Linear Modeling in R

William John Holden

26 April 2021

Abstract

These are my notes from CSE 635-50, Data Mining with Linear Models, which I took at the University of Louisville in Spring 2021 with Dr. Ahmed Desoky.

Preparing the data

Read in a data frame using `read.table`. If the file has a header, specify `read.table(filename, header = TRUE)`. If the data are separated by commas, use `read.csv`.

```
peppers = read.table("peppers.txt", header = TRUE)
```

You can peak inside the "structure" of the resulting data frame, use the `str` function.

```
str(airquality)
```

```
## 'data.frame':    153 obs. of  6 variables:
##  $ Ozone  : int  41 36 12 18 NA 28 23 19 8 NA ...
##  $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
##  $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
##  $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
##  $ Month  : int  5 5 5 5 5 5 5 5 5 5 ...
##  $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

Moments

The four "moments" are mean, variance, skewness, and kurtosis.

Moment	Name	Meaning
1	Mean	$E(x)$
2	Variance	$E(x-\mu)^2$
3	Skewness	$E(x-\mu)^3$
4	Kurtosis	$E(x-\mu)^4$

Skewness and kurtosis describe the shape of a distribution. Skewness indicates whether data biases towards to one side or the other; a negative skewness means the tail points left, a positive skewness means the tail points right. Kurtosis tells us about the "peakiness" of the data around the mean and the "fatness" of the tails. The standard normal has a kurtosis of 3. A distribution with a taller peak than the standard normal has a kurtosis greater than 3. A distribution with a shorter peak than the standard normal has a kurtosis less than 3.

```
library(moments)
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

```
var(mtcars$mpg)
```

```
## [1] 36.3241
```

```
sd(mtcars$mpg)
```

```
## [1] 6.026948
```

```
skewness(mtcars$mpg)
```

```
## [1] 0.6404399
```

```
kurtosis(mtcars$mpg)
```

```
## [1] 2.799467
```

Mean

$\mu = E(x)$, where x is the population random variable. The sample mean of size n is computed in the same way as the population.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} = m_1$$

Variance

Variance is defined as the sum of the squares of differences in the random variable and mean.

$$\sigma^2 = E(x-\mu)^2 = E(x^2 - 2\mu x + \mu^2) = E(x^2) - 2\mu E(x) + \mu^2 = E(x^2) - 2\mu^2 + mu^2 = E(x^2) - \mu^2$$

For sample variance we lose a degree of freedom.

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n-1} = \frac{\sum_{i=1}^{n} x_i^2}{n-1} - \bar{x}^2 = m_2$$

Skewness

There is no standard symbol for the third moment, m_3. $m_3 = E(x-\mu)^3$ and skewness is m_3/σ^3.

Kurtosis

There is also no standard symbol for the fourth moment, m_4. $m_4 = E(x-\mu)^4$ and kurtosis is m_4/σ^4.

Summary statistics

R has a convenience function `summary` to compute several summary statistics at once.

```
summary(mtcars)
```

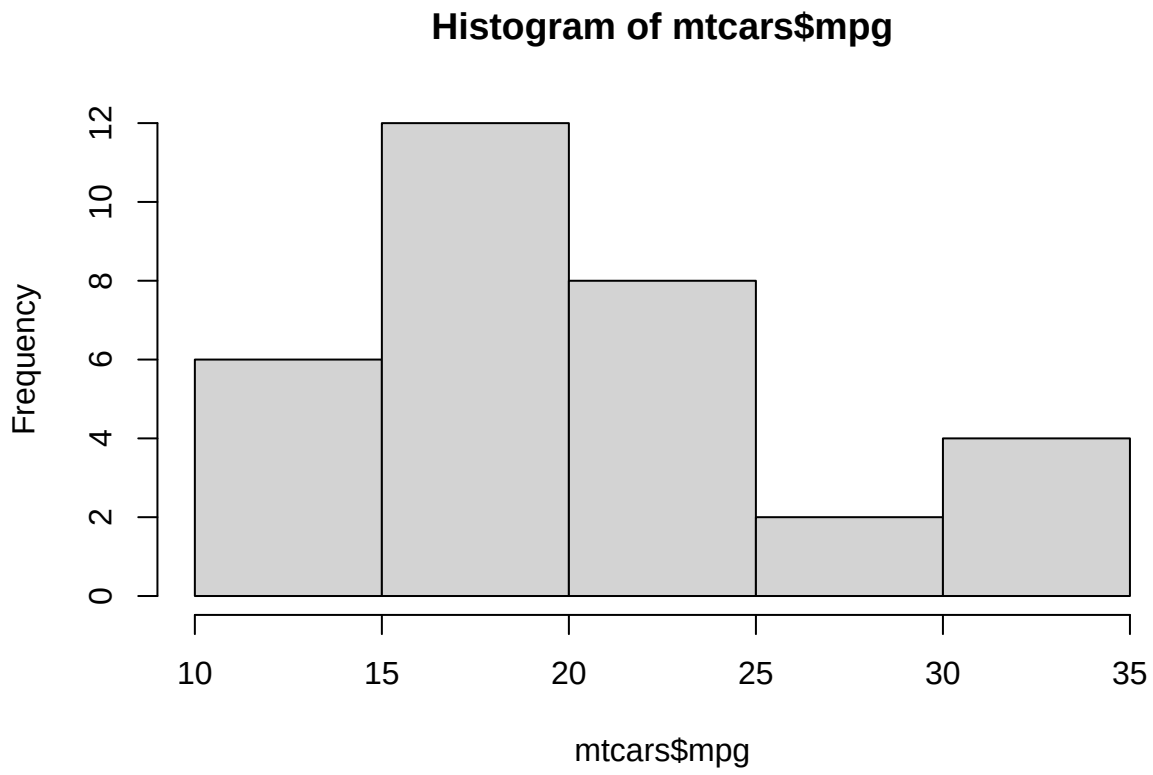
```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
```

```
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0
##       drat             wt             qsec             vs
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000
##        am              gear            carb
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000
##  Median :0.0000   Median :4.000   Median :2.000
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

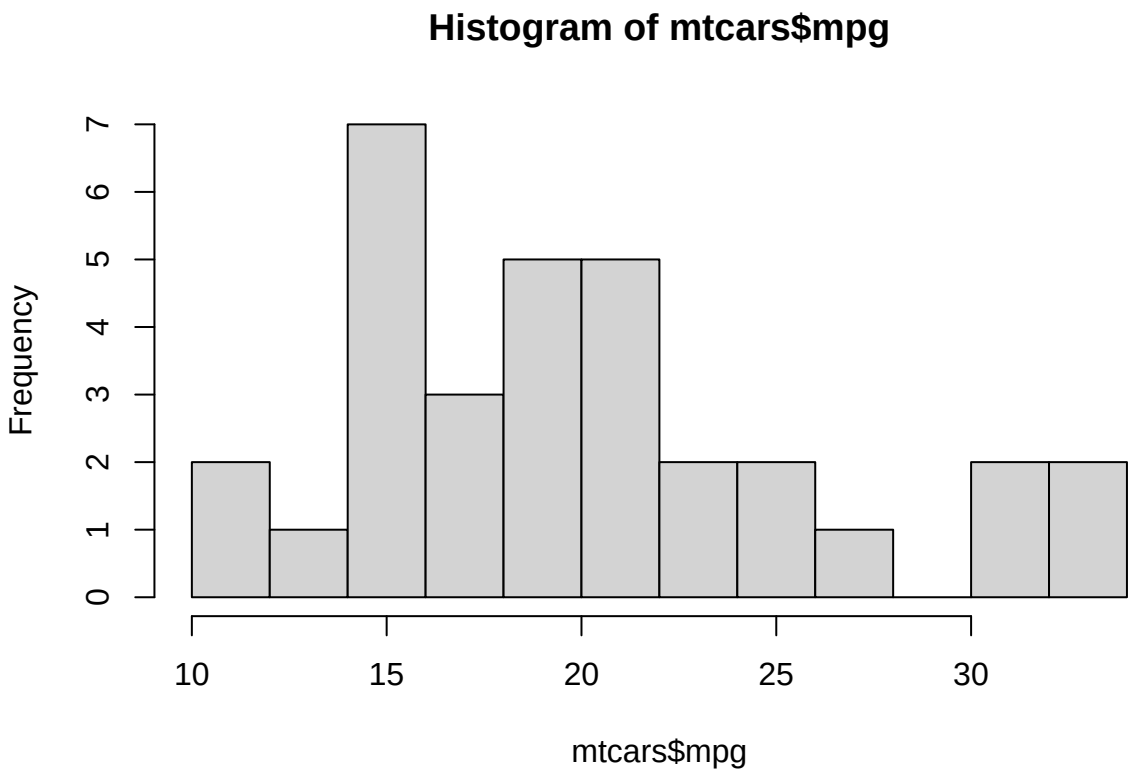
Histogram

An interesting use of the `hist` function is to tally the number of occurrences of values in a specified number of "breaks". If you don't want to actually show the histogram on your screen or in your document, then set the parameter `plot = FALSE`.

```
hist(mtcars$mpg)
```




```
hist(mtcars$mpg, breaks = 16)
```



`NA` Values

R defines a special value NA for empty values. This symbol is not quite the same as `NaN` and is very different from `NULL` You may see NA values from importing data from a file where the data is not complete.

To quickly drop all rows in a data frame containing NA, use the `na.omit` function.

```
nrow(airquality)
```

```
## [1] 153
```

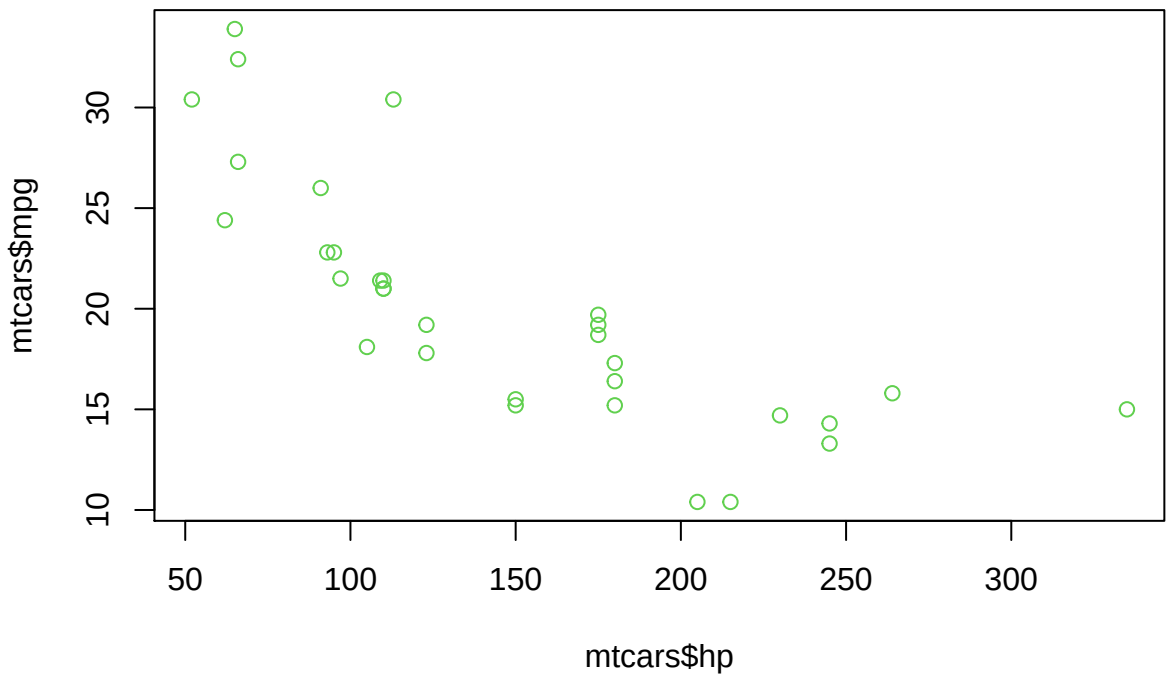
```
nrow(na.omit(airquality))
```

```
## [1] 111
```

Scatter Plot

A *formula* is a very common idiom in R. A formula has the form `y ~ x1 + x2 + x3 + \ldots + xn` where `y` is the response (dependent) variable and the `x`'s are free (independent) variables.

```
plot(mtcars$mpg ~ mtcars$hp, col = 3)
```



Linear Model

A linear model has the form

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_n X_n + \varepsilon$$

where X is a vector containing n independent variables, Y is the response variable, β is a vector of coefficients used in the model, and ε is the error. B is computed using

$$\beta = (X^T X)^{-1} X^T Y.$$

Note that each row of X is an observation, each column of X is an independent variable (X_1, X_2, and so on), and X should contain a column of ones. The column of ones is for the y-intercept.

```
y = c(75, 72, 99, 46, 80, 99, 10, 82)
x1 = c(23, 44, 57, 70, 58, 23, 71, 73)
x = cbind(rep(1, length(x1)), x1)
beta = solve(t(x) %*% x) %*% t(x) %*% y
beta
```

```
##           [,1]
##    108.2131742
## x1  -0.7224472
```

R provides the `lm` function to compute the linear model easily:

```
m1 = lm(y ~ x1)
summary(m1)
```

```
##
## Call:
## lm(formula = y ~ x1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -46.919 -12.881   1.489  16.898  31.966
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 108.2132    28.5146   3.795  0.00902 **
## x1           -0.7224     0.5113  -1.413  0.20742
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 27.69 on 6 degrees of freedom
## Multiple R-squared:  0.2496, Adjusted R-squared:  0.1246
## F-statistic: 1.996 on 1 and 6 DF,  p-value: 0.2074
```

Observe that the coefficients in the linear model are identical to those in β.

```
coef(m1)
```

```
## (Intercept)          x1
## 108.2131742  -0.7224472
```

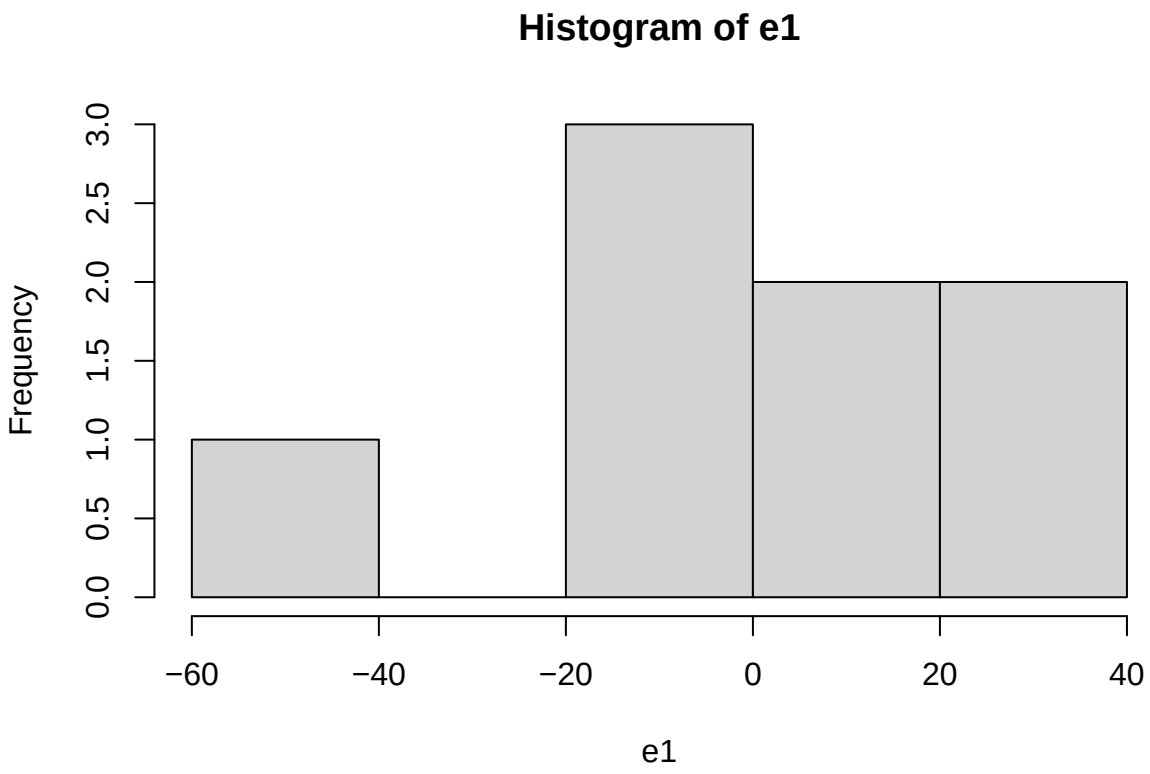
Sum of the Square of Errors (SSE)

Model accuracy is assessed by the sum of the square of error values (ε). First, subtract the predicted value value from the actual value.

```
p1 = predict(m1)
e1 = y - p1
summary(e1)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -46.919 -12.881   1.489   0.000  16.898  31.966
```

```
hist(e1)
```



Next, square each error value and compute their sum. The reason for squaring each error value is to prevent errors less than zero from canceling errors greater than zero. You can be a little bit clever with this calculation with vectorized operations, dot products, and cross products.

```
sum(e1^2)
```

```
## [1] 4599.641
```

```
sum(e1 * e1)
```

```
## [1] 4599.641
```

```
t(e1) %*% e1
```

```
##          [,1]
## [1,] 4599.641
```

In general, a lower SSE is better. The formula for β is derived from minimizing SSE. The "line of best fit" cannot be improved; its coefficients will be basically identical when calculated by any software on any computing machine. However, a non-linear model may give a better estimate.

Degree Two (Quadratic) Polynomial Model

You can still use `lm` for polynomials. Think of this like the chain rule: to compute $y = Ax^2$, first let $z = x^2$ and then compute $y = Az$. For the basis of comparison, I'll show again a degree one (linear) model, then a quadratic model.

```
# linear model
m2 = lm(mtcars$hp ~ mtcars$disp)
```

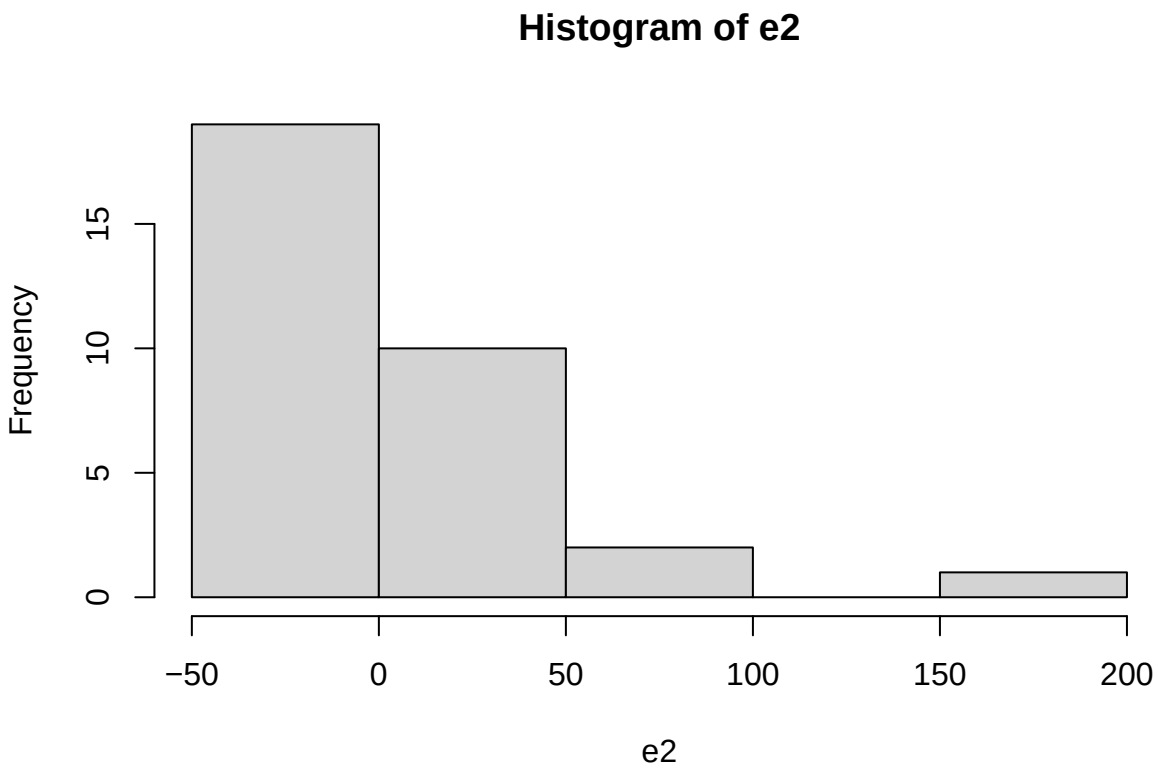
```r
summary(m2)
```

```
##
## Call:
## lm(formula = mtcars$hp ~ mtcars$disp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -48.623 -28.378  -6.558  13.588 157.562
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  45.7345    16.1289   2.836  0.00811 **
## mtcars$disp   0.4375     0.0618   7.080 7.14e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 42.65 on 30 degrees of freedom
## Multiple R-squared:  0.6256, Adjusted R-squared:  0.6131
## F-statistic: 50.13 on 1 and 30 DF,  p-value: 7.143e-08
```

```r
p2 = predict(m2)
e2 = mtcars$hp - p2
sum(e2^2)
```

```
## [1] 54560.19
```

```r
hist(e2)
```



Now for the quadratic model.

```
# quadratic model
disp.sq = mtcars$disp^2
m3 = lm(mtcars$hp ~ disp.sq)
summary(m3)
```

```
##
## Call:
## lm(formula = mtcars$hp ~ disp.sq)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -62.91 -32.17 -11.55  16.23 170.69
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 9.330e+01  1.218e+01   7.660 1.52e-08 ***
## disp.sq     7.837e-04  1.307e-04   5.995 1.42e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 47.01 on 30 degrees of freedom
## Multiple R-squared:  0.545,  Adjusted R-squared:  0.5298
## F-statistic: 35.94 on 1 and 30 DF,  p-value: 1.416e-06
```

```r
p3 = predict(m3)
# just for fun, reconstruct the same predictions directly using coefficients
c3 = coef(m3)
head(cbind(p3, c3[1] + c3[2] * disp.sq))
```

```
##         p3
## 1 113.3685 113.3685
## 2 113.3685 113.3685
## 3 102.4464 102.4464
## 4 145.4732 145.4732
## 5 194.8765 194.8765
## 6 132.9813 132.9813
```

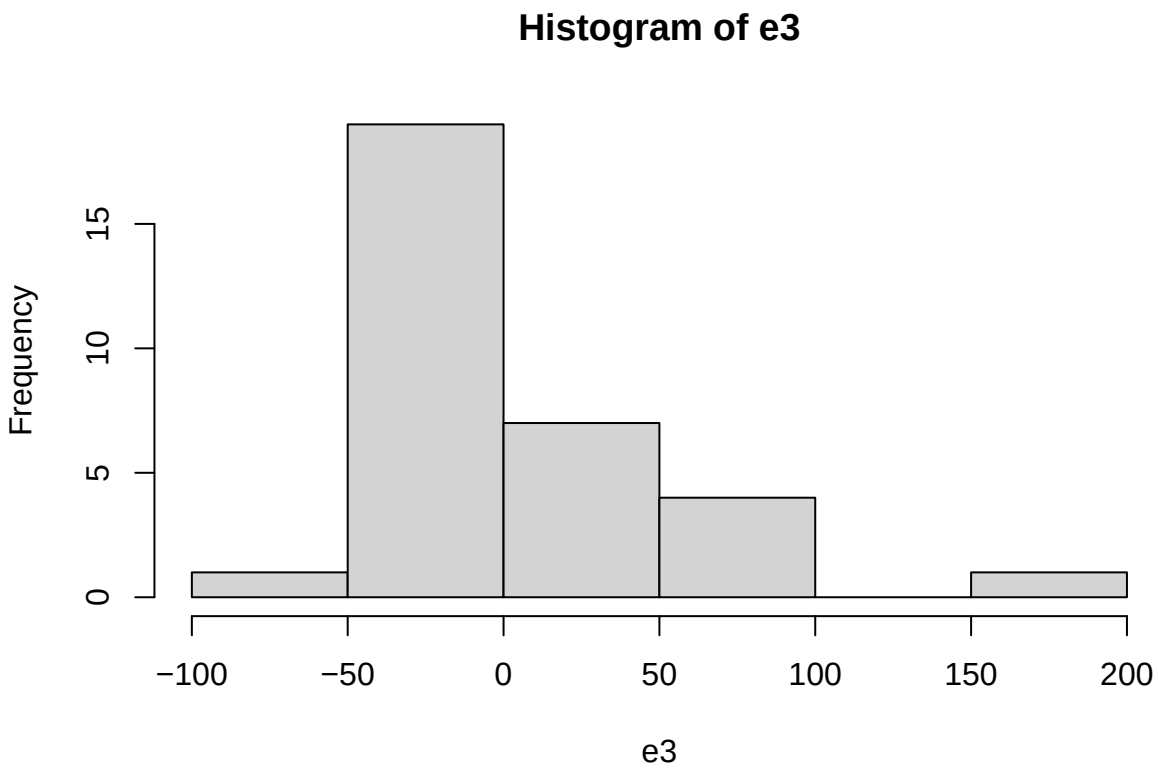
```r
e3 = mtcars$hp - p3
# we can also access the residuals from the model
head(cbind(e3, residuals(m3)))
```

```
##           e3
## 1  -3.368464  -3.368464
## 2  -3.368464  -3.368464
## 3  -9.446389  -9.446389
## 4 -35.473220 -35.473220
## 5 -19.876486 -19.876486
## 6 -27.981332 -27.981332
```

```r
sum(e3^2)
```

```
## [1] 66304.62
```

```r
hist(e3)
```



The sum of the squares of errors is higher in the quadratic model than the linear model (6.6304625×10^4 versus 5.4560191×10^4). We should also observe that the coefficient of determination, R^2, is lower for the quadratic model (0.5450076 versus 0.6255997).

You can make these polynomials as complicated as you want. For example, `lm(mtcars$hp ~ disp.sq + mtcars$disp)` would have worked.

Generalized Linear Model

The Generalized Linear Model (GLM) allows us to model variables as an exponential function. To compute predictions from a GLM, raise e to the power of the linear combination of coefficients fitted in the model.

```
m4 = glm(formula = hp ~ disp, data = mtcars, family = "poisson")
summary(m4)
```

```
##
## Call:
## glm(formula = hp ~ disp, family = "poisson", data = mtcars)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.8484  -2.5840  -0.6889   1.8185  11.2984
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 4.2667948  0.0351149  121.51   <2e-16 ***
## disp        0.0028553  0.0001161   24.59   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 958.27  on 31  degrees of freedom
## Residual deviance: 357.39  on 30  degrees of freedom
## AIC: 576.47
##
## Number of Fisher Scoring iterations: 4
```

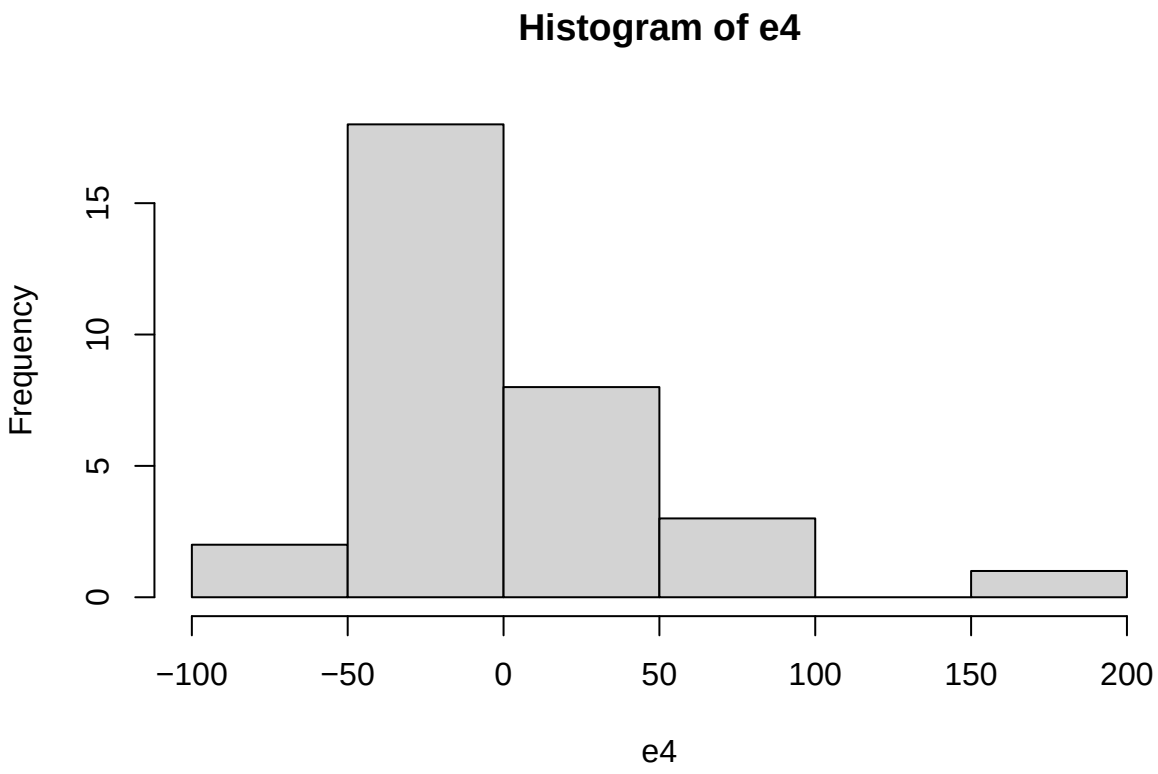
```
c4 = coef(m4)
# note that you have to use the exponential function
p4 = exp(predict(m4))
head(cbind(p4, exp(c4[1] + c4[2] * mtcars$disp)))
```

```
##                           p4
## Mazda RX4         112.57728 112.57728
## Mazda RX4 Wag     112.57728 112.57728
## Datsun 710         97.04407  97.04407
## Hornet 4 Drive    148.92720 148.92720
## Hornet Sportabout 199.27711 199.27711
## Valiant           135.53545 135.53545
```

```
e4 = mtcars$hp - p4
# the SSE for the GLM is between that of the linear and quadratic models
sum(e4^2)
```

```
## [1] 63631.76
```

```
hist(e4)
```



Entropy

Entropy is a statistic to quantify the amount of information in a random process $p(x)$. All values in $p = (p_1, p_2, \dots, p_n)$ are probabilities and their sum is $\sum p = 1$. The entropy function is defined as:

$$\text{Entropy}\,(X) = -\sum_{i=1}^{n} p_i \lg p_i$$

Note that lg is the base-two logarithm, $\log_2$.

R function

Here is a general-purpose function to compute entropy.

```
entropy <- function(vec, breaks = length(vec)) {
  h = hist(vec, breaks = breaks, plot = FALSE)
  n = sum(h$counts)
  stopifnot(n == length(vec))
  p = h$counts / n
  stopifnot(sum(p) == 1)
  q = p[p > 0]
  return(sum(-q * log2(q)))
}
```

Special case: constant process

The entropy of a constant process $p = (0, \dots, 0, 1, 0, \dots, 0)$ is 0.

```
entropy(rep(1, 1000))
```

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

Special case: uniform process

The entropy of a uniform process $p = (1/n, 1/n, 1/n, \dots, 1/n)$ is $\lg n$.

```
entropy(1:1000)
```

```
## [1] 9.963784
```

```
log2(1000)
```

```
## [1] 9.965784
```

General case

$$0 \leq \text{Entropy} \leq \lg n$$

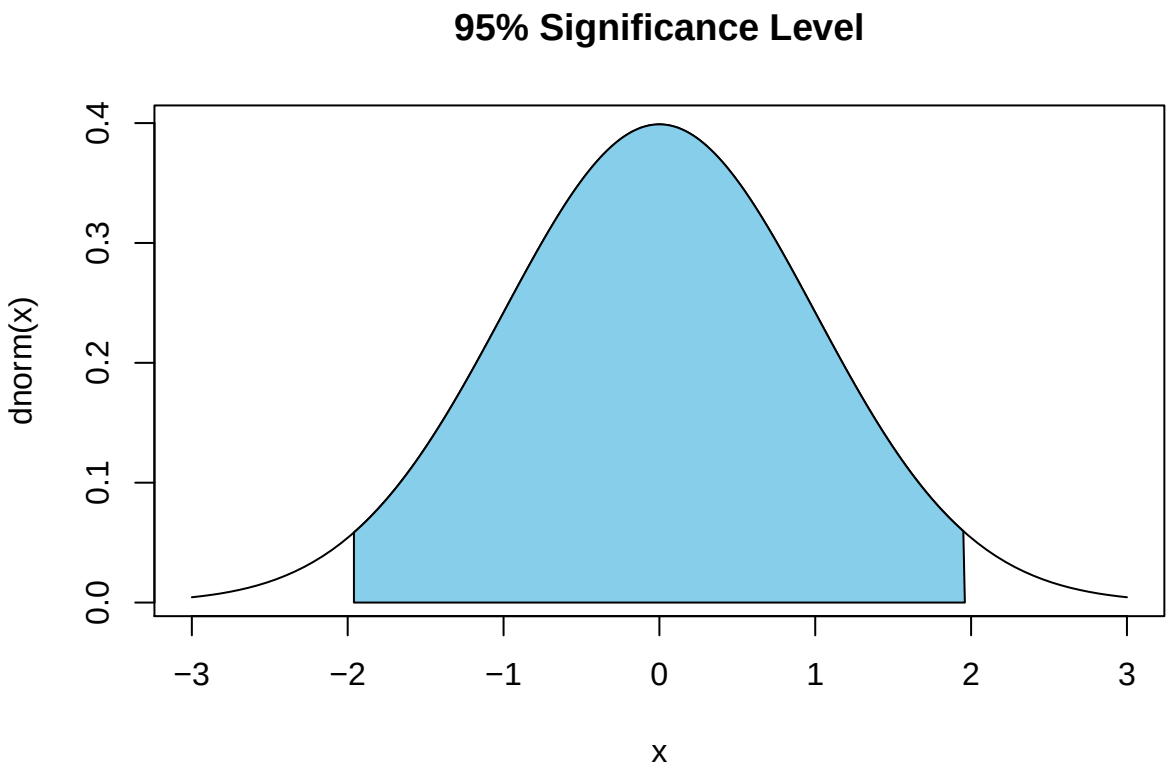
```
entropy(rchisq(1000, df = 4))
```

```
## [1] 8.394162
```

Hypothesis Testing

In hypothesis testing, we have a null hypothesis H_0 and an alternative hypothesis H_1. Prior to the experiment, we choose a level of significance α that will influence the level of certainty needed to reject or fail to reject the H_0. α is typically 0.05 or 5%.

```
left = qnorm((1 - .95)/2)
right = qnorm((1 + .95)/2)
cord.x <- c(left,seq(left,right,0.01),right)
cord.y <- c(0,dnorm(seq(left,right,0.01)),0)
curve(dnorm(x),xlim=c(-3,3),main='95% Significance Level')
polygon(cord.x,cord.y,col='skyblue')
```



The shaded area in the above plot shows the events that are not considered significant. Events that fall outside of the shaded area meet the significance level where we reject H_0.

A typical null hypothesis might be that the population mean is zero.

$$H_0 : \mu = 0$$

The alternate hypothesis for this test might be that the population mean is nonzero.

$$H_1 : \mu \neq 0$$

If the sample is large ($n > 30$), then we can use the z-Test.

$$z = \frac{\bar{x} - v}{\sigma / \sqrt{n}}$$

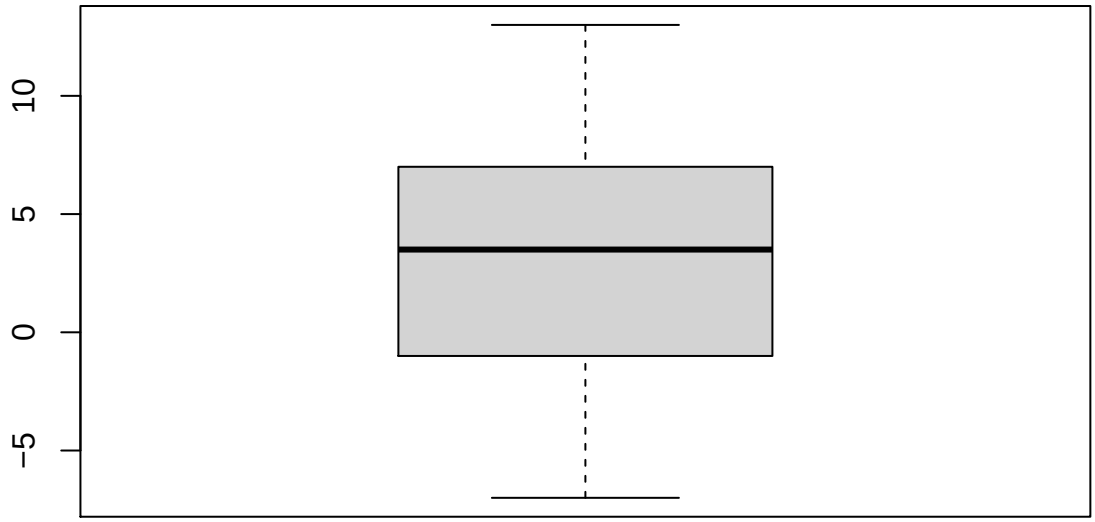
If the sample is small ($n \leq 30$), then we should use the t-Test.

Student t-Test

The t-Test uses both a test value $t = z$ and a critical value. The critical value depends on the number of degrees of freedom, $df = n - 1$.

One-Sample t-Test

```
# always look at the box plot when doing t-tests.
boxplot(peppers$angle)
```




```
mean(peppers$angle) / (sd(peppers$angle) / sqrt(length(peppers$angle)))
```

```
## [1] 3.174151
```

```
t.test(peppers$angle, mu = 0, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  peppers$angle
## t = 3.1742, df = 27, p-value = 0.003733
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  1.123883 5.233259
## sample estimates:
## mean of x
##  3.178571
```

Interpret this result as, "the probability that the population mean $\mu = 0$, given the size, average, and standard deviation of our sample, is only 0.3733366%." This is a low probability less than $\alpha = 0.05$, so we **reject** H_0 and instead accept H_1.

Observe also the confidence interval. The confidence interval (1.1238834, 5.2332594) does not contain 0,

which further indicates that $\mu \neq 0$.

Paired Samples t-Test

The *paired observations* t-Test is an easy way to compare samples. The idea is to subtract one sample from the other to reduce the problem from a two variables to one variable.

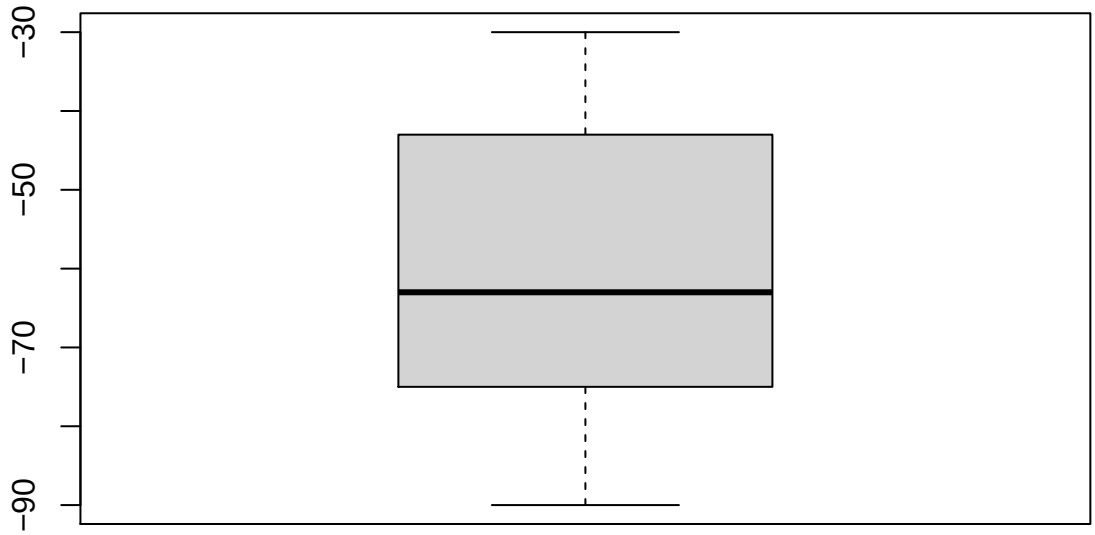
$$H_0 : \mu_X = \mu_Y \rightarrow \mu_X - \mu_Y = 0$$

$$H_1 : \mu_X \neq \mu_Y \rightarrow \mu_X - \mu_Y \neq 0$$

The t-statistic is

$$t = \frac{\bar{x} - \bar{y}}{\sqrt{\sigma^2_{x-y}/n}}.$$

```
x = c(30, 20, 60, 80, 40, 50, 60, 30, 70, 60)
y = c(73, 50, 128, 170, 87, 108, 135, 69, 148, 132)
boxplot(x - y)
```




```
(mean(x) - mean(y)) / sqrt(var(x - y) / length(x))
```

```
## [1] -9.67686
```

```
t.test(x - y, mu = 0, conf.level = 0.95)
```

```
##
##  One Sample t-test
##
## data:  x - y
## t = -9.6769, df = 9, p-value = 4.7e-06
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##  -74.02619 -45.97381
## sample estimates:
## mean of x
##       -60
```

Rejecting H_0 is quite obvious in this case. The p-value is much less than 0.05 and the confidence does not contain 0.

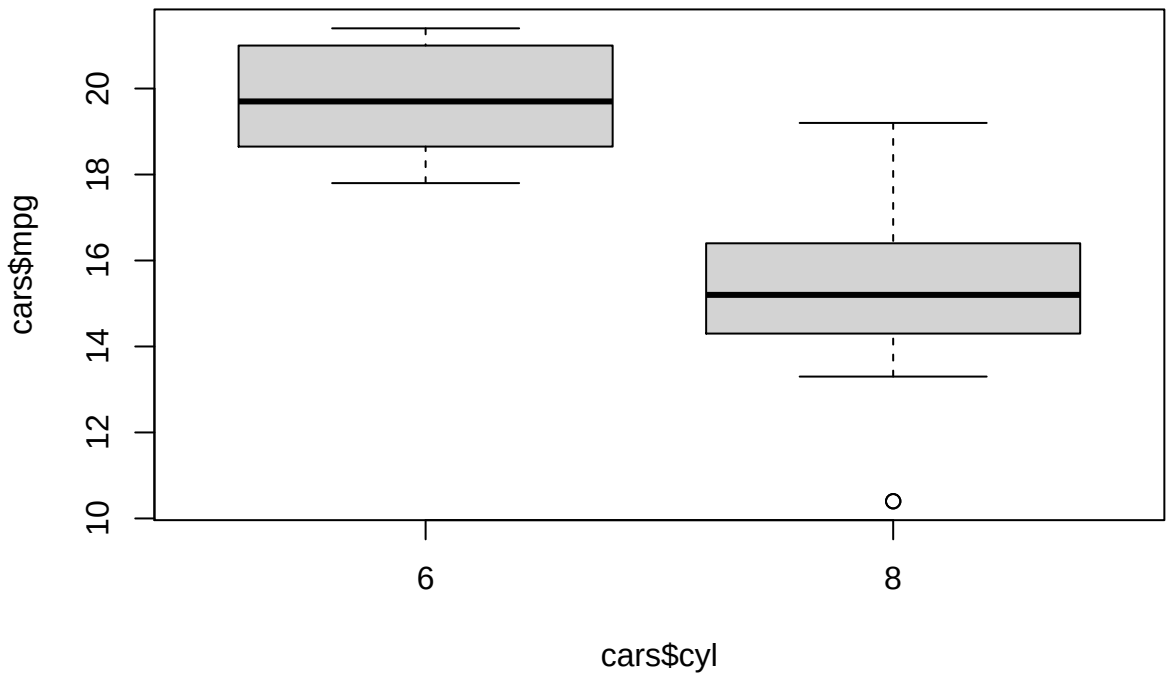
Unbalanced Two-Sample t-Test

The t-Test can also be used on *unbalanced* data sets where the size of the two samples are unequal.

```
cars = mtcars[mtcars$cyl > 4, ]
cars$cyl = as.factor(cars$cyl)
summary(cars$cyl)
```

```
##  6  8
##  7 14
```

```
boxplot(cars$mpg ~ cars$cyl)
```




```
t.test(mpg ~ cyl, data = cars, mu = 0, alt = "two.sided", var.eq = F, paired = F)
```

```
##
##  Welch Two Sample t-test
##
## data:  mpg by cyl
## t = 5.2911, df = 18.502, p-value = 4.54e-05
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  2.802925 6.482789
## sample estimates:
## mean in group 6 mean in group 8
##        19.74286        15.10000
```

ANOVA

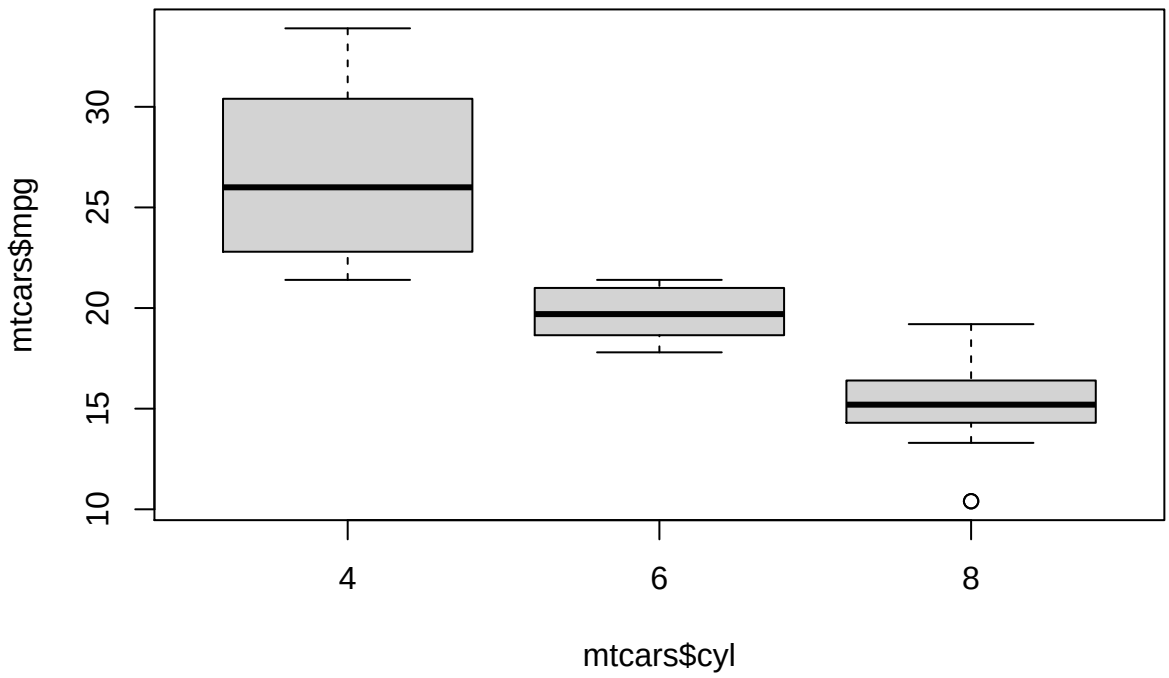
To compare the means of more than two samples, we need the Analysis of Variance (ANOVA) test.

$$H_0 : \mu(1) = \mu(2) = ... = \mu(k)$$

The alternative hypothesis H_1 is that at least one mean differs from the others.

The ANOVA test extracts the F-statistic from each sample as the basis for comparison.

```
boxplot(mtcars$mpg ~ mtcars$cyl)
```




```
# aov will give a different result if the free variable is left as a numeric value
m5 = aov(mtcars$mpg ~ as.factor(mtcars$cyl))
summary(m5)
```

```
##                       Df Sum Sq Mean Sq F value   Pr(>F)
## as.factor(mtcars$cyl)  2  824.8   412.4    39.7 4.98e-09 ***
## Residuals             29  301.3    10.4
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
m5$coef
```

```
##            (Intercept) as.factor(mtcars$cyl)6 as.factor(mtcars$cyl)8
##              26.663636              -6.920779             -11.563636
```

Interpret this result as, "the probability that the population mean efficiency for cars of 4, 6, and 8 cylinders are all equal, given this sample, is less than α."

The Tukey Honest Significant Differences function computes confidence intervals in the triangle of pairwise combinations of categories.

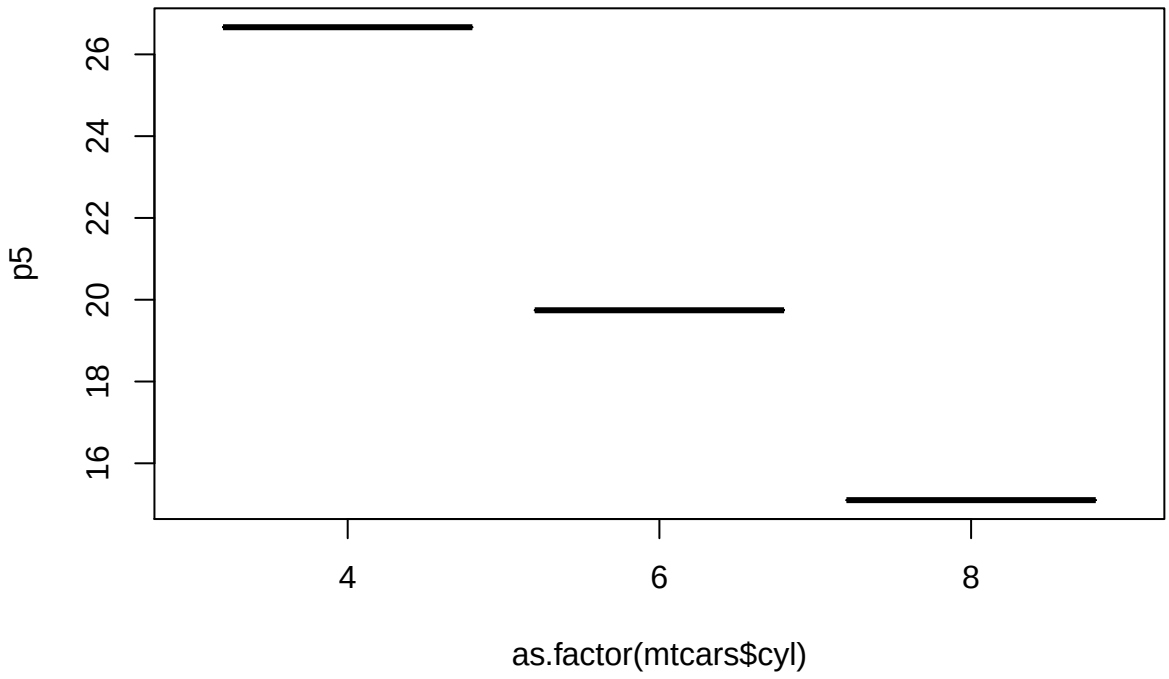
```
TukeyHSD(m5)
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = mtcars$mpg ~ as.factor(mtcars$cyl))
##
## $`as.factor(mtcars$cyl)`
```

```
##           diff        lwr        upr     p adj
## 6-4  -6.920779 -10.769350 -3.0722086 0.0003424
## 8-4 -11.563636 -14.770779 -8.3564942 0.0000000
## 8-6  -4.642857  -8.327583 -0.9581313 0.0112287
```

The model computed by the `aov` function can be used to form predictions.

```
p5 = predict(m5)
boxplot(p5 ~ as.factor(mtcars$cyl))
```




```
table(p5)
```

```
## p5
##             15.1 19.7428571428571 26.6636363636364
##               14                7               11
```

GLM will form the same predictions.

```
m6 = glm(mtcars$mpg ~ as.factor(mtcars$cyl))
summary(m6)
```

```
##
## Call:
## glm(formula = mtcars$mpg ~ as.factor(mtcars$cyl))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.2636  -1.8357   0.0286   1.3893   7.2364
##
```

```
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              26.6636     0.9718  27.437  < 2e-16 ***
## as.factor(mtcars$cyl)6  -6.9208     1.5583  -4.441 0.000119 ***
## as.factor(mtcars$cyl)8 -11.5636     1.2986  -8.905 8.57e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 10.38837)
##
##     Null deviance: 1126.05  on 31  degrees of freedom
## Residual deviance:  301.26  on 29  degrees of freedom
## AIC: 170.56
##
## Number of Fisher Scoring iterations: 2
```

```
p6 = predict(m6)
table(p6)
```

```
## p6
##             15.1 19.7428571428571 26.6636363636364
##               14                7               11
```

The aov function can be used for two-way experiments where there are two categorical independent variables. This is useful for randomized block design experiments.

```
summary(aov(data = mtcars, formula = mpg ~ factor(cyl) + factor(gear) + factor(am)))
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## factor(cyl)   2  824.8   412.4  41.598 7.91e-09 ***
## factor(gear)  2    8.3     4.1   0.416   0.6639
## factor(am)    1   35.3    35.3   3.556   0.0706 .
## Residuals    26  257.8     9.9
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The significance codes on the right side of the summary indicate how significant each independent is. Interpret this as, "the number of cylinders is extremely significant, having a manual or automatic transmission is weakly significant, and the number of forward gears is not at all significant for the efficiency of the car."

Factors

Many data sets contain numeric values that should be interpreted as categorical data. R calls these values "factors."

```
table(as.factor(mtcars$cyl))
```

```
##
##  4  6  8
## 11  7 14
```

Binarizing Data

The following data set contains pass/fail values that are coded as zero and one.

```
hours = c(0.50,0.75,1.00,1.25,1.50,1.75,1.75,2.00,2.25,2.50,2.75,3.00,3.25,3.50,4.00,4.25,
          4.50,4.75,5.00,5.50)
pass=c(0,0,0,0,0,0,1,0,1,0,1,0,1,0,1,1,1,1,1,1)
```

We can estimate the value of pass using a simple linear model.

```
m7 = lm(pass ~ hours)
summary(m7)
```

```
##
## Call:
## lm(formula = pass ~ hours)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.66715 -0.21262 -0.02053  0.17157  0.74339
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.15394    0.18315  -0.840 0.411655
## hours        0.23460    0.05813   4.036 0.000775 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3819 on 18 degrees of freedom
## Multiple R-squared:  0.4751, Adjusted R-squared:  0.4459
## F-statistic: 16.29 on 1 and 18 DF,  p-value: 0.0007751
```

But the predictions are real numbers.

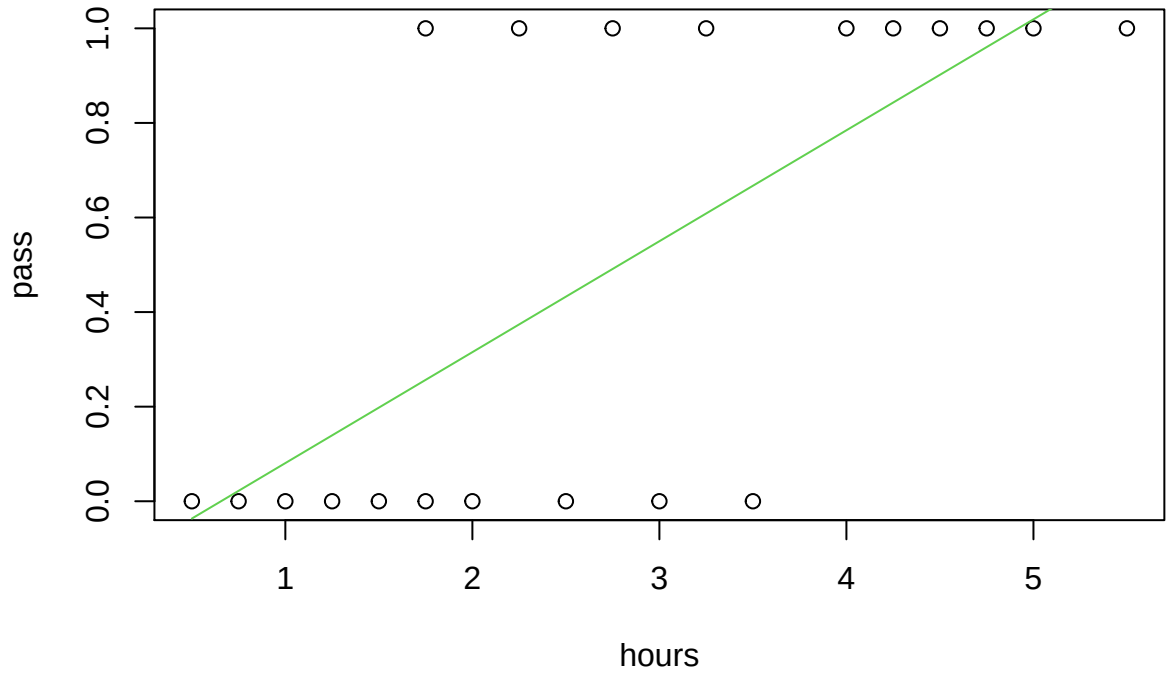
```
p7 = predict(m7)
```

We can use the `ifelse` function to clip these values to 0 and 1 around the predicted value 0.5.

```
ifelse(predict(m7) < 0.5, 0, 1)
```

```
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1  1
```

```
plot(pass ~ hours)
lines(p7 ~ hours, col = 3)
```



Confusion Matrix

We use the `table` function to assess the accuracy of a model.

```
t7 = table(ifelse(predict(m7) < 0.5, 0, 1), pass)
t7
```

```
##    pass
##     0 1
##   0 8 3
##   1 2 7
```

In this "confusion matrix," the values along the diagonal show accurate estimates.

```
acc7 = sum(diag(t7)) / sum(t7)
acc7
```

```
## [1] 0.75
```

The accuracy of this model is 75%. The probability of misclassification (PMC) is 25%.

Logit Model

The *logistic regression model* computes the probability $p = \Pr(y = 1|x)$ for $y \in \{0, 1\}$. Note that x may contain multiple variables, in which case ax is the dot product of the coefficient vector $\vec{a}$ and $\vec{x}$.

$$\text{logit}\,(y = 1|x) = \ln\left(\text{Odds}\,(y = 1|x)\right)$$

The logit model fits a linear model

$$\text{logit}\,(y=1|x) = ax+b$$

which means that

$$\text{Odds}\,(y=1|x) = e^{ax+b}$$

and therefore

$$\hat{p} = \Pr(y=1|x) = \frac{e^{ax+b}}{1+e^{ax+b}} = \frac{1}{1+e^{-ax-b}}.$$

The shape of this curve is called a sigmoid. In R, we fit a logit model with the `glm` function.

```
m8 = glm(pass ~ hours, family = "binomial"("logit"))
summary(m8)
```

```
##
## Call:
## glm(formula = pass ~ hours, family = binomial("logit"))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.70557  -0.57357  -0.04654   0.45470   1.82008
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -4.0777     1.7610  -2.316   0.0206 *
## hours         1.5046     0.6287   2.393   0.0167 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27.726  on 19  degrees of freedom
## Residual deviance: 16.060  on 18  degrees of freedom
## AIC: 20.06
##
## Number of Fisher Scoring iterations: 5
```

```
c8 = coef(m8)
p8 = 1 / (1 + exp(-c8[1] - c8[2] * hours))
head(cbind(p8, 1/(1+exp(-predict(m8)))))
```

```
##           p8
## 1 0.03471034 0.03471034
## 2 0.04977295 0.04977295
## 3 0.07089196 0.07089196
## 4 0.10002862 0.10002862
## 5 0.13934447 0.13934447
## 6 0.19083650 0.19083650
```

```
p8
```

```
##  [1] 0.03471034 0.04977295 0.07089196 0.10002862 0.13934447 0.19083650
##  [7] 0.19083650 0.25570318 0.33353024 0.42162653 0.51501086 0.60735865
## [13] 0.69261733 0.76648084 0.87444750 0.91027764 0.93662366 0.95561071
## [19] 0.96909707 0.98519444
```

These values should be interpreted as, "given x_i, the probability that $y = 1$ is $\hat{p}_i$."

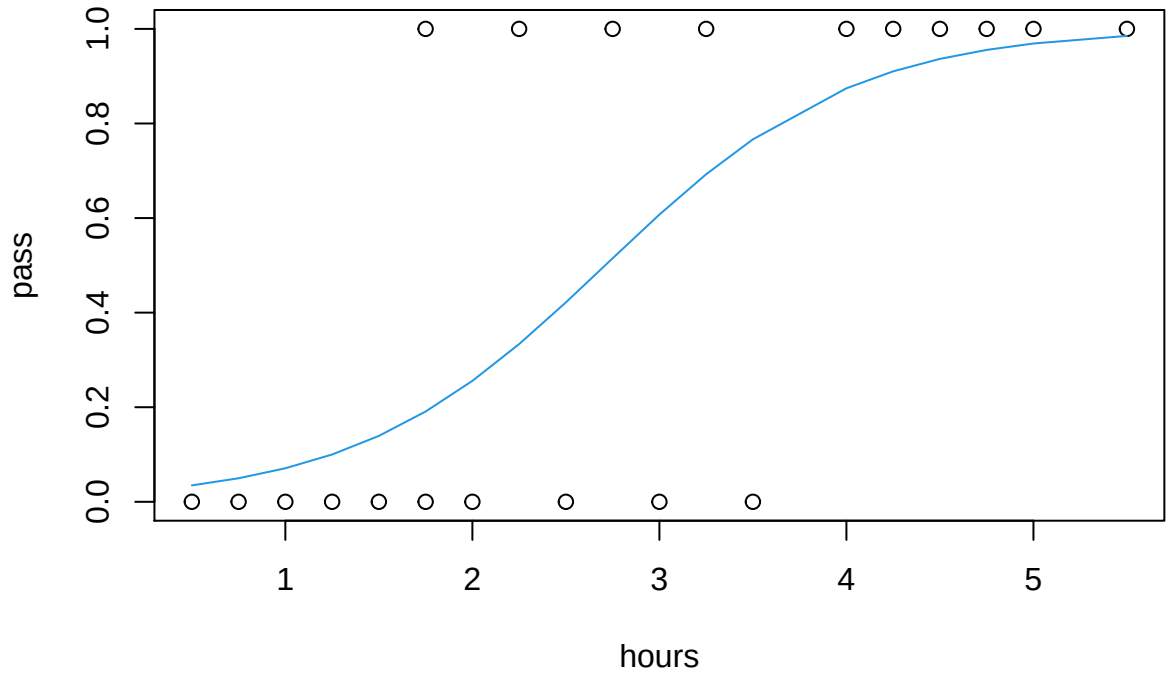
Binarize these predictions using `ifelse`.

```
t8 = table(ifelse(p8 < 0.5, 0, 1), pass)
acc8 = sum(diag(t8)) / sum(t8)
acc8
```

```
## [1] 0.8
```

This means that the logit model predicts passing scores with 80% accuracy.

```
plot(pass ~ hours)
lines(p8 ~ hours, col = 4)
```



Probit Model

The logistic regression (probit) transforms a binary y using the inverse of the standard normal distribution, Φ^{-1}.

$$\Pr\left(y = 1|x\right) = \Phi^{-1}\left(ax + b\right)$$

```
m9 = glm(pass ~ hours, family = "binomial"("probit"))
summary(m9)
```

```
##
## Call:
## glm(formula = pass ~ hours, family = binomial("probit"))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.70120  -0.56628  -0.05298   0.43203   1.82066
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.4728     0.9516  -2.599  0.00936 **
## hours         0.9127     0.3367   2.711  0.00672 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 27.726  on 19  degrees of freedom
## Residual deviance: 15.795  on 18  degrees of freedom
## AIC: 19.795
##
## Number of Fisher Scoring iterations: 6
```

```
c9 = matrix(coef(m9))
# all the predict function really does is this dot product
head(cbind(predict(m9), cbind(rep(1, length(hours)), hours) %*% c9))
```

```
##         [,1]       [,2]
## 1 -2.0164198 -2.0164198
## 2 -1.7882497 -1.7882497
## 3 -1.5600796 -1.5600796
## 4 -1.3319094 -1.3319094
## 5 -1.1037393 -1.1037393
## 6 -0.8755692 -0.8755692
```

```
p9 = pnorm(predict(m9))
t9 = table(ifelse(p9 < 0.5, 0, 1), pass)
t9
```

```
##    pass
##     0 1
##   0 8 2
##   1 2 8
```

```
acc9 = sum(diag(t9)) / sum(t9)
acc9
```

```
## [1] 0.8
```

The aov function displays the sum of the squares of errors.

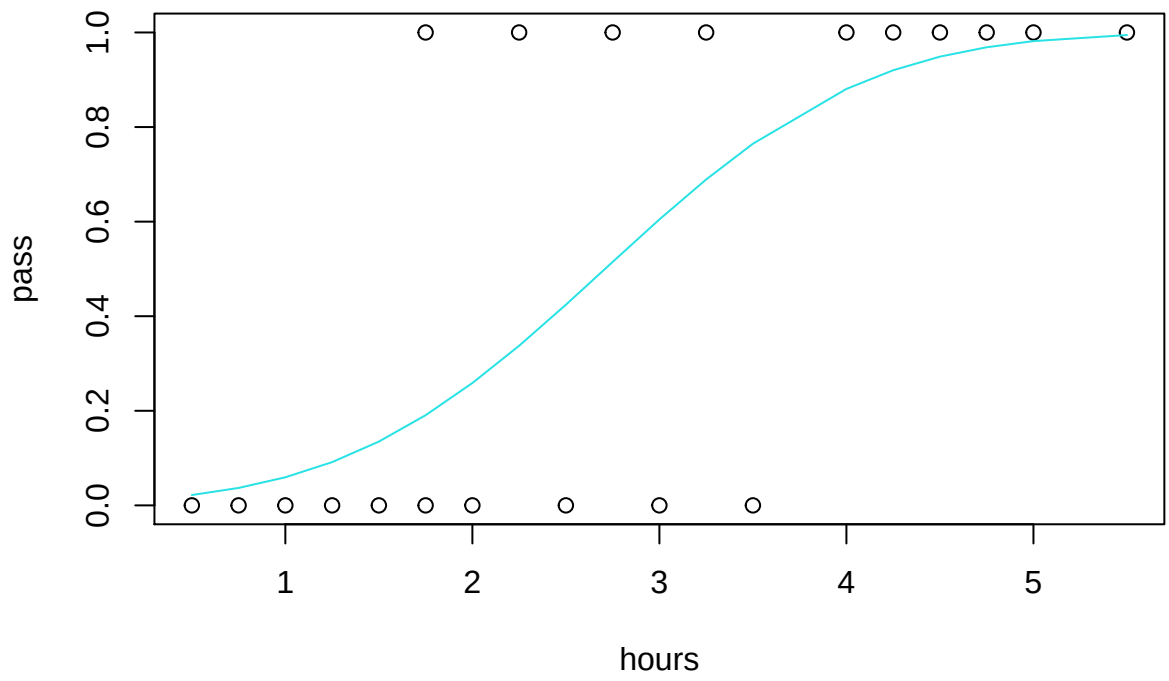
```
aov(m9)
```

```
## Call:
##    aov(formula = m9)
```

```
##
## Terms:
##                       hours Residuals
## Sum of Squares  2.375281  2.624719
## Deg. of Freedom        1        18
##
## Residual standard error: 0.3818609
## Estimated effects may be unbalanced
```

R^2 is computed from SS(model)/SS(model) + SSE(residuals). $0 \leq R^2 \leq 1$. R^2 may be identical for linear, logit, and probit models, even if the models have different accuracy/PMC.

```
plot(pass ~ hours)
lines(p9 ~ hours, col = 5)
```



Train-Test Split

A model is guaranteed to be optimal for the data it was fitted to. Does the model generalize to new data? Partitioning a dataset into non-overlapping testing and training subsets gives us a strong indication of the accuracy for a model.

We can reproduce the train-test split by seeding the random number generator.

```
set.seed(2021)
```

Now, generate the indices for the subsets.

```
indices = sample(2, nrow(iris), replace = TRUE, prob = c(.6, .4))
indices
```

```
##   [1] 1 2 2 1 2 2 2 1 2 2 1 2 2 1 2 1 1 2 2 1 1 2 1 1 1 2 2 2 2 2 1 2 2 2 1 1 2
##  [38] 1 2 1 1 2 1 2 1 2 2 1 1 2 1 2 1 1 1 1 1 1 2 2 1 1 1 1 2 1 1 2 1 1 1 2 1 2
##  [75] 1 1 1 1 2 2 1 1 2 2 1 1 2 2 1 1 1 2 2 1 1 2 2 2 2 2 1 1 1 2 1 1 1 1 1 2 1
## [112] 1 2 2 1 1 1 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 2 1 2 1 1 2 1 1 1 2 1 1 1 1 2 2
## [149] 1 1
```

```
table(indices)
```

```
## indices
##  1  2
## 89 61
```

Finally, assign values to data frames.

```
training = iris[indices == 1, ]
nrow(training)
```

```
## [1] 89
```

```
testing = iris[indices == 2, ]
nrow(testing)
```

```
## [1] 61
```

Multinom Model

The multinomial logistic prediction model allows more than two classes for y. The general idea is that if $y \in \{1, 2, 3\}$, then we compute $\Pr(y = 1|x)$ and $\Pr(y = 2|x)$. It is not necessary to compute $\Pr(y = 3|x)$, as this probability is given from the others. The sum of probabilities must be equal to 1. The multinomial logistic prediction model is not limited to only three classes.

$$\text{logit}(y = i|x) = \ln \frac{\Pr(y = i)}{1 - \Pr(y = i)}$$

The `multinom` function is in the `nnet` package.

```
library(nnet)
```

Split training/testing data.

```
set.seed(1776)
indices = sample(2, nrow(iris), replace = TRUE, prob = c(.6, .4))
training = iris[indices == 1, ]
testing = iris[indices == 2, ]
m10 = multinom(formula = Species ~ Sepal.Length + Sepal.Width + Petal.Length +
                 Petal.Width, data = training)
```

```
## # weights:  18 (10 variable)
## initial  value 88.987595
## iter  10 value 9.621663
## iter  20 value 3.844206
## iter  30 value 3.814678
## iter  40 value 3.806309
## iter  50 value 3.802712
```

```
## iter  60 value 3.802122
## iter  70 value 3.801984
## iter  80 value 3.801816
## iter  90 value 3.801690
## iter 100 value 3.801614
## final  value 3.801614
## stopped after 100 iterations
```

```
summary(m10)
```

```
## Call:
## multinom(formula = Species ~ Sepal.Length + Sepal.Width + Petal.Length +
##     Petal.Width, data = training)
##
## Coefficients:
##            (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
## versicolor    18.94555    -5.273733   -8.460874     12.70690  -0.8662309
## virginica    -26.54891    -7.792418   -9.010905     20.40105  14.5090739
##
## Std. Errors:
##            (Intercept) Sepal.Length Sepal.Width Petal.Length Petal.Width
## versicolor    39.76031     116.1282    215.2353     62.03196    54.94933
## virginica     40.71391     116.1771    215.2965     62.28595    55.18141
##
## Residual Deviance: 7.603228
## AIC: 27.60323
```

```
head(fitted.values(m10))
```

```
##      setosa   versicolor    virginica
## 1 1.0000000 2.173049e-09 1.506596e-29
## 2 0.9999996 4.289388e-07 6.479336e-27
## 4 0.9999968 3.190822e-06 2.096324e-25
## 6 1.0000000 5.760765e-10 3.278254e-28
## 7 0.9999999 6.487497e-08 7.790375e-27
## 8 1.0000000 3.057803e-08 6.219596e-28
```

```
table(predict(m10), training$Species)
```

```
##
##              setosa versicolor virginica
##   setosa         29          0         0
##   versicolor      0         24         1
##   virginica       0          1        26
```

This model is extremely accurate for its training data. Now, we evaluate the model with the testing data. This requires a little bit of linear algebra.

The coefficients are in a matrix

$$\beta = \begin{pmatrix} \beta_{01} & \beta_{02} \\ \beta_{11} & \beta_{12} \\ \vdots & \vdots \\ \beta_{n1} & \beta_{n2} \end{pmatrix}$$

which correspond to the n columns of the input matrix X. Notice that there is one extra row, β_0. This is the y-intercept. We need to augment our X matrix with a column of ones. The prediction will be formed

from the matrix product

$$(\Pr(y=1|\,X)\quad \Pr(y=2|\,X)) = X\beta = \begin{pmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{pmatrix} \begin{pmatrix} \beta_{01} & \beta_{02} \\ \beta_{11} & \beta_{12} \\ \vdots & \vdots \\ \beta_{n1} & \beta_{n2} \end{pmatrix}.$$

```
c10 = coef(m10)
x10 = cbind(rep(1, nrow(testing)), testing[,1], testing[,2], testing[,3], testing[,4])
# the %*% operator is for matrix multiplication
y10 = x10 %*% t(c10)
head(y10)
```

```
##      versicolor virginica
## [1,]  -16.57007 -62.58498
## [2,]  -20.26585 -66.48696
## [3,]  -11.17900 -55.50388
## [4,]  -21.95074 -68.46492
## [5,]  -26.35754 -71.44549
## [6,]  -20.03376 -64.91421
```

```
# exponentiate y to convert to predicted odds
y10 = exp(y10)
head(y10)
```

```
##        versicolor    virginica
## [1,] 6.363710e-08 6.602186e-28
## [2,] 1.579988e-09 1.333762e-29
## [3,] 1.396443e-05 7.851838e-25
## [4,] 2.930319e-10 1.845290e-30
## [5,] 3.573268e-12 9.367348e-32
## [6,] 1.992735e-09 6.428621e-29
```

```
pb = cbind(1/(1+y10[,1]+y10[,2]),y10[,1]/(1+y10[,1]+y10[,2]),y10[,2]/(1+y10[,1]+y10[,2]))
head(pb)
```

```
##           [,1]         [,2]         [,3]
## [1,] 0.9999999 6.363709e-08 6.602185e-28
## [2,] 1.0000000 1.579988e-09 1.333762e-29
## [3,] 0.9999860 1.396423e-05 7.851728e-25
## [4,] 1.0000000 2.930319e-10 1.845290e-30
## [5,] 1.0000000 3.573268e-12 9.367348e-32
## [6,] 1.0000000 1.992735e-09 6.428621e-29
```

```
# sum each row and verify that they are all 1's
unique(pb %*% c(1,1,1))
```

```
##      [,1]
## [1,]    1
## [2,]    1
## [3,]    1
```

```
# use list comprehension to select the most probable class
library(comprehenr)
p10 = to_vec(for(i in 1:nrow(testing)) which.max(pb[i,]))
t10 = table(p10, testing$Species)
```

```
acc10 = sum(diag(t10)) / sum(t10)
acc10
```

```
## [1] 0.9565217
```

Covariance

If variance is the average squared difference of a random variable x and its expected value $\bar{x}$, then *covariance* is the average product of the differences of two random variables x and x and their respective expected values $\bar{x}$ and $\bar{x}$.

$$\text{cov}(x, y) = \sum_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{n - 1}$$

Covariance is meaningful when x and y are *scaled* to $\bar{x} = \bar{y} = 0$ and $s_x = s_y = 1$.

```
x = 1:100
mean(x)
```

```
## [1] 50.5
```

```
sd(x)
```

```
## [1] 29.01149
```

```
x = scale(1:100)
mean(x)
```

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

```
sd(x)
```

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

The covariance statistic gives us a powerful method to observe correlation between two random variables. The strongest possible covariance is ± 1. This happens when there is a linear relationship between the two variables (including the identity relationship, $y = ax + b = 1x + 0 = x$). The weakest covariance occurs when there is no relationship between two variables. Weakly correlated variables have a covariance near 0.

```
cov(x, scale(20 * 1:100 + 21))
```

```
##      [,1]
## [1,]    1
```

```
cov(x, -x)
```

```
##      [,1]
## [1,]   -1
```

```
cov(x, scale((1:100)^2))
```

```
##           [,1]
## [1,] 0.9688545
```

```
cov(x, scale(exp(1:100)))
```

```
##          [,1]
## [1,] 0.252032
```

```
cov(x, scale(rnorm(100)))
```

```
##             [,1]
## [1,] -0.06504679
```

```
cov(rnorm(100), rnorm(100))
```

```
## [1] 0.08313274
```

```
cov(runif(100), runif(100))
```

```
## [1] 0.007329415
```

The `cov` function in R can produce a square matrix containing covariances for all pairwise combinations of variables in a data set.

```
# all columns except the fifth, which is not numeric
cov(scale(iris[,-5]))
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

The `cor` function handles scaling automatically.

```
cor(iris[,-5])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

The values in the correlation matrix, C, should be intuitive. For example, observe that covariance between `mpg` and `hp` is strong and negative. This makes sense: better efficiency in cars requires less horsepower.

```
cor(mtcars[,-2])
```

```
##             mpg       disp         hp        drat         wt        qsec
## mpg   1.0000000 -0.8475514 -0.7761684  0.68117191 -0.8676594  0.41868403
## disp -0.8475514  1.0000000  0.7909486 -0.71021393  0.8879799 -0.43369788
## hp   -0.7761684  0.7909486  1.0000000 -0.44875912  0.6587479 -0.70822339
## drat  0.6811719 -0.7102139 -0.4487591  1.00000000 -0.7124406  0.09120476
## wt   -0.8676594  0.8879799  0.6587479 -0.71244065  1.0000000 -0.17471588
## qsec  0.4186840 -0.4336979 -0.7082234  0.09120476 -0.1747159  1.00000000
## vs    0.6640389 -0.7104159 -0.7230967  0.44027846 -0.5549157  0.74453544
## am    0.5998324 -0.5912270 -0.2432043  0.71271113 -0.6924953 -0.22986086
## gear  0.4802848 -0.5555692 -0.1257043  0.69961013 -0.5832870 -0.21268223
## carb -0.5509251  0.3949769  0.7498125 -0.09078980  0.4276059 -0.65624923
##              vs          am       gear        carb
## mpg   0.6640389  0.59983243  0.4802848 -0.55092507
## disp -0.7104159 -0.59122704 -0.5555692  0.39497686
## hp   -0.7230967 -0.24320426 -0.1257043  0.74981247
## drat  0.4402785  0.71271113  0.6996101 -0.09078980
## wt   -0.5549157 -0.69249526 -0.5832870  0.42760594
## qsec  0.7445354 -0.22986086 -0.2126822 -0.65624923
## vs    1.0000000  0.16834512  0.2060233 -0.56960714
```

```
## am    0.1683451  1.00000000  0.7940588  0.05753435
## gear  0.2060233  0.79405876  1.0000000  0.27407284
## carb -0.5696071  0.05753435  0.2740728  1.00000000
```

The correlation matrix can be interpreted as the slopes of the lines of best fit between two variables.

Singular Value Decomposition

The Singular Value Decomposition (SVD) of a matrix is a means of extracting a diagonal matrix D from A where $U'AV = D$, $U'U = I$, and $V'V = I$. R implements this algorithm in the `svd` function.

```
c = cor(iris[,-5])
s = svd(c)
s$d
```

```
## [1] 2.91849782 0.91403047 0.14675688 0.02071484
```

```
attributes(s)
```

```
## $names
## [1] "d" "u" "v"
```

```
# the zapsmall function rounds some precision errors near zero
zapsmall(t(s$u) %*% c %*% s$v, digits = 5)
```

```
##        [,1]    [,2]    [,3]    [,4]
## [1,] 2.9185 0.00000 0.00000 0.00000
## [2,] 0.0000 0.91403 0.00000 0.00000
## [3,] 0.0000 0.00000 0.14676 0.00000
## [4,] 0.0000 0.00000 0.00000 0.02071
```

```
diag(t(s$u) %*% c %*% s$v)
```

```
## [1] 2.91849782 0.91403047 0.14675688 0.02071484
```

The matrix U contains orthonormal eigenvectors that will be used for Principal Component Analysis (PCA).

```
s$u
```

```
##            [,1]        [,2]       [,3]       [,4]
## [1,] -0.5210659 -0.37741762  0.7195664  0.2612863
## [2,]  0.2693474 -0.92329566 -0.2443818 -0.1235096
## [3,] -0.5804131 -0.02449161 -0.1421264 -0.8014492
## [4,] -0.5648565 -0.06694199 -0.6342727  0.5235971
```

Principal Component Analysis

Principal Component Analysis (PCA) allows us to compress the related variables of a data set. The principal components are computed from the matrix product of the scaled data set X with the matrix U found in SVD.

$$\text{PC} = XU$$

```
pc = scale(iris[,-5]) %*% s$u
summary(pc)
```

```
##        V1                 V2                  V3                  V4
##  Min.   :-3.2996   Min.   :-2.67732   Min.   :-1.00204   Min.   :-0.487849
```

```
##  1st Qu.:-1.3385   1st Qu.:-0.59205   1st Qu.:-0.19386   1st Qu.:-0.074428
##  Median :-0.4169   Median :-0.01744   Median :-0.02468   Median : 0.006805
##  Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.00000   Mean   : 0.000000
##  3rd Qu.: 2.0957   3rd Qu.: 0.59649   3rd Qu.: 0.25820   3rd Qu.: 0.090579
##  Max.   : 2.7651   Max.   : 2.64521   Max.   : 0.85456   Max.   : 0.468128
```

The principal components retain the same covariances as the original data set.

```
zapsmall(cov(pc), dig = 5)
```

```
##        [,1]    [,2]    [,3]    [,4]
## [1,] 2.9185 0.00000 0.00000 0.00000
## [2,] 0.0000 0.91403 0.00000 0.00000
## [3,] 0.0000 0.00000 0.14676 0.00000
## [4,] 0.0000 0.00000 0.00000 0.02071
```

The principal components, or a subset of the principal components, can be used to fit a model.

```
m11 = multinom(iris$Species ~ pc[,1])
```

```
## # weights:  9 (4 variable)
## initial  value 164.791843
## iter  10 value 25.587568
## iter  20 value 25.076351
## iter  30 value 25.062044
## iter  40 value 25.059980
## iter  50 value 25.058339
## iter  60 value 25.057430
## iter  70 value 25.056369
## iter  80 value 25.055474
## iter  90 value 25.055370
## final  value 25.055198
## converged
```

Predictions formed from a subset of the principal components can be stunningly accurate. Observe the performance of this model, which uses only a single PC.

```
table(predict(m11), iris$Species)
```

```
##
##              setosa versicolor virginica
##   setosa         50          0         0
##   versicolor      0         44         5
##   virginica       0          6        45
```

It is not always feasible to identify what these components actually correspond to.

The `prcomp` and `princomp` convenience functions perform PCA.

```
s$u
```

```
##            [,1]        [,2]       [,3]       [,4]
## [1,] -0.5210659 -0.37741762  0.7195664  0.2612863
## [2,]  0.2693474 -0.92329566 -0.2443818 -0.1235096
## [3,] -0.5804131 -0.02449161 -0.1421264 -0.8014492
## [4,] -0.5648565 -0.06694199 -0.6342727  0.5235971
```

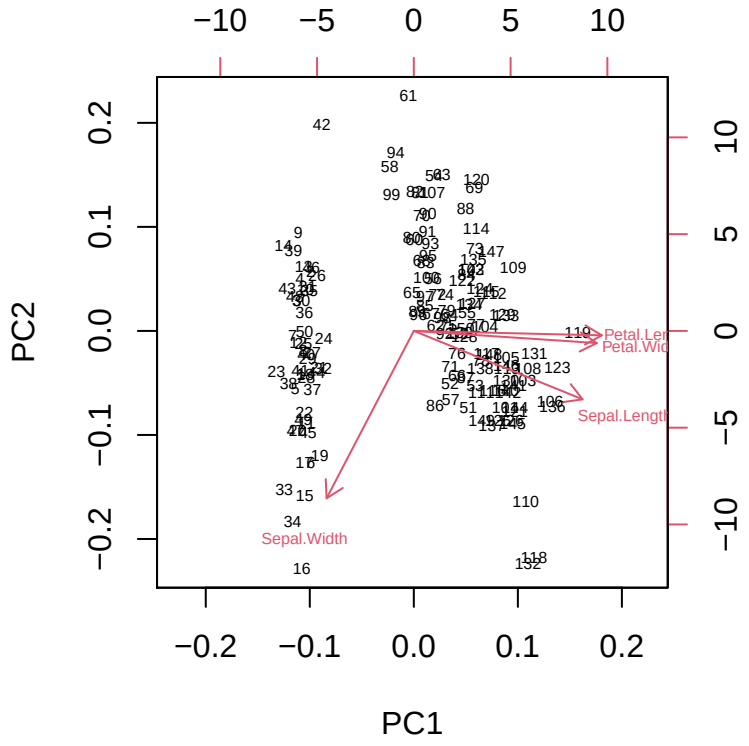
```
pc11 = prcomp(iris[,-5], center = TRUE, scale = TRUE)
attributes(pc11)
```

```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

The values in `pc11$x` are the same values computed in `pc` earlier.

The `biplot` function can help us to visualize the weights of the free variables.

```
biplot(pc11, cex = .5)
```




```
pc12 = prcomp(mtcars[,-2], center = TRUE, scale = TRUE)
pc12
```

```
## Standard deviations (1, .., p=10):
##  [1] 2.3870112 1.6260695 0.7789541 0.5192262 0.4722086 0.4539880 0.3674303
##  [8] 0.3432990 0.2773822 0.1510772
##
## Rotation (n x k) = (10 x 10):
##              PC1          PC2         PC3          PC4          PC5         PC6
## mpg  -0.3933031  0.002673234 -0.19432743  0.02213387 -0.10381360  0.1503227
## disp  0.3968373 -0.036747437 -0.09413658 -0.25602403 -0.43475427  0.2675614
## hp    0.3521714  0.260647937  0.11660175  0.06748839 -0.53300670 -0.1514492
## drat -0.3200402  0.265069816  0.18888714 -0.85559750 -0.04320425 -0.2270672
## wt    0.3786780 -0.129871176  0.31259764 -0.24514105  0.02124494  0.4674935
## qsec -0.2052711 -0.469488227  0.42851226 -0.06822117  0.13855899  0.4030371
## vs   -0.3239576 -0.241530899  0.47020447  0.21355079 -0.56485159 -0.2238605
```

```
## am   -0.2598732  0.421482792 -0.20277762  0.03115801 -0.15237443  0.5636751
## gear -0.2267859  0.456632869  0.30269814  0.26463392 -0.07007775  0.2601658
## carb  0.2278920  0.422680625  0.51857846  0.12651437  0.38394347 -0.1216518
##               PC7         PC8         PC9        PC10
## mpg   0.305940514 -0.77382321 -0.25180659  0.13367787
## disp  0.216505822 -0.03451067 -0.20451881 -0.64518656
## hp   -0.005979696 -0.29645344  0.55661884  0.29173515
## drat  0.021359531  0.04829742  0.05349434  0.02258840
## wt   -0.026928884  0.01579313 -0.36077093  0.57601501
## qsec  0.016444236 -0.15529536  0.54309474 -0.21952502
## vs   -0.268083495  0.08496762 -0.34712591 -0.03553290
## am   -0.602377109  0.04803261  0.06069941 -0.05414093
## gear  0.618108817  0.35049404  0.01907061  0.02306342
## carb -0.203107400 -0.39121681 -0.18007569 -0.30908675
```

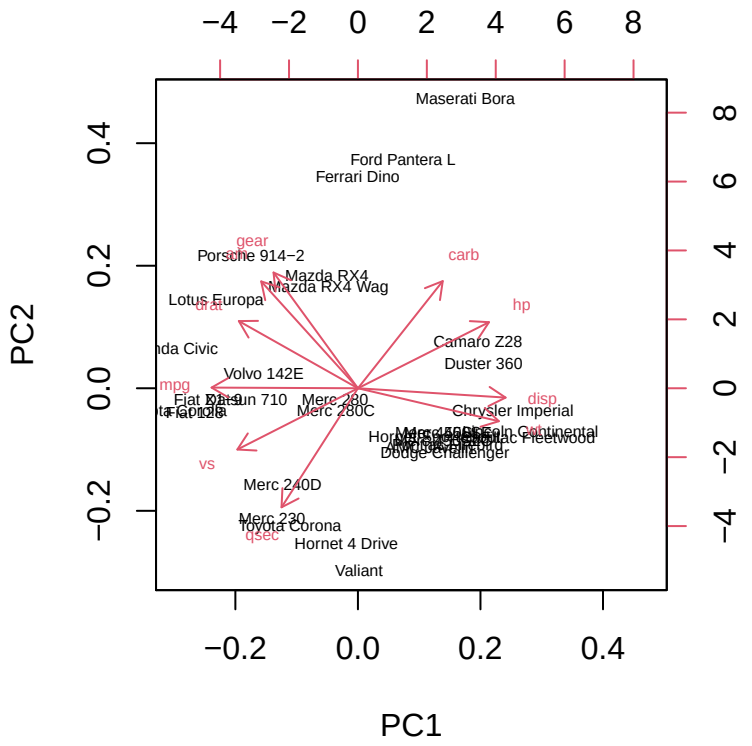
The `summary` of the `prcomp` class shows the individual and cumulative significance of each component.

```
summary(pc12)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4    PC5     PC6    PC7
## Standard deviation     2.3870 1.6261 0.77895 0.51923 0.4722 0.45399 0.3674
## Proportion of Variance 0.5698 0.2644 0.06068 0.02696 0.0223 0.02061 0.0135
## Cumulative Proportion  0.5698 0.8342 0.89487 0.92183 0.9441 0.96474 0.9782
##                           PC8     PC9    PC10
## Standard deviation     0.34330 0.27738 0.15108
## Proportion of Variance 0.01179 0.00769 0.00228
## Cumulative Proportion  0.99002 0.99772 1.00000
```

PC1 relates strongly to efficiency, displacement, and horsepower. PC1 accounts for 56.98% of all variance in the data set. PC2 accounts for another 26.44% of variance in the data set, and has more influence on the number of gears, carburetors, and quarter mile time. We might conjecture that PC1 and PC2 could be interpreted as the size and speed of the car.

```
biplot(pc12, cex = .5)
```




```
m12 = multinom(mtcars$cyl ~ pc12$x[,1])
```

```
## # weights:  9 (4 variable)
## initial  value 35.155593
## iter  10 value 0.684501
## iter  20 value 0.028064
## iter  30 value 0.017481
## iter  40 value 0.012504
## iter  50 value 0.009071
## iter  60 value 0.003600
## iter  70 value 0.001523
## iter  80 value 0.000751
## iter  90 value 0.000687
## iter 100 value 0.000555
## final  value 0.000555
## stopped after 100 iterations
```

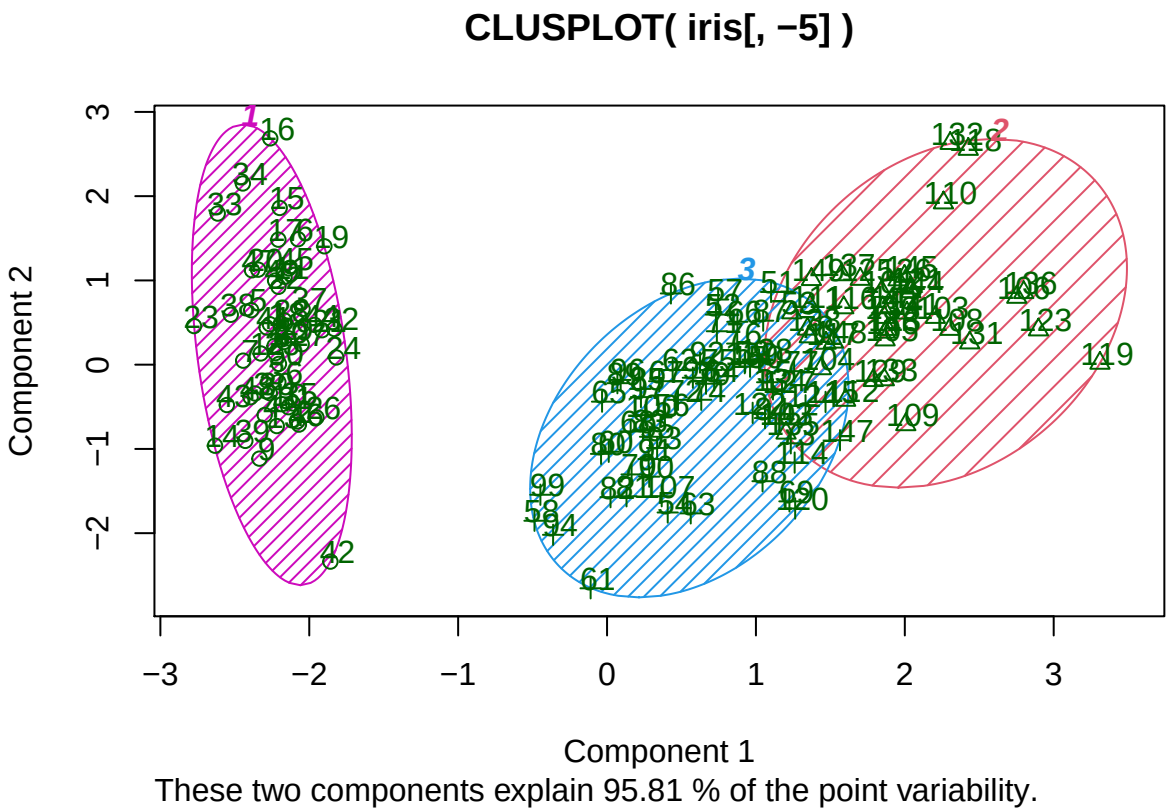
```
table(mtcars$cyl, predict(m12))
```

```
##
##      4  6  8
##   4 11  0  0
##   6  0  7  0
##   8  0  0 14
```

k-Means Clustering

Clustering is another solution to modeling data with a categorical target variable. With k-means, we have to know how many categories are in the target.

```
set.seed(1985)
k = kmeans(iris[,-5], 3)
library(cluster)
# compare the appearance of this cluster plot to the PCA biplot earlier!
clusplot(iris[,-5], k$cluster, color=TRUE, shade=TRUE, labels=2, lines=0)
```



The results are exposed in the `cluster` property.

```
k$cluster
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 2 2 2 2 3 2 2 2 2
## [112] 2 2 3 3 2 2 2 2 3 2 3 2 3 2 2 3 3 2 2 2 2 2 3 2 2 2 2 3 2 2 2 3 2 2 2 3 2
## [149] 2 3
```

The `kmeans` function uses randomness, so the cluster numbers may be different in each run.

```
kmeans(iris[,-5], 3)$cluster
```

```
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [75] 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 2 1 1 1 1
## [112] 1 1 2 2 1 1 1 1 2 1 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 1 2 1 1 1 2 1 1 1 2 1
```

```
## [149] 1 2
```

```
kmeans(iris[,-5], 3)$cluster
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1
## [112] 1 1 3 3 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 1 1 1 1 3 1 1 1 3 1 1 1 3 1
## [149] 1 3
```

This means that accurate classifications are not guaranteed to be along the diagonal of the `table` matrix.

```
t13 = table(iris$Species, k$cluster)
t13
```

```
##
##                1  2  3
##   setosa      50  0  0
##   versicolor   0  2 48
##   virginica    0 36 14
```

The solution is to assume that the majority element for each column was an accurate classification.

```
sum(apply(t13, 1, max)) / sum(t13)
```

```
## [1] 0.8933333
```

Function Reference

R Function	Usage
`library`	Load a library
`read.table`	Parse a file as a data frame
`read.csv`	Parse comma-separated values as a data frame
`head`	Show the first six elements of a vector, matrix, table, or data frame
`str`	Show the structure of an R data frame
`nrow`	Count the rows of a data frame or matrix
`ncol`	Count the columns of a data frame or matrix
`na.omit`	Drops `NA` values from a dataframe
`as.factor`	Specify that a vector contains categorical data
`[,-n]`	Return all rows and all columns except n from a data set
`c`	Create a column vector
`cbind`	Bind vectors as the columns of a matrix
`rm`	Delete a value from R's environment
`summary`	Summarize data objects or models
`mean`	Calculate the means of each column
`sd`	Calculate the standard deviation for one column
`var`	Calculate the variance in one column or covariance among all columns
`quantile`	Find the bounds that split a data set into four equal subsets.
`range`	Find the extrema of a data set
`skewness`	Compute the skewness of a distribution
`kurtosis`	Compute the kurtosis of a distribution
`hist`	Render a histogram for a data set
`plot`	Draw a scatter plot of x and y values on the Cartesian plane
`lines`	Draw lines between x and y coordinates on a plot
`barplot`	Draw a bar plot
`boxplot`	Draw a box-and-whisker plot of a distribution

R Function	Usage
`lm`	Fit a linear model of the form `y ~ x1 + x2 + ... + xn`
`glm`	Fit a generalized linear model (optionally as a logit or probit)
`multinom`	Fit a multinomial logistic regression model
`coef`	Extract the coefficients from a model
`predict`	Compute a linear combination from a list of observations
`sum`	Compute the sum of a vector or matrix
`exp`	Exponentiate e to some power
`version`	Show the R software version
`function`	Define a pure function
`stopifnot`	Halt a function if a condition is not met
`return`	Stop a function and return a value. Requires parenthesis.
`pnorm`	Find the cumulative probability for the normal distribution
`t.test`	The Student t-Test
`aov`	The Analysis of Variance (ANOVA) test
`TukeyHSD`	Confidence intervals between pairs of variables
`ifelse`	Return values based on the outcome of a conditional statement
`table`	Count predicted and actual values in tabular form
`diag`	Extract the values from the diagonal (M_{ii}) of a matrix
`set.seed`	Specify the seed for the pseudo-random number generator
`sample`	Generate indices that can be used for train/test splits
`%*%`	Matrix multiplication and matrix-vector multiplication
`t`	Transpose a matrix
`to_vec`	List comprehension from the `comprendr` package
`scale`	Subtract the sample mean and divide by standard deviation
`cov`	Find the covariance in the numerical columns of a matrix
`svd`	Singular value decomposition of matrix A, $U'AV = D$
`prcomp`	Perform Principal Component Analysis (PCA)
`kmeans`	Discover clusters in a data set using the k-means algorithm

Version

```
version
```

```
##                _
## platform       x86_64-w64-mingw32
## arch           x86_64
## os             mingw32
## system         x86_64, mingw32
## status
## major          4
## minor          0.5
## year           2021
## month          03
## day            31
## svn rev        80133
## language       R
## version.string R version 4.0.5 (2021-03-31)
## nickname       Shake and Throw
```