Statistical Modeling

Robert B. Gramacy
Virginia Tech Department of Statistics

bobby.gramacy.com

Summary Statistics

R includes many functions for summarizing data. E.g.,

```
> load("dow30.RData")
> mean(dow30$Open)
[1] 56.97535
> min(dow30$Open)
[1] 6.9
> max(dow30$Open)
[1] 211.15
> range(dow30$Open)
[1]   6.90 211.15
> sd(dow30$Open)
[1] 38.44013
```

The empirical distribution of data can be summarized in a variety of ways.

```
> quantile(dow30$Open,
+     probs=c(0, 0.25, 0.5, 0.75, 1))
     0%     25%     50%     75%    100%
  6.900  27.790  49.450  75.935 211.150
> fivenum(dow30$Open)
[1]   6.90  27.79  49.45  75.95 211.15
> IQR(dow30$Open)
[1] 48.145
> summary(dow30$Open)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   6.90   27.79   49.45   56.98   75.94  211.20
```

Correlation and covariance

The simplest way to characterize the relationship between two random variables is through their correlation.

Recall the birth weight example.

```
> cor(b6c$WTGAIN, b6c$DBWT)
[1] 0.1751866
```

By default, this calculates Pearson's correlation:

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1^n}(x_i - \bar{x})}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})}}$$

which works best for normally distributed data.

`cor()` can also do Spearman's ρ and Kendall's τ.

- A non-parametric (distribution free) correlation:

$$\rho = \frac{n(\sum_i x_i y_i) - (\sum_i x_i)(\sum_i y_i)}{\sqrt{n(\sum_i x_i^2) - (\sum_i x_i)^2}\sqrt{n(\sum_i y_i^2) - (\sum_i y_i)^2}}$$

```
> cor(b6c$WTGAIN, b6c$DBWT, method="spearman")
[1] 0.1776192
```

- A rank-based correlation. *(Heavy computation.)*

$$\tau = \frac{n_c - n_d}{1/2n(n-1)}$$

```
> cor(b6c$WTGAIN, b6c$DBWT, method="kendall")
[1] 0.1214556
```

The definition of covariance is

$$\mathbb{C}\mathrm{ov}(X, Y) = \mathbb{E}\{(X - \mathbb{E}(X))(Y - \mathbb{E}(Y))\},$$

whose sample version appears in the numerator of the Pearson correlation.

- So it is correlation scaled by the product of (square-roots) of marginal variances.
- And this idea can be extended to the Spearman and Kendall versions.
- `cov()` takes the same arguments as `cor()`.

Principal Components & Factors

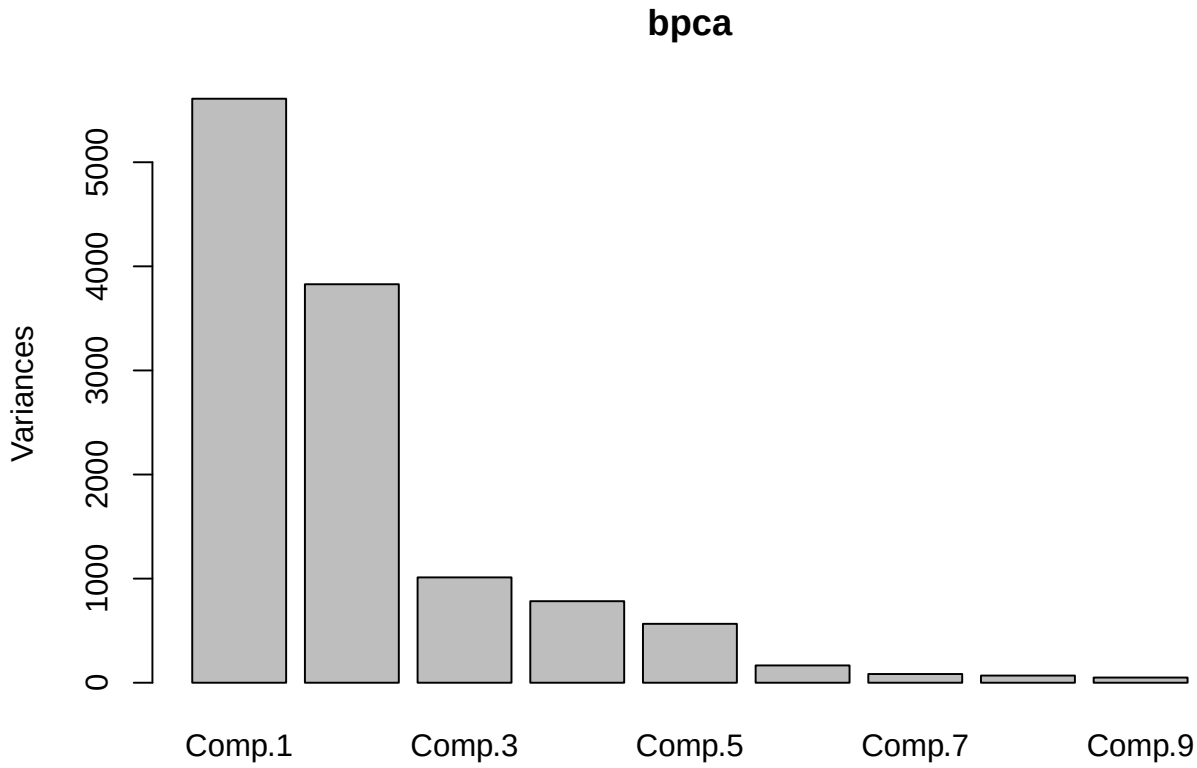
PCA breaks a set of (possibly correlated) variables into a set of uncorrelated ones.

- Under the hood it uses SVD (`prcomp()`) or an eigen-decomposition (`princomp()`).
- The output is a projection, potentially to a lower dimensional space called the principal components.
- As such, it is a dimension reduction tool,
- aiming to reduce correlated high dimensional data into a small number of "directions" of large variability.

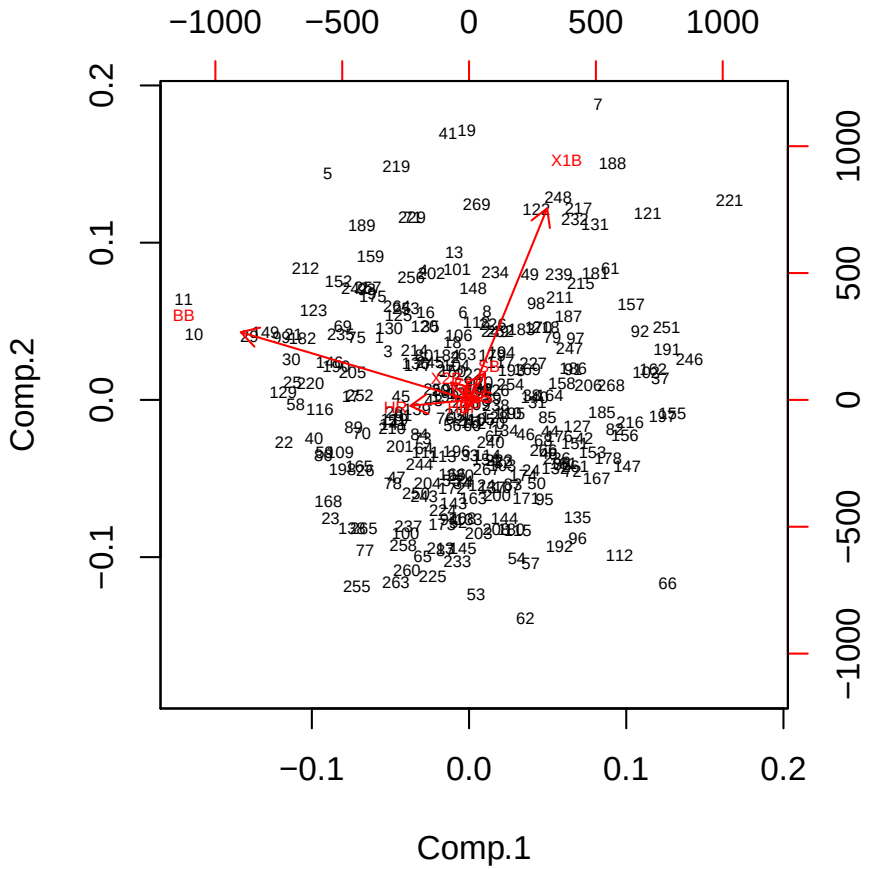
You can give it a matrix `x` or a data frame and `formula`.

Consider the batting data again.

```
> bat08 <- read.csv("bat08.csv")
> bat08 <- transform(bat08, X1B=H-X2B-X3B-HR)
> b08pca <- princomp(~X1B+X2B+X3B+HR+BB+HBP+SF+SB+CS,
+                    data=bat08)
> plot(bpca)
```



A `biplot()` graphically displays the contributions of each of the variables to a pair of principal components, and shows individual observations on the same scale.




```
> biplot(bpca, cex=0.5)
```

The function `factanal()` provides a similar functionality for factor analysis (FA).

- Factor analysis is similar to PCA in that it finds a lower-dimensional representation.
- It involves specifying a probability model, usually MVN.

$$\mathbf{x} = \mathbf{\Lambda f} + \mathbf{e}, \quad \text{usually} \;\; \mathbf{e} \sim \mathcal{N}_n$$

- and inference by maximum likelihood.

More choices, and extra computational overhead, but allows tests of hypotheses (for the number of factors).

```
> bpca <- factanal(~X1B+X2B+X3B+HR+BB+HBP+SF+SB+CS,
+                  data=bt0008, factors=2)
> loadings(update(bfact, factors=4))

Loadings:
    Factor1 Factor2 Factor3 Factor4
X1B  0.114  -0.110   0.535  -0.286
X2B -0.257   0.126   0.395   0.213
X3B  0.170  -0.120   0.251  -0.151
HR  -0.127   0.364           0.545
BB           0.984           0.105
HBP                          0.370
SF   0.103   0.181   0.689
SB   0.702           0.161  -0.107
CS   0.827  -0.129
...
```

Probability distributions

Many statistical tests work by calculating a test statistic and then comparing it to a value from a theoretical distribution.

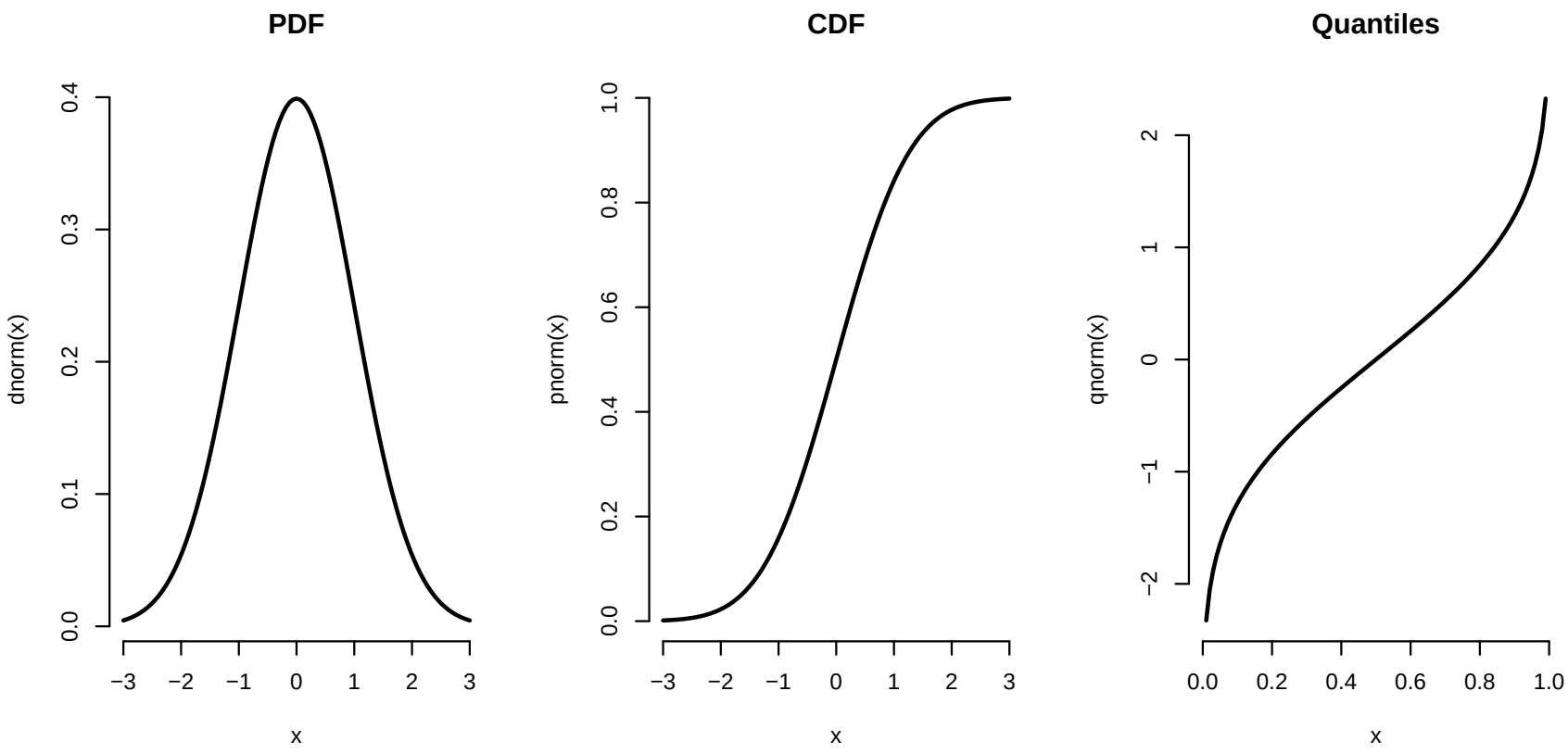
R provides a set of functions to calculate

- densities, distributions, and quantiles

from common families.

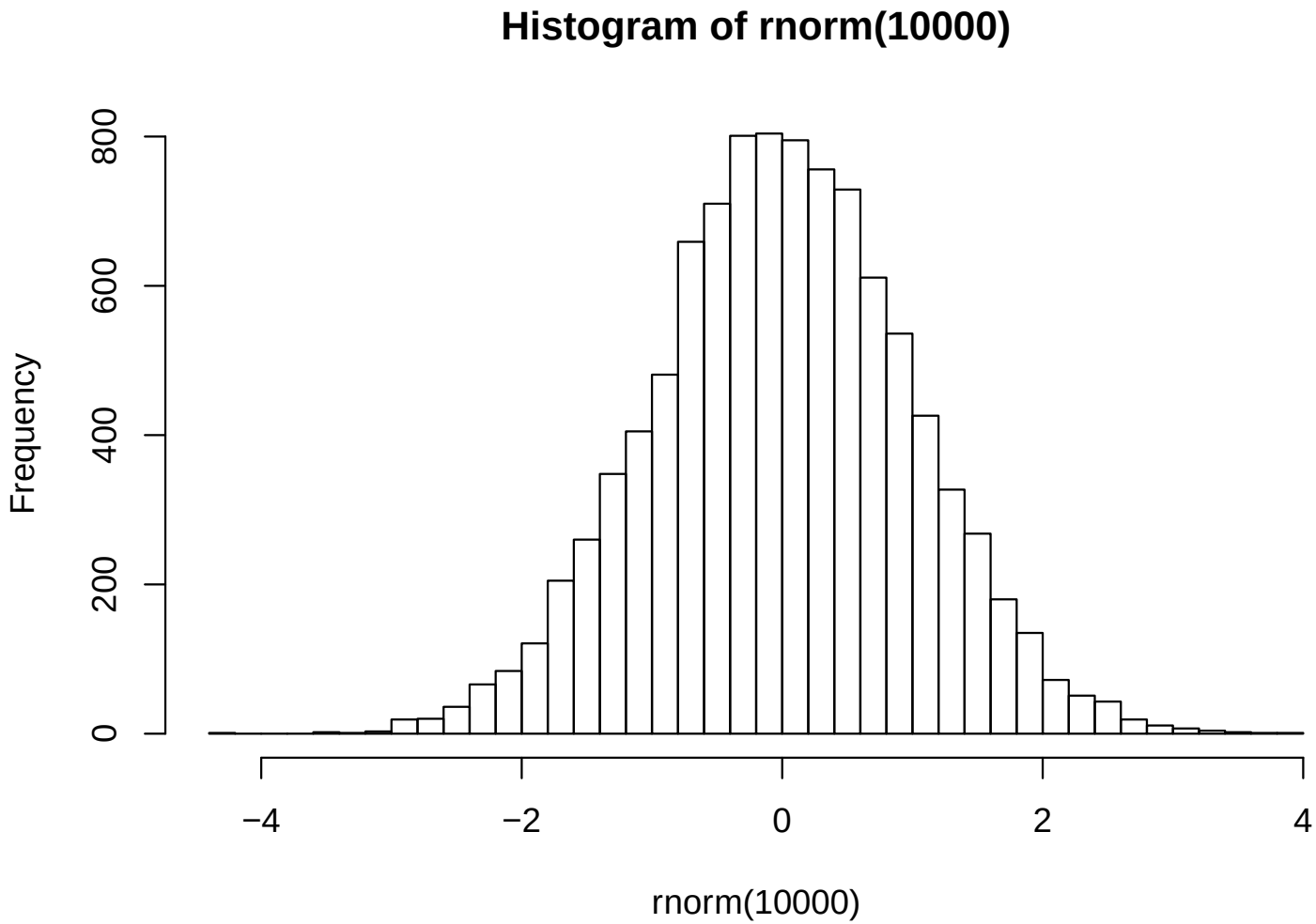
- You can also generate random variables, which is important for Monte Carlo experiments.

```
> par(mfrow=c(1,3), lwd=2, bty="n")
> plot(dnorm, -3, 3, main="PDF")
> plot(pnorm, -3, 3, main="CDF")
> plot(qnorm, 0, 1, main="Quantiles")
```



The `rnorm()` generates deviates from the normal distribution.

```
> hist(rnorm(10000), breaks=50)
```



The same suite of four functions

- `d*()`, `p*()`, `q*()`, and `r*()`

are available for most common distributions, i.e., where `*` is

- `beta`, `binom`, `cauchy`, `chisq`, `exp`, `f`, `gamma`, `geometric`, `lnorm`, `pois`, `t`, and more ...

They share common arguments to specify the x-value, distribution or quantile, number of samples, units of output

- `x`, `log`, `q`, `p`, `lower.tail`, `log.p`, `n`

but differ in distribution-specific arguments. E.g.,

- The `*norm()` functions take optional `mean=` and `sd=` arguments, whereas `*pois()` takes `lambda=`

Statistical Tests

R automates many types of conventional, and cutting-edge, statistical hypothesis tests.

- We'll introduce some of the basics here,
- starting with continuous data,
- and refer to others alongside their modeling routines, like `lm()` and `glm()`, etc.

The most basic ones are based on comparing means of normally distributed data.

- E.g., asking if the mean of some data is close to what you expected before designing the experiment.
- `t.test()` can help with that, and a lot more.

For example, consider an experiment on tire durability performed by the National Highway Traffic Safety Administration (NHTSA) in 2003.

- They stress tested tires of many brands on specialized equipment in extreme conditions.

Consider a particular tire type `"H"` from *Pathfinder*.

```
> tires <- read.csv("tires.csv")
> hfail <- subset(tires, Tire_Type=="H" &
+   Speed_At_Failure_km_h == 160)$Time_To_Failure
> hfail
 [1] 10.00 16.67 13.58 13.53 16.83  7.62  4.25
 [8] 10.67  4.42  4.25
> mean(hfail)
[1] 10.182
```

If, *a priori*, we thought these tires would last about nine hours, should we be surprised to find they lasted more than ten?

```
> t.test(hfail, mu=9)

One Sample t-test

data:  hfail
t = 0.7569, df = 9, p-value = 0.4684
alternative hypothesis: true mean is not equal to 9
95 percent confidence interval:
  6.649536 13.714464
sample estimates:
mean of x
   10.182
```

- No, we should not.

Another common situation is when you have two groups of observations, and you want to know if there is a significant difference between their means.

There were three tires that had the same speed rating, at 180km/h.

```
> efail <- subset(tires, Tire_Type=="E" &
+   Speed_At_Failure_km_h == 180)$Time_To_Failure
> dfail <- subset(tires, Tire_Type=="D" &
+   Speed_At_Failure_km_h == 180)$Time_To_Failure
> bfail <- subset(tires, Tire_Type=="B" &
+   Speed_At_Failure_km_h == 180)$Time_To_Failure
```

Starting with types "D" and "E".

```
> t.test(efail, dfail)

Welch Two Sample t-test
data:  efail and dfail
t = -2.5042, df = 8.961, p-value = 0.03373
alternative hypothesis:
  true difference in means is not equal to 0
95 percent confidence interval:
 -0.82222528 -0.04148901
sample estimates:
mean of x mean of y
 4.321000  4.752857
```

`t.test()` can also work with paired data.

- This is useful for comparing data from two *different entities* under otherwise *identical conditions*.
- We'll see some examples in Lecture 8 on Monte Carlo experiments.

A non-parametric equivalent to the t-test is available via `wilcox.test()`.

Comparing variances proceeds very similarly: via

- `var.test()` uses F-tests, or
- `bartlett.test()` for Bartlett tests.

Analysis of Variance

ANOVA means many things.

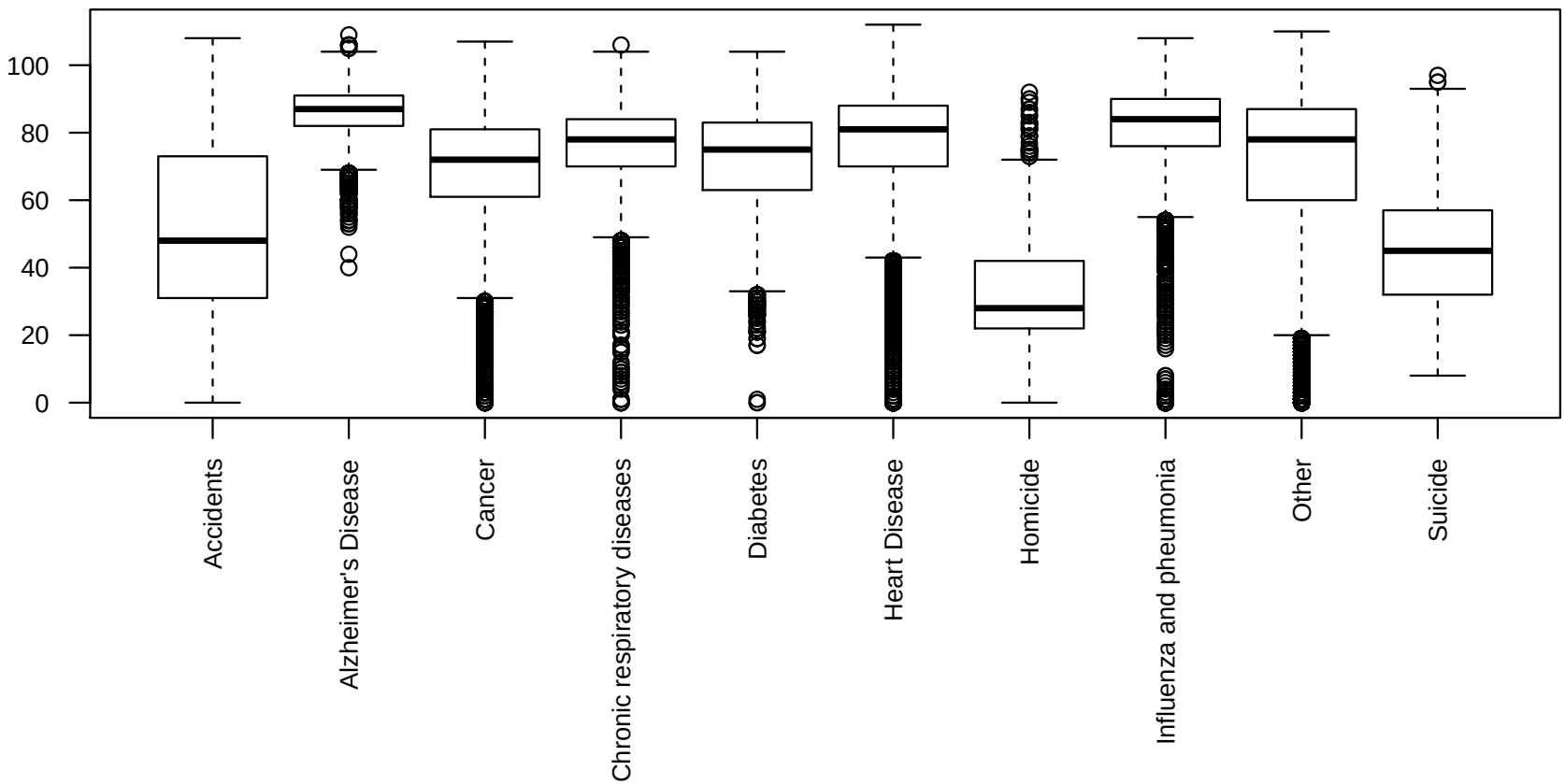
- At its most basic it gives a way of comparing means across more than two groups.
- It asks the question: *Is the between-group variability larger than the within-group variability?* by calculating sums of squares.

Two functions offer ANOVA functionality in R

- `aov()`: a crude interface specifically for grouped data
- `anova()`: a generic method that works on model output (e.g., `lm()`, `glm()`, etc.)

Consider the 2006 US mortality data set.

```
> mort06 <- read.csv("mort06.csv")
> boxplot(age~Cause, data=mort06, cex.axis=0.75)
```



Here is how to do the ANOVA with `anova()`.

```
> anova(lm(age~Cause, data=mort06))
Analysis of Variance Table

Response: age
              Df   Sum Sq Mean Sq F value    Pr(>F)
Cause          9 15727886 1747543  5893.3 < 2.2e-16 ***
Residuals 243034 72067515     297
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

- Grouping by `Cause` is significant (assuming normality).
- A non-parametric equivalent is `kruskal.test()`.
- Compare variances by group using `flinger.test()`.

Distribution Tests

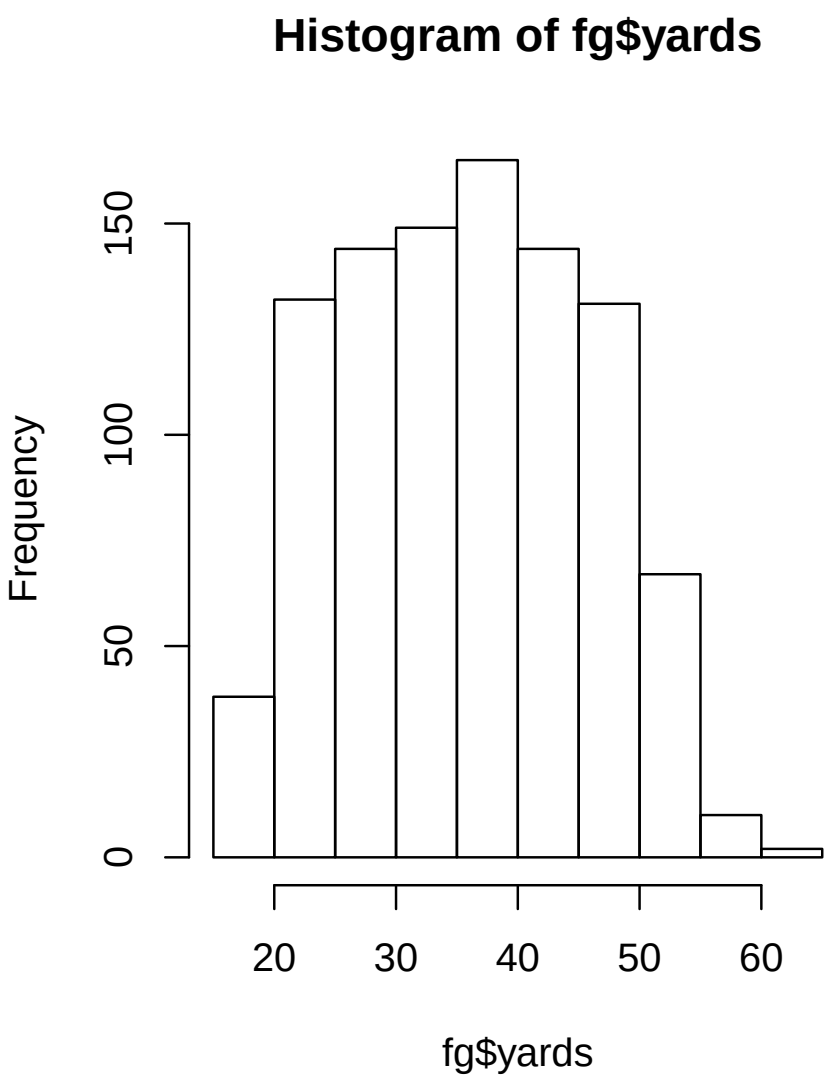
Often we want to check if data have a particular distribution,

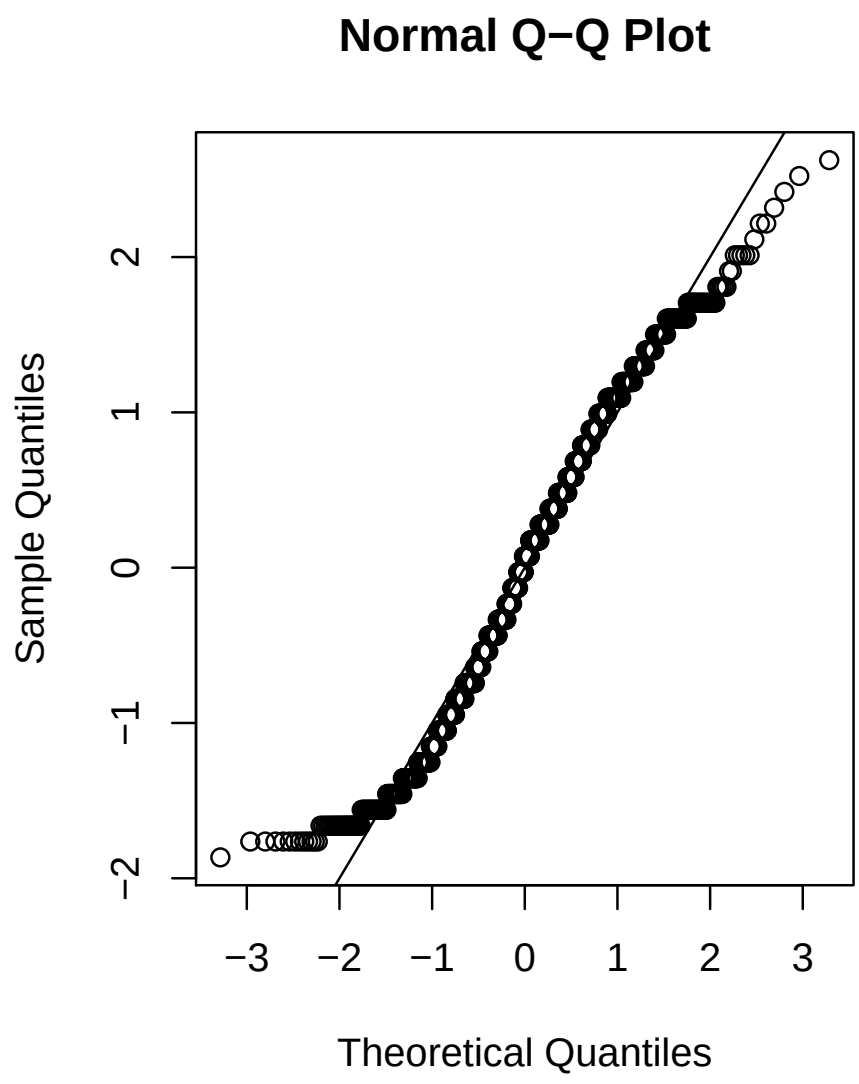
- e.g., Gaussian,

because many modeling/testing methods depend on a Guassianity assumption being true.

One option is visualization. Consider data on field goals in the NFL during 2005.

```
> fg <- read.csv("fg.csv")
> par(mfrow=c(1,2))
> hist(fg$yards)
> yn <- (fg$yards - mean(fg$yards))/sd(fg$yards)
> qqnorm(yn); abline(0,1)
```


Histogram of fg$yards
Frequency
0
50
100
150
20
30
40
50
60
fg$yards


Normal Q−Q Plot
Sample Quantiles
−2
−1
0
1
2
−3
−2
−1
0
1
2
3
Theoretical Quantiles

To test for normality you can try the Shapiro-Wilk test.

```
> shapiro.test(fg$yards)

Shapiro-Wilk normality test

data:  fg$yards
W = 0.9728, p-value = 1.307e-12
```

- Rejects the null hypothesis of Gaussianity.

A Kolmogorov–Smirnov (KS) test allows you to compare against any distribution; Try `ks.test(x, y, ...)` where `y=` specifies the distribution, either as

- data values, in which case the function checks that `x` and `y` have the same distribution,
- or a distribution function like `pnorm`.

```
> ks.test(fg$yards, pnorm,
+         mean=mean(fg$yards), sd=sd(fg$yards))
One-sample Kolmogorov-Smirnov test
data:  fg$yards
D = 0.0665, p-value = 0.0003349
alternative hypothesis: two-sided
```

Discrete Data

There is a different set of tests for looking at the statistical significance of discrete random variables, like counts or proportions.

E.g., if you have a set of data

- with several different groups of observations
- and are measuring the probability of success in each group,

you can use `prop.test()` to measure whether the difference between groups is statistically significant.

For example, consider data on field goal success versus stadium type (inside or outside).

```
> fgtab <- t(fgtab[3:4,])
> fgtab
```

```
      FG good FG no
Both       53    14
In        152    24
Out       582   125
```

We can check if there is a statistically significant difference in success among the groups.

```
> prop.test(fgtab)

   3-sample test for equality of proportions
   without continuity correction

data:  fgtab
X-squared = 2.3298, df = 2, p-value = 0.312
alternative hypothesis: two.sided
sample estimates:
   prop 1    prop 2    prop 3
0.7910448 0.8636364 0.8231966
```

- No difference.

Other options for discrete data include.

- Binomial tests for data on trials with two outcomes; see `binom.test()`.
- Fisher's exact test, checking for independence in two categorical variables; see `fisher.test()`.
- χ-squared tests check for differences in the proportion of categories from two populations; see `chisq.test()`.
- The Friedman rank-sum test is a non-parametric counterpart to ANOVA for categorical data; see `friedman.test()`

Linear Models

A linear regression assumes that there is a linear relationship between the response variable, Y_i, and the predictors $x_{i1}, \ldots, x_{ip}$.

$$Y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i, \quad \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2),$$

for observations $i = 1, \ldots, n$.

Inferring the unknown $\hat{\beta}_j$ provides information that is descriptive and predictive.

- We can test if $\beta_j = 0$ to determine relevance.
- We can predict $Y(x)$ for new x.

Consider fitting a linear model for `runs` as the response (y), and batting and base running data as predictors (x) for the same team data we used earlier.

```
> blm <- lm(R~X1B+X2B+X3B+HR+BB+HBP+SF+SB+CS,
+           data=bt0008)
> coef(blm)
  (Intercept)           X1B           X2B           X3B
-507.16019759    0.56704867    0.69110420    1.15836091
           HR            BB           HBP            SF
   1.47438916    0.30117665    0.37749717    0.87218094
           SB            CS
   0.04369407   -0.01533245
```

More information is available via `summary(blm)`; see R output,

- including info on standard errors, t-statistics and p-values.
- Those also form the basis of confidence intervals.

```
> confint(blm)
                    2.5 %       97.5 %
(Intercept) -570.85828008 -443.4621151
X1B            0.51583022    0.6182671
X2B            0.57449582    0.8077126
X3B            0.81752968    1.4991921
HR             1.37432941    1.5744489
BB             0.25570041    0.3466529
HBP            0.16077399    0.5942203
SF             0.49451857    1.2498433
SB            -0.07349342    0.1608816
CS            -0.32152716    0.2908623
```

```
> XtXi <- solve(t(X) %*% X)
> bhat <- XtXi %*% t(X) %*% y
> df <- length(y) - ncol(X)
> s2 <- sum((X %*% bhat - y)^2)/df
> SEs <- sqrt(diag(XtXi)*s2)
> tvals <- bhat/SEs
> pvals <- 2*pt(abs(tvals), df=df, lower.tail=FALSE)
> t(pvals)
                1          X1B          X2B
[1,] 1.896707e-39 1.249261e-60 1.381926e-25
              X3B           HR           BB
[1,] 1.343378e-10 1.662058e-83 3.018015e-30
              HBP           SF           SB
[1,] 0.0007020266 8.328536e-06 0.4634872
             CS
[1,] 0.9215298
```

The last two x's, SB and CS,may not be useful. We may

- get better estimates of the other coefficients,
- and tighter estimates of standard errors (better predictions) without them.

One option is to do a new `lm()` command without them.

- Or we can `update()` the existing fit.

```
> blm2 <-  update(blm, R~X1B+X2B+X3B+HR+BB+HBP+SF)
> summary(blm)$r.squared
[1] 0.9144079
> summary(blm2)$r.squared
[1] 0.9141863
```

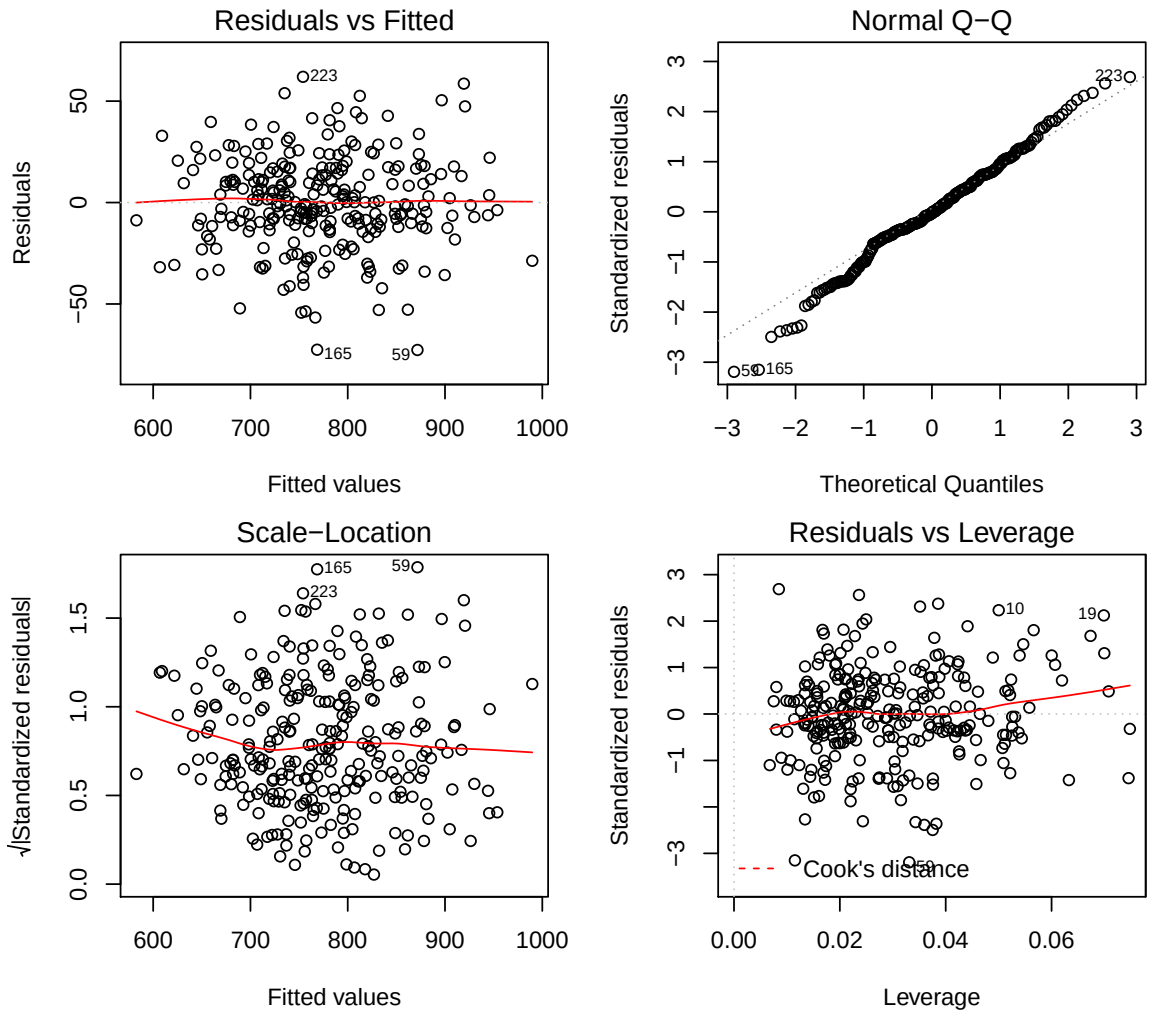
These models are nested, so to formally *test* which is better we can use an ANOVA technique.

- Normalized ratio's of R^2s have an F-distribution.

```
> anova(blm, blm2)
Analysis of Variance Table

Model 1: R ~ X1B + X2B + X3B + HR + BB + HBP + SF + SB + CS
Model 2: R ~ X1B + X2B + X3B + HR + BB + HBP + SF
  Res.Df    RSS Df Sum of Sq      F Pr(>F)
1    260 140078
2    262 140440 -2   -362.56 0.3365 0.7146
```

`plot.lm()`, can help check modeling assumptions.

```
> par(mfrow=c(2,2), mar=c(4,4,2,2))
> plot(blm2)
```



`predict.lm()` can help with forecasts and associated uncertainties.

```
> Xf <- bt0008[sample(1:nrow(bt0008), 2),]
> predict(blm2, newdata=Xf,
+       interval="prediction", se.fit=TRUE)
$fit
         fit      lwr      upr
133 831.4485 784.9997 877.8973
52  740.8836 694.8188 786.9484

$se.fit
     133       52
4.519411 3.356061

$residual.scale
[1] 23.15234
```

Subset selection and shrinkage

Variable selection in linear (or any) model(s) is a hard problem.

- F-tests/ANOVA are crude implements.
- Especially with searching through $O(p^2)$ possible interaction terms.

R provides several built-in and add-on methods that are better, and better automated.

The simplest is `step()` which works with several model objects, including `lm` and `glm`.

`step()` automates a greedy search by finding next variable (or interaction) to add or remove using an information criteria (AIC or BIC).

```
> blmn <- lm(R~1, data=bt0008[,-c(1,2,13)])
> blm.aic <- step(blm, scope=~.+.^2,
+     direction="both")
> extractAIC(blm)
[1]   10.000 1707.913
> extractAIC(blm2)
[1]    8.000 1704.611
> extractAIC(blm.aic)
[1]   14.000 1696.637
```

Shrinkage methods include ridge regression and lasso.

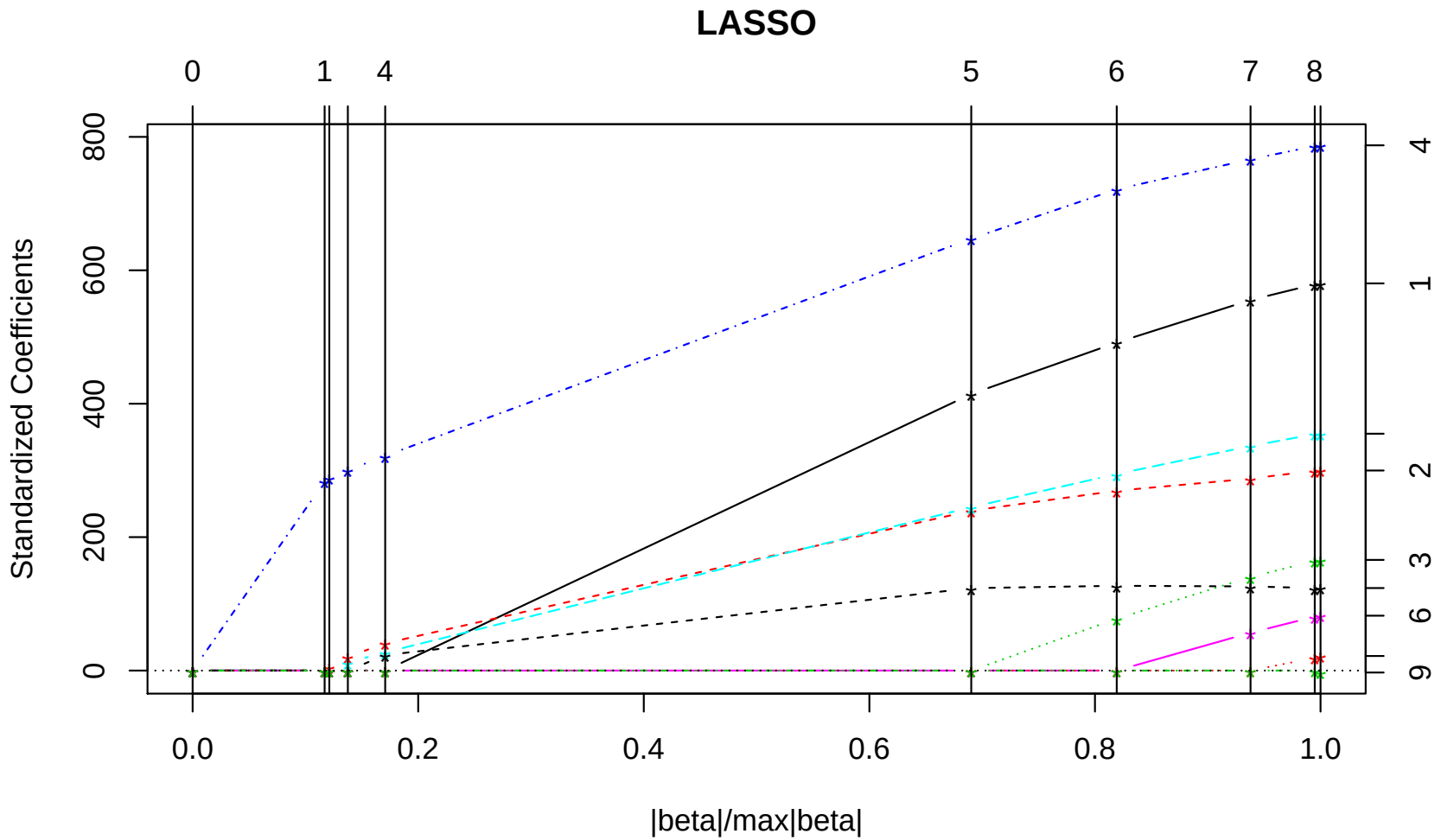
- see, e.g., `lm.ridge()` in the `MASS` library.
- lasso is part of the `lars` family/package.

Lasso solves the following optimization problem:

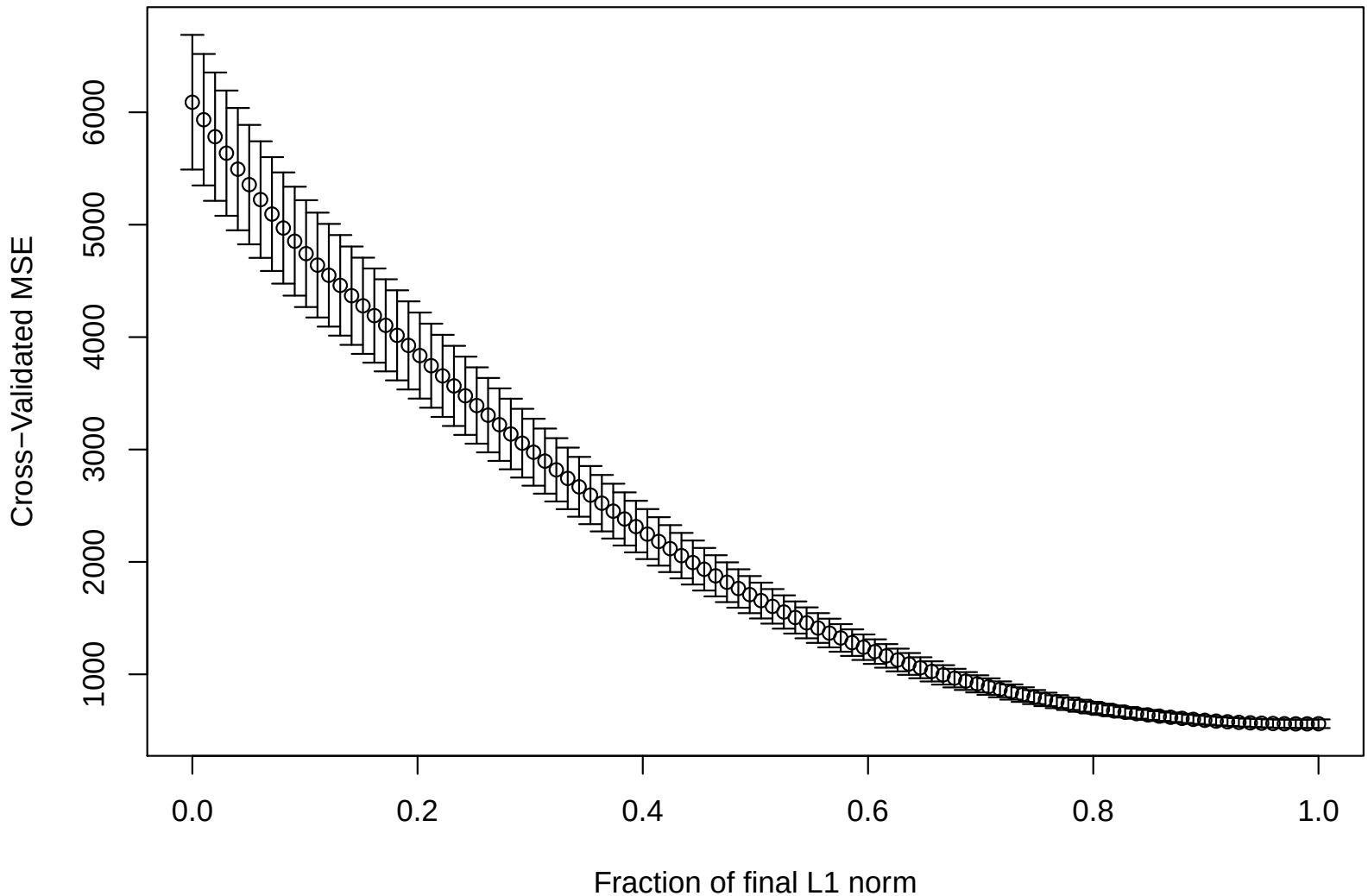
$$\arg\min_{\beta} \sum_{i=1}^{n} (y_i - x_i^\top \beta)^2 + \lambda \sum_{j=1}^{p} |\beta_i|.$$

- Cross validation (CV) is usually used to choose lambda.
- Due to the nature of the L_1 penalty, solutions $\hat{\beta}$ can be sparse.

```
> library(lars)
Loaded lars 1.1
> blasso <- lars(X[,-1], y); plot(blasso)
```



```
> blasso.cv <- cv.lars(X[,-1],y)
```



Eyeballing the 1-SE rule leads to choosing a fraction of 0.8.

```
> coef(blasso, s=0.8, mode="fraction")
      X1B       X2B       X3B        HR        BB
0.4691099 0.6108537 0.4615540 1.3302964 0.2435906
      HBP        SF        SB        CS
0.0000000 0.8924589 0.0000000 0.0000000
```

The `monomvn` package provides an automation

```
> regress(X[,-1], y, method="lasso")$b[,1][-1]
[1] 0.5328276 0.6564672 0.9103220 1.4217583
[5] 0.2798753 0.2189508 0.8907541 0.0000000
[9] 0.0000000
```

- It can do the same for ridge and other methods.

Principal Components Regression

Coming full circle, pairing a rotation of the predictors with regression (PCR) works well.

- It has connections with ridge regression.

You could do it by hand.

```
> Zp <- cbind(1, X[,-1] %*% bpca$loadings[,1:3])
> theta <- solve(t(Zp) %*% Zp) %*% t(Zp) %*% y
> bpcr <- c(theta[1], bpca$loadings[,1:3] %*% theta[-1])
> bpcr
 [1] -281.62759024    0.61930970    0.46810539
 [4]   -0.02909064    0.42129688    0.54849463
 [7]    0.02980921    0.05745549   -0.58625651
[10]   -0.19045098
```

Or, you can use routines from the `pls` package.

```
> library(pls)
> bpcr2 <- pcr(y ~ X[,-1], ncomp = 3)
> as.numeric(coef(bpcr2))
[1]  0.61930970  0.46810539 -0.02909064
[4]  0.42129688  0.54849463  0.02980921
[7]  0.05745549 -0.58625651 -0.19045098
```

One usually choses `ncomp` by CV, e.g., with `monomvn`.

```
> bpcrcv <- regress(X[,-1], y, method="pcr")
> bpcrcv$b[,1][-1]
[1]  0.62892810  0.62951157 -0.03155156
[4]  1.51824323  0.30711252  0.14710112
[7]  0.09972421  0.08039534 -0.05171339
> bpcrcv$ncomp
[1] 4
```

The `pls` package also supports partial least squares regression (PLSR),

- which is similar to PC, but rotates *with* response, y.
- It works very similarly, complete with an automation in `monomvn`.

Other similar methods:

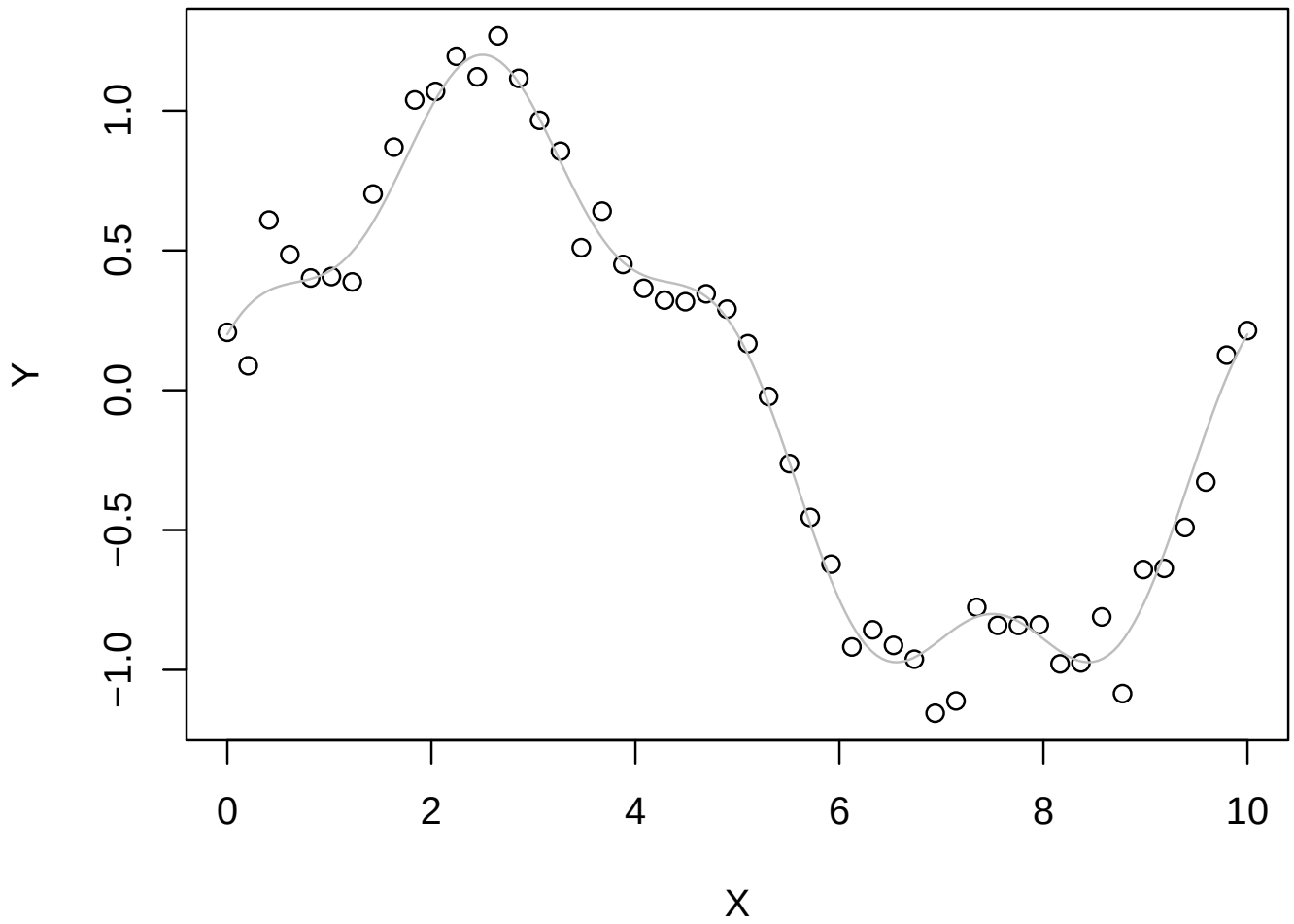
- Elastic net in the `elasticnet` package.,
- Stepwise and "forward stagewise" automations in the `lars` package.
- Bayesian methods like `blasso()` and the Horseshoe `bhs()` in the `monomvn` package.

Nonlinear regression/smoothing

R has several non-linear regression, or sometimes called smoothing methods.

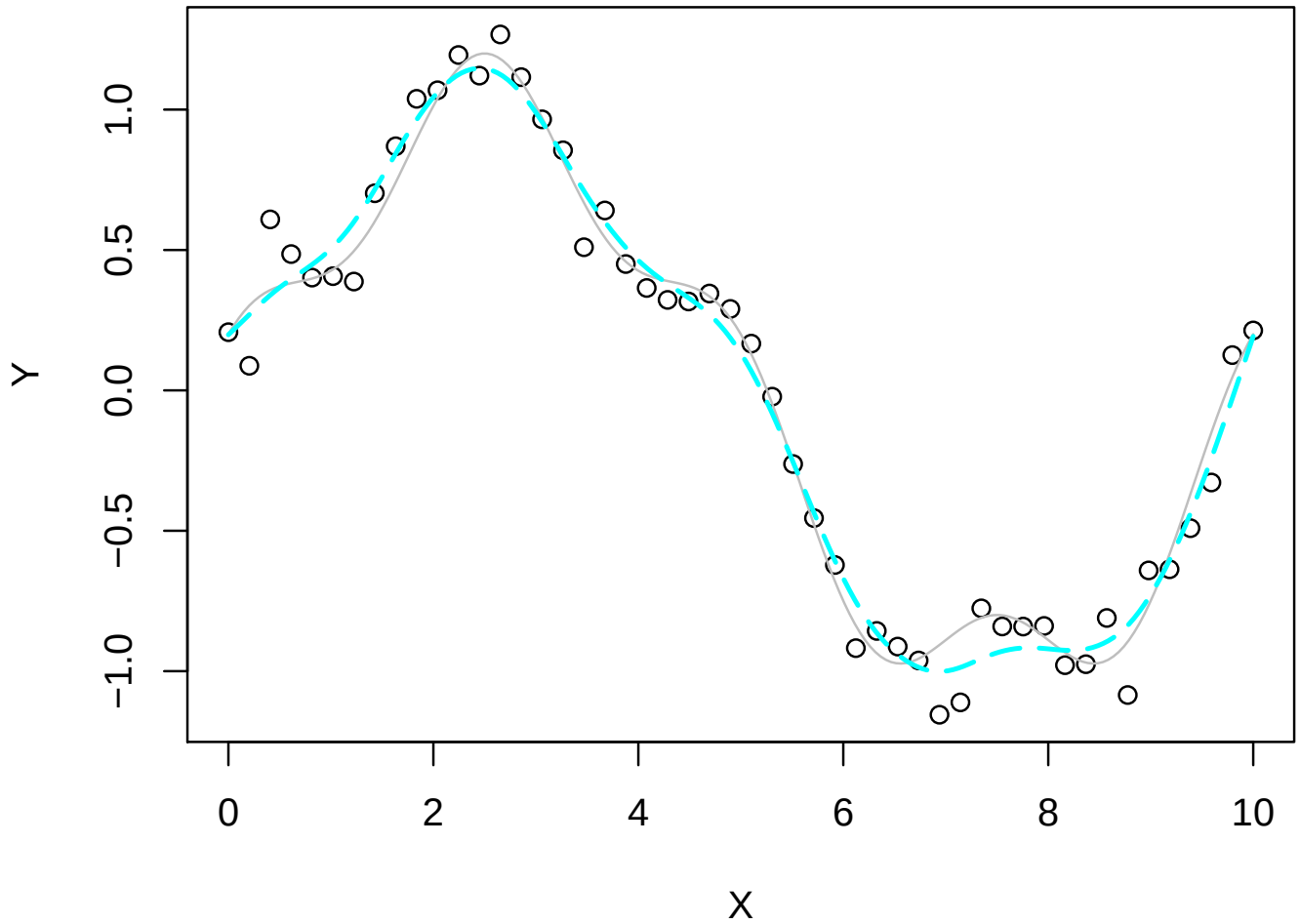
Consider the following x-y pairs.

```
> X <- seq(0,10,length=50)
> Ytrue <- (sin(pi*X/5) + 0.2*cos(4*pi*X/5))
> Y <- Ytrue + rnorm(length(Ytrue), sd=0.1)
> plot(X,Y)

> XX <- seq(0,10,length=199)
> YYtrue <- (sin(pi*XX/5) + 0.2*cos(4*pi*XX/5))
> lines(XX, YYtrue, col="gray", type="l")
```



- Smoothing splines are perfect for 1d and 2d data.

```
> library(splines)
> fit1 <- smooth.spline(X, Y, df=11)
> YY1 <- as.matrix(predict(fit4, data.frame(X=XX))$y)
> lines(XX, YY1, col=5, lty=5, lwd=2)
```



CV can be used to choose the fidelity, via the degrees of freedom parameter `df=`.

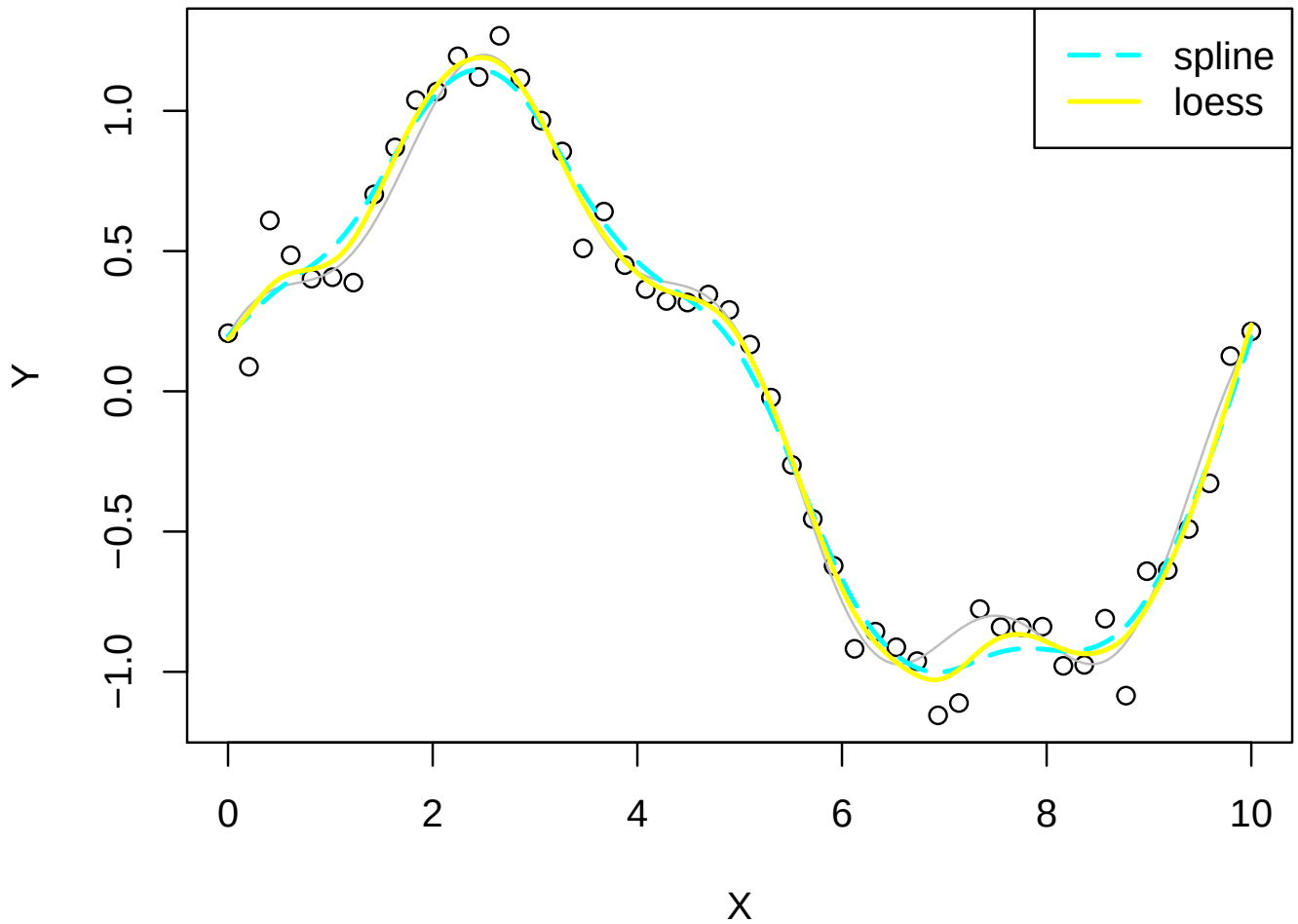
- Smoothing splines are basically a ridge regression using a special cubic polynomial basis set.

Another type of local polynomial smoothing is offered by `loess()`.

- These work well in ≤ 4 dimensions
- and have a tuning parameter called `span=`.

```
> fit3 <- loess(Y~X, span=0.5)
> YY1 <- as.matrix(predict(fit3, data.frame(X=XX)))
```

```
> lines(XX, YY2, col=7, lty=7, lwd=2)
> legend("topright", c("spline", "loess"),
+        lty=c(5,7), col=c(5,7), lwd=2)
```



Classification

What about when the response variable is categorical?

- E.g., two categories: $Y \in \{0, 1\}$.

You could try to predict the probability that $Y_i = 1$ through a linear function of the predictor variables

$$P(Y_i = 1|x_i, \beta) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}.$$

- The problem is that the linear function isn't constrained to give values in $[0, 1]$, i.e., probabilities.

One way to proceed is with a sigmoidal transformation.

Logistic Regression

An inverse logit transformation leads to logistic regression.

Let $\eta_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}$. Then,

$$P(Y_i = 1 | x_i, \beta) = \frac{e^{\eta_i}}{1 + e^{\eta_i}}.$$

Logistic regression is a special case of a Generalized Linear Model (GLM) for dealing with binary (or binomial) responses.

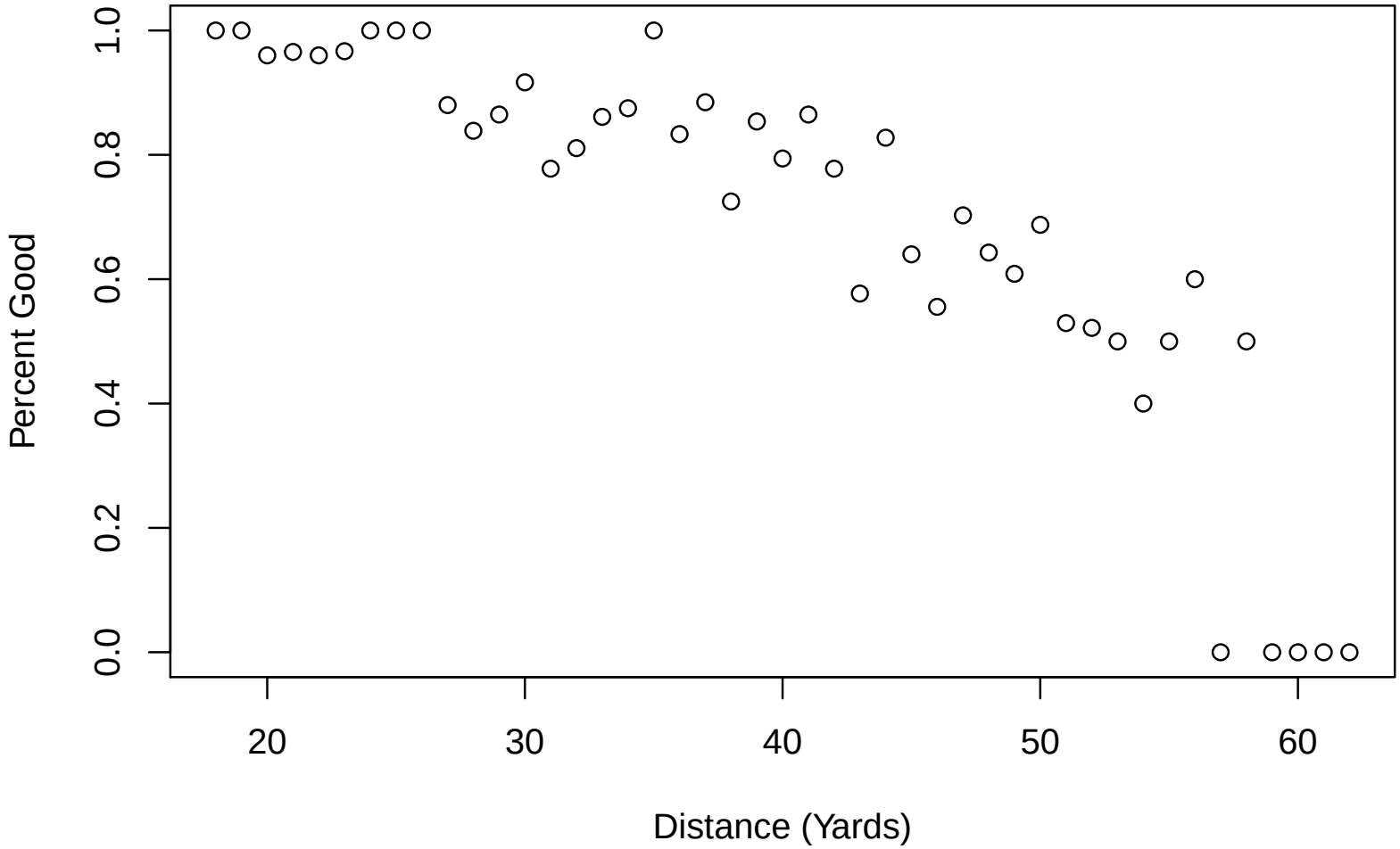
Other GLMs include

- Poisson responses $\rightarrow$ log-linear models.
- Gamma responses, ...

Consider our field goals data again.

- Each time a kicker attempts a field goal there is a chance that the goal will be successful
- and a chance it will fail.
- The probability varies according to distance.

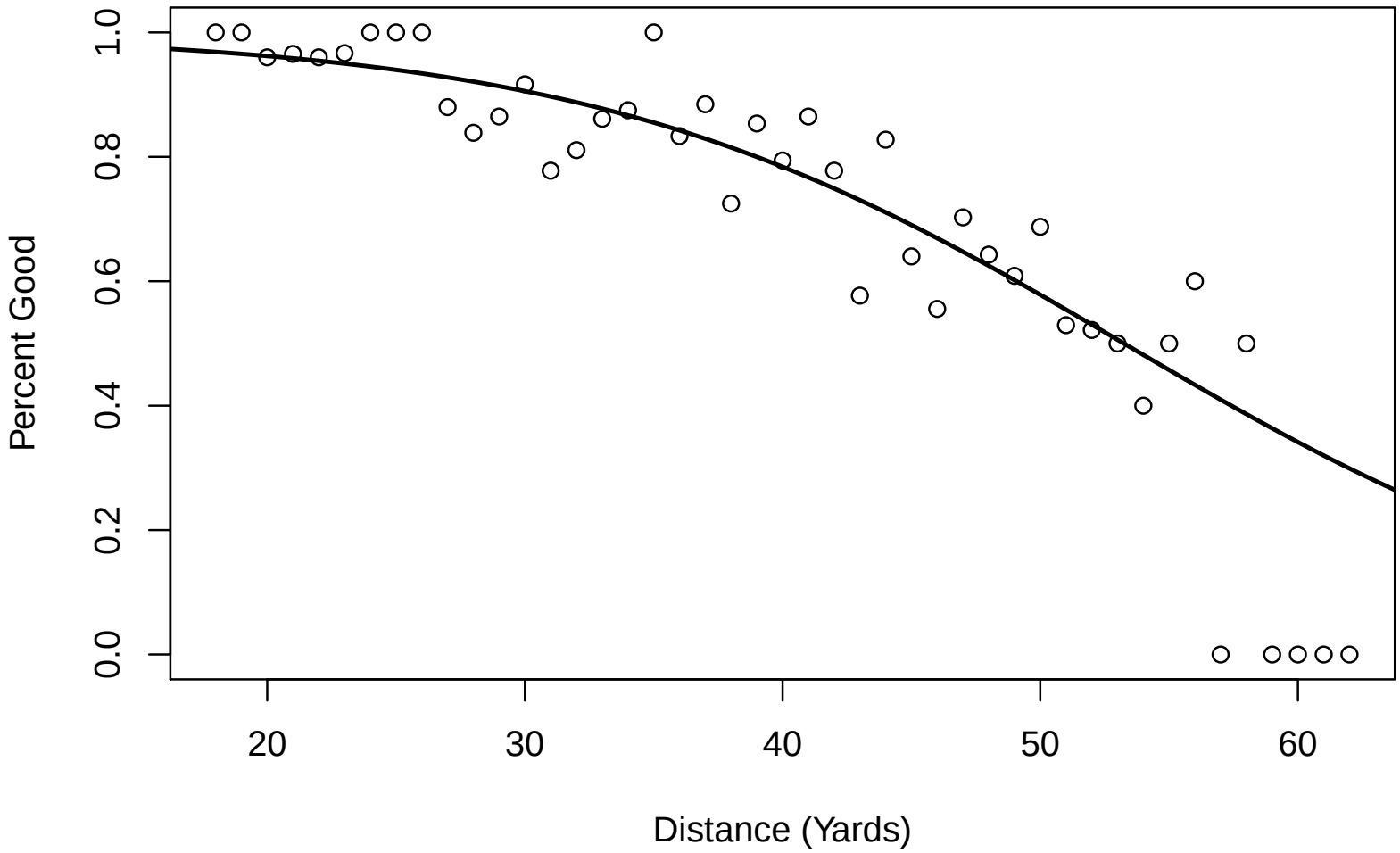
```
> fglr <- transform(fg,
+       good=as.factor(ifelse(play.type=="FG good",
+         "good", "bad")))
> fgt <- table(fglr$good, fglr$yards)
> plot(colnames(fgt), fgt["good",]/
+    (fgt["good",]+fgt["bad",]),
+    xlab="Distance (Yards)", ylab="Percent Good")
```



- Ideal for modeling with logistic regression, which in R is a `glm()` with `family="binomial"`.

```
> fglogit <- glm(good~yards, data=fglr,
+                family="binomial")
> summary(fglogit)

Call:
glm(formula=good~yards, family="binomial", data=fglr)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  5.178856   0.416201  12.443   <2e-16 ***
yards       -0.097261   0.009892  -9.832   <2e-16 ***

    Null deviance: 978.90  on 981  degrees of freedom
Residual deviance: 861.22  on 980  degrees of freedom
AIC: 865.22

Number of Fisher Scoring iterations: 5
```

```
> xx <- data.frame(yards=seq(15,65,length=1000))
> p <- predict(fglogit, newdata=xx, type="response")
> lines(xx[,1], p, lwd=2)
```



Extensions.

- You can use `step()` with `glm()`.
- For > 2 classes, try `multinom` which uses neural networks.
- The `textir` package has a function called `mnlm` which does L_1 penalized multinomial regression.
- The `glmnet` package has similar functionality but requires more computation for CV.
- The `reglogit` package offers a Bayesian lasso logistic regression.
- For ordered categories try `plor()` from the `MASS` library.

Discriminant analysis

Another approach is to try separate observations by parameterizing a (possibly high dimensional) $(p-1)$-dimensional function through the p-dimensional predictor space.

- Linear discriminant analysis (`lda()`) finds the best linear combination of features, constructing a separating hyperplane.
- Quadratic discriminant analysis (`qda()`) uses quadratic functions.
- Flexible discriminant analysis (`fda()` in the `mda` package) replaces the linear regressions of LDA by flexible non-parametric ones.

Primarily, DA is tailored to two-class data, although there are extensions.

To illustrate, consider the Spambase data set from HP Labs.

- It contains 58 attributes for 4601 emails, 1813 of which are spam.
- Lets read in the data and create training and validation sets so we can see how well the methods do at classifying spam.

```
> spam <- read.csv("spam.csv")
> testi <- sample(1:nrow(spam), 1000)
> train <- spam[-testi,]
> test <- spam[testi,]
```

```
> spam.lda <- lda(spam~., data=train)
> p.lda <- predict(spam.lda, newdata=test)$class
> table(actual=test$spam, lda=p.lda)
      lda
actual   0   1
     0 580  22
     1  86 312

> spam.qda <- qda(spam~., data=train)
> p.qda <- predict(spam.qda, newdata=test)$class
> table(actual=test$spam, qda=p.qda)
      qda
actual   0   1
     0 452 150
     1  14 384
```

```
> library(mda)
> spam.fda <- fda(spam~., data=train)
> p.fda <- predict(spam.fda, newdata=test, type="class")
> table(actual=test$spam, qda=p.fda)
      qda
actual   0   1
     0 580  22
     1  86 312
```

- `mda` also provides a similar technique called mixture discriminant analysis.

Trees

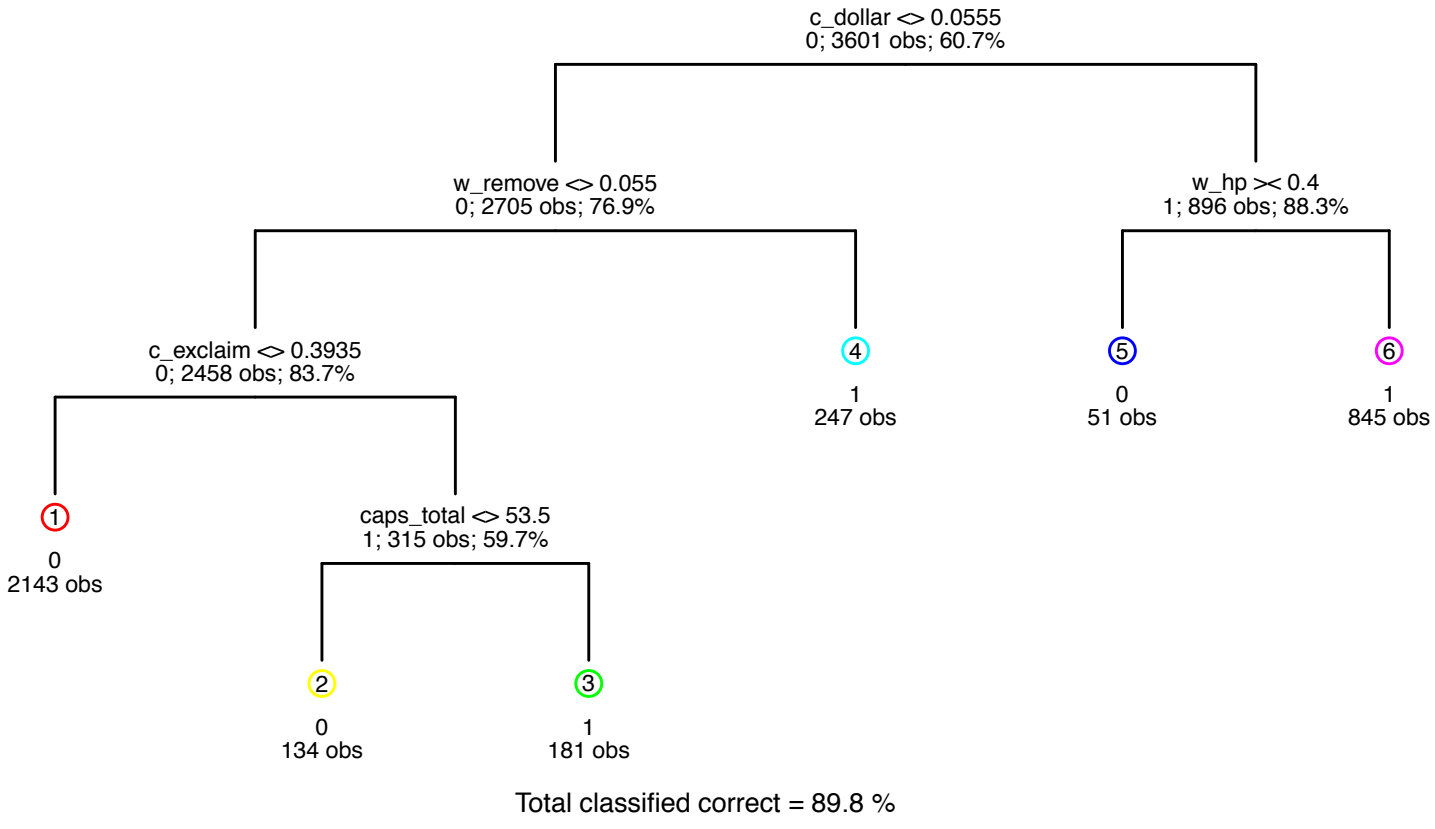
Tree models are great for classification (although they are also used for regression).

- They are often applauded for their interpretive aspects,
- but modern methods are also highly accurate.

The CART framework builds up the tree iteratively.

- Partition the data (recursively) along an input direction, x_j, chosen via an information criterion.
- Repeat until no further splits are possible.
- Prune splits subject to an out-of-sample validation scheme (e.g., CV).

```
> library(rpart)
> spam.rp <- rpart(spam~., data=train)
> train <- transform(train, spam=as.factor(spam))
> library(maptree)
> draw.tree(spam.rp, cex=0.5, nodeinfo=TRUE)
```



Total classified correct = 89.8 %

Other popular tree methods:

- Random Forests in the `randomForest` package: small trees with bagging
- Bayesian version via BART in `BayesTree`.
- Online classification via Dynamic Trees in `dynaTree`.

Other machine learning approaches:

- Nearest Neighbor via `knn()`.
- Neural Networks via the `nnet` package.
- Support Vector Machines (SVMs) via `e1071`.
- ...

Most support regression as well as classification.