated from an observed sample. When there are no ties in the x's, the denominator simply counts all pairs: $n_c + n_d = n(n-1)/2$. The numerator is large in absolute value when there is a strong association, positive (many concordant) or negative (many discordant). Like ρ and ρ_s, τ is a proportion: $-1 \leq \tau \leq 1$ by design, with equality only if all pairs are concordant or discordant, which only happens when all observations perfectly fall on a line.

R code below counts those pairs by looping over (x_i, y_i) in x-order. Pre-ordering over x avoids a double `for` loop in favor of factorized inequalities with indices like `(i + 1):n`.

```
o <- order(x)                                        ## process pairs ...
yo <- y[o]                                           ## ... (x, y) in  ...
xo <- x[o]                                           ## ... order of x
concord <- discord <- rep(NA, n)
for(i in 1:(n - 1)) {
  xu <- xo[i] != xo[(i + 1):n]                       ## catch ties in xs
  yties <- sum((yo[i] == yo[(i + 1):n])*xu)          ## catch ties in ys
  concord[i] <- sum((yo[i] < yo[(i + 1):n])*xu) + yties/2
  discord[i] <- sum((yo[i] > yo[(i + 1):n])*xu) + yties/2
}
```

Lines of code defining `xu` and `yties` are crucial when there are ties in x and y, respectively. Notice how `xu` is used in a product to zero-out any subsequent counts via inequalities, whereas `yties` augments with a fractional component. The last ten rows of counts, again in order of `x`, are provided in Table 12.1.

```
kable(cbind(cbind(x=round(x, 3), y)[o,], concord, discord)[(n - 10):n,],
  caption="Partial table of counts of concordant and discordant pairs.")
```

Printing a table like this isn't essential, but can help for understanding and for carrying out your own calculation with pen and paper, say on an exam. Take the first of those rows. To figure out what belongs in the `concord` column, scan with your eyes down the `y` column and count how many of those are bigger than the one in the current (e.g., first) row. Then do the same, but counting smaller ones, for `discord`. Repeat for the next row, filling out `concord` and `discord` entries looking only at `y`-values. Ignore the `x` column, which is ordered and only provided for reference.

Code found below follows Eq. (12.4) to complete a $\hat{\tau}$ calculation. Using `na.rm=TRUE` is important, since the last entries of `concord` and `discord` are `NA`, leading to blank cells in the last row of Table 12.1.

```
nc <- sum(concord, na.rm=TRUE)
nd <- sum(discord, na.rm=TRUE)
```

TABLE 12.1: Partial table of counts of concordant and discordant pairs.

x	y	concord	discord
0.061	47.52	8	2
0.114	53.70	5	4
0.235	47.97	6	2
0.355	29.98	7	0
0.570	32.38	6	0
0.623	71.38	0	5
0.636	50.71	4	0
0.918	59.66	1	2
1.052	57.65	2	0
1.601	69.38	0	1
1.855	57.83		

```
tauhat <- (nc - nd)/(nc + nd)
tauhat
```

```
## [1] 0.2751
```

In our SES/score example there are no ties in `y`, but there are several in `x`.

```
c(length(unique(y))/n, length(unique(x))/n, nc + nd, choose(n, 2))
```

```
## [1]   1.0000   0.8065 458.0000 465.0000
```

Consequently, if you were to switch labels for `y` and `x` you'd get a slightly different `tauhat`. That's unsatisfying, but there's an easy fix. Just do it again with `x` and `y` swapped, and combine the two results. There are many equally good ways of doing that. One option is to simply add them and divide by two. Or, a fancier solution involves aggregating counts:

$$\hat{\tau} = \frac{n_c^{yx} - n_d^{yx} + n_c^{xy} - n_d^{xy}}{n_c^{yx} + n_d^{yx} + n_c^{xy} + n_d^{xy}} \tag{12.5}$$

I'll leave that for you to try as an exercise in §12.5. In the meantime, note that the library calculation is symmetric, and therefore similar to our by-hand calculation, but not identical.

```
c(cor(y, x, method="kendall"), cor(x, y, method="kendall"))
```

```
## [1] 0.273 0.273
```

Most of the discrepancy can be accounted for by symmetrizing (12.5); however, there's also a slight difference in how ties are tabulated. Ultimately it doesn't matter much which version of $\hat{\tau}^2$ is used – remember, the statistic is your choice – and it's probably simplest to use the library since `for` loops are cumbersome. I just wanted to give you a glimpse under the hood.

Like with Spearman's ρ_s in Eq. (12.2), competing hypotheses target a population τ which

corresponds to an estimate calculated from data, $\hat{\tau}$, and almost exclusively checks for zero association.

$$\begin{array}{ll} \mathcal{H}_0 : \tau = 0 & \text{null hypothesis (“what if”)} \\ \mathcal{H}_1 : \tau \neq 0 & \text{alternative hypothesis} \end{array} \tag{12.6}$$

All that's left is to decide how to study $\hat{\tau}$'s sampling distribution under the null.

Kendall's τ via MC

Interestingly, MC for Kendall's τ is nearly identical to Spearman's ρ_s. Below, I sample from `1:n` for both y and x populations directly. You can think of them as ranks if you want to, but they're not used that way in construction of τ. In fact, any mechanism for sampling random `Ys` and `Xs` values would do.

```
Taus <- rep(NA, N)
for(i in 1:N) {
  Ys <- sample(1:n, n)                      ## random Ys (or ranks)
  Xs <- sample(1:n, n)                      ## random Xs (or ranks)
  Taus[i] <- cor(Xs, Ys, method="kendall")  ## Kendall's tau calculation
}
```

If you're concerned about ties, `rank.ties` could be used to mimic any ties present in the data. Although not ranks, that function would work as long as ordered values like `1:n` are used, and which may subsequently be permuted randomly. I'm content to move along with a simple setup, skip the visual this time and move straight on to p-value calculation.

```
pval.mc <- 2*mean(Taus >= tauhat)
pval.mc
```

```
## [1] 0.02883
```

Like with Spearman's ρ_s, a reject of the null.

Kendall's τ via math

There's an SSP counting method for this one, too, but again I'm skipping it in favor of `pKendall` in `SuppDists`. Before calculating a p-value, I'd like to take a look at the mass of the null (12.6) distribution via `dKendall`. See Figure 12.4. A view like this isn't strictly necessary to complete a testing procedure, but I think it helps explore the range of possibilities, especially along the x-axis in the plot.

```
nc2 <- choose(n, 2)
tgrid <- ((-nc2):nc2)/nc2
plot(tgrid, dKendall(tgrid, n), type="l", lwd=2)
abline(v=c(tauhat, -tauhat), lty=1:2, col=2, lwd=2)
legend("topright", c("obs", "reflect"), lty=1:2, lwd=2, col=2, bty="n")
```

Although the graph *looks* continuous, like a density plot, τ's support (counting concordant and discordant pairs) is the discrete set of integers from $-(n(n-1)/2$ to $n(n-1)/2$. Calculating a p-value involves summing that mass into the tails.

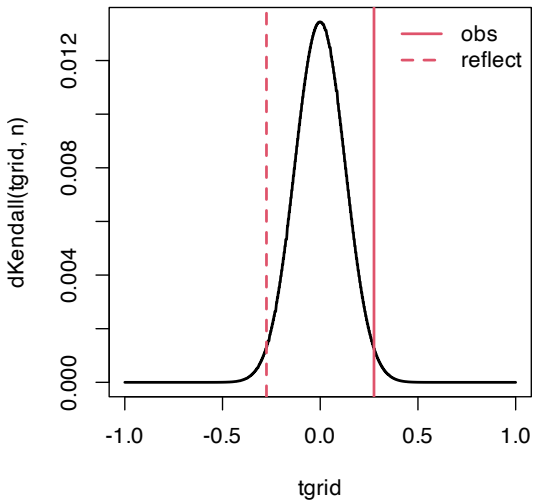

FIGURE 12.4: Exact Kendall's τ sampling mass for the SES/scores example.

```
pval.exact <- 2*pKendall(tauhat, n, lower.tail=FALSE)
pval.exact
```

```
## [1] 0.02874
```

There's no built-in library function to compare this exact result to because, like with Spearman, `cor.test` ignores `exact=TRUE` in the presence of ties. The CLT approximation behind the `exact=FALSE` goes like this. Under $\mathcal{H}_0$, $\mathbb{E}\{\hat{\tau}\} = 0$ since when $\tau = 0$ you have about the same number of concordant as discordant pairs: $\mathbb{E}\{N_c - N_d\} = 0$. It may be shown that $\mathbb{V}\text{ar}\{\hat{\tau}\} = 2(2n+5)/9n(n-1)$. Combining those:

$$Z = \frac{3\tau\sqrt{n(n-1)}}{\sqrt{2(2n+5)}} = \frac{(n_c - n_d)\sqrt{18}}{\sqrt{n(n-1)(2n+5)}} \sim \mathcal{N}(0,1) \quad \text{as} \quad n \to \infty.$$

The first option is preferred when using a $\hat{\tau}^2$ value calculated from `cor(..., method="kendall")`. When working on pen-and-paper, the second one might be simpler. All are asymptotic approximations, meaning that it doesn't matter which you use for large n, and it's hard to judge accuracy for smaller n. In R …

```
z <- (nc - nd)*sqrt(18)/sqrt(n*(n - 1)*(2*n + 5))
pval.clt <- 2*pnorm(-abs(z))
```

Here's the library version of the CLT, which provides a similar estimate.

```
pval.cltlib <- cor.test(y, x, method="kendall",
  exact=FALSE, correct=FALSE)$p.value
c(mc=pval.mc, exact=pval.exact, clt=pval.clt, cltlib=pval.cltlib)
```

```
##      mc   exact     clt  cltlib
## 0.02883 0.02874 0.03223 0.03204
```

Those CLT results don't match up exactly because my $\hat{\tau}$ has not been symmetrized, and there are other small details with how ties are calculated. In case you're curious, the continuity correction is ± 1 in the numerator depending on what side of zero the observed statistic is on.

```
zc <- (nc - nd - sign(tauhat))*sqrt(18)/sqrt(n*(n - 1)*(2*n + 5))
c(byhand=2*pnorm(-abs(zc)), lib=cor.test(y, x, method="kendall",
  exact=FALSE)$p.value)
```

```
##  byhand     lib
## 0.03362 0.03204
```

Again, no match because of symmetrization, etc. All of these p-values are in the same ballpark and lead to the same conclusion at the 5% level.

12.3 Trending?

One use case for non-P correlation tests applies to a single sample $y_1, \ldots, y_n$ where a set of coordinates, the x_i's, are derived from the order (or time) in (at) which each y_i sample was collected. That order may be determined by any indexing, but it's almost always time-of-collection. In that setup, an association measured by ρ_s or τ can test for trend[6] over those indices. It's not possible to use Pearson's ρ in this context because the x coordinate, i.e., an ordered set of indices, cannot plausibly be regarded as marginally Gaussian.

Climate variables are a great example, like precipitation in the continental United States. Data in usprecip.csv[7] was originally downloaded from the National Oceanic and Atmospheric Administration (NOAA) National Centers for Environmental Information, Climate at a Glance: National Time Series[8] pages. (Apologies for using "National" three times in one sentence.) These data contain more than a century of precipitation measurements, recorded as space-aggregated inches of rainfall. Figure 12.5 provides a visual. For such so-called time series[9], it's customary to "connect the dots" with lines as a cue to a left-to-right interpretation as time wanders on.

```
precip <- read.csv("usprecip.csv")
x <- precip$date
y <- precip$inches
plot(x, y, type="b", xlab="year", ylab="inches")
```

These data can help answer an important climate question. Is rainfall increasing over time; is there a trend? Looking at Figure 12.5, to my eye the answer is yes, but there's a lot of noise. We know how to be more scientific, more statistical than that. Humans are hard-wired to see patterns, even when they don't exist. It's best to get ourselves out of the loop and let data speak for themselves through mathematics and probability.

[6] https://en.wikipedia.org/wiki/Trend_analysis
[7] https://bobby.gramacy.com/hipp0/usprecip.csv
[8] https://www.ncei.noaa.gov/access/monitoring/climate-at-a-glance/national/time-series
[9] https://en.wikipedia.org/wiki/Time_series

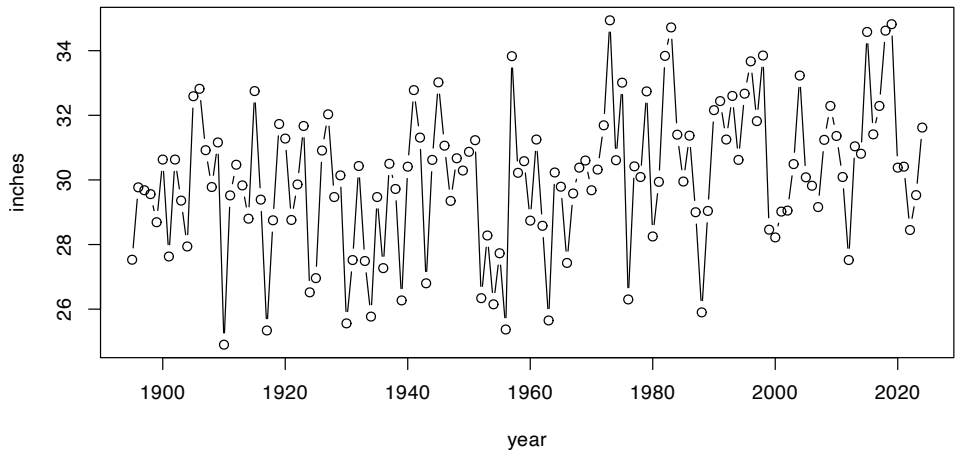

FIGURE 12.5: NOAA USA precipitation time series.

Here are our two rank correlation coefficients for a non-P analysis.

```
rshat <- cor(y, x, method="spearman")
tauhat <- cor(y, x, method="kendall")
c(rhos=rshat, tau=tauhat)
```

```
##   rhos    tau
## 0.2773 0.1865
```

There are some degree-2 ties in the observed tallies, even though measurements are recorded to two decimal places.

```
n <- length(y)
ty <- table(y)
d2ties <- sum(ty > 1)
ty[ty > 1]
```

```
## y
## 27.52 29.47 29.68 30.09 30.38 30.41 30.62 30.63 31.25 32.29
##     2     2     2     2     2     2     2     2     2     2
```

There are not a substantial number, but noteworthy. Just to be on the safe side, I've decided to incorporate `rank.ties` into my MC analysis. The `for` loop in R below combines both Spearman's ρ_s and Kendall's τ calculations, since they share many aspects of their MC virtualization.

```
source("rank_ties.R")
Taus <- Rs <- rep(NA, N)
for(i in 1:N) {
  RXs <- sample(1:n, n)
  Yis <- rank.ties(1:n, d2ties, 2)            ## deal with ties in y
  RYs <- sample(Yis$ranks, n)
```

```
  Rs[i] <- cor(RXs, RYs)                       ## for rho_s
  Taus[i] <- cor(RXs, RYs, method="kendall")   ## for taus_s
}
```

Figure 12.6 provides histograms summarizing both empirical sampling distributions, for Spearman's ρ_s and Kendall's τ on the same axes. In order to ensure that the bins have commensurate heights, I have provided a `breaks=20` argument to both `hist()` commands. This is purely cosmetic, and doesn't affect p-value calculations below.

```
hist(Rs, main="", xlim=c(-0.5, 0.7),
  ylim=c(0, 300000), breaks=20)                ## same number of bins
hist(Taus, add=TRUE, col=0, border=2, breaks=20)  ## for both histograms
abline(v=c(rshat, -rshat), lty=1:2, lwd=2)
abline(v=c(tauhat, -tauhat), lty=1:2, col=2, lwd=2)
legend("topright", c("rshat", "reflect"), lty=1:2, lwd=2, bty="n")
legend("right", c("tauhat", "reflect"), lty=1:2, lwd=2, col=2, bty="n")
```

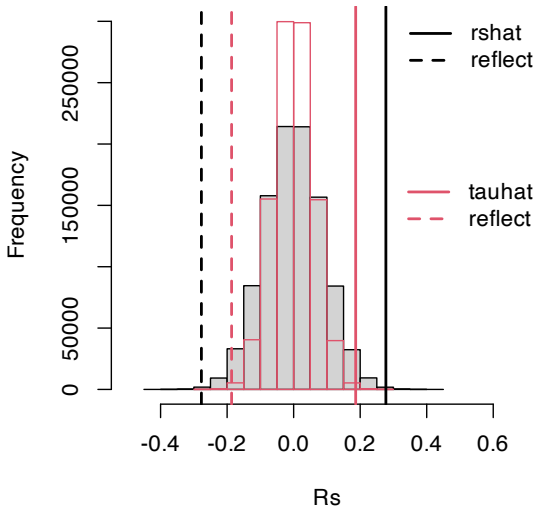

FIGURE 12.6: Empirical sampling distribution of $\hat{\rho}_s$ and $\hat{\tau}$ under null hypotheses (12.2) and (12.6), testing trend in continental US precipitation over a century.

See how the black/gray-filled histogram for ρ_s is slightly wider than the red/transparent one for τ (which is more peaked). So too is the observed statistic $\hat{\rho}_s$ and its reflection compared to $\hat{\tau}$. Consequently, p-values calculated from either are similar.

```
pval.rhos.mc <- 2*mean(Rs >= rshat)
pval.tau.mc <- 2*mean(Taus >= tauhat)
c(rhos=pval.rhos.mc, tau=pval.tau.mc)
```

```
##     rhos      tau
## 0.001530 0.001614
```

This is an easy reject of $\mathcal{H}_0$ under either test at the 5% level. We may conclude that there

TABLE 12.2: Precipitation LM/SLR fit.

	est	stderr	t stat	p val
(Intercept)	-3.7971	9.734	-0.3901	0.6971
x	0.0173	0.005	3.4815	0.0007

is indeed a trend over the years – a trend which appears to be increasing, but hold that thought for a moment. Just to double-check everything, the code below calculates p-values for both tests using math, by hand, and via library calculations. I provided `exact=FALSE` to the library-based Spearman test because it spits out a warning otherwise.

```
pval.rhos.math <- 2*pSpearman(rshat, n, lower.tail=FALSE)
pval.tau.math <- 2*pKendall(tauhat, n, lower.tail=FALSE)
pval.rhos.lib <- cor.test(y, x, method="spearman", exact=FALSE)$p.value
pval.tau.lib <- cor.test(y, x, method="kendall")$p.value
rbind(mc=c(rhos=pval.rhos.mc, tau=pval.tau.mc),
  math=c(rhos=pval.rhos.math, tau=pval.tau.math),
  lib=c(rhos=pval.rhos.lib, tau=pval.tau.lib))
```

```
##          rhos      tau
## mc   0.001530 0.001614
## math 0.001404 0.001561
## lib  0.001403 0.001657
```

All give about the same result, which is reassuring. Recall that our MC respects the distribution of ties observed in the data, whereas the by-hand "math" ones do not. I haven't provided by-hand versions of CLT-based tests here. Perhaps let me encourage you to check those on your own if you're curious.

So we've established that there's likely a trend in these data, by not what that trend is. Determining the yearly increase in precipitation over the last century or so would require estimating a slope. We know how to do that parametrically from §7.6. Here are tidy library calculations.

```
fit <- lm(y ~ x)
sumcoef <- summary(fit)$coefficients
colnames(sumcoef) <- c("est", "stderr", "t stat", "p val")
kable(round(sumcoef, 4), caption="Precipitation LM/SLR fit.")
```

Observe in the `summary` provided by Table 12.2 that estimated slope $\hat{\beta}_1 = 0.0173$ is positive and that the p-value for the null of $\beta_1 = 0$ is small at $\phi_1 = 7 \times 10^{-4}$. So each year precipitation increases by about 0.0173 inches. And although small, this estimate is statistically significant. Over a century, that adds up to almost 2 inches, or about 10%. No wonder the continental US has seen historic flash flooding of late[10].

That analysis of slopes leverages a Gaussian assumption. How could we do that non-P? I'm so glad you asked.

[10] https://www.nytimes.com/interactive/2025/07/16/weather/flash-floods.html

12.4 Non-P lines

Correlation is linear, and non-P SLR analysis leans on that connection. In fact, testing is an application of Spearman's ρ_s, and confidence interval (CI) calculations use Kendall's τ. Emphasis throughout is on slope, with intercepts being treated as a nuisance parameter[11]. That's fine – slopes are the most interesting parts of lines.

The setup has similarities to §7.5. Data are comprised of (x_i, y_i) pairs, and we assume

$$\text{Model:} \qquad \mathbb{E}\{Y_i\} = \beta_0 + \beta_1 x_i \qquad \text{for} \qquad i = 1, \ldots, n. \tag{12.7}$$

Each Y_i is treated as independent of the others given x_i. Similarities end there. In this chapter, no distribution is assumed for random errors; for points around the line. They're in Eq. (12.7) implicitly, of course, since that's what the expectation is integrating over. Without committing to a distributional form for them, there's no need to introduce any ε_i into the model description. Likewise, there's no σ^2.

We haven't gotten rid of all parameters; there's still β_0 and β_1, and these are the main inferential quantities of interest – especially β_1. Intercepts and slopes are unavoidable when working with lines. Crucially, there are no probabilistic parameters because there are no explicit distributions.

Below I shall cover testing and CI calculation separately. As I said, each involves a different technology from earlier in the chapter. The chapter concludes with some commentary on prediction. Throughout I shall use the US precipitation data as a running example. Reanalysis of the frat dash example from from Chapter 7 is another good choice, and is left to you as homework exercise. Math score vs. SES would also work.

Testing

Suppose we wish to test the null hypothesis that $\beta_1 = \beta_1^{(0)}$ for some value of $\beta_1^{(0)}$. So the same setup as Eq. (7.20), where $\beta_1^{(0)} = 0$, but here I'm being slightly more generic and allowing for any slope $\beta_1^{(0)}$ to be tested. Usually interest lies in testing for a zero slope since that means no linear relationship; no power in x to predict Y. I have a somewhat unconventional test in mind here.

Material in §7.6 can be similarly generic, and indeed the MC therein used a value of `beta1 <- 0` that can easily be changed. Here things are a little different since the value of $\beta_1^{(0)}$ is important at the outset no matter how a sampling distribution is derived.

I said that slope testing is an application of Spearman's ρ_s. Actually, it's Spearman's ρ_s on so-called *zero-intercept* or *intercept-free residuals*. Let

$$u_i = y_i - \beta_1^{(0)} x_i \qquad \text{for} \qquad i = 1, \ldots, n.$$

So that's just like e_i in Eq. (7.12) except that $\beta_0 = 0$. In fact this is inconsequential. Any value β_0, including some estimate $\widehat{\beta}_0$, would work fine. Considering that, however, a choice of zero, or in other words "intercept-free", is the simplest option. Now, perform a Spearman's

[11] https://en.wikipedia.org/wiki/Nuisance_parameter

ρ_s-test on $(x_1, u_1), \ldots, (x_1, u_n)$. The outcome of that test directly determines the outcome for the slope $\beta_1^{(0)}$.

Consider testing if the 100-year rise in precipitation of 2 inches, alluded to above in §12.3, were supported by these data. That would correspond to testing $\beta_1 = 0.02$. If you'd like, you can go back through this analysis with $\beta_1 = 0$, but I think you already know how that'll turn out considering our earlier test for trend. This twist, of testing for 2 inches in a century, helps keep things interesting. Or at least I think so.

Here we go in code.

```
beta1 <- 0.02
u <- y - beta1*x
rs.ux <- cor(u, x, method="spearman")
n - length(unique(u))
```

```
## [1] 2
```

Notice how there's just two ties in `u` even though raw `y`-values had several. This is because each of the `x`-values involved in the tie is unique. I've decided to forgo any special handling of ties going forward.

First via MC, cutting-and-pasting from above

```
Rs <- rep(NA, N)
for(i in 1:N) {
  RYs <- sample(1:n, n)
  RXs <- sample(1:n, n)
  Rs[i] <- cor(RXs, RYs)
}
pval.mc <- 2*mean(Rs <= rs.ux)    ## this time in the left tail
```

Then via math and library calculations, again cutting-and-pasting and ignoring by-hand CLT calculations.

```
pval.exact <- 2*pSpearman(rs.ux, n)
pval.lib <- cor.test(u, x, method="spearman", exact=FALSE)$p.value
c(mc=pval.mc, exact=pval.exact, lib=pval.lib)
```

```
##     mc  exact    lib
## 0.3543 0.3554 0.3554
```

These are all about the same and there's not enough evidence to reject the null. Put together a press release! Newsflash: continental US now gets about two more inches of rain per year, on average (year-over-year), than it used to in the early 1900s. By the way, there's no library function that I'm aware of that automates all of these steps. You can write one!

Confidence Interval

Have a look at Eq. (12.3). Each check for a concordant or discordant pair involves rise over run – a calculation for slope – over each pair of pairs (x_i, y_i) and (x_j, y_j):

$$s_{ij} = \frac{y_i - y_j}{x_i - x_j}, \quad \text{for} \quad 1 \le i < j \le n. \tag{12.8}$$

Here's how to calculate that in R via outer products.

```
s <- outer(y, y, '-')/outer(x, x, '-')
```

Except the only problem with that is that it does all (i, j) pairs, not just the ones where $i < j$. The output object `s` is an $n \times n$ matrix.

```
dim(s)
```

```
## [1] 130 130
```

I wish to extract the upper triangle of that matrix to avoid duplication. Although convenient mathematically, I don't need double-indexing, as s_{ij}, in code below. A vector of `s`-values, as provided by `upper.tri` indexing, is sufficient.

```
s <- s[upper.tri(s)]
```

Each entry of `s` is a slope, estimated from one pair of points on the scatterplot in Figure 12.5. Be careful to check here that there are no `NaN` values as might arise if any pair of $x_i = x_j$ for $i \neq j$. Eliminate any of those.

```
s <- s[is.finite(s)]    ## there aren't any in the precip example
ns <- length(s)
```

Let n_s denote the number of slopes which remain. If there are no ties in `x` then $n_s = n(n-1)/2$.

```
c(ns, n*(n - 1)/2)
```

```
## [1] 8385 8385
```

A histogram of those n_s values is shown in Figure 12.7. This visual is for inspection purposes only. Those `s` values do not comprise a sampling distribution. They're just a collection of (all non-infinite) slopes derived from pairs of points in the data. The reason I say that is I don't want you to see that most of those s_{ij} values straddle zero and think that means slope is likely zero, or that a CI straddles zero. Those samples aren't weighted by their mass under the sampling distribution. Neither are the larger s_{ij} values weighted that fill out heavy tails in the histogram, both positive and negative. What really matters for a CI is where the extremes of that empirical distribution lie relative to those s_{ij} values. That's what Kendall's τ is for.

```
hist(s, main="")
```

Let α denote level desired for the CI. Find $q_{\alpha/2}$ under Kendall's distribution for τ. You could do that in closed form, or via MC.

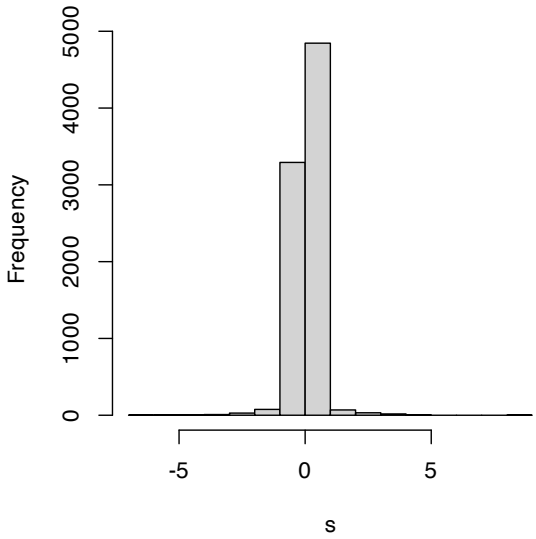

FIGURE 12.7: Pointwise slopes from precipitation data. (This is not a sampling distribution.)

```
alpha <- 0.05
q <- qKendall(1 - alpha/2, n)
c(q, quantile(Taus, 1 - alpha/2))    ## Taus were simulated earlier
```

```
##         97.5%
## 0.1160 0.1161
```

Both should give about the same result. Now $q \in [-1, 1]$, the range of any correlation coefficient. Using q with `s` requires re-scaling to integers $1, \ldots, n_s$. I've done this below defining quantities q_r and $q_l = 1 - q_r$ bracketing $(1 - \alpha/2) \times 100$ percent of the distribution of slopes in a way that I think is both sensible and transparent.

```
qr <- ceiling(ns*(q + 1)/2)          ## ceiling for integer
ql <- ns - qr
```

There are other good choices here. I used `ceiling(...)` in order to round up to the nearest larger integer to be conservative – possibly making the interval a little wider than `round(...)`, which would also work. Indices q_l and s_r define the largest and smallest entries of s_{ij}, stored in `s`, that bracket the interval of interest. Specifically, sort (forming order statistics) entries of `s` providing $s^{(1)}, \ldots, s^{(n_s)}$, and take the q_l and q_r entry in that sorted order.

```
so <- sort(s)
CI <- c(so[ql], so[qr])
CI
```

```
## [1] 0.005873 0.025634
```

That CI does not include zero, but it does include 0.02, supporting results from earlier tests for trend and slope, respectively.

Prediction

As with P-based linear modeling, knowing what to use for prediction is easy. Here's a version of Eqs. (7.9)–(7.10) based on medians:

$$\hat{\beta}_1 = s^{(n_s/2)} \quad \text{and} \quad \hat{\beta}_0 = y^{(n/2)} - \hat{\beta}_1 x^{(n/2)}. \tag{12.9}$$

In R:

```
b1 <- median(s)
medx <- median(x)
b0 <- median(y) - b1*medx
c(b0=b0, b1=b1)
```

```
##       b0       b1
## -0.13972  0.01551
```

Prediction for any new x_p could follow $\hat{\beta}_0 + \hat{\beta}_1 x_p$. Quantifying uncertainty is the hard part. I have not seen this addressed in the literature (the "math way"), but MC is not hard if you follow your nose. Suppose predictions were desired for the following grid of x_p values.

```
xp <- seq(min(x), max(x), length=100)
```

R code below loops over Kendall's τ samples similar to a CI calculation. I used a smaller N compared to examples earlier in the chapter to speed things up a bit. Filling out $n_p = 100$ predictions for `xp` many thousands (or millions) of times is lots of work. We still get a great visual with a coarser sample, as I'll show you momentarily.

Each sampled slope is combined with an estimate of intercept via a (single) bootstrap sample median $y^{(n/2)}$. This choice acknowledges randomness in y without additionally imposing it on x, which are fixed timesteps (years) in this precipitation example. An ordinary bootstrap would jointly sample (x, y), which is not what I want here.

```
N <- 100000 ## smaller N for faster execution
Yps <- matrix(NA, nrow=N, ncol=length(xp))
Es <- rep(NA, N)
for(i in 1:N) {

  ## Kendall's tau slope sampling
  RXs <- sample(1:n, n)
  RYs <- sample(1:n, n)
  Taus <- cor(RXs, RYs, method="kendall")
  si <- round(ns*(Taus + 1)/2)
  B1s <- so[si]

  ## combine with bootstrap median of Y (not X)
  Ys <- sample(y, n, replace=TRUE)
```

```
  B0s <- median(Ys) - B1s*medx

  ## put them together and predict
  Yps[i,] <- B0s + B1s*xp

  ## residual quantile assumes symmetric distribution
  Es[i] <- quantile(abs(y - (B0s + B1s*x)), 0.95)
}
```

In the second half of the loop above, sampled slope(s) and intercept(s) are put together to derive samples of predictive $Y(x_p)$ which may be used for CI calculation. Finally, the 95% quantile of observed absolute residuals, e, from that sampled line are saved so that a predictive interval (PI) may be constructed. Using an absolute value here makes an implicit assumption about symmetry of errors, which might not always be appropriate.

The code chunk below extracts quantiles from those sampled $Y(x_p)$ values and $Y(x_p) + e$ for CI and PI error bar calculations, respectively. A grand average of $Y(x_p)$ samples is also calculated, so that this may be compared with our point-estimate (12.9).

```
## Confidence Interval
CI.l <- apply(Yps, 2, quantile, prob=0.025)
CI.u <- apply(Yps, 2, quantile, prob=0.975)

## Predictive Interval
Emat <- matrix(rep(Es, length(xp)), nrow=N, ncol=length(xp))
PI.l <- apply(Yps - Emat, 2, quantile, prob=0.025)
PI.u <- apply(Yps + Emat, 2, quantile, prob=0.975)

## Prediction, for comparison to median-based calculation
Ypmean <- colMeans(Yps)
```

Figure 12.8 shows those predictive quantities overlayed onto a scatterplot of the data. The shape of that summary, particularly its hyperbolic error bars, looks similar to ones you might get for an ordinary P-based analysis. See, for example, Figure 7.10.

```
plot(x, y, type="b", ylim=range(c(y, PI.l, PI.u)),
  xlab="year", ylab="inches")
abline(b0, b1, col=3, lwd=3)          ## median-based
lines(xp, Ypmean, lwd=3)              ## average of medians
lines(xp, CI.l, lty=2, lwd=3)
lines(xp, CI.u, lty=2, lwd=3)
lines(xp, PI.l, col=2, lty=2, lwd=3)
lines(xp, PI.u, col=2, lty=2, lwd=3)
legend("topleft", c("95% CI", "95% PI"), lty=2, col=1:2, lwd=3, bty="n")
legend("bottomright", c("mean", "(b0, b1)"), col=c(1,3), lwd=3, bty="n")
```

This time, our analysis is free of many of the assumptions underlying a simple linear model (SLR). Conditional independence is used for y_i values, and symmetry is imposed on residuals,

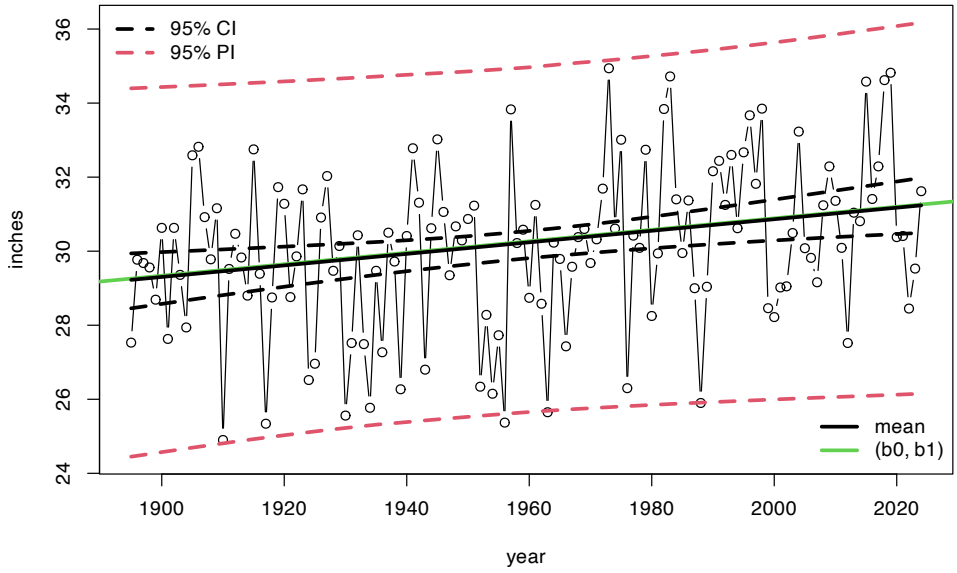

FIGURE 12.8: Non-P predictive distribution for precipitation data.

but only for predictive purposes. That's it. Observe that our median-based estimate (12.9) is nearly identical to the average of medians from the MC loop. These are equivalent.

12.5 Homework exercises

These exercises help gain experience with non-P correlation and regression. Don't forget to try each test multiple ways, e.g., via MC and math calculations. If you're looking for more practice, any of the questions on P-based tests in §7.8 are appropriate as well. Needless to say, the intention here is that you use a non-P test.

Do these problems with your own code, or code from this book. Unless stated explicitly otherwise, avoid use of libraries in your solutions.

#1: Symmetrizing Kendall's τ

Explore (12.5) and *at least one* of your own ideas for symmetrizing an estimate for Kendall's τ. Verify that it works, i.e., that you get the same answer when you swap `x` and `y`, and compare to what the library provides: `cor(y, x, method="kendall")`.

#2: Upgrade `monty.cor`

Upgrade `monty.cor` from §7.8 to include non-P options via Spearman's ρ_s and Kendall's τ. Support MC and closed-form/math alternatives including asymptotic approximations via CLT. Incorporate your solution to #1.

Hint: many of the following questions are easier after this one is squared away.

#3: Voting and income

The file voting.csv[12] linked from the book webpage contains median income in 2023 by state (and Washington, DC), and percent voting for Trump in the 2024 election by state (and DC). These data were collected from two separate Wikipedia pages on median income by state[13] and US presidential election results[14]. See Figure 12.9.

```
voting <- read.csv("voting.csv")
plot(voting$income, voting$trump, type="n",
  xlab="median state income in 1000 USD",
  ylab="percent voting Trump/Republican")
text(voting$income, voting$trump, labels=voting$state, cex=0.7)
```

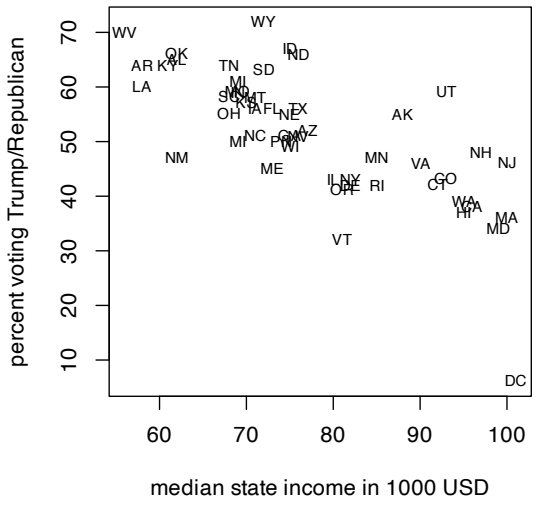

FIGURE 12.9: Percent Republican (Trump) vote by median state income during the 2024 US election season.

Is there a correlation between wealth at the state level, and propensity to vote Republican?

#4: SES vs. math score

Revisit the analysis in §12.1–12.2 two ways.

a. Use school 100.

```
y <- nels$score[nels$school == 100]
x <- nels$ses[nels$school == 100]
```

b. Use all schools.

```
y <- nels$score
x <- nels$ses
```

[12] https://bobby.gramacy.com/hipp0/voting.csv
[13] https://en.wikipedia.org/wiki/List_of_U.S._states_and_territories_by_income
[14] https://en.wikipedia.org/wiki/2024_United_States_presidential_election#Results_by_state

Comment on any linear association between SES and score in these two contexts. Use both Spearman- and Kendall-based tests via MC and closed-form calculations.

#5: Non-Gaussian correlation, redux

In §7.8 #6 some data was presented and you were asked to evaluate if a Gaussian correlation (Pearson's ρ) was appropriate, and adjust as necessary. Revisit that data here with non-P correlation via Spearman's ρ_s and Kendall's τ. How do results compare?

#6: Average January temperature in VA

Data in VAjantemp.csv[15] linked on the book webpage provides average January temperatures (in degrees Fahrenheit) in Virginia for a period of more than 100 years. These data came from a similar NOAA search query[16] as the precipitation data.

```
VAjantemp <- read.csv("VAjantemp.csv")
```

Is there evidence of a trend in this time series? Use both Spearman- and Kendall-based tests.

#7: Mosquitoes

Data provided below come from Hribar (2019) and were download via the VecDyn[17] database. They are yearly observed counts of mosquitoes (genus and species *Aedes taeniorhynchus*, aggregated from weekly) from 1998 until 2019 in Big Pine Key, Florida, USA.

```
mosquitoes <- c(902, 1442, 847, 2322, 801, 455, 847, 26, 366, 79, 256,
  196, 84, 439, 76, 107, 60, 122, 84, 172, 102, 85)
```

The first observation corresponds to 1998, and the last one to 2019. Is there evidence of trend in these data?

#8: Library functions `monty.nPslr` and `predict.monty.nPslr`

Write the non-P version of `monty.slr` and `predict.monty.slr` from §7.8. Automate testing and CI calculation for slopes with the former, and prediction (with CI and PI) with the latter. *You won't be able to share as much information between the two `*.nPslr` functions as with their `*.slr` counterparts, but it'll be nice if they otherwise have the same feel for the user (you).*

Hint: many of the following questions are easier after this one is squared away.

#9: Frat dash, redux

Revisit the frat dash example from Chapter 7 with a test of the null hypothesis that $\beta_1 = 0$, CI for β_1 and predictions, including CI and PI, providing a non-P analog of Figure 7.11.

[15] https://bobby.gramacy.com/hipp0/VAjantemp.csv
[16] https://www.ncei.noaa.gov/access/monitoring/climate-at-a-glance/statewide/time-series
[17] https://vectorbyte.crc.nd.edu/vecdyn-detail/622

#10: Rent for all dwellings

Revisit rent.csv[18], linked from the book webpage and first explored in §7.8 #15. Except, now, base your analysis on all dwellings, not just those smaller than 20,000 square feet.

```
rent <- read.csv("rent.csv")
y <- rent$Rent
x <- rent$SqFt
```

Answer the following.

a. Why were those large dwellings excluded in §7.8, and why are they less of a concern here, in a non-P setup? Considering that non-P setup …
b. Is square-feet a useful predictor of rent?
c. How much more does an additional 1,000 square feet cost? Provide a CI.
d. Provide a visual of your fit, and overlay a prediction with CI and PI error bars for dwellings ranging from zero to 20,000 square feet. *Note here I'm still focusing on smaller dwellings, limiting x_p, mostly to have a cleaner visual.*

How do your answers to #b–#d compare to the P-analysis in Chapter 7.

#11: MPG vs. weight

File mileage.csv[19] linked from the book webpage contains gasoline mileage (`$mpg`) ratings for cars and light trucks of various makes and models, including information on their weight (`$wt`, in hundreds of pounds), and other specs. These data come from a study published by the U.S. Environmental Protection Agency in 1991.

```
mileage <- read.csv("mileage.csv")
```

Answer the following with a non-P analysis.

a. Is weight a useful predictor of mileage?
b. How much does each additional 100 pounds "cost" you in terms of gasoline efficiency? Provide a CI.
c. Provide a visual of your fit, and overlay predictions with CI and PI error bars for cars weighing 1,700 to 6,500 pounds.
d. Can you foresee any problems with extrapolating your prediction to even heavier vehicles?

#12: Import elasticity

This question is meant to whet your appetite for Chapter 13.

Data recorded in imports.csv[20] contain import values and gross domestic product (GDP), both in billions of US dollars (USD), for a collection of countries during the early 2000s. The plot in Figure 12.10 provides a visual of these data on a log-log scale.

[18] https://bobby.gramacy.com/hipp0/rent.csv
[19] https://bobby.gramacy.com/hipp0/mileage.csv
[20] https://bobby.gramacy.com/hipp0/imports.csv

```
imports <- read.csv("imports.csv")
x <- log(imports$imports)
y <- log(imports$gdp)
plot(x, y, type="n", xlab="log imports", ylab="log GDP",
  xlim=c(-2.5, 7.5), ylim=c(-1, 10))
text(x, y, labels=imports$country, cex=0.7)
```

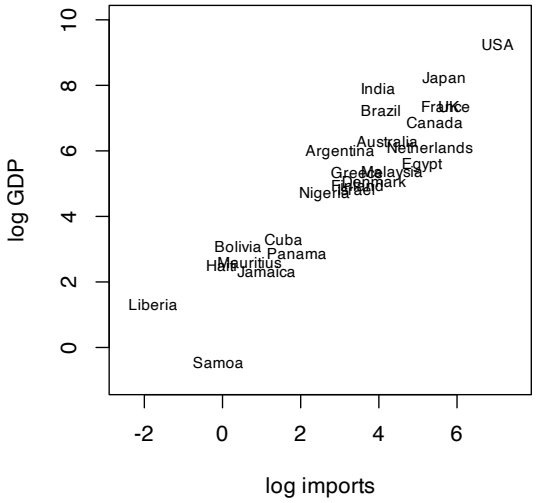

FIGURE 12.10: Gross domestic product (GDP) versus imports, both in billions of USD and on a log-log scale.

There are two reasons for log-log modeling here. The first one is the subject of §13.1, and basically it has to do with the challenge of linear modeling on multiplicative scales. For our purposes here, let's just say that x and y look better when logged. (I encourage you to make the same plot with unlogged coordinates. You'll see what I mean.)

The second one is more practical. In economics, the estimated slope $\hat{\beta}_1$ of a linear model is known as the elasticity[21] of the two variables under study. Economic theory says that imports and GDP should have an elasticity of one. Do these data support this hypothesis?

[21] https://en.wikipedia.org/wiki/Elasticity_(economics)

13

Fancy regression

My aim for this chapter is to offer a treat at the end – some dessert. Also, I really wanted to have thirteen chapters. A lucky number!

Regression is, in my opinion, the most important subject in statistics. Many models, methods, and procedures can be interpreted through the lens of regression, and you already know lots about it (Chapters 7 and 12). If it seems like fitting lines through points couldn't possibly be that useful, my hope is that you'll have a different perspective by the end of this chapter. Perhaps I'll convince you to take a second class in statistics. I hope this whets your appetite for more. Also, I've left myself a number of loose ends from earlier in the book. Gotta ties those up.

Although long, this chapter will move fast, and I'll only scratch the surface of many topics. We'll use some fancy math without lots of fanfare, especially linear algebra and vector calculus. My hope is that this won't be off-putting. I don't think you need to fully understand those bits to appreciate what's going on. For the most part, I shall use tools that are familiar to you, but you might not have realized that they can be used in that way.

Since you already know many of the tools, I'm going to (for the first time in the book) favor the classical, math-based/non-MC approach to inference. This will allow me to move more quickly. I'll come back to MC towards the end (§13.5) to address something that *is* genuinely new: model selection for multiple linear regression (MLR). Once I explain to you what MLR is (§13.3), you'll see why "model selection" is necessary. Things get complicated quickly and it helps to have a principled approach to exploring an MLR landscape. Some additional MC considerations are left for you as exercises in §13.6.

First, I have lots to say about simple linear regression (SLR). SLR is limiting because not many relationships neatly follow Gaussian noise added to a straight line (7.14)–(7.15). Fortunately, SLR need not be applied so simply. Lines needn't be "straight" and Gaussian noise needn't be additive. You may have heard of generalized linear models (GLMs)[1], but that's not what we're doing. We'll use ordinary LMs, but fancy. GLMs are super cool, but for another book, another course.

One last disclaimer: I shall favor a parametric (P) approach to inference, even though many of the methods have non-P versions. Again, that allows me to move more quickly, leaving some interesting explorations to your homework.

[1] https://en.wikipedia.org/wiki/Generalized_linear_model

13.1 Log-transforms

This section is composed of three data analysis vignettes, introducing use of log-transforms on inputs and outputs. Commentary on other, non-log possibilities is provided. The idea is to reveal a pattern and to develop data-analytic instincts.

Used pickup trucks

Consider the selling price of second-hand pickup trucks regressed on model year (i.e., age). Data linked from pickups.csv[2] on the book webpage were scraped from Craigslist[3] by a colleague of mine, Matt Taddy[4] while we were both at The University of Chicago[5]. I owe a lot of the following presentation to Matt; although over the years it has evolved toward my own tastes.

```
pickups <- read.csv("pickups.csv")
fit <- lm(price ~ year, data=pickups)
```

Here's something new to discuss right at the very start. Notice how I refer to the names of columns in the formula `price ~ year` even though those variables are not defined as objects. By providing `data=pickups`, using a formula allows me to refer to columns as if those variables were defined in the current environment. This is handy, as we shall see as examples progress. Another thing that'll be different about this chapter is that I'm usually not defining `x` and `y` variables, explicitly. Rather, I'll favor variable names that are more descriptive of their contents/columns, relying on you to keep track of which are inputs (`$year`) and outputs (`$price`).

Figure 13.1 provides two views summarizing our SLR fit. On the left is a conventional scatterplot with the best-fitting, least-squares line overlayed. On the right are fitted-values and residuals $(\hat{y}_i, e_i)$. Recall that fitted values are perfectly correlated with inputs, $r_{\hat{y}c} = 1$. So it's as if `$year` were on the x-axis rather than $\hat{y}_i$-values. In this instance, it may have been simpler to use `$year`, but when we get to MLR §13.3, there's good reason to prefer $\hat{y}_i$.

```
par(mfrow=c(1, 2))
plot(pickups$year, pickups$price, xlab="year", ylab="price")
abline(fit, lwd=2)
legend("topleft", "least squares line", lwd=2, bty="n")
plot(fit$fitted, fit$resid,
  xlab="fitted values (yhat)", ylab="residuals (ei)")
abline(h=0, col=2, lty=2, lwd=2)
legend("topleft", "zero residual", col=2, lty=2, lwd=2, bty="n")
```

Although the fit looks fairly decent on the left, the right panel reveals that residuals are not iid. All "linearness" has been extracted, but nonlinear pattern is left over. I see two patterns.

[2] https://bobby.gramacy.com/hipp0/pickups.csv
[3] https://chicago.craigslist.org/
[4] https://taddylab.com/
[5] https://www.uchicago.edu

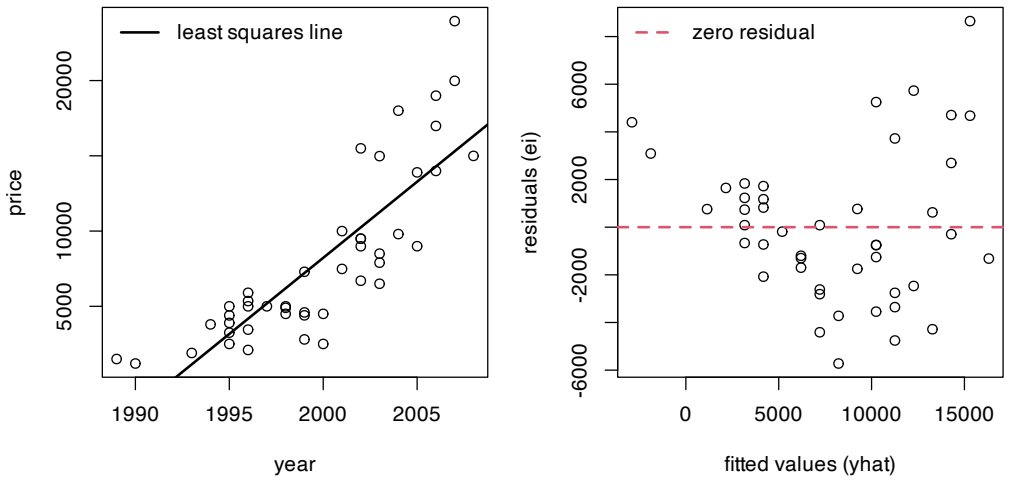

FIGURE 13.1: Pickup truck data and SLR fit (left) along with residuals versus fitted values (right).

One is that the line first under-predicts (positive residuals until about 5,000), then over-predicts (negative residuals until about 12,000), and then under-predicts again, on average. The second pattern is that the magnitude of the residuals, in absolute value, seems to be increasing from left to right. So there is a nonlinear mean pattern and a nonconstant variance pattern, violating two (basically all!) SLR modeling assumptions.

Quick digression on residual diagnostics. An SLR model (7.14) assumes $\varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$, but we don't observe ε_i values. Residuals $e_i \neq \varepsilon_i$, and it may not be that e_i values are iid or even Gaussian. However, it turns out that both things are *almost* true. I'll show you more in §13.3 once some additional scaffolding is in place. It stands to reason that if the assumed errors follow a particular distribution, then observed ones would be similar.

Checking for patterns in residuals, or deviations from Gaussian in their distribution, is a go-to diagnostic for detecting poor SLR fit. Many regression textbooks devote entire chapters to residual diagnostics. I'm not going to do that here. It's an important subject, but not an exciting one. If you're curious, read more on studentized residuals[6] and QQ-plots[7]. Besides plotting raw residuals versus fitted values, those two are the next most common tool for detecting issues.

Alright, back to pickup trucks. Another downside to the fit in Figure 13.1 is that predictions go negative for older trucks. Mentally extrapolate the line in the left panel to 1985 or older. Eventually you cross the x-axis. That doesn't make a whole lot of sense. Would someone be willing to pay you to take the truck from them? I suppose stranger things could happen, but I wouldn't bet on it.

Making a log-transformation of the response (y, or `$price` in this instance) solves all three problems, like magic. Check it …

```
pickups$lprice <- log(pickups$price)     ## augment the pickups data frame
lfit <- lm(lprice ~ year, data=pickups)
```

[6] https://en.wikipedia.org/wiki/Studentized_residual
[7] https://en.wikipedia.org/wiki/Q–Q_plot

As with other uses of log in this book, the base is e but it doesn't really matter as long as you're consistent. Logarithms are an example of "variance stabilizing transformations" because they squash large values more than small ones. Square roots are also variance stabilizing, as are many other functions.

Both log and square root are monotonic, so they preserve direction (increasing or decreasing). Both are also nonlinear, so they can impart or negate curvature. Neither can take a negative argument, so you can't use them if any $y_i < 0$ without some other adjustment. Fortunately for `pickups$price`, those are all positive. One difference between $\log(\cdot)$ and $\sqrt{\cdot}$ is that the former can have negative values in its image: $\log y < 0$ if $0 < y < 1$. Square roots are always positive.

```
par(mfrow=c(1, 2))
plot(pickups$year, pickups$lprice, xlab="year", ylab="log price")
abline(lfit, lwd=2)
legend("topleft", "least squares line", lwd=2, bty="n")
plot(lfit$fitted, lfit$resid,
  xlab="fitted values (log yhat)", ylab="residuals (ei)")
abline(h=0, col=2, lty=2, lwd=2)
legend("bottomleft", "zero residual", col=2, lty=2, lwd=2, bty="n")
```

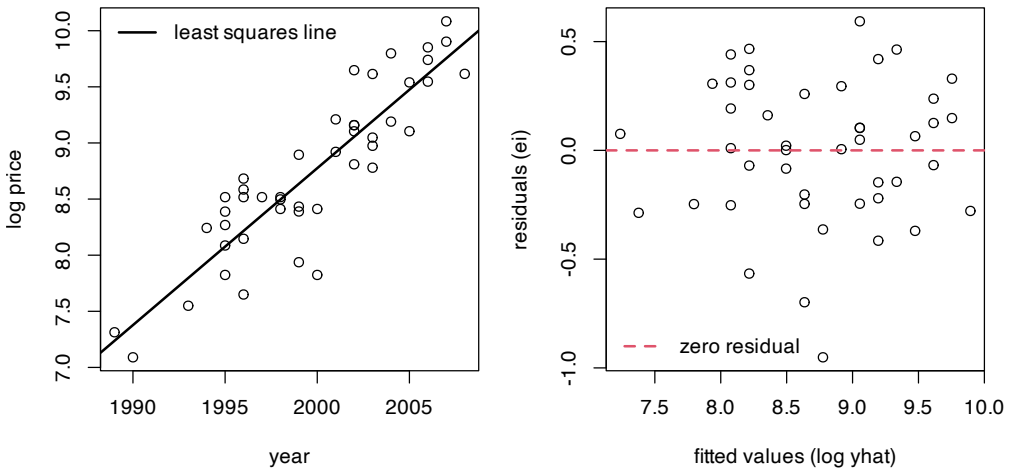

FIGURE 13.2: Pickup truck fit with logged response.

Figure 13.2 provides a log-outputs view of Figure 13.1. That's a much better fit, right? No more oscillation between over- and under-predicting, and no more increasing variance from left to right. A downside to the view here is that log-price is hard to interpret. Ideally, we'd have a model, and predictions, that are back on the original output scale (dollars).

If

$$\text{Model:} \quad \mathbb{E}\{\log Y\} = \beta_0 + \beta_1 x \quad \text{then} \quad Y \approx e^{\beta}_0 e^{\beta_1 x}.$$

This is an approximation, since you technically can't reverse expectations, which are integrals, and logs. For more on that, see Jensen's inequality[8]. But it's close. The message here

[8] https://en.wikipedia.org/wiki/Jensen's_inequality

is that an additive (linear) model on logged outputs is a multiplicative (nonlinear) model back on the original scale.

You may cringe at those exponents, but they're ideal for this setting. Predictions for y, no matter what's estimated for intercept β_0 and slope β_1, will never go negative for any input x. The image of an exponential is never negative: $e^x > 0$ for all $x \in \mathbb{R}$.

Therefore, predictions made in log-space will organically squish and expand back on the original output scale.

```
xp <- seq(min(pickups$year), max(pickups$year), length=1000)
p <- predict(lfit, newdata=data.frame(year=xp), interval="prediction")
```

Observe how I'm using `xp`, in keeping with our earlier use of x_p for prediction in Chapters 7 and 12. However, that grid is fed as `newdata` into `predict` with an indication that those quantities should be interpreted as `$year` values via `data.frame`. Figure 13.3 shows back-transformed predictive means and PI error bars.

```
plot(pickups$year, pickups$price, ylim=range(exp(p)),
  xlab="year", ylab="price")
lines(xp, exp(p[,1]), lwd=2)
lines(xp, exp(p[,2]), col=2, lty=2, lwd=2)
lines(xp, exp(p[,3]), col=2, lty=2, lwd=2)
legend("topleft", c("mean", "95% PI"), col=1:2, lty=1:2, lwd=2, bty="n")
```

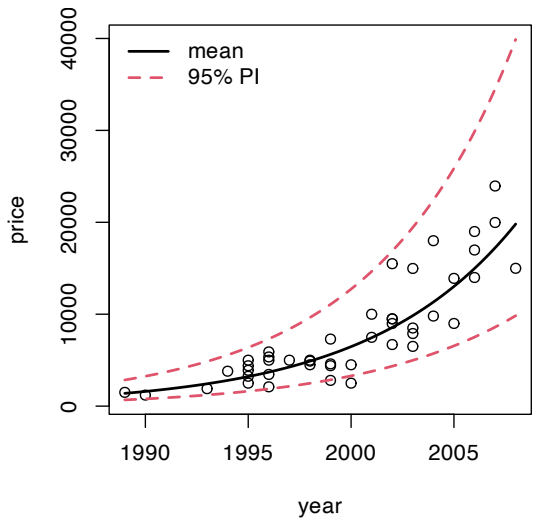

FIGURE 13.3: Back-transformed predictions for pickup trucks.

The first time I saw a visual like the one in Figure 13.3 was eye-opening. Lines are pretty basic, but it's up to you as the practitioner to decide what, exactly, you want to model linearly. You get to choose what the input and output variables are. Someone may have collected raw x values in dollars, but it's fine to choose to model $f(x)$ just as it's fine to collect heights in inches and decide to model centimeters instead.

What's important is how results are communicated. If you report predictions on log-dollars

folks probably won't understand. People aren't good at thinking in logarithms. Other times, preferred units are cultural. (Try telling a German that you're 6'2"; they think in metric.) You get to choose what *features* are modeled, to use the machine learning jargon, but back-transforms are key to communication. Put it back the way you found it!

Finally, with log-transforms and exp back-transforms, you'll never look foolish predicting a negative price or other nonsense. Observe in the figure that the error bars forming PIs shrink to stay positive. A variance stabilizing transformation organically gives you nonconstant, monotonic variance on the original scale.

Mammal weight

I like to think of log-transforms as "unbunching". Looking at Figure 13.3, y-values are bunched up at small values, and spread out for larger ones. Input x-values are more-or-less uniform. Take the log of the y values, and they'll be on a different, more uniformly-spread scale (Figure 13.2, left). Leave the x's alone.

Sometimes, there's bunching in both inputs *and* outputs. Data in mammals.csv[9] on the book webpage contains pairs of weights for the body (kilograms) and brain (grams) of 84 mammals. Early examples with these data are provided by Weisberg (1985) and Rousseeuw and Leroy (1987). See Figure 13.4, showing the original data (left) and a SLR fit to log-log input and output variables (right).

```
mammals <- read.csv("mammals.csv")
lbody <- log(mammals$body)
lbrain <- log(mammals$brain)
llfit <- lm(lbrain ~ lbody)
```

See what I mean about bunching on both axes? You may recall a similar GDP–Imports log-log fit from §12.5. The analysis here, for mammal weights, could easily be ported over to that setting.

```
par(mfrow=c(1, 2))
plot(mammals$body, mammals$brain,
  xlab="body weight (kg)", ylab="brain weight (g)")
plot(lbody, lbrain, xlab="log body weight", ylab="log brain weight")
abline(llfit, lwd=2)
legend("bottomright", "log-log fit", lwd=2, bty="n")
```

Consider a multiplicative model $\mathbb{E}\{Y \mid x\} = ax^b$. Take logs of both sides to get

$$\text{Model:} \qquad \log(\mathbb{E}\{Y \mid x\}) = \log(a) + b\log(x) \equiv \beta_0 + \beta_1 \log(x).$$

So the log-log model is appropriate whenever things are linearly related on a multiplicative, or percentage scale, i.e., when $\%y$ changes linearly with $\%x$. In that situation, estimated slope $\hat{\beta}_1$ is known as *elasticity*. It may be interesting to test, for these data, whether brain and body weight have an elasticity of one, meaning that there's a 1:1 relationship on percentage terms.

[9] https://bobby.gramacy.com/hipp0/mammals.csv

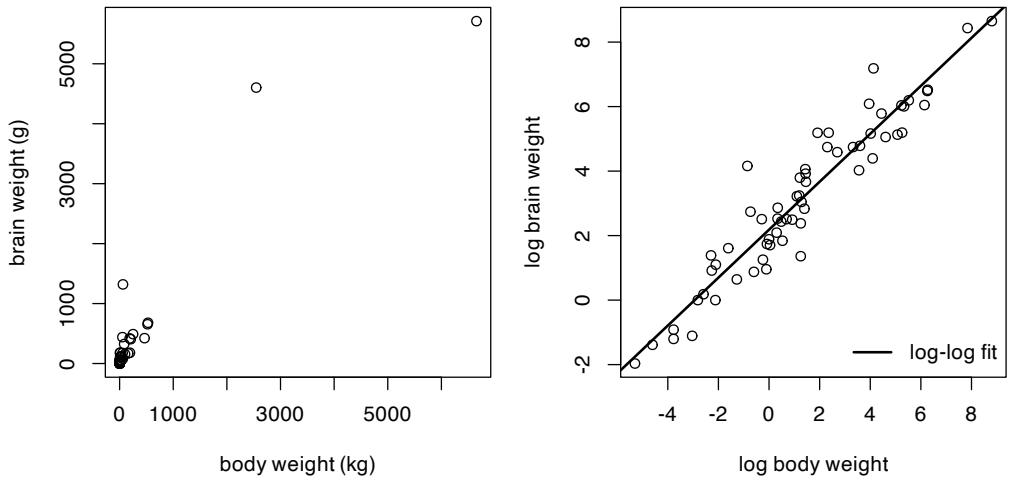

FIGURE 13.4: Raw mammals data (left) and log-log SLR fit (right).

```
b1 <- as.numeric(coef(llfit)[2])
se1 <- summary(llfit)$coefficients[2, 2]
t <- (b1 - 1)/se1
pval <- pt(-abs(t), nrow(mammals) - 2)
c(b1=b1, pval=pval)
```

```
##        b1      pval
## 7.432e-01 1.578e-11
```

No. We can reject $\mathcal{H}_0 : \beta_1 = 1$. Apparently, brains of large mammals tend to be smaller, proportionately speaking, than those of smaller mammals. I find that counterintuitive. You'd think that you'd need a bigger brain to manage more body. Or at least I would, but I'm not a biologist.

How about a prediction back on the original scale?

```
xp <- seq(min(mammals$body), max(mammals$body), length=1000)
p <- predict(llfit, newdata=data.frame(lbody=log(xp)),
  interval="prediction")
```

Be careful to transform the predictive grid on the way *into* `predict`, and then undo everything that comes out. Figure 13.5 shows the resulting predictive distribution via mean and 95% PI error bars.

```
plot(mammals$body, mammals$brain, ylim=range(exp(p)),
  xlab="body weight", ylab="brain weight")
lines(xp, exp(p[,1]), lwd=2)
lines(xp, exp(p[,2]), col=2, lty=2, lwd=2)
lines(xp, exp(p[,3]), col=2, lty=2, lwd=2)
legend("topleft", c("mean", "95% PI"), col=1:2, lty=1:2, lwd=2, bty="n")
```

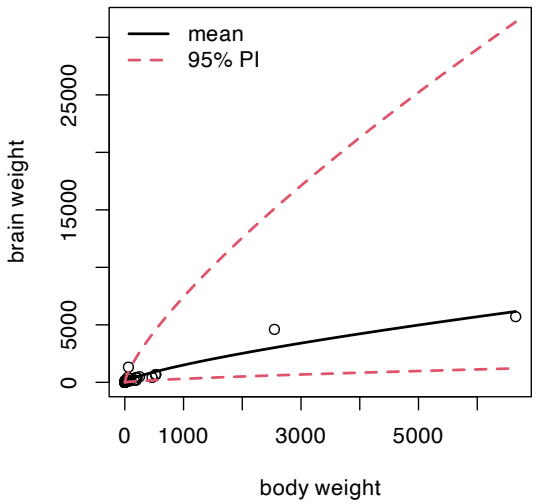

FIGURE 13.5: Predictive distribution for mammals' weights.

Exponents, combined with very few large mammals in the data set, translates into wide error bars for predictions for those large mammals. Consequently, this view isn't as pretty as some others we've seen, but it is what it is. At least we'll never predict that a mammal could somehow have negative brain weight. That's even worse than having to pay someone to take your pickup truck. You're better off donating it to NPR[10].

Price elasticity

People tend to buy less stuff when it's more expensive. Shocker! Sellers hope to find the sweet spot, balancing price, sales, and inventory, to maximize profit. Our final example comes from a subset of "Retail Scanner Data" found in the Nielsen-Kilts SCANTRACK database[11]. Data in confood.csv[12] on the book webpage record supermarket sales of canned food produced by Consolidated Foods[13] at various price points.

```
confood <- read.csv("confood.csv")
lprice <- log(confood$price)
lsales <- log(confood$sales)
llfit <- lm(lsales ~ lprice)
```

Raw data values (left) and log-log versions with SLR fit (right) are shown in Figure 13.6.

```
par(mfrow=c(1, 2))
plot(confood$price, confood$sales,
  xlab="price ($)", ylab="sales volume (units)")
plot(lprice, lsales, xlab="log price", ylab="log sales")
abline(llfit, lwd=2)
legend("topright", "log-log fit", lwd=2, bty="n")
```

[10] https://www.npr.org
[11] https://www.chicagobooth.edu/research/kilts/research-data/nielseniq
[12] https://bobby.gramacy.com/hipp0/confood.csv
[13] https://en.wikipedia.org/wiki/Sara_Lee_Corporation

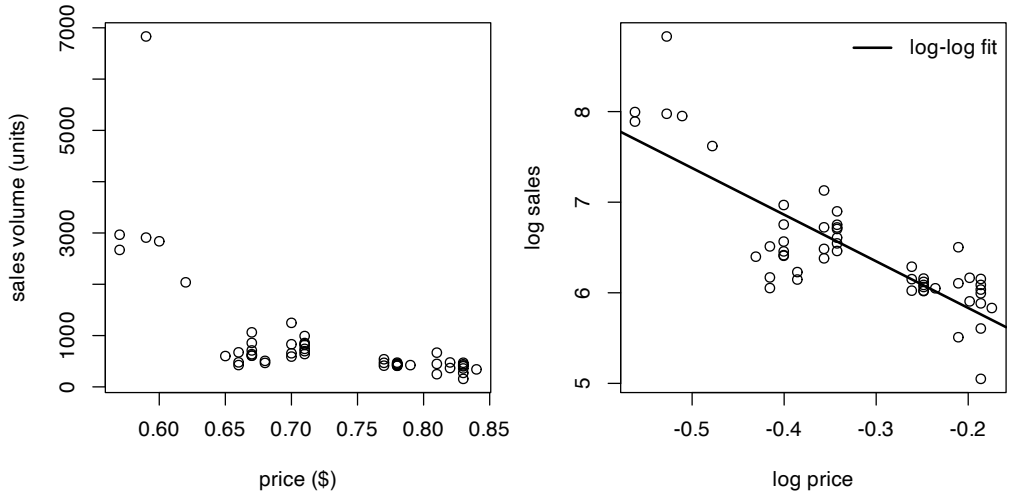

FIGURE 13.6: Raw Consolidated Foods data (left) and log-log SLR fit (right).

TABLE 13.1: Confoods log-log LM/SLR fit.

	est	stderr	t stat	p val
(Intercept)	4.803	0.1744	27.53	0
lprice	-5.148	0.5098	-10.10	0

The view on the original scale is less pathological than the one in Figure 13.4 (left), but clearly nonlinear. Sales never dip below zero, and you wouldn't want to predict that either. Bunching is less pronounced, and perhaps `log(price)` isn't necessary, since prices seems to vary uniformly (possibly no bunching on the x-axis). However, it can sometimes be advantageous to use $\log x$ whenever logging y, automatically. Log-log lends interpretability through an elasticity (or percent) interpretation. Definitely don't do it if it makes things look worse though.

Table 13.1 provides a summary of coefficients and standard errors for a log-log fit to Consolidated Foods sales. Everything is highly significant. Apparently, if you increase the price by 1%, then sales decrease by about 5%. Makes sense.

```
sumcoef <- summary(llfit)$coefficients
colnames(sumcoef) <- c("est", "stderr", "t stat", "p val")
kable(round(sumcoef, 4), caption="Confoods log-log LM/SLR fit.")
```

The sweet spot in that relationship depends on costs and how those compare to revenue: sales volume times price. For that, it can help to have predictions back on the original scale.

```
xp <- seq(min(confood$price), max(confood$price), length=1000)
p <- predict(llfit, newdata=data.frame(lprice=log(xp)),
  interval="prediction")
```

See Figure 13.7. Something like this could be invaluable to decision-making at several levels: supermarket, manufacturing, logistics, sales/promotions, etc.

```
plot(confood$price, confood$sales, ylim=range(exp(p)),
   xlab="price ($)", ylab="sales volume (units)")
lines(xp, exp(p[,1]), lwd=2)
lines(xp, exp(p[,2]), col=2, lty=2, lwd=2)
lines(xp, exp(p[,3]), col=2, lty=2, lwd=2)
legend("topright", c("mean", "95% PI"), col=1:2, lty=1:2, lwd=2, bty="n")
```

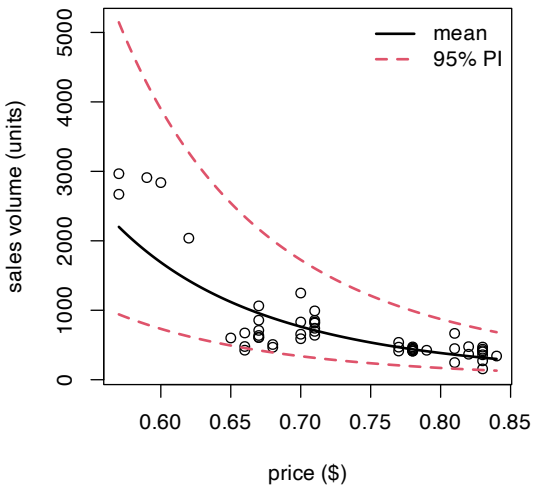

FIGURE 13.7: Predictive distribution for Consolidated Foods sales at varying price points.

I hope you're starting to see what I mean by "SLR need not be applied so simply". Not many relationships are linear, but lines are good approximations, at least locally. You can always transform your outputs, and/or inputs, and get additional flexibility. You can really do whatever you want, although logarithms are a good place to start. If log doesn't work, try square root. If neither of those helps, you might need to try something different. I have some ideas for you; read on.

Be careful, if you change the scale of outputs (e.g., with $\log y$) then don't make any direct quantitative comparisons to models fit on other/original scales. That means don't compare standard errors or R^2 values. Comparisons across transformations of inputs are fine though, as long as outputs are held constant. In fact, you don't need to commit to just one.

13.2 Polynomials

If straight lines, possibly on a transformed space, aren't providing a good fit, try upgrading to polynomials. Taylor's theorem[14] says any curve may be approximated, locally, by a low-degree polynomial. Weierstrass' theorem[15] says that you can approximate any function,

[14] https://en.wikipedia.org/wiki/Taylor's_theorem
[15] https://en.wikipedia.org/wiki/Stone–Weierstrass_theorem

globally over any bounded range, with a high-enough degree polynomial. The sweet spot, statistically speaking, is somewhere in-between.

What am I saying? Try

$$\text{Model:} \qquad Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \varepsilon_i, \quad \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2), \quad i = 1, \ldots, n. \tag{13.1}$$

Technically, this is a multiple linear regression (MLR) with powered-up features derived from scalar x_i. So I'm doing things a little out of order; most of the details for MLR come next in §13.3. Just bear with me. I'll show you that you already know how to code it up.

You can choose to power-up as much as you'd like, say up to degree $\beta_d x_i^d$, but be careful. When $n \leq d-1$ you get a "perfect fit", with the polynomial going exactly through all pairs (x_i, y_i), for $i = 1, \ldots, n$. I say "perfect", but it's not "ideal", since it means $\varepsilon_i = 0$ for all i. That's not very Gaussian. As in any fitting exercise, it's better to let the data help you decide, and when in doubt keep it simple to avoid overfit. Check if you need to go up to degree d by testing the hippopotamus that $\beta_d = 0$. I'll show you.

Polynomial features like this are a kind of transformation of x; one that fits into the linear framework, albeit in higher dimension (an MLR). You can always combine this with other transformations, like a log if you have bunching or want to ensure positivity. Here is an example where both are used at once. (I told you we'd move fast.)

Consider an experiment relating soybean weight to various factors. See Davidian and Giltinan (1995), with data provided by `nmle` on CRAN (Pinheiro et al., 2025). Here we shall analyze a subset of these data and focus on a time–weight relationship.

```
library(nlme)    ## don't forget install.packages("nmle")
data(Soybean)
soy <- Soybean[Soybean$Time < 65 & Soybean$Variety == "P",]
```

Figure 13.8 shows two views into a fit on these data. The left panel uses original units, but has a fit derived from a linear model on the log-response.

```
x <- soy$Time
y <- log(soy$weight)                        ## transform output with y
lfit <- lm(y ~ x)                           ## (log) linear model fit
xp <- seq(min(x), max(x), length=1000)
p <- predict(lfit, newdata=data.frame(x=xp), interval="prediction")
```

When you look at the data, you can see why: there's bunching in the y-axis. Plus, soybean weights can't be negative. The residual analysis summarized in the right panel shows that the fit (to $\log y$) isn't ideal. There's nonlinearity left over in the residuals. Looking back at the left panel, perhaps it would've been possible to see the over-, then under-, then over-predicting pattern there, too. A residual analysis makes that much clearer by "zooming" in. What to do?

```
par(mfrow=c(1, 2))

## view in original space (left panel)
plot(soy$Time, soy$weight, xlab="time (days)", ylab="weight (g)")
```

```
lines(xp, exp(p[,1]), lwd=2)
lines(xp, exp(p[,2]), col=2, lty=2, lwd=2)
lines(xp, exp(p[,3]), col=2, lty=2, lwd=2)
legend("topleft", c("mean", "95% PI"), col=1:2, lty=1:2, lwd=2, bty="n")

## residual analysis (right panel)
plot(lfit$fitted, lfit$residual,
  xlab="fitted values (yhat)", ylab="residuals (e)")
abline(h=0, col=2, lty=2, lwd=2)
legend("bottom", "zero residual", col=2, lty=2, lwd=2, bty="n")
```

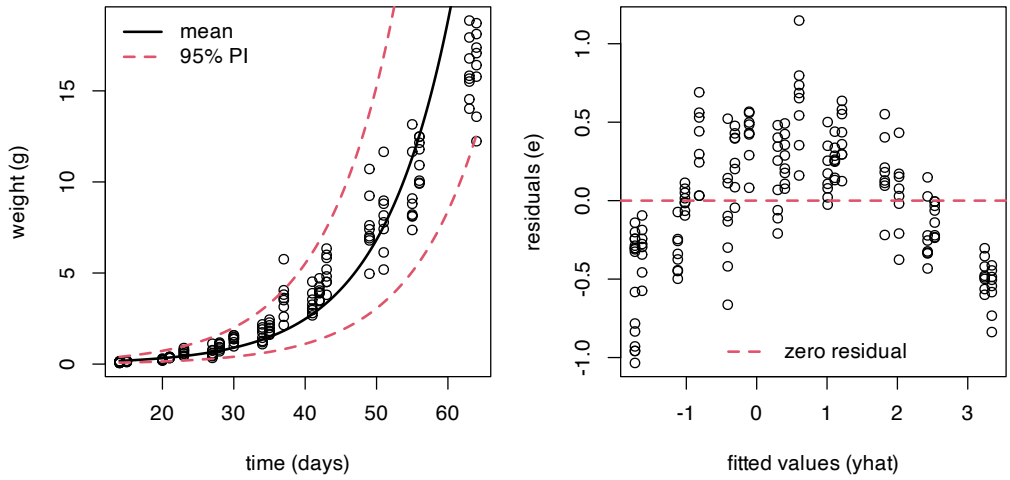

FIGURE 13.8: Fit to soybean data with log-response, back on original scale (left) and residual diagnostics on a log-scale (right).

Figure 13.9 shows what happens when you create a degree-2 feature for x, which is `$Time`, and use it in a degree-2 polynomial mean – a quadratic regression.

```
x2 <- x^2                                  ## degree-2 training feature
lfit2 <- lm(y ~ x + x2)                    ## technically our first MLR
xp2 <- xp^2                                ## degree-2 testing feature
p2 <- predict(lfit2, newdata=data.frame(x=xp, x2=xp2),
  interval="prediction")
```

I know I'm moving lightning quick. Hold on tight. The left panel focuses on the polynomial part. Look how adding the new feature `x2`, which is really `$Time^2`, lends curvature to the surface when plotted as a function of `xp`-values only. Notice how `xp2`-values are used in `predict`, but they don't feature later in the plot.

```
par(mfrow=c(1, 2))

## left panel, view in log y space
plot(x, y)
```

```
lines(xp, p2[,1], lwd=2)
lines(xp, p2[,2], col=2, lty=2, lwd=2)
lines(xp, p2[,3], col=2, lty=2, lwd=2)

## right panel, view in original y space
plot(soy$Time, soy$weight,
  xlab="time (days)", ylab="weight (g)")
lines(xp, exp(p2[,1]), lwd=2)
lines(xp, exp(p2[,2]), col=2, lty=2, lwd=2)
lines(xp, exp(p2[,3]), col=2, lty=2, lwd=2)
legend("topleft", c("mean", "95% PI"), col=1:2, lty=1:2, lwd=2, bty="n")
```

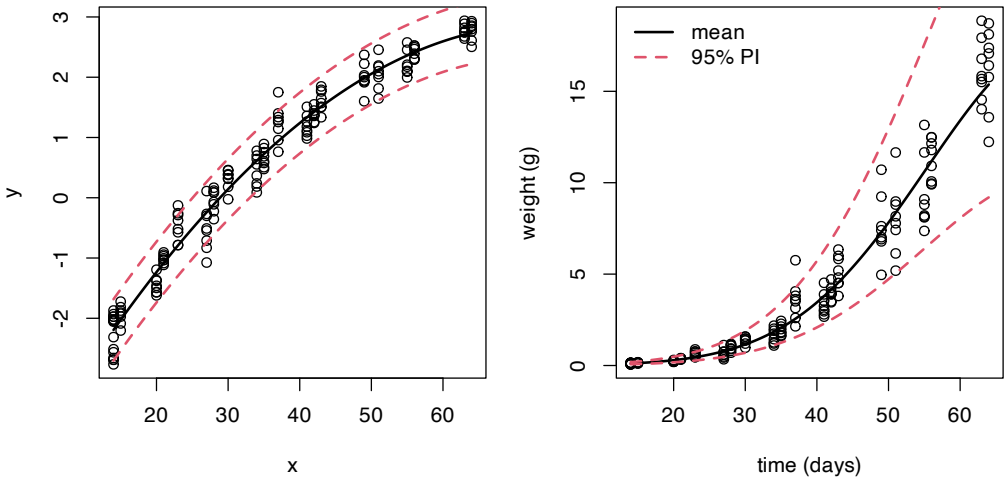

FIGURE 13.9: Degree-2 fit to soybean data with log-response, on log-scale (left) and back on the original scale (right).

The right panel in the figure undoes everything, putting our visual back on the original scale(s). Comparing to the left panel of Figure 13.8, differences are subtle. To my eye we now have a more nuanced, tighter nonlinear fit.

How can I know for sure that it was worthwhile to include an additional degree-2 feature? I can test $\mathcal{H}_0 : \beta_2 = 0$. Table 13.2 provides a summary of coefficients, standard errors, t statistics and p-values. Have a look at the new row corresponding to β_2, and notice the zero p-value (after rounding).

```
sumcoef <- summary(lfit2)$coefficients
colnames(sumcoef) <- c("est", "stderr", "t stat", "p val")
kable(round(sumcoef, 4), caption="Soybean LM on log-y, quadratic fit.")
```

It could be that an even higher-degree polynomial would provide a better fit. That's easy to check.

```
x3 <- x^3
lfit3 <- lm(y ~ x + x2 + x3)
```

TABLE 13.2: Soybean LM on log-y, quadratic fit.

	est	stderr	t stat	p val
(Intercept)	-4.8233	0.1159	-41.62	0
x	0.2074	0.0067	30.77	0
x2	-0.0014	0.0001	-16.00	0

```
pval3 <- summary(lfit3)$coefficients[4, 4]
pval3
```

```
## [1] 0.1447
```

There's not enough evidence to reject $\mathcal{H}_0 : \beta_3 = 0$, so it must be that a degree-2 polynomial is sufficient for these data.

There was a lot going on in that example: logs, polynomials, residual diagnostics and converting back and forth between original and transformed scales. Hopefully each building block is not too much to take in, on its own. The only real magic here is the `lm` command, which can apparently take in more than one `x`-value if you string them together with `+` in the formula: `lm(y ~ x + x2 + x3)`. What's going on under the hood?

13.3 Multiple linear regression

Many data analysis settings involve more than one independent variable or factor which affects the dependent or response variable. Reviewing some of the examples we explored previously, here are some possible extensions. There are multi-factor asset pricing models (beyond CAPM). Demand for a product may depend on prices of competing brands, advertising, household attributes, etc. House prices depend on more than just size.

In SLR, the conditional mean of Y depends on x. *Multiple linear regression* (MLR) extends this idea to include more than one independent variable. As with SLR, there are several ways to write out the model. Two of them have direct SLR analogs, but my favorite is the third one, which is new.

$$\begin{aligned}
1: &\qquad Y_i = \beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d + \varepsilon_i, &\qquad \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2), \\
2: &\qquad Y_i \mid x_{i1}, \ldots, x_{id} \overset{\text{ind}}{\sim} \mathcal{N}(\beta_0 + \beta_1 x_1 + \cdots + \beta_d x_d, \sigma^2) &\qquad i = 1, \ldots, n \\
3: &\qquad Y \sim \mathcal{N}_n(X\beta, \mathbb{I}_n \sigma^2). &\qquad (13.2)
\end{aligned}$$

Above, $\mathcal{N}_n$ indicates an n-variate multivariate normal (MVN)[16] distribution. The new, vectorized approach (13.2) is easier to work with mathematically if you're not afraid of some linear algebra[17], but it requires some unpacking.

- $Y = (Y_1, \ldots, Y_n)^\top$, with y (still an n-vector) being in used non-random contexts;
- β is a $p = (d + 1)$-vector $(\beta_0, \beta_1, \ldots, \beta_d)^\top$ of unknown regression coefficients;

[16] https://en.wikipedia.org/wiki/Multivariate_normal_distribution
[17] https://en.wikipedia.org/wiki/Linear_algebra

- $\mathbb{I}_n$ is an $n \times n$ identity matrix;
- X is an $n \times p$ design matrix[18] with leading column of 1s, and each of d coordinates stacked as columns thereafter. Its i^{th} row is $x_i^\top = (1,\ x_{i1}, \cdots, x_{id})$.

All of this applies to SLR as well, taking $d = 1$, but it could represent overkill in that setting. Eq. (13.2) is still a "linear" model even though $\beta_0 + \sum_{j=1}^{d} \beta_j x_j$ is not a strictly a "line" unless $d = 1$. Linear simply refers to the way in which X and β are combined, as a matrix-vector product. Linear algebra, if it can be distilled down to a phrase, is merely an "abstraction of sums of products", and other operations on those quantities. When $d = 2$ the "linear" surface is a plane[19] in two dimensions. For $d = 3, 4, ...$ we say hyperplane[20].

Quick digression on notation. Sometimes, I'll write $x_{\cdot j}$ to refer to the j^{th} column of X. Other times, I'll use x_j, interchangeably. A $\cdot$ is intended to remind you of row indexing for any $i = 1, \ldots, n$. In both cases, using j rather than i is intended to signal column indexing, whereas i is reserved for rows. Usually, my choice about one or the other is settled by a spur-of-the-moment decision.

Alright, back to MLR. Notice how polynomial regression (13.1) is an MLR with $x_j = x^j$ for $j = 1, \ldots, d$. In fact, that allows me to begin an illustration and draw a comparison to library-based calculations from §13.2. The n-vector y is already stored in `y`, containing `log(soy$weight)` values. An $n \times p$ design matrix X, where $p = 3$ for the quadratic case, may be built as follows.

```
X <- cbind(1, x, x2)
```

Recall that `x = soy$time` and `x2` $\equiv x^2$. If it bothers you that I'm using a polynomial regression to illustrate MLR, don't worry. I've got plenty more MLR examples coming.

Importantly, key assumptions underlying MLR are the same as SLR. The conditional mean of Y_i is linear in the x_{ij} variables, for $j = 1, \ldots, p$. Additive errors ε_i, representing noisy deviations from the line, are still Gaussian, independent from each other, and identically distributed (i.e., they have constant variance, σ^2).

Interpretation of regression coefficients can be extended from the single covariate case through partial derivatives

$$\beta_j = \frac{\partial\, \mathbb{E}\{Y \mid x_1, \ldots, x_d\}}{\partial x_j}.$$

That is, holding all other variables constant, β_j is the average change in Y per unit change in x_j.

Therein lies the biggest challenge: there's more to keep track of with more columns of X. Consequently, one looks through fitted values $\hat{y}_i = \hat{y}_i = b_0 + b_1 x_{i1} + \cdots + b_d x_{id}$, where b is any estimate of β. That way, residuals are unchanged as $e_i = y_i - \hat{y}_i$, and residual diagnostics need not proceed on an $x_{\cdot j}$ basis. Least squares is also unchanged; just minimize the residual sum of squares $\sum_{i=1}^{n} e_i^2$ via calculus: take a gradient and solve for the best b-vector. It may not surprise you that the answer is the same via the likelihood. I'll take that route, in part because it allows me to fill in some gaps left earlier.

[18] https://en.wikipedia.org/wiki/Design_matrix
[19] https://en.wikipedia.org/wiki/Plane_(mathematics)
[20] https://en.wikipedia.org/wiki/Hyperplane

MLR inference and analysis

Calculating the MLE follows Alg. 3.1 where a likelihood is provided by Eq. (13.2). Using an MVN density with mean $X\beta$ and variance $\sigma^2\mathbb{I}_n$, we have

$$\begin{aligned} L(\beta, \sigma^2) &= (2\pi\sigma^2)^{-n/2} \exp\left\{-\frac{1}{2\sigma^2}(y - X\beta)^\top (y - X\beta)\right\} \\ \ell(\beta, \sigma^2) \equiv \log L &= c - \frac{n}{2}\log\sigma^2 - \frac{1}{2\sigma^2}(y - X\beta)^\top (y - X\beta) \\ &= c - \frac{n}{2}\log\sigma^2 - \frac{1}{2\sigma^2}(y^\top y - 2y^\top X\beta + \beta^\top X^\top X\beta). \end{aligned} \tag{13.3}$$

Even if you're not super comfortable with linear algebra, I bet you can follow the development above if you're patient with it. The term $(y - X\beta)^\top (y - X\beta)$, which is really a dot product[21], is one way to write out squared Euclidean distance[22] in p dimensions.

$$||y - X\beta||^2 = (y - X\beta)^\top (y - X\beta) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_{i1} - \cdots - \beta_d x_{id})^2 \tag{13.4}$$

The rest is factorizing products, simplifying and dropping terms (subsumed in c) which are constant with respect to parameters σ^2 and β. Note that $y^\top X\beta = \beta^\top X^\top y$ since transposes reverse the order of matrix/vector products and both evaluate to scalars.

Returning back to our expression for the log-likelihood above, take partial derivatives with respect to β and solve.

$$\begin{aligned} 0 = \frac{\partial \ell}{\partial \beta}\Big|_{\beta=\hat{\beta}} &= \frac{1}{\sigma^2}(X^\top y + X^\top X\hat{\beta}) \\ X^\top X\hat{\beta} &= X^\top y \end{aligned}$$

As long as $X^\top X$ is of invertible, which happens when X is of full rank[23],

$$\hat{\beta} = (X^\top X)^{-1} X^\top y. \tag{13.5}$$

Look closely at that equation, it might be familiar. Does $b_1 = s_{yx}/s_{xx}$ ring any bells? It's easy to verify that Eq. (13.5) agrees with any of the $d = 1$ solutions in Chapter 7. I'll leave that to you and continue with the `soy` polynomial regression illustration.

```
XtX <- t(X) %*% X             ## t is transpose, %*% is matrix product
XtXi <- solve(XtX)            ## solve is inv with only one argument
Xty  <- t(X) %*% y            ## %*% also for matrix-vector product
bhat <- drop(XtXi %*% Xty)    ## drop from a 1xp matrix to p-vector
rbind(coef(lfit2), bhat)
```

```
##      (Intercept)      x        x2
##           -4.823 0.2074 -0.001396
## bhat      -4.823 0.2074 -0.001396
```

[21] https://en.wikipedia.org/wiki/Dot_product
[22] https://en.wikipedia.org/wiki/Euclidean_distance
[23] https://en.wikipedia.org/wiki/Rank_(linear_algebra)

Those are the same. I bet you can do that as a one-liner, but I didn't because I wanted to go through it slowly. Also, I'll need some of those intermediate building blocks later.

An important result for MVNs involves projection[24]. Think of projection in this context as a marginal comprised of a linear combination of the components of a random vector. We already know that marginals of MVNs are Gaussian, and it turns out that projections are, too. The particular result I need here is the following: If $Y \sim \mathcal{N}_n(\mu, \Sigma)$ and P is a $p \times n$ matrix, then $PY \sim \mathcal{N}_p(P\mu, P\Sigma P^\top)$.

What is that useful for? Let $P = (X^\top X)^{-1} X^\top$ so that $\hat{\beta} = Py$. Observe that

$$\begin{aligned} PX\beta &= (X^\top X)^{-1} X^\top X \beta = \beta, \\ \text{and} \quad PP^\top &= (X^\top X)^{-1} X^\top X (X^\top X)^{-1} = (X^\top X)^{-1} \end{aligned}$$

since $X^\top X$ is symmetric, and then so is its inverse. Therefore, using the MVN result above, we obtain

$$\hat{\beta} = PY \sim \mathcal{N}_p(\beta, \sigma^2 (X^\top X)^{-1}) \tag{13.6}$$

giving the sampling distribution of $\hat{\beta}$. This works for all p components of $\hat{\beta}$ at once. Isn't that neat? You get one (vectorized) formula for the entire set of coefficients, slope(s) and intercept combined. No need to separate them out like Eqs. (7.25) and (7.23). And their covariance (7.27) comes for free!

I saved `XtXi` earlier, how prescient! Before that can be used to extract standard errors for p-values and CIs, I need an estimate of σ^2. Plugging in $\hat{\beta}$ into the log-likelihood (13.3) with Eq. (13.4) gives

$$\ell(\hat{\beta}, \sigma^2) = -\frac{n}{2} \log \sigma^2 - \frac{1}{2\sigma^2} ||y - X\hat{\beta}||^2.$$

Differentiating with respect to σ^2 and solving, yields

$$\begin{aligned} 0 = \frac{\partial \ell}{\partial \sigma^2}\Big|_{(\hat{\beta}, \hat{\sigma}^2)} &= -\frac{n}{2\hat{\sigma}^2} + \frac{1}{2\hat{\sigma}^4} ||y - X\beta||^2 \\ \hat{\sigma}^2 &= \frac{1}{n} ||y - X\hat{\beta}||^2. \end{aligned}$$

One can argue that this induces bias which is corrected with

$$s^2 = \frac{1}{n-p} ||y - X\hat{\beta}||^2 \tag{13.7}$$

where $p = d + 1$ is the degrees of freedom (DoF): the number of parameters estimated via $\hat{\beta}$ before calculating variance. As with s^2 back in Chapter 3, Eq. (3.16), it may be shown that $\sigma^2 \sim (n-p)s^2/\chi^2_{n-p}$ following Cochran's theorem[25], but I'll spare you the details.

[24] https://en.wikipedia.org/wiki/Projection_(linear_algebra)
[25] https://en.wikipedia.org/wiki/Cochran's_theorem

TABLE 13.3: By-hand re-do of soybean summary.

	est	stderr	t stat	p val
(Intercept)	-4.8233	0.1159	-41.62	0
x	0.2074	0.0067	30.77	0
x2	-0.0014	0.0001	-16.00	0

```
p <- ncol(X)
n <- nrow(X)
s2 <- sum((y - X %*% bhat)^2)/(n - p)
c(hand=s2, lib=summary(lfit2)$sigma^2)
```

```
##    hand     lib
## 0.06457 0.06457
```

Also like in Chapter 3, estimating σ^2 changes an MVN into a (multivariate) Student-t. Table 13.3 is the by-hand version of Table 13.2. It seems sorta silly to put two copies of the same table, but it's important to check this so there's no smoke-and-mirrors.

```
se <- sqrt(s2*diag(XtXi))
t <- bhat/se
pval <- 2*pt(-abs(t), n - p)
tab <- cbind(bhat, se, t, pval)
colnames(tab) <- c("est", "stderr", "t stat", "p val")
rownames(tab) <- c("(Intercept)", "x", "x2")
kable(round(tab, 4), caption="By-hand re-do of soybean summary.")
```

MLR leverage and Prediction

Indulge me for one more thing. (Actually, I've got two more things, but if you look at them in a certain way they're the same.) The projection trick can be used a second time to derive the distribution of errors, e. Let

$$H = X(X^\top X)^{-1}X^\top \tag{13.8}$$

so that $\hat{y} = Hy = X\hat{\beta}$. Sometimes, H is called the *hat matrix* because it "puts the hat on y". This is one of those rare instances of a clever name from the stats community.

```
H <- X %*% XtXi %*% t(X)
```

H is a special kind of matrix called an orthogonal projection[26], with many nice properties: $H^\top = H$, $H^2 = H$, $(\mathbb{I}_n - H)^\top = (\mathbb{I}_n - H)$, and $(\mathbb{I}_n - H)^2 = (\mathbb{I}_n - H)$. Eponymously, Hy and $y - Hy$ are orthogonal, meaning their product is the identity. How is that useful?

Observe that

$$e = Y - \hat{Y} = Y - X\hat{\beta} = Y - X(X^\top X)^{-1}X^\top Y \equiv (\mathbb{I}_n - H)Y.$$

[26] https://en.wikipedia.org/wiki/Projection_(linear_algebra)#Orthogonal_projections

If $Y \sim N_n(X\beta, \sigma^2 I)$ is correct, our result about projections of MVNs gives

$$\begin{aligned} & e \sim N_n((\mathbb{I}_n - H)X\beta, \sigma^2(\mathbb{I}_n - H)(\mathbb{I}_n - H)^\top). \\ \text{Therefore} \quad & e \sim N_n(0, \sigma^2(\mathbb{I}_n - H)), \end{aligned} \tag{13.9}$$

since H is a projection and $X\beta - HX\beta = 0$. In words, residual vector e is MVN like ε, but its covariance is not a scaled identity. Residuals are correlated, not independent. That's what leads to the hyperbolic shape of predictive error bars, which I'm coming to next.

A really good check of understanding here is the following trivia question. Which line is closer to the data: (a) the estimated line using MLE $\widehat{\beta}$; or (b) the true line using β? I'll give you a hint. The answer involves comparing Eq. (13.9) to the distribution for ε implied by Eq. (13.2). Maybe you'll see that on a quiz.

Components on the diagonal of H, which are often notated as $h_i = \text{diag}(H)_i$, are referred as the leverage[27] or *influence* of point (x_i, y_i). Quantity h_i measures how much (x_i, y_i) pulls e_i toward zero in $\mathbb{V}\text{ar}\{e\}$. Importantly, h_i only depends on x_i, not y_i. That's another good trivia question.

In 1d, leverage and distance between x_i and $\bar{x}$ are pretty much the same thing, so the metaphor works. A lever exerts more force if you push/pull on it far from the pivot joint, $\bar{x}$. With $p = 3$ columns of X, especially ones that are powers of a single x, the notion of distance becomes more complicated. Figure 13.10 provides a visual of the leverage of the training data points in the soybean quadratic regression.

```
h <- diag(H)
jit <- rnorm(length(x), 0, 0.5)  ## horizontal jitter
plot(x + jit, h, pch=18,
  xlab="inputs xi (time)", ylab="leverage hi")
```

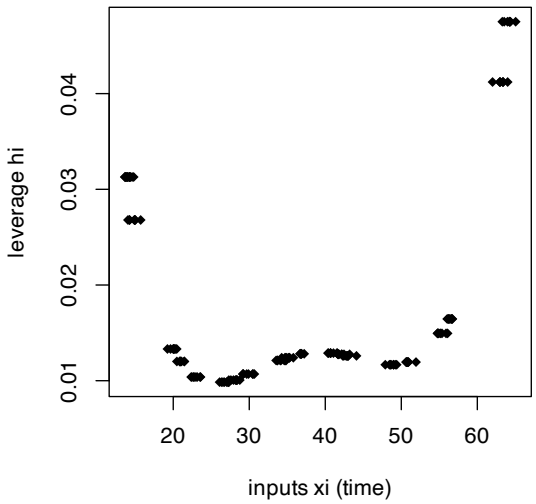

FIGURE 13.10: Illustrating leverage for the soybean quadratic regression.

There are many repeated x-values in that example, so there are many fewer than $n = 168$

[27] https://en.wikipedia.org/wiki/Leverage_(statistics)

unique leverages (20). I've jitterd them a little on the x-axis to help reveal their multitude. As you can see in the figure, distance from the middle of the x-values contributes to leverage, but it's not the whole story.

If you re-purpose the hat matrix (13.8) with a predictive vector x_p, rather than training X, you can use it for prediction:

$$h_p = x_p^\top (X^\top X)^{-1} x_p.$$

Understanding the distribution of predictive errors, $e_p = 1 + h_p$ is the hard part; predicting with $x_p^\top \hat{\beta}$ is easy. I know I'm moving fast here, but similar projection arguments for e with H can be used to deduce that $\mathbb{V}\text{ar}\{Y_p \mid x_p\} = \sigma^2(1 + h_p) = \sigma^2(1 + x_p^\top (X^\top X)^{-1} x_p)$. Notice the $1+$ here rather than $\mathbb{I}_n -$ in Eq. (13.9). That's not a mistake. Residuals e are pulled to zero by their leverage. The opposite thing happens when predicting. That pull is translated into uncertainty, imparting a hyperbolic shape.

Here that is in code, for multiple x_p at once, forming predictive matrix X_p.

```
Xp <- cbind(1, xp, xp2)
Hp <- Xp %*% XtXi %*% t(Xp)
Spred <- s2*(diag(1, nrow(Xp)) + Hp)          ## for PIs
spred <- sqrt(diag(Spred))
Sfit <- s2*Hp                                 ## for CIs if desired
sfit <- sqrt(diag(Sfit))                      ## not used below
q <- qt(0.975, n - p)
PI.u <- drop(Xp %*% bhat) + q*spred           ## checking upper PI only
round(mean((PI.u - p2[,3])^2), 8)             ## lower would be similar
```

```
## [1] 0
```

As you can see, the upper PI error bar is within machine precision as compared to the one calculated earlier via `predict` in §13.2. A comparison on the lower one would be similar.

13.4 Dummies and Interactions

A dummy variable[28] is smarter than it sounds. It allows you to use a categorical input, like a grouping variable for ANOVA (§6.3), in an MLR. For example, one of the columns of the pickup truck data is vehicle make.

```
table(pickups$make)
```

```
##
## Dodge  Ford   GMC
##    10    12    23
```

[28] https://en.wikipedia.org/wiki/Dummy_variable_(statistics)

Dummies can be used to convert a non-numeric grouping into a real-valued predictor (actually binary[29]) that can be used with MLR linear algebra, as outlined in the previous section. For example, here are two dummy variables for the three categories of pickup truck `$make`, notated as w_i for $i = 1, \ldots, n$.

$$x_{i2} = \mathbb{I}_{\{w_i=\text{Ford}\}} \quad \text{and} \quad x_{i3} = \mathbb{I}_{\{w_i=\text{GMC}\}}$$

These two new columns of X can represent each w_i for all three categories: Dodge is $(x_{i2}, x_{i3}) = (0, 0)$, Ford is $(1, 0)$, and GMC is $(0, 0)$. Notice that Dodge's representation is in the negative space of neither Ford nor GMC. It is said to be "in the intercept" since slopes β_2 and β_3 in the MLR would be zero:

$$\text{Model:} \qquad Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \varepsilon.$$

Recall that $x_{\cdot 1}$ in the pickup truck MLR is year. One may create a design matrix containing dummy variables in R as follows.

```
X <- cbind(1, pickups$year, pickups$make == "Ford", pickups$make == "GMC")
y <- log(pickups$price)
bhat <- drop(solve(t(X) %*% X) %*% t(X) %*% y)
```

Or, R will do it for you if you provide a categorical input in the `lm` formula.

```
fit <- lm(lprice ~ year + make, data=pickups)
rbind(coef(fit), bhat)
```

```
##      (Intercept)   year makeFord makeGMC
##           -272.7 0.1407   0.1257  0.1355
## bhat      -272.7 0.1407   0.1257  0.1355
```

R builds dummy variables alphanumerically unless you `relevel` the categorical `factor` to specify a different order. That puts the first `pickups$make`, which is Dodge, in the intercept. Was it worthwhile including these two additional degrees of freedom? Table 13.4 shows that the answer is no. Neither of the p-values for the two dummy variables is below the usual 5% threshold.

```
sumcoef <- summary(fit)$coefficients
colnames(sumcoef) <- c("est", "stderr", "t stat", "p val")
kable(round(sumcoef, 4), caption="Pickup truck LM/SLR fit with dummies.")
```

A word of caution here: that conclusion was reached by doing two t-tests simultaneously, when technically they must be one-at-a-time. A better way is via partial f-testing, which is the subject of §13.5.

You can even do an MLR entirely with dummy variables, and guess what? That's an ANOVA! Recall the frozen treats example from Chapter 6. Scoop weights are written out a little differently below, but the `ydf` object should be the same as the `data.frame` used in §6.3.

[29] https://en.wikipedia.org/wiki/Binary_number

TABLE 13.4: Pickup truck LM/SLR fit with dummies.

	est	stderr	t stat	p val
(Intercept)	-272.6906	22.0835	-12.3482	0.0000
year	0.1407	0.0110	12.7430	0.0000
makeFord	0.1257	0.1435	0.8758	0.3862
makeGMC	0.1355	0.1276	1.0620	0.2945

```
## data from Chapter 6
sorbet <- c(4.0, 5.5, 4.9, 3.8, 4.3, 4.4, 5.1, 4.1, 4.0)
icecream <- c(3.0, 3.7, 3.4, 3.6, 3.0, 3.5, 4.0)
fatfree <- c(3.5, 3.4, 3.7, 3.4, 3.9, 4.2, 4.0, 4.0, 3.8, 3.8)

## data.frame representation
ydf <- data.frame(
  weight=c(sorbet, icecream, fatfree),
  type=c(rep("sorbet", length(sorbet)), rep("icecream", length(icecream)),
    rep("fatfree", length(fatfree))))

## linear model fit and f statistic
fit <- lm(weight ~ type, data=ydf)
summary(fit)$fstatistic
```

```
## value numdf dendf
##  11.9   2.0  23.0
```

Check that this is the same as what was provided in Table 6.1. The difference here is that sums of squares are not directly involved. Hold that thought until §13.5. Although an MLR model (13.2) is superficially different than an ANOVA one (6.5), they are operationally equivalent. They have the same number of parameters, assume Gaussians whose means are determined by d levels or slopes (and intercept) and common variance σ^2. Group means can be recovered from an MLR fit with dummies by adding the intercept to the slopes.

```
bhat <- coef(fit)
rbind(
  c(fatfree=mean(fatfree), icecream=mean(icecream), sorbet=mean(sorbet)),
  c(bhat[1], bhat[1] + bhat[-1]))
```

```
##      fatfree icecream sorbet
## [1,]    3.77    3.457  4.456
## [2,]    3.77    3.457  4.456
```

That's dummies – pretty smart, right? Now I get to pivot to something entirely different, but equally important in an MLR context.

Interactions

So far we've considered the impact of each predictor variable x_j in an additive way. A valuable extension which keeps us in the MLR framework, but allows x_j to affect the mean in a less independent fashion, is through multiplicative *interaction effects*:

$$\text{Model:} \qquad Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \beta_3 x_{i1} x_{i2} + \cdots + \varepsilon_i,$$

so that, for example

$$\frac{\partial \mathbb{E}\{Y \mid x_1, x_2\}}{\partial x_1} = \beta_1 + \beta_3 x_2.$$

Interactions are useful whenever the rate of change in the response due to one input depends on the setting of another input.

I like to think of interaction terms as new features, $x_{1:2} = x_1 x_2$ derived from a product of two other base inputs or features x_2 and x_3. Sometimes, you'll see regression coefficients notated similarly as $\beta_{1:2}$ or β_{12} rather than β_3. You can even interact and input with itself, like $x_1 x_1$, building a quadratic term x_1^2.

Consider the connection between college (i.e., bachelor's degree) grades and grades in a Masters of Business Administration (MBA) program. You might think that students who performed well during their bachelor's study would also do well in an MBA program. A model capturing that might look like the following, using some shorthand:

$$\text{Model:} \qquad \text{GPA}^{\text{MBA}} = \beta_0 + \beta_1 \text{GPA}^{\text{Bach}} + \varepsilon.$$

Here's a fit to some data collected at the University of Chicago Booth School of Business[30], and available on the book webpage as grades.csv[31].

```
grades <- read.csv("grades.csv")
fit1 <- lm(MBAGPA ~ BachGPA, data=grades)
summary(fit1)$coefficients
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   2.5899    0.31206   8.299 1.199e-11
## BachGPA       0.2627    0.09244   2.842 6.066e-03
```

I'm skipping the formal table environment here to help streamline things a bit. Hope you don't mind. You can see from the `summary` above that the slope on `BachGPA` is positive and statistically significant. For every one point increase in college GPA, expected MBA GPA increases by about 0.26 points.

What other things might affect MBA GPA? The data file contains an `age` column; we could try incorporating that.

```
fit2 <- lm(MBAGPA ~ BachGPA + Age, data=grades)
summary(fit2)$coefficients
```

[30] https://www.chicagobooth.edu/
[31] https://bobby.gramacy.com/hipp0/grades.csv

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  3.66400   0.444203   8.248 1.645e-11
## BachGPA      0.19495   0.088793   2.196 3.194e-02
## Age         -0.03024   0.009446  -3.201 2.173e-03
```

Yes, age matters and the coefficient is negative. Apparently, older students tend not to do as well as younger ones. I don't know about you, but that doesn't feel right to me. Many MBA students enroll after they've worked in industry, gaining valuable experience in the workforce after graduating with their bachelor's. The very best students, with the highest GPAs, get the best jobs and may be the most likely to delay further education. In other words, what's really going on might be more nuanced. It may be worth entertaining an interaction between age and college GPA.

In R, there are two clean ways to invoke an interactions via `lm`. One is to explicitly write out the new feature as `BachGPA:Age`, along with anything else. Or, include both so-called "marginal effects" `BachGPA` and `Age` as well as their interaction `BachGPA:AGE` with shorthand `BachGPA*Age`.

```
fit3 <- lm(MBAGPA ~ BachGPA*Age, data=grades)
## or lm(MBAGPA ~ BachGPA + Age + BachGPA:Age, data=grades)
summary(fit3)$coefficients
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.27964    2.59259 -0.1079  0.91447
## BachGPA      1.36936    0.76596  1.7878  0.07886
## Age          0.10974    0.09117  1.2036  0.23346
## BachGPA:Age -0.04181    0.02709 -1.5434  0.12798
```

What happened? We have a fit that makes more sense, that's what. Both `BachGPA` and `Age` have positive coefficients. The interaction `BachGPA:Age` has a negative coefficient, meaning that how well you did in college matters less over time. That's the good news.

The bad news is that none of the coefficients are statistically significant. This happened because our columns of X, the design matrix, are multicollinear[32]. More on that is coming next in §13.5. For now, it simply means that we've used too many DoFs to explore variability in our data, making it difficult to detect (an over-complicated) signal from noise.

How about including the interaction effect, but not both marginal effects? That's one fewer DoF, a more parsimonious[33] model.

```
fit4 <- lm(MBAGPA ~ BachGPA*Age - Age, data=grades)
## same as lm(MBAGPA ~ BachGPA + BachGPA:Age)
summary(fit4)$coefficients
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept)  2.820494   0.296928   9.499 1.227e-13
## BachGPA      0.455750   0.103026   4.424 4.073e-05
## BachGPA:Age -0.009377   0.002786  -3.366 1.323e-03
```

Would you look at that! Everything is statistically significant and the story makes sense.

[32] https://en.wikipedia.org/wiki/Multicollinearity
[33] https://en.wikipedia.org/wiki/Occam's_razor

People who did better in college tend to do better in graduate school, but the effect is weakened for older students who've gained valuable experience in industry before enrolling in an MBA.

It is interesting to contrast predictive surfaces under a *main effects*-only model (`fit2`), meaning without interactions, and one with interactions (`fit4`). Have a look at Figure 13.11. There's a lot going on in the code for the figure, and I realize this is the first time you're seeing an image and contour plot. Redder is lower and yellower/whiter is higher. Contours trace out evenly-spaced zero-gradient ridges adding precision to a visual topography provided by color. Pretty cool, right?

```
## calculate predictions under both models
ggrid <- seq(min(grades$BachGPA), max(grades$BachGPA), length=100)
agrid <- seq(min(grades$Age), max(grades$Age), length=100)
g <- expand.grid(ggrid, agrid)
p2 <- predict(fit2, newdata=data.frame(BachGPA=g[,1], Age=g[,2]))
p4 <- predict(fit4, newdata=data.frame(BachGPA=g[,1], Age=g[,2]))

## ensure same color palette for both images
cs <- heat.colors(128)
bs <- seq(min(grades$MBAGPA), max(grades$MBAGPA), length=129)

## main effects on left, interactions on right
par(mfrow=c(1, 2))
image(ggrid, agrid, matrix(p2, ncol=100), col=cs, breaks=bs,
  xlab="Bachelors GPA", "ylab"="Age")
contour(ggrid, agrid, matrix(p2, ncol=100), add=TRUE)
image(ggrid, agrid, matrix(p4, ncol=100), col=cs, breaks=bs,
  xlab="Bachelors GPA", "ylab"="Age")
contour(ggrid, agrid, matrix(p4, ncol=100), add=TRUE)
```

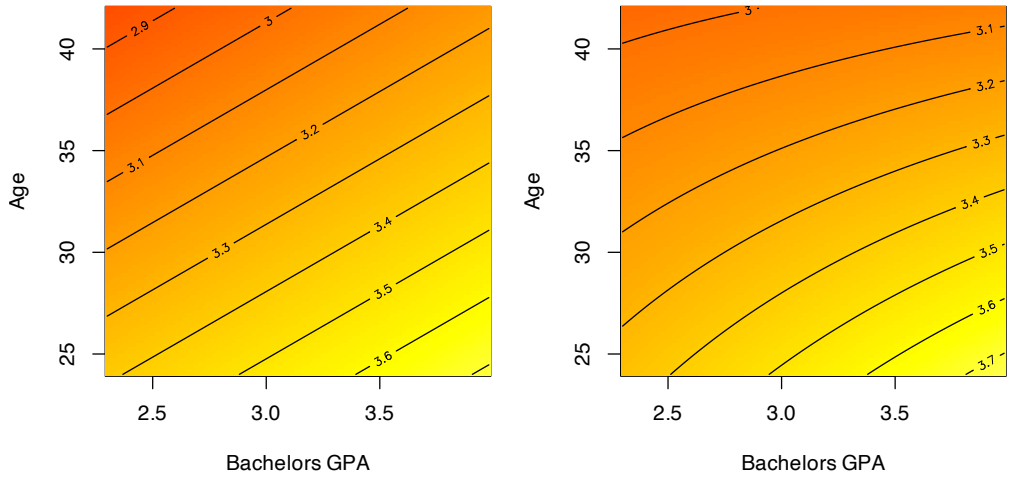

FIGURE 13.11: Comparing main effects only (left) to main effects plus interaction (right) on the MBA grades data.

The main thing to notice in the figure is that contour lines are straight on the left, tracing

out a plane in 2D. On the right, those lines are curved because variables are interacting. Polynomial regression leverages nonlinearity in features (powers of x) to wring non-linearity out of an inherently linear framework (13.2). Interactions provide nonlinear response surfaces *across* pairs of inputs.

There are shorthands in R to obtain model fits with all main effects and/or all interactions.

```
fit5 <- lm(MBAGPA ~ ., data=grades)          ## all main effects .
fit6 <- lm(MBAGPA ~ . + .^2, data=grades)    ## plus interactions .^2
```

If you use one of these, be careful to ensure that the contents of the `data=` argument only has columns that you intend to use. If, for example, you have augmented the data frame with transformed versions (like with `pickups` in §13.1), you'll get something other than intended/wrong results.

I'm going to leave it to you to look at `fit5` and `fit6` later (§13.6). We've explored a lot of models already. How do you know which ones are really the best?

13.5 Model selection

We've made it to the last non-homework/Appendix section in the book, and I promised you some Monte Carlo (MC). Choosing the "right" MLR is hard because there could be many predictors (inputs), and many features derived from those predictors (quadratic, interaction, log-transform), and chances are that many of them are collinear with one another.

Multicollinearity

What does that last thing mean? It means that the individual inputs, columns $x_{\cdot j}$ of X, don't have "unique" information about our response y. I put "unique" in quotes because the right word here is orthogonal[34], but it's a bit nerdy. Consider the pickup truck data. One of the columns is `$miles`, which is collinear with `$year`. Older vehicles tend to have been driven farther. Those predictors don't have unique information about the response, `$price`. Sometimes collinearity is unavoidable, other times it's in there on purpose, like when introducing interaction or quadratic terms.

Multicollinearity means that the columns of X covary with one another, so they muddy the water when extracting correlation with y in a linear model. In symbols, when multicollinearity is high $X^\top X$ is unstable relative to $X^\top y$ in $\hat{\beta} = (X^\top X)^{-1} X^\top y$. The result may be spuriously large components $\hat{\beta}_j$ and/or standard errors (13.6) on the diagonal of $(X^\top X)$.

I'm going to pivot to a different example, where the degree of multicollinearity is high. You'll have a chance to play more with pickup trucks on your homework. Consider a survey of employees at a company who've been asked to rate their boss on seven criteria. The seventh criterion is "overall", and will serve as the response, y. Other criteria include $x_{\cdot 2}$, opportunity to learn new things; $x_{\cdot 3}$, does not allow special privileges; and $x_{\cdot 4}$, raises based on performance; etc.

You'll find the data in boss.csv[35] on the book webpage.

[34] https://en.wikipedia.org/wiki/Orthogonality
[35] https://bobby.gramacy.com/hipp0/boss.csv

```
boss <- read.csv("boss.csv")
cor(boss[,3:5])
```

```
##        X2     X3     X4
## X2 1.0000 0.4933 0.6403
## X3 0.4933 1.0000 0.4455
## X4 0.6403 0.4455 1.0000
```

To keep the output matrix small, the code above looks at correlation (which is normalized covariance) for just three inputs. See how $x_{\cdot 4}$ is 49% the same, and $x_{\cdot 5}$ is 64% the same, as $x_{\cdot 3}$. That's serious multicollinearity. Consequently, when you fit an MLR with all those columns, t-tests indicate that only x_1 is useful (on its own) for predicting y.

```
full <- lm(Y ~ ., data=boss)  ## shorthand . for X1 + ... + x6
summary(full)$coefficients
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 10.78708    11.5893  0.9308 0.3616337
## X1           0.61319     0.1610  3.8090 0.0009029
## X2           0.32033     0.1685  1.9009 0.0699253
## X3          -0.07305     0.1357 -0.5382 0.5955939
## X4           0.08173     0.2215  0.3690 0.7154801
## X5           0.03838     0.1470  0.2611 0.7963343
## X6          -0.21706     0.1782 -1.2180 0.2355770
```

Yet, if I drop several of the multicollinear columns, and fit a simpler/more parsimonious model …

```
base <- lm(Y ~ X1 + X2, data=boss)
summary(base)$coefficients
```

```
##             Estimate Std. Error t value  Pr(>|t|)
## (Intercept)   9.8709     7.0612   1.398 1.735e-01
## X1            0.6435     0.1185   5.432 9.574e-06
## X2            0.2112     0.1344   1.571 1.278e-01
```

… I see that both `$X1` and `$X2` are useful for predicting `$Y`. What gives? Multicollinearity makes it hard to see the marginal effect of a particular input, $x_{\cdot j}$. You can't get a look at one of them at a time when they are so similar to, and covary with, one another. Which leads me to the subject of this section. How does one decide which fit is best, `full` or `base`? Do I need predictors `$X3`, …, `$X6`, in addition to `$X1` and `$X2`, or is a simpler model with $d = 2$ inputs sufficient?

How to select a model? Here I'm focusing on selecting one from just two options. Baby steps. Later I'll mention how to scale that up to look at many combinations. As with any testing setup, one must choose a statistic and derive its sampling distribution under a null hypothesis. We need something that summarizes the whole fit of a model.

How about r^2? Recall that r^2 for SLR is the correlation between y and $\hat{y}$ values. It's the same for MLR, only $\hat{y}$ is more complicated. There are two r^2 values, one for each model.

```
r2f <- summary(full)$r.squared   ## same as cor(boss$y, full$fitted)
r2b <- summary(base)$r.squared   ## same as cor(boss$y, base$fitted)
c(full=r2f, base=r2b)
```

```
##   full   base
## 0.7326 0.7080
```

Since these two models are nested, meaning that `full` contains all slope (and intercept) coefficients involved in `base`, its r^2 will be bigger. More coefficients $\beta_3, \ldots, \beta_6$ mean more DoF, a smaller residual sum of squares, and a "better" fit. I say "better" because it just means the points are closer to the line (the hyperplane). It doesn't mean that you'll get better predictions for new data. The more complicated, `full`, model could just be fitting the noise, or overfitting[36].

How to distill those two r^2 values down to one statistic for testing? Why not look at their difference? Since they're nested, putting the larger one first, $r^2_\Delta = r^2_f - r^2_b$, guarantees a value in $[0, 1]$.

```
r2diff <- r2f - r2b
r2diff
```

```
## [1] 0.02459
```

So we get a 2.5% better fit from an additional …

```
pf <- length(coef(full))
pb <- length(coef(base))
pdiff <- pf - pb
c(full=pf, base=pb, diff=pdiff)
```

```
## full base diff
##    7    3    4
```

… $p_\Delta = 4$ parameters. Is that worth it?

I've put the cart before the horse a little, jumping right to the statistic without mentioning what the competing hypotheses are. My preference would be to write hypotheses generically using "complexity" via p_f and p_b and the difference between them p_Δ. In a way, this is what the machine learning community does, and I'll say more about that below. But it doesn't mesh well with a classical statistics setup. Perhaps that makes stats old-fashioned, but actually both mindsets serve us well in this context, and there's comfort (I find) in convention.

Taking inspiration from ANOVA (6.6), consider the following hypotheses.

$$\begin{aligned} \mathcal{H}_0 &: \beta_{p_{\text{base}}+1} = \beta_{p_{\text{base}}+2} = \cdots = \beta_{p_{\text{full}}} = 0 \\ \mathcal{H}_1 &: \text{at least one } \beta_j \neq 0 \text{ for } j > p_{\text{base}} \end{aligned} \tag{13.10}$$

See how that's a formal, but long-winded way of asking the model selection question of

[36] https://en.wikipedia.org/wiki/Overfitting

interest? $\mathcal{H}_0$ assumes that any additional coefficients are zero (the `base` model), whereas $\mathcal{H}_1$ says at least one is useful (`full`). Connection to ANOVA is not just coincidental or inspirational. As with dummy variables in §13.4, MLR subsumes ANOVA in many respects. The advantage to having something like Eq. (13.10) is that it tells you exactly what to simulate via MC.

Before jumping in, it's worth pointing out that individual t-tests for particular coefficients don't apply here. Those are one-at-a-time procedures, whose standard errors and p-values must be interpreted as conditional on other parameters being in the model.

You could, of course, do a sequence of t-tests, singling out one β_j at a time. However, performing multiple smaller tests – asking multiple questions, statistically speaking – is inferior to performing a single big test – just one question – if the latter is really what you're after. Recall the multiple testing hazard from §6.4. After rejecting the null for the big question, if that's what happens, it might be worthwhile to perform other, smaller tests.

Model selection via MC

First, here's a little setup to make it easier when we get to the loop. In particular, I need a data frame (`Dfs` below) that I can use to hold randomly generated Y-values along with original inputs X.

```
X <- as.matrix(boss[,-1])
n <- nrow(X)
Dfs <- boss                                   ## auxiliary df for sims
```

Code below encodes $\mathcal{H}_0$, and extracts useful quantities for simulation.

```
beta <- c(as.numeric(coef(base)), rep(0, 4))  ## coefficients under H0
mu <- beta[1] + X %*% beta[-1]                ## mean/prediction under H0
s2 <- summary(base)$sigma^2                   ## est. uncertainty under H0
```

Ok, ready? Generate data under the null (`base`), then calculate fits for `base` and `full`, and save their difference in R^2s.

```
N <- 100000
R2fs <- R2bs <- R2diffs <- rep(NA, N)
for(i in 1:N) {

  ## simulate data under H0 (via residuals)
  sigma2 <- s2  ## or sigma2 <- (n - pb)*s2/rchisq(1, n - pb)
  Ys <- rnorm(n, mu, sqrt(sigma2))

  ## put simulated Ys along with the data Xs
  Dfs$Y <- Ys

  ## and then fit both models to these simulated data
  Fs <- lm(Y ~ X1 + X2 + X3 + X4 + X5 + X6, data=Dfs)     # H1
  Bs <- lm(Y ~ X1 + X2, data=Dfs)                         # H0
```

```
  ## extract both R2 values
  R2fs[i] <- summary(Fs)$r.squared
  R2bs[i] <- summary(Bs)$r.squared

  ## save their difference
  R2diffs[i] <- R2fs[i] - R2bs[i]
}
```

You may have noticed that I took a little shortcut by not sampling σ^2 values via inverse-χ^2. That's definitely doable if you'd like, but results are pretty much the same because n is large. Figure 13.12 shows the empirical sampling distribution of R^2_Δ values, alongside our observed value from the data.

```
hist(R2diffs, main="", xlab="sampled R2diff values")
abline(v=r2diff, col=2, lwd=2)
legend("right", "observed", col=2, lwd=2, bty="n")
```

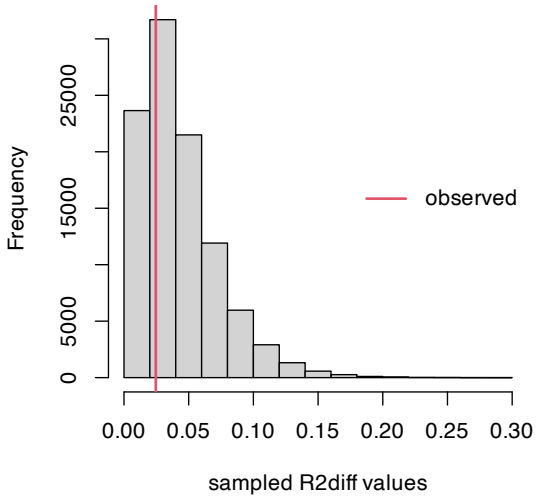

FIGURE 13.12: Sampling distribution of R^2_Δ for the boss data example.

We only care about large observed r^2_Δ values, since small ones support the null. So this is a right-tailed test. Clearly there's not enough evidence to reject $\mathcal{H}_0$, since the observed value is near the mode. It never hurts to calculate a p-value.

```
pval.mcr <- mean(R2diffs > r2diff)
pval.mcr
```

```
## [1] 0.6854
```

In conclusion, the additional flexibility offered by those four extra coefficients is not helpful. That doesn't mean that none of them are useful on their own. It just means that in the presence of `$X1` and `$X2`, you don't need any of `$X3` … `$X4`. Due to collinearity, it may be that one of those would be useful if one of `$X1` or `$X2` were dropped from the model. Hold that thought.

Model selection via math

R^2_Δ is beautifully intuitive. It takes something well-understood from two similar models and pits them against one another. Unfortunately, its sampling distribution is not available in closed form. (Although the MC is simple enough.) Fortunately, some clever mathematicians showed that a similar statistic does have a known distribution.

The story goes like this: R^2 is a squared correlation, which is a ratio of (co-)variances, which are residual sums of squares (SSs), which have a χ^2 distribution, and ratios of χ^2s have an F distribution. Sound familiar? That's the same as the ANOVA F story.

If you don't care about an ANOVA connection, you can go directly to the f statistic through r^2. The formula below connects the two, which is known as a *partial f statistic*.

$$f = \frac{R^2_\Delta / p_\Delta}{(1 - R^2_{\text{full}})/(n - p_{\text{full}})} = \frac{(\text{SSE}_{\text{full}} - \text{SSE}_{\text{base}})/p_\Delta}{\text{SSE}_{\text{full}}/(n - p_{\text{full}})} \tag{13.11}$$

When the `base` model is null or empty, meaning that it only has an intercept term β_0 and no slopes (i.e., $p_{\text{base}=1}$), an ordinary or full f statistic is obtained. When dummy variables exclusively encode groups, as described in §13.4, an ANOVA f statistic (6.11) is recovered.

```
f <- ((r2f - r2b)/pdiff)/((1 - r2f)/(n - pf))
f
```

```
## [1] 0.5287
```

Its random analog follows $F \sim F_{p_\Delta, n - p_{\text{full}}}$. Figure 13.13 provides a visual, along with the observed value of the statistic for comparison.

```
fs <- seq(0, 5, length=1000)
plot(fs, df(fs, pf - pb, n - pf), type="l")
abline(v=f, col=2, lwd=2)
legend("right", "observed", col=2, lwd=2, bty="n")
```

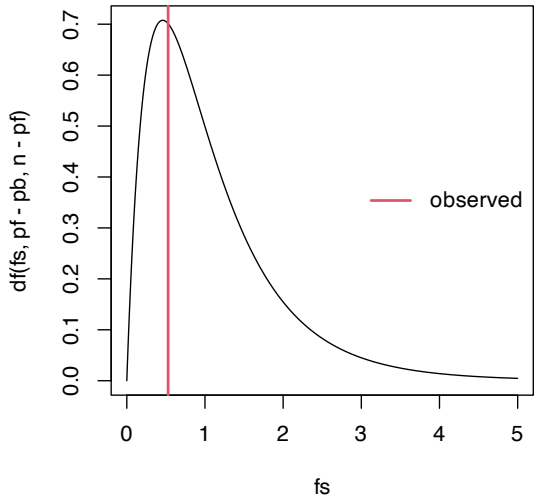

FIGURE 13.13: Sampling distribution of F for the boss data example.

That view is similar to Figure 13.12 excepting labels on the x-axis. However, p-values under the two tests (MC via r^2 and closed form via f) differ slightly …

```
pval.f <- pf(f, pdiff, n - pf, lower.tail=FALSE)
c(MCr=pval.mcr, f=pval.f)
```

```
##    MCr      f
## 0.6854 0.7158
```

… because they're not based on exactly the same statistic, and so they have different power (see A.2). Yet, if you similarly normalize the R^2 values …

```
Fs <- (R2diffs/pdiff)/((1 - R2fs)/(n - pf))
pval.mcf <- mean(Fs > f)
pval.mcf
```

```
## [1] 0.7166
```

… then MC and closed-form p-values agree as $N \to \infty$.

One downside to this analysis, whether in closed form or via MC, is that comparisons must be nested. One model must be a special case of the other with some of its coefficients implicitly being zero (13.10). Things are more complicated when an arbitrary comparison is made, but it is possible.

You can even automate a search for good MLR models, which some might call data mining[37]. There are various ways to do that, but most studies start with what is known as stepwise regression[38]. See `?step` in R. Other approaches include least angle regression[39], which is a family or methods of which the well-known LASSO[40] is a special case. See `lars` for R (Hastie and Efron, 2022).

13.6 Homework exercises

These exercises help gain experience with transformations, dummy variables, interactions and model selection in MLR. These are somewhat more open-ended than others in the book, as this chapter is a bit of a bonus. Most test several skills at once. Some of them are quite involved. Good luck!

Do these problems with your own code, or code from this book. In contrast with other chapters, you're encouraged to use library functions (such as `lm`) as long as they have been discussed in the chapter. Don't just grab whatever off the web or CRAN.

[37] https://en.wikipedia.org/wiki/Data_mining
[38] https://en.wikipedia.org/wiki/Stepwise_regression
[39] https://en.wikipedia.org/wiki/Least-angle_regression
[40] https://en.wikipedia.org/wiki/Lasso_(statistics)

#1: Non-P SLR with log-transform

Revisit our pickup truck $\log y$ analysis with a non-P approach. Specifically, recreate Figure 13.3 using the methods behind Figure 12.8.

#2: Non-P log-log SLR

Revisit our Consolidated Foods log-log analysis with a non-P approach. Specifically, recreate Figure 13.7 using the methods behind Figure 12.8.

#3: Import elasticity, redux

Revisit #12 from §12.5, where a non-P test was performed to determine if import elasticity conformed with economic theory. Provide predictions back on the original scale. You may choose to do this via non-P or P methods.

#4: Telemarketing

Data linked in telemkt.csv[41] on the book webpage records the number of `$calls` made by telemarketers after they've been on the job for a certain amount of time (`$months`).

```
telemkt <- read.csv("telemkt.csv")
```

Fit an appropriate polynomial regression to these data and provide a prediction, complete with error bars, over the range of `$months` in the data. Provide statistical justification for each term in your polynomial, beginning with a linear model and ending with the highest degree you've found to be useful.

#5: Electricity demand

The file elec.csv[42] on the book webpage contains data on the rate, measured in megawatts (`$MW`), of electricity delivered to Gulf Energy[43] customers in Alabama. Also provided are average daily temperature readings (`$temp` in degrees Fahrenheit) for each day corresponding to the megawatt reading.

```
elec <- read.csv("elec.csv")
```

Your task is to build a model to predict electricity usage from temperature. Begin by plotting these data so that you can entertain sensible transformations. Once you've decided on your model, use it to make predictions for a grid of temperatures spanning those provided in the data. Always include an assessment of uncertainty with any prediction.

Hint: compare the following transformation `log(100 - elec$temp)` to other simple ones like (ordinary) log or square-root. Why is this one better?

#6: Asymptotic MLR

Take second derivatives $\partial^2\ell/\partial\beta^2$, and any expectations required for calculating the Fisher information, and thereby verify that the asymptotic distribution of the MLE for $\hat{\beta}$ is exact (13.6).

[41] https://bobby.gramacy.com/hipp0/telemkt.csv
[42] https://bobby.gramacy.com/hipp0/elec.csv
[43] https://gulfenergyinfo.com

#7: Miles with pickups

Incorporate `$miles` into our Craigslist pickup analysis via MLR. Does it help to use a log-transform? Is this new predictor collinear with `$year`? Do you prefer a model fit with `$miles` (or `$lmiles`) and `$year` or just the one with `$year`. How about including `$make`? Back your conclusions up with statistical analysis.

I know we dismissed it back in §13.3, but we never did the partial f-test.

#8: Mileage, redux

Return to mileage.csv[44] first studied in §12.5 and perform parametric MLR analysis ignoring the `$MakeModel` column.

```
mileage <- read.csv("mileage.csv")
mileage <- mileage[,-1]
```

Explore fits for these data. Should any of the variables be transformed? Should any be eliminated? What about interactions? Report on your best fit (with any transformed variables) in two settings: (a) without any interactions, but with all main effects you find useful; (b) with interactions (and main effects) that you feel are helpful.

Provide two `$mpg` predictions, including error bars, for a Honda CRV with the following specifications.

```
crv <- c(vol=75, hp=119, sp=115, wt=30.5)
```

One (a) using main effects only; and another (b) with interactions.

#9: MBA grades with GMAT

Incorporate `$GMAT` into our bachelor's/MBA GPA analysis. Consider it both as a main effect, and in interactions with other variables. Try to find what you think is the best fit, both statistically and what makes sense to you.

#10: Gender, age and income

Data provided in census2000.csv[45] on the book webpage contains records from the USA National Census exercise in 2000. In this question, and the next three, I'll ask you to work on a subset of these data for working-age residents.

```
census <- read.csv("census2000.csv")
workers <- (census$hours > 500) & (census$income > 5000) & (census$age < 60)
census <- census[workers,]
```

In this question, explore (log) wage rate (income per hour), as defined in the code below, as a function of age and gender.

[44] https://bobby.gramacy.com/hipp0/mileage.csv
[45] https://bobby.gramacy.com/hipp0/census2000.csv

```
lWR <- log(census$income/census$hours)
age <- census$age
sex <- census$sex
```

Ignore all other columns, some of which are involved in the next question(s).

Fit an MLR that allows you to predict (log) wage rate as a function of age and gender. You may wish to consider adding polynomial and interaction effects, or make other transformations. However, do not split the data into separate sets! Keep everything together and use appropriate MLR constructs to differentiate your fit as necessary.

You may wish to begin by plotting the data. Be warned: there are a lot of data points ($n = 25403$), and looking at all of them could be a hot mess if you're not careful. Figure 13.14 offers a visual based on averages that you may find inspiring.

```
men <- sex == "M"
malemean <- tapply(lWR[men], age[men], mean)
femalemean <- tapply(lWR[!men], age[!men], mean)
plot(18:59, malemean, type="l", lwd=2, col=4, xlab="age",
     ylab="avg log-wage rate", xlim=c(19,60), ylim=c(1.8,3))
lines(18:59, femalemean, lwd=2, col=6)
text(x = rep(60,2), y = c(malemean[42],femalemean[42]),
     labels=c("M","F"), col=c(4,6))
```

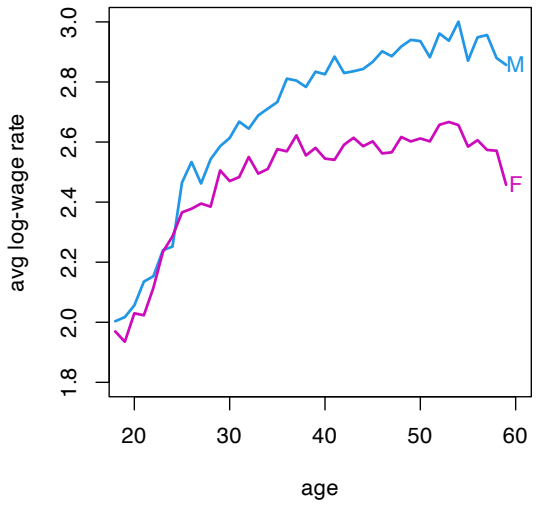

FIGURE 13.14: Average log-wage rate from the 2000 census by age and gender.

Feel free to use that visual as a basis of your own presentation, but note that it does not make use of any model fits/MLR. That's your job. As always, ensure that your predictions contain error bars to quantify uncertainty.

#11: Education and income

Using the census data from #10, explore the relationship between education (`$edu`) and log-wage rate. Use the same `workers` subset, but only use `$edu`, a categorical variable, and

don't use any of the other predictors. That makes this an ANOVA, but I want you to take an MLR approach. Is education a good predictor of pay? Are all of those education categories useful? If not, consider combining any that seem indistinct from the "intercept". Report on what you find. Provide a prediction, with PI, for (un-logged) wage rate for each education category.

#12: Combining #10 and #11

Combine your analysis of (log) wage rate based on age, gender, and education. Try to find the best fitting model using the tools you know. Comment on what the data says affects pay. What affects it most/least? Considering that eventually you'll get older (I hope), but it's hard to change your gender, what can you do in order to meet your income goals?

#13: All census columns

You might think I was just stretching things out so that I'd have thirteen questions in Chapter 13. Perhaps I was, subconsciously.

Now, use all of the columns of the census data. That means additionally incorporate `$marital` and `$race`. Look back at #12 and answer the same questions about what affects pay, and what you can control.

Hint: one thing that makes this question challenging is that not all categories of `$marital` and `$race` are "useful". They're too granular.

```
table(census[,3:4])
```

```
##            race
## marital     Asian Black NativeAmerican Other White
##   Divorced     54   334             27   176  2467
##   Married     531  1004            109   907 12387
##   Separated    55   257             10   186   710
##   Single      218   930             47   552  4154
##   Widow         5    61              3    19   200
```

This is especially so when combined with the eight categories of `$edu`. Explore reducing the number of categories by combining some of them. For example, it may be easy to justify combining "Divorced" and "Separated" categories. Only do that if it also makes sense statistically.

A

Loaded terms

This appendix is organized in three parts: §A.1 fleshes out the idea of an empirical distribution; §A.2 is on power analysis; and §A.3 is on Monte Carlo (MC) error. Although these are important topics, I avoided details earlier so as not to derail our conversation. I think you can appreciate these concepts in passing, and without extensive detail. But if you're curious, here is slightly less quick overview.

A.1 Empirical distribution

I use the term empirical distribution, or ECDF[1], a bunch in the book. Actually, ECDF stands for "empirical cumulative distribution function", but that's a mouthful. I like ECDF as an acronym for a variety of reasons. One is that there's an R function called `ecdf` which helps with calculations and visualization. I'll get more into that soon. The other is that ED is a terrible acronym because it can easily be confused with a certain health issue, so I'll steer clear.

All calculations based on MC that require visualizing or calculating a confidence interval (CI) or p-value are implicitly leveraging an ECDF. Some non-P procedures, even those that avoid MC, make use of an ECDF, though sometimes indirectly. Anything based on ranks is using one, too. If you've gotten to this point in the book – here at the very back – without skipping ahead to this appendix, you probably have a vague idea of what an ECDF is simply by suggestion. I'm ok with you name-dropping ECDF without deep understanding. Fake it 'till you make it! I sometimes use the word manifold[2] without deep understanding. I try to keep it light when I do. I'll be the first admit that I'm not that kind of mathematician.

Loosely, an ECDF is a distribution that is defined by a discrete collection of values or examples, rather than on a particular functional form. Here's a simple example. Suppose you had a sample of data values.

```
y <- rnorm(30, mean=12, sd=3)
```

Those `y`-values came from a Gaussian, an actual (known) distribution with particular values for parameters. Those are not material to our discussion. In fact, I'm going to ask you to forget about the Gaussian, except as a ground truth (an actual distribution) to compare an empirical one to. The ECDF for `y` is the step function that uses sorted `y` values on the x-axis, and a normalized "staircase" of "+1s" on the y-axis. Here's what I mean.

[1] https://en.wikipedia.org/wiki/Empirical_distribution_function
[2] https://en.wikipedia.org/wiki/Manifold

```
y <- sort(y)          ## these on the x-axis
n <- length(y)
ey <- (1:n)/n         ## these on the y-axis
```

Figure A.1 plots that wonky staircase (left panel) and compares it to what's produced by the `ecdf` function in R. They're right on top of one another, so I've tried to adjust the size of the points and line types so that both are still visible. Each stair, as climbed from left to right, always has a vertical height of $1/n$, but the width of its horizontal platform is determined by the gap between adjacent sorted $y^{(i)}$-values, or order statistics (§8.1).

```
par(mfrow=c(1, 2))
sfun <- stepfun(y, c(0, ey))
ygrid <- seq(min(y) - 5, max(y) + 5, length=1000)
plot(sfun, xlab="y", ylab="ecdf", main="", xlim=range(ygrid), lwd=2)
lines(ecdf(y), col=2, lwd=2, lty=2, cex=0.75)
legend("left", c("by hand", "ecdf"), col=1:2, lwd=2, lty=1:2, bty="n")
plot(ygrid, pnorm(ygrid, 12, 3), lwd=2, type="l", xlab="y", ylab="e/cdf")
lines(ecdf(y), col=2, lwd=2, lty=2)
legend("left", c("truth", "ecdf"), col=1:2, lwd=2, lty=1:2, bty="n")
```

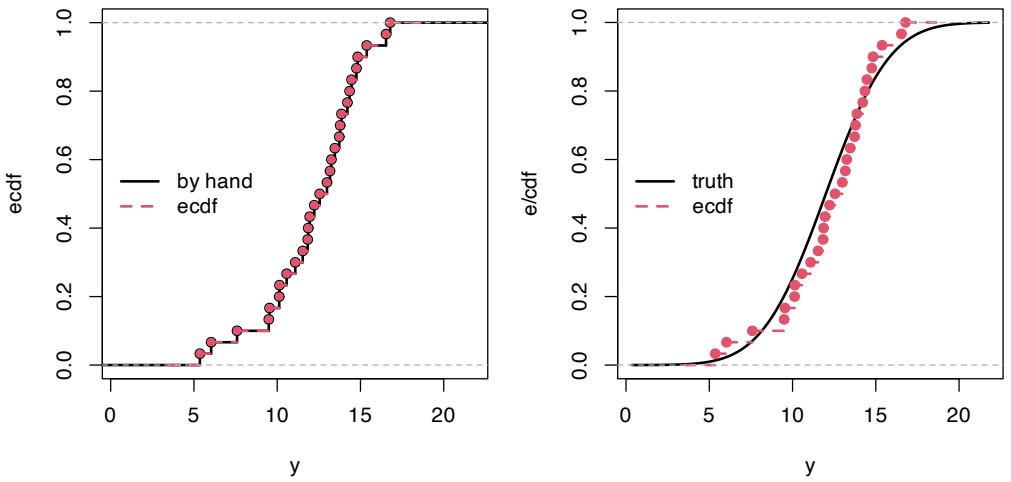

FIGURE A.1: Comparing the by-hand ECDF to R's `ecdf` (left), and to the real CDF (right). The `ecdf` in both panels is the same.

The right panel of the figure offers a comparison to the true Gaussian cumulative distribution function (CDF)[3] used to generate `y`. Here we see that the empirical one is similar to the truth. As you might imagine, the ECDF gets closer to the true CDF as n is increased. I encourage you to try this example with `n <- 1000` or larger. This is why, when we use a really big N in a MC, we get accurate results. Our ECDF approximation to the sampling distribution is then very close to the true CDF of the sampling distribution. I'll come back to that in a moment.

R's `ecdf` actually returns a function that can be evaluated for any y-value, sort-of like

[3] https://en.wikipedia.org/wiki/Cumulative_distribution_function

`pnorm` in R, but not specifically for a Gaussian – for any `y`-values. Working with ranks, like in Chapters 9–12, is like working directly on an ECDF, except skipping the divide-by-n part in the code block above. Notice how the distribution function – any of the ones in Figure A.1 – is a mapping between the domain of `y`, i.e., on the real number line/x-axis, to the interval $(0, 1)$ on the y-axis. Observations that are clumpy on the x-axis (those are the `y` values) are uniform on the y-axis.

Inverting that logic … an inverse CDF can take something random uniform, like ranks from random permutations (via `sample`), and impart on it a non-uniform distribution. This is what quantile functions, like `qnorm` or `qbinom` (any `q*`) do. The empirical analog is simply `quantile` in R.

Most people find it hard to intuit (cumulative) distribution functions. I'm one of those people. Density or mass functions are easier. Whereas CDFs look like s-curves, or sigmoids[4], non-decreasing from left to right, densities go up and come back down, like a bell curve or similar. Technically, a probability density (or mass) function (PDF)[5] is the derivative of a CDF. Or, vice versa, a CDF is the indefinite integral of a PDF from the left (usually $-\infty$). For samples of continuous random variables, like our `y` vector, defining an ECDF, it's hard to take a derivative without making additional assumptions about smoothness, in a calculus sense. Recall that derivatives are defined as limits.

Histograms have been our go-to for visualizing that derivative empirically. You might say they offer an "empirical (probability) density function (EPDF)", although that isn't really a thing. (Don't go around talking about EPDFs. Anyone who knows this stuff will think you're making things up. Even though it probably should be a thing.)

The closest thing to an EPDF is kernel density estimation (KDE)[6]. KDE is like a smooth version of a histogram. Whereas the ECDF is completely nonparametric (non-P; see §8.1), histograms and KDEs are a bit of a hybrid. They are still considered non-P, but since histograms have a bin size and KDE involves a bandwidth parameter (and a choice of kernel), there's still a quantity that needs to be "dialed in". Somewhat confusingly, the command to fit a KDE in R is called `density`.

```
kde <- density(y)
```

Figure A.2 offers a visual comparison between histogram and KDE (left panel), and between KDE and truth (right). In both cases, the "parameter" is estimated using an automatic procedure behind the scenes, embedded in `hist` and `density` commands, respectively.

```
par(mfrow=c(1, 2))
hist(y, freq=FALSE, xlim=range(ygrid), ylim=range(kde$y), main="")
lines(kde, col=2, lwd=2, lty=2)
legend("topleft", "kde", col=2, lty=2, lwd=2, bty="n")
plot(ygrid, dnorm(ygrid, 12, 3), ylim=range(kde$y),
  lwd=2, type="l", xlab="y", ylab="pdf")
lines(kde, col=2, lwd=2, lty=2)
legend("topleft", c("truth", "kde"), col=1:2, lwd=2,
  lty=1:2, bty="n")
```

[4] https://en.wikipedia.org/wiki/Sigmoid_function
[5] https://en.wikipedia.org/wiki/Probability_density_function
[6] https://en.wikipedia.org/wiki/Kernel_density_estimation

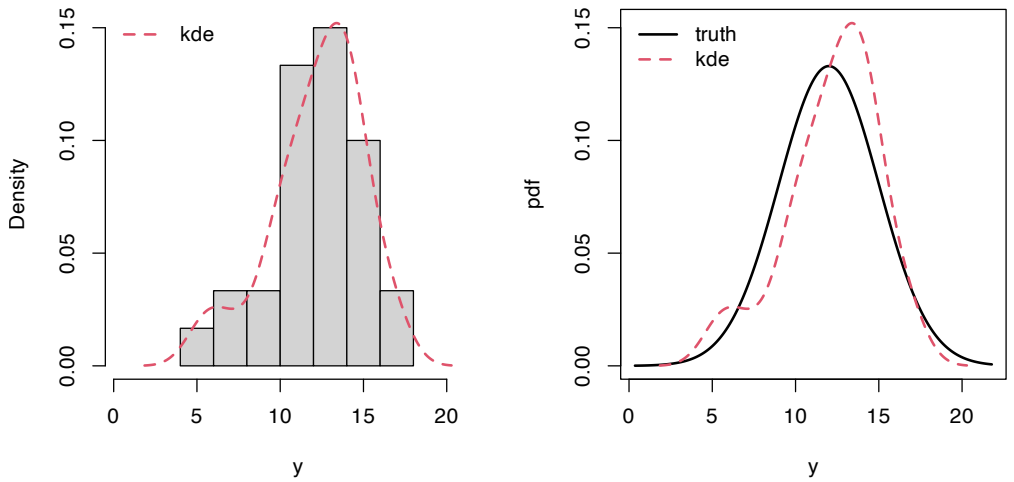

FIGURE A.2: Comparing histogram to kernel density (left) to the true Gaussian (right).

So, in conclusion, a sample may represent a distribution via an ECDF. The bigger that sample, the more accurate the representation. This is key to a MC calculation, where sample size N controls accuracy. Whereas an ECDF may be a crude approximation in the context of a small sample of data points of size n, a sampling distribution based on big-N MC samples can be highly accurate. It can be lots better than an asymptotic approximation, for example based on the central limit theorem (CLT; Chapter 4). Size N gives you a dial on that accuracy, whereas n is fixed. MC p-value calculations essentially replace integrals from unknown distributions (or ones that are out of reach without hard math) with sums (or averages). In so doing, one is implicitly using an ECDF in that calculation. Quantiles of an MC sample can be used to calculate CIs.

A.2 Power analysis

I'll be the first to admit this isn't my area of expertise, which in part explains why power doesn't play a more prominent role in the book. It's also difficult stuff. My intention here is to provide some background for curious, engaged readers. Most of this material is derivative of Wasserman (2004), based on notes for a short course I once gave to a hedge fund in Chicago.

There are two types of errors we can make when basing decisions on the outcome of hypothesis tests like these. A *type I error* occurs when the null hypothesis ($\mathcal{H}_0$) is true, but it is rejected. This came up in Chapter 1, and when discussing the multiple testing hazard in §6.4. Type I error is sometimes called a "false positive", indicating that a given condition is present (e.g., $\theta \neq \theta_0$) when it is actually not present.

The type I error rate, or *significance level*, is the probability of rejecting a null hypothesis, given that it is true. This is the α we choose when comparing to a p-value or constructing a CI.

A type II error[7] occurs when the null hypothesis is false but erroneously fails to be rejected. This is sometimes called a "false negative", which happens when we fail to believe a truth, usually due to lack of enough evidence. Type II error is denoted by β and is related to the power of a test, denoted by $1 - \beta$.

We often think of them separately, Type I/II errors, α and β: choosing one, usually $\alpha = 0.05$, and acknowledging β without appreciating just how intertwined they are. Delving a little into their interplay requires a few definitions.

Let Y generically represent random observations from a data generating process, e.g., $Y_1, \ldots, Y_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$. Let $S \equiv S(Y)$ be a statistic, e.g., $S = \bar{Y}$, and let c be a *critical value* used to determine when a statistic leads to rejecting $\mathcal{H}_0$, e.g., reject $\mathcal{H}_0$ when $S > c$. The *rejection region* R is the set of y-values, i.e., data sets, that would lead to rejecting $\mathcal{H}_0$.

$$R = \{y : S(y) > c\}$$

The *power function* of a test with rejection region R is defined by $\beta(\theta) = \mathbb{P}_\theta(Y \in R)$, whereas its size is $\alpha = \sup_{\theta \in \mathcal{H}_0} \beta(\theta)$. Technically, a test is said to have level α if its size is less than or equal to α, though most don't think of level as an inequality.

Suppose, with our Gaussian data (known σ^2) we want to test $\mathcal{H}_0 : \mu \leq 0$ versus $\mathcal{H}_1 : \mu > 0$ using $S = \bar{Y}$ and reject $\mathcal{H}_0$ when $S > c$. The power function is

$$\begin{aligned}\beta(\mu) &= \mathbb{P}_\mu(\bar{Y} > c) = \mathbb{P}_\mu\left(\frac{\sqrt{n}(\bar{Y} - \mu)}{\sigma} > \frac{\sqrt{n}(c - \mu)}{\sigma}\right) \\ &= \mathbb{P}\left(Z > \frac{\sqrt{n}(c - \mu)}{\sigma}\right) = 1 - \Phi\left(\frac{\sqrt{n}(c - \mu)}{\sigma}\right).\end{aligned}$$

Take $\sigma = 1$ and $n = 25$. Figure A.3, left, traces out power as $\beta(\mu, c)$. Observe that power is increasing in μ.

```
mu <- c <- seq(-1, 1, length=100)
g <- expand.grid(c, mu)
beta <- function(x) { 1 - pnorm(5*(x[,2] - x[,1])) }
par(mfrow=c(1, 2))
image(mu, c, matrix(beta(g), ncol=100), col=heat.colors(128))
plot(mu, beta(cbind(mu, 0)), type="l", lwd=2, ylab="beta(mu)")
abline(v=0, col="gray", lty=2)
text(c(-0.5, 0.5), c(0.5, 0.5), c("H0", "H1"))
```

Fixing $c = 0$, say, the right panel of the figure shows the resulting slice through the 2D image. Then, α is the spot on the graph where power crosses the y-axis, since $\mathcal{H}_0 : \mu \leq 0$.

$$\alpha = \max \mu \leq 0 \beta(\mu) = \beta(0) = 1 - \Phi\left(\frac{\sqrt{n}c}{\sigma}\right)$$

Turning things around, $c = \frac{\sigma \Phi^{-1}(1-\alpha)}{\sqrt{n}}$. We reject when $\bar{y} > \sigma \Phi^{-1}(1-\alpha)/\sqrt{n}$, or equivalently when $\bar{y}/\sigma/\sqrt{n} > q_{1-\alpha}$.

It would be desirable to find the test with highest power under $\mathcal{H}_1$, among all such size α

[7] https://en.wikipedia.org/wiki/Type_I_and_type_II_errors#Type_II_error

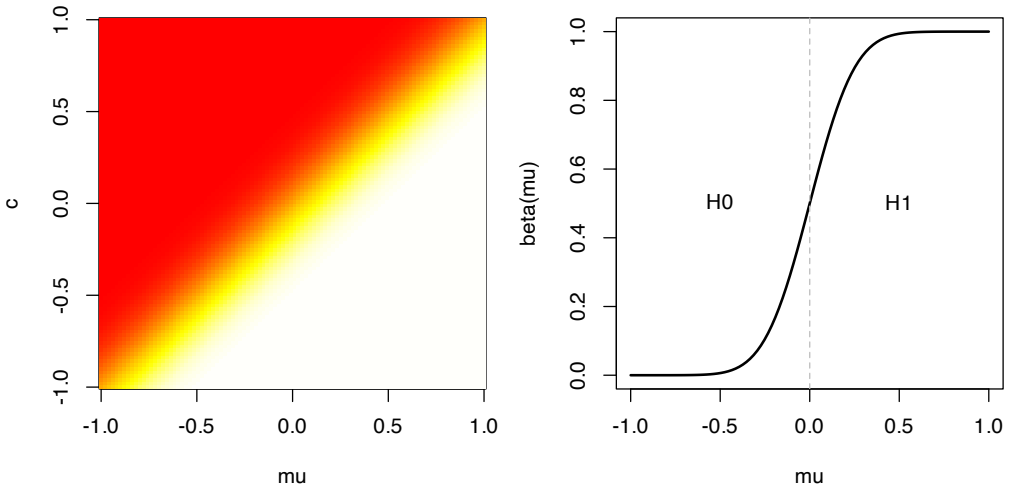

FIGURE A.3: Power for a Gaussian example (left) and a slice at $c = 0$ (right).

tests. Such a test, if it exists, is called uniformly most powerful[8]. Finding most powerful tests is hard and, in many cases, they don't exist. For that reason, focus is typically on generic recipes, like tests based on averages, a Wald test, or tests based on maximum likelihood.

Whenever you have two tests of the same, or similar hypotheses, they can be compared based on power and size. In practice this is difficult. Ironically, many such analyses involve MC.

A.3 Monte Carlo error

You might think this discussion would be more prominent in a book that evangelizes MC. I'm burying it in an appendix. This is an important topic, but also it isn't. If you're worried that you don't have an accurate MC estimate of something, like a p-value or a CI, just make N bigger.

The N values I used in simulations has tended to drift higher, as models and inferential goals became more complex, as chapters progressed. $N = 1{,}000$ was enough until Chapter 5. $N = 10{,}000$ sufficed for a while, and then by Chapter 9 we touched $N = 1$ million (M). I even suggested trying 10 M at one point. Then things cooled off a little, and I was back down to 100,000 by Chapter 13.

How did I decide what to use? I'd run the MC a couple of times and take note of what I got, and how much it changed from one run to the next. If things seemed to vary more than I'd like, I'd make N bigger. If I thought I could make N smaller, I would. In early chapters I didn't want to frighten you off, so I tried to make N as small as possible. It's not an exact science, but it's not rocket science either. (If you're using MC for rocket science, or brain surgery, make N really big please!)

The idea of "running the MC a couple of times" can be formalized into a robust assessment

[8] https://en.wikipedia.org/wiki/Uniformly_most_powerful_test

of uncertainty. Take a two-sample Bernoulli test of §5.1, for example. The function `do.MC` below serves as a wrapper around code for that experiment, set up in a way that allows me to entertain calculations for various N, returning a p-value estimate and timing details.

```
do.MC <- function(N=100)
 {
  ## for timing
  tic <- proc.time()[3]

  ## pasted data from Chapter 5
  sy <- 18
  ny <- 30
  sx <- 19
  nx <- 53

  ## pasted estimates from Chapter 5
  that.d <- sy/ny - sx/nx
  that.pool <- (sy + sx)/(ny + nx)

  ## pasted MC loop from Chapter 5
  That.ds <- rep(NA, N)
  for(i in 1:N) {
    Sy <- sum(rbinom(ny, 1, that.pool))
    Sx <- sum(rbinom(nx, 1, that.pool))
    That.ds[i] <- Sy/ny - Sx/nx
  }

  ## pasted p-value calculation from Chapter 5
  pval.mc <- 2*mean(That.ds >= that.d)

  ## finish timing and return
  toc <- proc.time()[3]
  return(list(est=pval.mc, time=toc - tic))
}
```

Now, the code block below calls `do.MC` for a range of N values in a grid, `Ngrid`. Although I generally prefer powers of ten for N, this code uses powers of two to stretch things out a little for dramatic effect. Fair warning: the first few N-values on the grid are fast, but the last few are very slow. I do thirty reps of each so that I may assess MC variability in the output, which is the whole point of this study.

```
Ngrid <- 2^(7:20)
reps <- 30
time <- est <- matrix(NA, length(Ngrid), reps)
for(i in 1:length(Ngrid)) {
  for(r in 1:reps) {
    out <- do.MC(Ngrid[i])
    est[i,r] <- out$est
    time[i,r] <- out$time
```

```
  }
}
```

Figure A.4 plots the standard deviation of those p-values, assessed over the thirty reps (left) and average compute times (right). Both panels use a $\log_{10}$ axis, so things aren't stretched or compressed too much horizontally. The right panel is in log-log so that an exponential increase in runtime, owing to exponentially increasing N across `Ngrid`, appears nearly linear.

```
sd.est <- apply(est, 1, sd)
atime <- rowMeans(time)
par(mfrow=c(1, 2))
plot(log10(Ngrid), sd.est, type="b", lwd=2,
  xlab="log_10 N", ylab="sd(pval)")
plot(log10(Ngrid), log10(atime), type="b", lwd=2,
  xlab="log_10 N", ylab="log_10 time (seconds)")
```

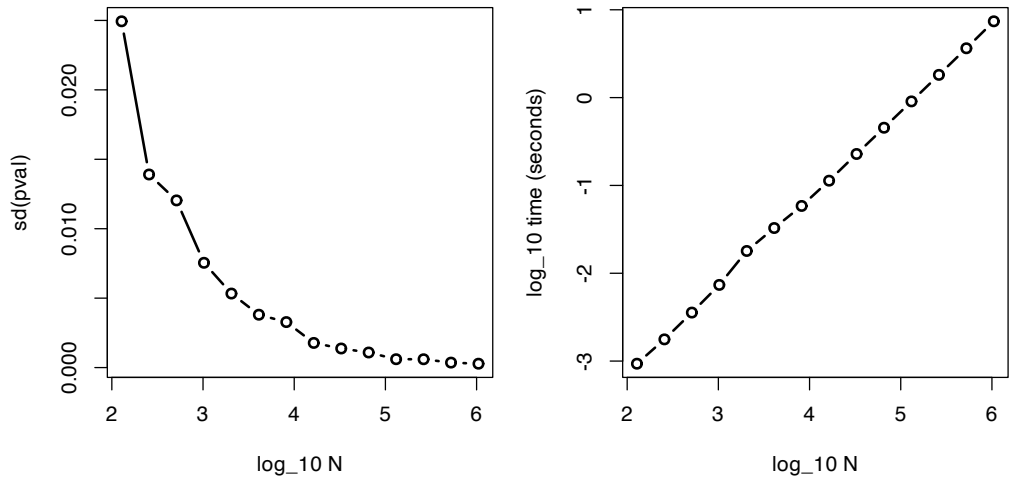

FIGURE A.4: Exploring standard deviation of p-value estimates (left) and execution time (right) for an increasing number of MC "trials" N. Both plots use a $\log_{10}$ x-axis, and the right panel uses $\log_{10}$ on both axes.

The right panel shows that computing time scales commensurately with computational effort, N. That one is boring, so I dispensed with it first. The left panel shows diminishing returns on accuracy as N is increased. In this example, after $N = 10^5$ or so, p-values are accurate to within about 0.001. More precisely …

```
c(Ngrid[11], 2*sd.est[11])
```

```
## [1] 1.311e+05 1.206e-03
```

In other words, when $N = 1.3107 \times 10^5$ (since powers of two), we get a standard deviation of 6.0289×10^{-4}. Therefore, MC error is assessed as plus-or-minus two times that value with 95% probability, using a CLT argument. I chose thirty reps to have a big enough n, which is actually `reps`, to use the CLT.

What just happened? I performed an experiment on a computer code – a computer simulation experiment[9]. You could even fit a regression, possibly after transformation, to interpolate or smooth those points and assess predictive uncertainty. That's known as *surrogate modeling*, or *computer model emulation*. I know someone who wrote a whole book[10] on that. I'll try to keep it simple here. After all, this is a pretty basic computer experiment.

Getting that estimate required a decent amount of work. The total time of the entire double-`for` loop simulation was …

```
round(sum(time)/60)    ## convert to minutes
```

```
## [1] 7
```

And that was just for a Chapter 5 example. I don't recommend going to such great lengths generally, even though 7 minutes isn't that long in the landscape of computer simulation experiments, generally speaking. Here's something I might suggest instead.

Fixate on a particular N value, like $N = 10{,}000$, as a good starting point. Run your MC with that N a number of times. Ten repetitions are probably sufficient, but I'll stick with thirty here for an accurate CLT approximation.

```
N <- 10000
est <- rep(NA, reps)
for(r in 1:reps) {
  out <- do.MC(N)
  est[r] <- out$est
}
```

Alright, now I have thirty p-value estimates, and so with 95% probability the p-value lies in the following range.

```
quantile(est, c(0.025, 0.975))
```

```
##    2.5%   97.5%
## 0.03093 0.04171
```

If that p-value interval is too wide, just increase N and try again. Or, if that's too much work, salvage the $30 \times N$ that you've already done! Check this out. We have thirty sample p-values already, and the CLT says a good estimate of the population p-value is the sample average.

```
pval <- mean(est)
pval
```

```
## [1] 0.03612
```

That estimate is based on thirty times more simulations than any individual estimate is. Since all MC simulations are iid, the effective N is like 3×10^5. However, without even more

[9] https://en.wikipedia.org/wiki/Computer_simulation
[10] https://bobby.gramacy.com/surrogates/

simulations like those in Figure A.4, we may not know how accurate that aggregate p-value is.

Or do we? The CLT also tells us something about the accuracy of sample average estimates. Let $\bar{P}_r$ be an average-based p-value estimator based on r repetitions.

$$\mathbb{V}\text{ar}\{\bar{P}_r\} = \frac{\sigma^2}{r} \quad \text{as} \quad r \to \infty.$$

I'm really just regurgitating Eq. (4.1) with different letters, using $\bar{Y}_n \equiv \bar{P}_n$ and $n \equiv r$. An estimate for σ^2 is easy …

```
s2 <- var(est)
se <- sqrt(s2/reps)
se
```

```
## [1] 0.0005621
```

Using that standard error, a CLT approximation allows our p-value interval to be updated via Student-t.

```
q <- qt(c(0.975), reps - 1)
pval + c(-1, 1)*q*se
```

```
## [1] 0.03497 0.03727
```

That interval is for all $N' = N \times r = 3 \times 10^5$ simulations.

But wait, that's not all! If `s2` is trustworthy as an estimator of σ^2, then we may further *deduce* what the interval would be for *any* value of N'. For example, consider $N' = 1\text{M}$.

```
Nprime <- reps*N
reps.1M <- reps*10^6/Nprime       ## effective number of reps for 1M
reps.1M
```

```
## [1] 100
```

```
se.1M <- sqrt(s2/reps.1M)
pval + c(-1, 1)*q*se.1M
```

```
## [1] 0.03549 0.03675
```

Any changes to the p-value estimate from a 3.3 times larger MC simulation would only affect the third decimal place, and by at most two "points" for that digit.

The actual estimate of the p-value remains unchanged without doing more MC simulations. By deducing the new interval, we may make an informed decision about whether or not more simulations would be worth the time and computing energy. If that improvement is worth the extra effort, computationally speaking, then go for it! I'll pass.

So why did I leave this to the appendix? Because, by the time you get to the end of this book, you already know how to do it! It just may not have occurred to you yet. Estimating

MC error is a matter of turning a computer simulation into an experiment, and performing statistical inference. A virtuous cycle!

B

Coded subroutines

This appendix is composed of several small vignettes for code that is not explained in detail elsewhere in the book, but may be downloaded from the book webpage. Each does something that I – or I think you will – find helpful in the course of explaining material or working on homework problems. But, I felt it would be a distraction to cover these in much detail in the main body of a chapter.

B.1 Discrete p-values

In §1.4 I used a subroutine for p-value calculation and visualization with a discrete distribution, specifically the binomial. I thought you might find it handy in your `monty.*` "library" functions for your homework when working with discrete distributions. It was designed to be somewhat generic, not only for the binomial. One challenge in any p-value calculation is knowing what tail of the sampling distribution your observed test statistic is in. This determines the reflection for visualization, and sum-based (area under the curve) calculations tallying tail probabilities. This is what is automated by `pval.discrete` as provided by pval_discrete.R[1] on the book webpage.

If you look in that file, you'll find the code below, which is pasted here as a backup, in case you have a printed copy of the book without access to the book webpage. The file has an additional comment block at the top which explains how to use the function, and what its arguments are. I'll re-verbalize those here. The first, `s` argument is the observed statistic, a scalar. The second, `x` argument represents the domain of the discrete mass, and `xd` is its mass (like from `xd <- dbinom(x, ...)`). Finally, the flag `vis` lets the user specify if an optional visual should be provided.

```
pval.discrete <- function(s, x, xd, vis=FALSE)
  {
    ## some checks
    lx <- length(x)
    if(lx != length(xd)) stop("x, xd length mismatch")
    if(any(xd < 0)) stop("density must be positive")
    if(any(diff(x) <= 0)) stop("x must be increasing")

    ## normalize
    eps <- 2/sum(xd)
    xd <- xd/sum(xd)
```

[1] https://bobby.gramacy.com/hipp0/pval_discrete.R

```
  ## calculate mode using "lower reflected density" rule
  mode <- which.max(xd)
  if(s <= x[mode]) {               ## stat is in left tail
    il <- x <= s                   ## indices of left tail
    ps <- sum(xd[il])              ## left tail pval
    d <- max(xd[il]) + eps         ## max dens for right tail
    ri <- min(which(x >= mode & xd <= d))
    r <- x[ri]                     ## reflected stat value
    pr <- sum(xd[ri:lx])           ## right tail pval
  } else {                         ## stat is in the right tail
    ir <- x >= s                   ## indices of right tail
    ps <- sum(xd[ir])              ## pval in right tail
    d <- max(xd[ir]) + eps         ## max dens for left tail
    ri <- max(which(x <= mode & xd <= d))
    r <- x[ri]                     ## reflected stat value
    pr <- sum(xd[1:ri])            ## left tail pval
  }

  ## pval from both tails
  p <- ps + pr

  ## possibly plot
  if(vis) {

    ## plot empirical sampling density
    sfun <- stepfun(x[-1], xd)
    ran <- range(c(s, r, x[xd > 0]))
    plot(sfun, main="", xlim=ran, xlab="test stat",
      ylab="sampling distribution (mass)", lwd=2)

    ## show test stat and reflection
    abline(v=c(s, r), lty=1:2, col=2, lwd=2)

    ## put legend
    legend("bottom", c("s", "reflect"), col=2, lty=1:2, lwd=2, bty="n")
  }

  ## return p-value
  return(p)
}
```

The function output is a simple p-value estimate.

One thing to notice about this subroutine, and others provided in this appendix, is the liberal use of checks and explanations in comments. These are easy to overlook in a first coding pass. But anyone with experience can tell you that they're saviors in the long run. It's hard to get in a coding headspace, and remember everything you were thinking when you first wrote some code. Reusing code involves preventing misuse (those sanity checks at the top of the function), and comments that explain each code block (sometimes each line). The most important user of your code is your future self. Be kind to your future self.

B.2 Random ties in ranks

In §9.2 we entertained the possibility of tied ranks in MC simulations, say for Wilcoxon rank sum tests (§9.2) and signed ranks tests (§9.3). Similar situations came up again in Chapter 12. These are not easy to deal with in subset sum problem (SSP)-based closed-form calculations, but are rather easier stochastically. The function `rank.ties` in rank_ties.R[2] on the book webpage allows you to randomly generate rank-averaged ties in a permutation of integers.

The first argument is `ranks`, which is either `1:n` for an n-sized sample, or the output of `out$ranks` in a daisy chain. Outputs are discussed in more detail below. The second and third arguments specify the nature of ties requested, via how many (`ntie`) and at what `degree`. The final argument `uniq` specifies which of `ranks` remain as unaveraged (untied) and are available for forming a tie of the desired degree.

```
rank.ties <- function(ranks, ntie, degree, uniq=ranks)
 {
  ## check for degree > 2
  if(degree < 2) stop("must have degree > 2")

  ## check for enough left to make ties, avoiding infinite loop below
  if(length(uniq) < degree*ntie + 1)
    stop("too few uniq to create ties")

  ## check increasing ranks
  n <- length(ranks)
  if(any(ranks[-1] < ranks[-n]))
    stop("ranks must be increasing")

  ## check uniq is subset of ranks
  if(any(!(uniq %in% ranks)))
    stop("uniq should be subset of ranks")

  ## inits
  degree <- degree - 1    ## when counting ties, the first one is free
  fail <- 0               ## avoid infinite loop

  ## find each tie
  while(ntie > 0) {

    ## pick one remaining rank at random
    u <- sample(uniq, 1)
    useq <- u:(u + degree)

    ## check that its next tie neighbors are there too
    if(all(useq %in% uniq)) {
```

[2] https://bobby.gramacy.com/hipp0/rank_ties.R

```
      ranks[useq] <- mean(ranks[useq])    ## replace ranks with average
      uniq <- setdiff(uniq, useq)         ## remove from the uniq list
      ntie <- ntie - 1                    ## one less tie to find
      fail <- 0

    } else {  ## if not, try again

      fail <- fail + 1                    ## infinite loop safeguard
      if(fail > 10*length(ranks))
        stop("unable to find ties")
    }
  }

  ## return all ranks and which ones are not tied
  return(list(ranks=ranks, uniq=uniq))
}
```

The output is a `list` object with `ranks` modified to include any desired ties, at random, and the remaining list of untied (`out$uniq`) ranks. These can be fed in as inputs to another call, in a daisy chain, in order to have (multiple) ties of multiple degrees. There are many checks in place in this subroutine to ensure that the ties requested are reasonable given the other arguments. They are not fool-proof, but cover many situations I encountered during development. Although ties of varying degree can be requested in any order, I recommend starting with higher-degree ties first since those are harder to find. Doing low-degree ties first could make it impossible to find runs of unaveraged ranks of higher degree.

The location of the ties is chosen at random, but `out$ranks` are still in the order provided by the input `ranks` argument. When used in a MC based on ranks, one would still need to randomly permute them, as in a loop which had no ties.

B.3 Subset sum distribution

The Wilcoxon rank sum (§9.2), signed ranks (§9.3), squared ranks (§11.1) and Kruskal-Wallis (KW; §11.2) tests all work with the SSP as a subroutine. Here, I provide details for rank sum and squared ranks tests, leaving signed ranks and KW-testing details to you. First, below is some generic discussion on SSP solving.

SSP Solving

I know of two packages on CRAN that can help with SSP solving. One is `FLSSS` (Liu, 2025), which is what I used for a long time. While I was busy writing this book, it seemed that it had been abandoned by its maintainer. It was still on the *archive* part of CRAN (the A in the acronym), but it could no longer be installed with `install.packages`. Although you could still install the package from source, I scrambled to find an alternative because I didn't want to burden you with that. I found `RccpAlgos` (Wood, 2025), which provides a similar set of solvers. Since then, `FLSSS` came back on the main part of CRAN, with many improvements. I should have been more patient.

The functions in ranks.R[3] on the book webpage provide an implementation using SSP solvers from both packages, although the ones from `FLSSS` are commented out in some cases. At the time of writing, I had observed that `RcppAlgos` was faster than I remembered `FLSSS` being, because the former used compiled subroutines via `Rcpp` (Eddelbuettel and Balamuta, 2018). But then, after `FLSSS` came back from hiatus, I saw that it had undergone a major upgrade to use a similar kit, including `RcppParallel` (Allaire et al., 2025). It's an arms race! For specific statistical testing problems, like via `wilcox.test` built into R, there are even faster solvers, e.g., via `pwilcox`. These do not deploy general-purpose SSP subroutines, so they are less easily extendable to other, similar situations (like squared ranks).

Below, I'll take you through each subroutine in ranks.R in turn, with brief commentary. These begin with rank sum-specific routines, and then pivot to squared ranks.

Wilcoxon rank sum

First is the density function. Technically this is a mass function, since the distribution over ranks is discrete, but in R all mass and density functions start as `d*`. The `k` argument is the observed value of the statistic, like $\bar{r}$ for rank sum or $\bar{r}^+$ for signed ranks. Arguments `nx` and `ny` specify the size of the two samples. In this book I have typically used/labeled those the other way around, first `ny` and then `nx`. But I wrote these before making that choice and decided if it ain't broke don't fix it. You may always call the function with them specified in the other order, first providing `ny` and then `nx` via `dranks(k, ny, nx)`. It will work fine. Make sure you're consistent. If your $\bar{r}$ is from the sum for the y group, as in §9.2, then be sure to provide `ny` first then `nx`.

```
dranks <- function(k, nx, ny)
  {
    N <- nx + ny
    ## numer <- length(FLSSS(nx, 1:N, k, 0.1, solutionNeed=sum(1:N)))
    numer <- partitionsCount(1:N, nx, target=k)
    denom <- choose(N, nx)
    return(as.numeric(numer/denom))
  }
```

Here are a few other details for `dranks`. Notice that the combined sample size, `ny + nx` is called `N` not `n`. I wrote this before considering MC solutions, involving N. The choice for the symbol, `N` , doesn't affect anything since it's a variable in the namespace[4] local to the `dranks` function scope. The `partitionCount` function is what solves the SSP. Read that as finding a subset of size `nx` using integers `1:N` targeting a sum of `k`. That function counts how many ways that can be done, and then this number is normalized by $\binom{N}{n_x}$ to obtain the mass. Alternatively, `FLSSS` may be used by swapping comments. This one has a clunkier syntax.

Next, `pranks` evaluates the CDF, and is implemented simply as a sum of `dranks` from the left. There's not that much else to it.

```
pranks <- function(k, nx, ny)
  {
```

[3] https://bobby.gramacy.com/hipp0/ranks.R
[4] https://en.wikipedia.org/wiki/Namespace

```
  ## nothing to do?
  if(k == 0) return(0)

  ## sum up all density evaluations from 1:k
  p <- 0
  for(i in 1:k) p <- p + dranks(i, nx, ny)

  ## done
  return(p)
}
```

Finally, the code below provides a version of `wilcox.test` that uses `pranks` rather than `pwilcox`. It doesn't cover all cases. I really just wrote it to check that I was getting the same results as `wilcox.test`. In particular, it doesn't allow you to do a signed ranks test, though it could be modified to do so. (See homework exercises in §9.5.) I decided to name it after Mann and Whitney to share the love, since they came up with a similar method around the time of Wilcoxon, but they generally don't get as much name recognition.

```
mannwhitney.test <- function(x, y,
  alternative=c("two-tailed", "less", "greater"))
  {
    ## check alternative argument
    alternative <- match.arg(alternative)

    ## extract lengths
    nx <- length(x)
    ny <- length(y)
    N <- nx + ny

    ## get ranks
    r <- rank(c(x, y))
    rx <- r[seq_along(x)]

    ## check for ties and possibly warn
    if(length(r) - length(unique(r)) > 0.1*N)
      warning("more than 10% ties, suggest using normal approximation")

    ## calculate test statistic
    t <- sum(rx)

    ## go through alternatives for p-value
    if(alternative == "less") phi <- pranks(t, nx, ny)
    else if(alternative == "greater") phi <- 1 - pranks(t - 1, nx, ny)
    else { ## two-sided
      phi <- 2*min(pranks(t, nx, ny), 1 - pranks(t - 1, nx, ny))
    }

    ## return results
```

```
    return(list(stat=t, p.value=phi, alternative=alternative))
  }
```

As above, it doesn't matter if you flip `x` and `y` arguments.

Squared ranks

This test is due to Conover (1999), and my implementation is based on the verbal description provided therein. First is the density function. It helps to externally normalize that density when called repeatedly, for example when plotting like in Figure 11.1. That's what's facilitated by `dsqranks.denom` below.

```
dsqranks.denom <- function(nx, ny)
  {
    N <- nx + ny
    r2 <- (1:N)^2
    sr2 <- sum(r2)

    denom <- 0
    for(i in 1:sr2) {  ## expensive, could be faster if approximate

      ## solve subset sum problem
      ## denom <- denom + length(FLSSS(nx, r2, i, 0.1, solutionNeed=sr2))
      denom <- denom + nrow(partitionsGeneral(r2, nx, target=i))

    }

    return(denom)
  }
```

Notice how it loops over every possible squared rank from 1 to the sum of `ny + nx` squared. A calculation based on `FLSSS` is commented out, as with earlier functions for Wilcoxon tests. The active code is based on a routine from `RcppAlgos`, but not the same routine as for Wilcoxon tests. For some reason, `partitionsCount` does not work in this context. It spits out a warning which is a little cryptic, but I deduced that it would rather you use `partitionsGeneral` to enumerate all solutions. Actual solutions aren't required by this application, only their count.

The `dsqranks` function, calculating the mass, can take a pre-calculated normalization, or it can calculate its own normalization (default). Notice how it's a special case of the `dsqranks.denom` function. Normalization is where most of the work is involved, computationally speaking.

```
dsqranks <- function(k, nx, ny, denom=NULL)
  {
    N <- nx + ny
    r2 <- (1:N)^2

    ## solve subset sum problem
    ## numer <- length(FLSSS(nx, r2, k, 0.1, solutionNeed=sum(r2)))
```

```
    numer <- nrow(partitionsGeneral(r2, nx, target=k))

    ## offload to another function in case pre-calculated
    if(is.null(denom)) denom <- dsqranks.denom(nx, ny)

    return(as.numeric(numer/denom))
  }
```

Calculating mass isn't useful directly for testing, but can help with visualization. Testing requires summing over tail probabilities to calculate p-values, as provided by `psqranks`. This is the same as evaluating the mass over all tail events. Although that could involve calls to `dsqranks`, I found it more expedient to extract out the important aspects of `dsqranks` and its normalization.

```
psqranks <- function(k, nx, ny)
  {
    ## nothing to do?
    if(k == 0) return(0)

    ## avoiding duplicated denom computation in multiple dsqranks calls
    N <- nx + ny
    r2 <- (1:N)^2
    sr2 <- sum(r2)

    ## calculate the common denominator to all density calculations
    denom <- dsqranks.denom(nx, ny)

    ## sum up all density evaluations from 1:k
    p <- 0
    for(i in 1:k) {
      ## p <- p + length(FLSSS(nx, r2, i, 0.1, solutionNeed=sr2))/denom
      p <- p + as.numeric(nrow(partitionsGeneral(r2, nx, target=i))/denom)
    }

    ## done
    return(p)
  }
```

To check that everything was working, I built a squared ranks test wrapper, which is provided below.

```
sqranks.test <- function(x, y,
  alternative=c("two-tailed", "less", "greater"))
  {
    ## check alternative argument
    alternative <- match.arg(alternative)

    ## data dimensions
    nx <- length(x)
```

```
  ny <- length(y)
  N <- nx + ny

  ## calculate absolute discrepancies from the (estimated) mean
  xbar <- mean(x)
  ybar <- mean(y)
  u <- abs(x - xbar)
  v <- abs(y - ybar)

  ## calculate the observed ranks, and t
  r <- rank(c(u, v))
  ru <- r[seq_along(u)]
  rv <- r[-seq_along(u)]

  ## check for ties and possibly warn
  if(length(r) - length(unique(r)) > 0.1*N)
    warning("more than 10% ties, suggest using normal approximation")

  ## calculate test statistic
  t <- sum(ru^2)

  ## go through alternatives for p-value
  if(alternative == "less") phi <- psqranks(t, nx, ny)
  else if(alternative == "greater") phi <- 1 - psqranks(t-1, nx, ny)
  else { ## two-sided
    phi <- 2*min(psqranks(t, nx, ny), 1 - psqranks(t-1, nx, ny))
  }

  ## return results
  return(list(stat=t, p.value=phi, alternative=alternative))
}
```

The output of this function matches what `conover(..., do.exact=TRUE)` provides, using a library function from `ANSM5` (Spencer, 2024) on CRAN. Like the shoutout to `pwilcox` and `wilcox.test` above, `conover` is faster than `sqranks.test` because its implementation uses purpose-built (rather than general purpose) subroutines to enumerate sums of ranks. To be honest, I couldn't figure out what was going on under the hood in `conover`. One upside to `sqranks.test` is that it's super straightforward and easy to adapt to other similar tests.

Bibliography

Agresti, A. and Coull, B. (1998). Approximate is better than "exact" for interval estimation of binomial proportions. *The American Statistician*, 52(2):119–126.

Al-Ghamdi, A. (2001). Analysis of time headways on urban roads: case study from Riyadh. *Journal of Transportation Engineering*, 127(4):289–294.

Allaire, J., Francois, R., Ushey, K., Vandenbrouck, G., Geelnard, M., and Intel (2025). `RcppParallel`*: Parallel Programming Tools for* `Rcpp`. R package version 5.1.10.

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., and Chang, W. (2018). `rmarkdown`*: Dynamic Documents for R*. R package version 1.10.

Anderson, E. (1935). The irises of the Gaspe Peninsula. *Bulletin of the American Iris Society*, 59:2–5.

Arnholt, A. and Evans, B. (2023). *BSDA: Basic Statistics and Data Analysis*. R package version 1.2.2.

Assaf, A., Beresford, S., Risica, P., Aragaki, A., Brunner, R., Bowen, D., Naughton, M., Rosal, M., Snetselaar, L., and Wenger, N. (2016). Low-fat dietary pattern intervention and health-related quality of life: The women's health initiative randomized controlled dietary modification trial. *Journal of the Academy of Nutrition and Dietetics*, 116(2):259–271.

Barrick, R. and Showers, W. (1994). Thermophysiology of Tyrannosaurus rex; evidence from oxygen isotopes. *Science*, 265:222–224.

Becker, R., Chambers, J., and Wilks, A. (1988). *The New S Language*. Wadsworth & Brooks/Cole.

Behrens, W. (1928). *Ein beitrag zur fehlerberechnung bei wenigen beobachtungen*. Institut für Pflanzenbau.

Best, D. and Roberts, D. (1975). Algorithm AS 89: The upper tail probabilities of Spearman's rho. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 24(3):377–379.

Canty, A. and Ripley, B. (2024). `boot`*: Bootstrap* `R` *(S-Plus) Functions*. R package version 1.3-31.

Chernoff, H. and Lehmann, E. (1954). The Use of Maximum Likelihood Estimates in χ^2 Tests for Goodness of Fit. *The Annals of Mathematical Statistics*, 25(3):579–586.

Clopper, C. and Pearson, E. (1934). The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26:404–413.

Conover, W. (1999). *Practical nonparametric statistics*. Wiley.

Davidian, M. and Giltinan, D. (1995). *Nonlinear Models for Repeated Measurement Data*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, Boca Raton, FL.

Davison, A. and Hinkley, D. (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press, Cambridge.

Dwass, M. (1957). Modified randomization tests for nonparametric hypotheses. *Annals of Mathematical Statistics*, 28(1):118–128.

Eddelbuettel, D. and Balamuta, J. (2018). Extending R with C++: A Brief Introduction to Rcpp. *The American Statistician*, 72(1):28–36.

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26.

Fay, M. (2023). asht*: Applied Statistical Hypothesis Tests*. R package version 1.0.1.

Fay, M. and Shaw, P. (2010). Exact and asymptotic weighted logrank tests for interval censored data: The interval R package. *Journal of Statistical Software*, 36(2):1–34.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7 (Part II):179–188.

Genz, A. and Bretz, F. (2009). *Computation of Multivariate Normal and t Probabilities*. Lecture Notes in Statistics. Springer-Verlag, Heidelberg.

Greenberg, D. and Kosters, M. (1970). Income guarantees and the working poor: The effect of income maintenance programs on the hours of work of male family heads. *RAND Reports*.

Hammersley, J. and Handscomb, D. (1964). *Monte Carlo Methods*. Chapman & Hall, London.

Harris, J. (1912). A simple test of the goodness of fit of Mendelian ratios. *The American Naturalist*, 46(552):741–745.

Hastie, T. and Efron, B. (2022). lars*: Least Angle Regression, Lasso and Forward Stagewise*. R package version 1.3.

Hoff, P. (2009). *A first course in Bayesian statistical methods*, volume 580. Springer.

Hope, A. C. (1968). A simplified Monte Carlo significance test procedure. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 30(3):582–598.

Hribar, L. (2019). Dataset for mosquito collections on Big Pine Key, Florida, USA. *Data in Brief*, 26:104516.

Jones, O., Maillardet, R., and Robinson, A. (2009). *Scientific programming and simulation using R*. CRC Press, Boca Raton, FL.

Kendall, M. (1948). *Rank Correlation Methods*. C. Griffin.

Liu, C. (2025). FLSSS*: Mining Rigs for Problems in the Subset Sum Family*. R package version 9.2.0.

Malik, H. (1970). Estimation of the parameters of the Pareto distribution. *Metrika*, 15(1):126–132.

Mann, H. and Whitney, D. (1947). On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, pages 50–60.

Millard, S. (2013). *EnvStats: An R Package for Environmental Statistics*. Springer, New York.

Nandi, S., Mukherjee, P., Tambe, S., Kumar, R., and Kulkarni, B. (2002). Reaction modeling and optimization using neural networks and genetic algorithms: case study involving TS-1-catalyzed hydroxylation of benzene. *Industrial & engineering chemistry research*, 41(9):2159–2169.

Patefield, W. M. (1981). Algorithm AS 159: an efficient method of generating random $r \times c$ tables with given row and column totals. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 30(1):91–97.

Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 50(302):157–175.

Pinheiro, J., Bates, D., and R Core Team (2025). `nlme`*: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-168.

Ramsey, F. and Schafer, D. (2013). *The Statistical Sleuth: A course in methods of data analysis, 3rd edition*. Brooks/Cole Cengage Learning.

Ramsey, F., Schafer, D., Sifneos, J., Turlach, B., Horton, N., Loi, L., Aloisio, K., Zhang, R., and Pruim, R. (2024). `Sleuth3`*: Data Sets from Ramsey and Schafer's "Statistical Sleuth (3rd Ed)"*. R package version 1.0-6.

Rousseeuw, P. and Leroy, A. (1987). *Robust Regression and Outlier Detection*. Wiley, New Jersey.

Simpson, J., Olsen, A., and Eden, J. (1975). A Bayesian analysis of a multiplicative treatment effect in weather modification. *Technometrics*, 17(2):161–166.

Smeeton, N., Spencer, N., and Sprent, P. (2025). *Applied Nonparametric Statistical Methods*. Chapman & Hall/CRC, Boca Raton, FL.

Solomon, P., Adams, F., Silver, A., Zimmer, J., and DeVeaux, R. (2002). Ginkgo for memory enhancement: a randomized controlled trial. *Jama*, 288(7):835–840.

Spencer, N. (2024). `ANSM5`*: Functions and Data for the Book "Applied Nonparametric Statistical Methods", 5th Edition*. R package version 1.1.1.

Stefan, R. and Cheche, T. (2016). Coin toss modeling. *arXiv preprint arXiv:1612.06705*.

Torcher, K. (1969). *Art of Simulation*. Hodder & Stoughton Ltd, England.

Wasserman, L. (2004). *All of Statistics: a concise course in statistical inference*. Springer, New York.

Wasserman, L. (2006). *All of Nonparametric Statistics*. Springer, New York.

Weisberg, S. (1985). *Applied Linear Regression, 2nd edition*. Wiley, New Jersey.

Welch, B. (1938). The significance of the difference between two means when the population variances are unequal. *Biometrika*, 29(3/4):350–362.

Wheeler, B. (2025). `SuppDists`*: Supplementary Distributions*. R package version 1.1-9.9.

Wickham, H. (2016). `ggplot2`*: Elegant Graphics for Data Analysis*. Springer-Verlag, New York, NY.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.

Wood, J. (2025). `RcppAlgos`*: High Performance Tools for Combinatorics and Computational Mathematics*. R package version 2.9.3.

Xie, Y. (2015). *Dynamic Documents with R and* `knitr`. Chapman & Hall/CRC, Boca Raton, Florida, 2nd edition.

Xie, Y. (2018a). `bookdown`*: Authoring Books and Technical Documents with Rmarkdown*. R package version 0.9.

Xie, Y. (2018b). `knitr`*: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.20.

Index