R Tutorial

In R, comments are indicated in code with a "#" character.

# 1 Reading in Data and Basic Statistical Functions

## 1.1 Read in the data.

### a) Read the data in as a .csv file.

student <- read.csv('/Users/class.csv')

[read.csv()](https://www.rdocumentation.org/packages/utils/versions/3.4.0/topics/read.table)

### b) Read the data in as a .xls file.

First we need to install the [gdata](#GDATA) package, and then call the package to use.

# The package we need to read in a .xls file (gdata) must first be  
# installed and then called to use  
library(gdata)  
  
student\_xls <- read.xls('/Users/class.xls', 1)

[read.xls()](https://www.rdocumentation.org/packages/gdata/versions/2.18.0/topics/read.xls)

### c) Read the data in as a .json file.

First we need to install the [rjson](#RJSON) (install.packages(pkgs='rjson')), and then call the package to use. There is more code involved in reading a .json file into R so it becomes a proper data frame, however we will not at this time dive into the explanation for all this code, but it should become evident throughout the tutorial.

# The package we need to read in a .json file (rjson) must first be  
# installed and then called to use  
library(rjson)  
  
temp <- fromJSON(file = '/Users/class.json')  
temp <- do.call('rbind', temp)  
temp <- data.frame(temp, stringsAsFactors = TRUE)  
temp <- transform(temp, Name=unlist(Name), Sex=unlist(Sex), Age=unlist(Age),   
 Height=unlist(Height), Weight=unlist(Weight))  
temp$Name <- as.factor(temp$Name)  
temp$Sex <- as.factor(temp$Sex)  
temp$Age <- as.integer(temp$Age)  
  
student\_json <- temp

[fromJSON()](https://www.rdocumentation.org/packages/rjson/versions/0.2.15/topics/fromJSON)

## 1.2 Find the dimensions of the data set.

Information about an R [data frame](#DataFrame) is available by calling the [dim()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/dim) function, with the data name as an argument.

dim(student)

## [1] 19 5

## 1.3 Find basic information about the data set.

str(student)

## 'data.frame': 19 obs. of 5 variables:  
## $ Name : Factor w/ 19 levels "Alfred","Alice",..: 1 2 3 4 5 6 7 8 9 10 ...  
## $ Sex : Factor w/ 2 levels "F","M": 2 1 1 1 2 2 1 1 2 2 ...  
## $ Age : int 14 13 13 14 14 12 12 15 13 12 ...  
## $ Height: num 69 56.5 65.3 62.8 63.5 57.3 59.8 62.5 62.5 59 ...  
## $ Weight: num 112 84 98 102 102 ...

[str()](https://www.rdocumentation.org/packages/utils/versions/3.4.0/topics/str)

## 1.4 Look at the first 5 observations.

The first 5 observations of a [data frame](#DataFrame) are available by calling the [head()](https://www.rdocumentation.org/packages/utils/versions/3.4.0/topics/head) function, with the data name as an argument. By default, head() returns 4 observations, but we can alter the function to return 5 observations in the way shown below. The [tail()](https://www.rdocumentation.org/packages/utils/versions/3.4.0/topics/head) function is analogous and returns the last observations.

head(student, n=5)

## Name Sex Age Height Weight  
## 1 Alfred M 14 69.0 112.5  
## 2 Alice F 13 56.5 84.0  
## 3 Barbara F 13 65.3 98.0  
## 4 Carol F 14 62.8 102.5  
## 5 Henry M 14 63.5 102.5

## 1.5 Calculate mean of numeric variables.

# We must apply the is.numeric function to the data set which returns a   
# matrix of booleans that we then use to subset the dataset to return   
# only numeric variables   
  
# Then we can use the colMeans function to return the mean of   
# column variables  
colMeans(student[sapply(student, is.numeric)])

## Age Height Weight   
## 13.31579 62.33684 100.02632

[colMeans()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/colSums) [sapply()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/lapply) [is.numeric](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/numeric)

## 1.6 Compute summary statistics of the data set.

Summary statistics of a [data frame](#DataFrame) are available by calling the [summary()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/summary) function, with the data name as an argument.

summary(student)

## Name Sex Age Height Weight   
## Alfred : 1 F: 9 Min. :11.00 Min. :51.30 Min. : 50.50   
## Alice : 1 M:10 1st Qu.:12.00 1st Qu.:58.25 1st Qu.: 84.25   
## Barbara: 1 Median :13.00 Median :62.80 Median : 99.50   
## Carol : 1 Mean :13.32 Mean :62.34 Mean :100.03   
## Henry : 1 3rd Qu.:14.50 3rd Qu.:65.90 3rd Qu.:112.25   
## James : 1 Max. :16.00 Max. :72.00 Max. :150.00   
## (Other):13

## 1.7 Descriptive statistics functions applied to columns of the data set.

# Notice the subsetting of student with the $ character   
sd(student$Weight)

## [1] 22.77393

sum(student$Weight)

## [1] 1900.5

length(student$Weight)

## [1] 19

max(student$Weight)

## [1] 150

min(student$Weight)

## [1] 50.5

median(student$Weight)

## [1] 99.5

[sd()](https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/sd) [sum()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/sum) [length()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/length) [max()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/Extremes) [min()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/Extremes) [median()](https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/median)

## 1.8 Produce a one-way table to describe the frequency of a variable.

### a) Produce a one-way table of a discrete variable.

table(student$Age)

##   
## 11 12 13 14 15 16   
## 2 5 3 4 4 1

### b) Produce a one-way table of a categorical variable.

table(student$Sex)

##   
## F M   
## 9 10

[table()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/table)

## 1.9 Produce a two-way table to visualize the frequency of two categorical (or discrete) variables.

table(student$Age, student$Sex)

##   
## F M  
## 11 1 1  
## 12 2 3  
## 13 2 1  
## 14 2 2  
## 15 2 2  
## 16 0 1

[table()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/table)

## 1.10 Select a subset of the data that meets a certain criterion.

# The "," character tells R to select all columns of the data set  
females <- student[which(student$Sex == 'F'), ]  
head(females, n=5)

## Name Sex Age Height Weight  
## 2 Alice F 13 56.5 84.0  
## 3 Barbara F 13 65.3 98.0  
## 4 Carol F 14 62.8 102.5  
## 7 Jane F 12 59.8 84.5  
## 8 Janet F 15 62.5 112.5

[which()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/which)

## 1.11 Determine the correlation between two continuous variables.

height\_weight <- subset(student, select = c(Height, Weight))  
cor(height\_weight, method = "pearson")

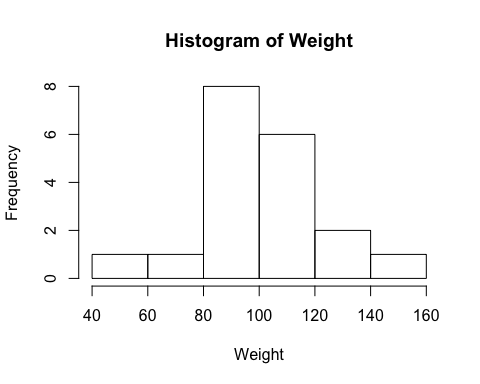
## Height Weight  
## Height 1.0000000 0.8777852  
## Weight 0.8777852 1.0000000

[cor()](https://www.rdocumentation.org/packages/WGCNA/versions/1.51/topics/cor)

# 2 Basic Graphing and Plotting Functions

## 2.1 Visualize a single continuous variable by producing a histogram.

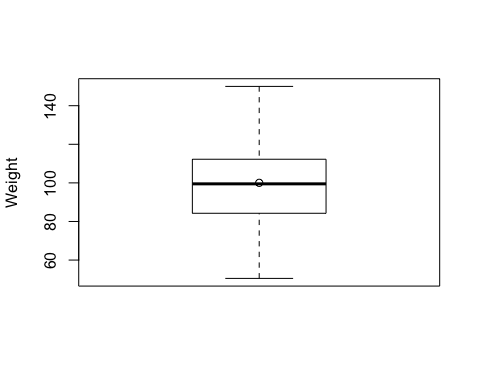
# Setting student$Weight to a new variable “Weight” cleans up the labeling of   
# the histogram   
Weight <- student$Weight  
hist(Weight)



[hist()](https://www.rdocumentation.org/packages/graphics/versions/3.4.0/topics/hist)

## 2.2 Visualize a single continuous variable by producing a boxplot.

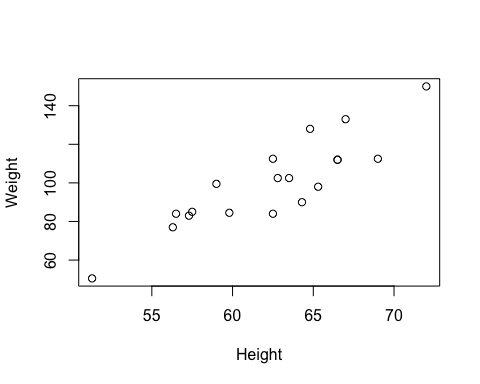
# points(mean(Weight)) tells R to plot the mean of the variable  
# on the boxplot   
boxplot(Weight, ylab="Weight")  
points(mean(Weight))



[boxplot()](https://www.rdocumentation.org/packages/graphics/versions/3.4.0/topics/boxplot) [points()](https://www.rdocumentation.org/packages/graphics/versions/3.4.0/topics/points)

## 2.3 Visualize two continuous variables by producing a scatterplot.

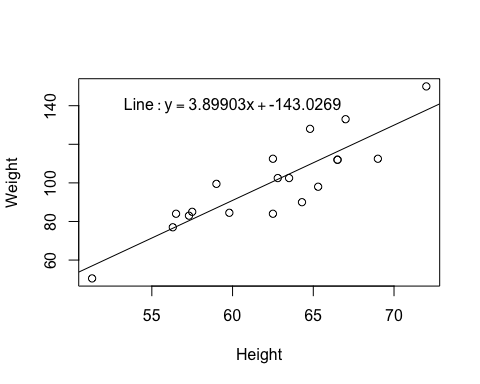
Height <- student$Height  
# Notice here you specify the x variable first and then the y variable   
plot(Height, Weight)



[plot()](https://www.rdocumentation.org/packages/graphics/versions/3.4.0/topics/plot)

## 2.4 Visualize a relationship between two continuous variables by producing a scatterplot and a plotted line of best fit.

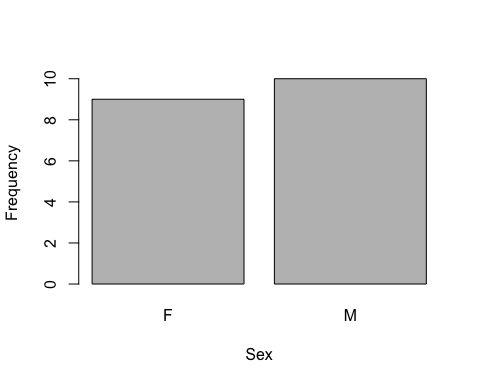
plot(Height, Weight)  
  
# lm() models Weight as a function of Height and returns the parameters   
# of the line of best fit  
model <- lm(Weight~Height)  
coeff <- coef(model)  
intercept <- as.matrix(coeff[1])[1]  
slope <- as.matrix(coeff[2])[1]  
  
# abline() prints the line of best fit   
abline(lm(Weight~Height))  
  
# text() prints the equation of the line of best fit, with the first   
# two arguments specifying the x and y location, respectively, of where   
# the text should be printed on the graph   
text(60, 140, bquote(Line: y == .(slope) \* x + .(intercept)))



[lm()](https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/lm) [coef()](https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/coef) [as.matrix()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/matrix) [abline()](https://www.rdocumentation.org/packages/graphics/versions/3.4.0/topics/abline) [text()](https://www.rdocumentation.org/packages/graphics/versions/3.4.0/topics/text) [bquote()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/bquote)

## 2.5 Visualize a categorical variable by producing a bar chart.

counts <- table(student$Sex)  
# beside = TRUE indicates to print the bars side by side instead of on top of   
# each other   
# names.arg indicates which names to use to label the bars   
barplot(counts, beside=TRUE, ylab= "Frequency", xlab= "Sex", names.arg=names(counts))

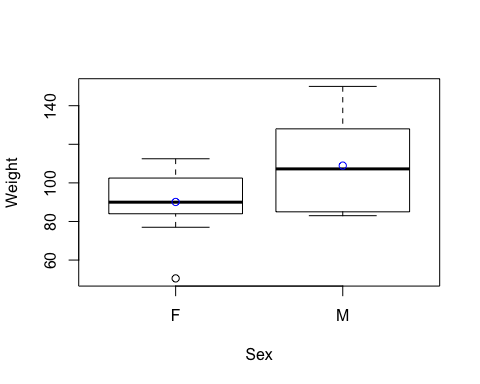


[barplot()](https://www.rdocumentation.org/packages/graphics/versions/3.4.0/topics/barplot) [names()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/names)

## 2.6 Visualize a continuous variable, grouped by a categorical variable, using side-by-side boxplots.

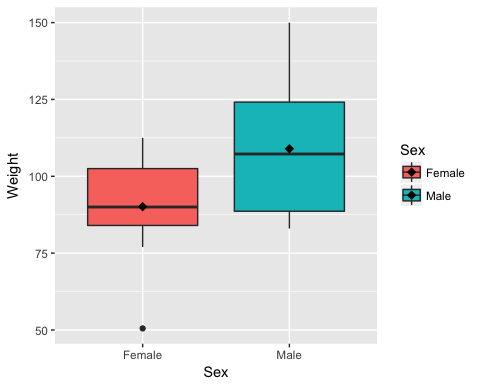
### a) Simple side-by-side boxplot without color.

# Subset data set to return only female weights, and then only male weights   
Female\_Weight <- student[which(student$Sex == 'F'), "Weight"]  
Male\_Weight <- student[which(student$Sex == 'M'), "Weight"]  
  
# Find the mean of both arrays   
means <- c(mean(Female\_Weight), mean(Male\_Weight))  
   
# Syntax indicates Weight as a function of Sex   
boxplot(student$Weight ~ student$Sex, ylab= "Weight", xlab= "Sex")  
  
# Plot means on boxplots in blue   
points(means, col= "blue")



### b) More advanced side-by-side boxplot with color.

library(ggplot2)  
student$Sex <- factor(student$Sex, levels = c("F","M"),   
 labels = c("Female", "Male"))  
ggplot(data = student, aes(x = Sex, y = Weight, fill = Sex)) +   
 geom\_boxplot() + stat\_summary(fun.y = mean,   
 color = "black", geom = "point",   
 shape = 18, size = 3)



[ggplot2](http://www.statmethods.net/advgraphs/ggplot2.html) [factor()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/factor) [c()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/c) [aes()](https://www.rdocumentation.org/packages/ggplot2/versions/2.2.1/topics/aes) [geom\_boxplot()](https://www.rdocumentation.org/packages/ggplot2/versions/2.2.1/topics/geom_boxplot) [stat\_summary()](https://www.rdocumentation.org/packages/ggplot2/versions/2.2.1/topics/stat_summary_bin)

# 3 Basic Data Wrangling and Manipulation

## 3.1 Create a new variable in a data set as a function of existing variables in the data set.

# Notice here how you can create the BMI column in the data set just by   
# naming it   
student$BMI <- student$Weight / (student$Height)\*\*2 \* 703  
head(student, n=5)

## Name Sex Age Height Weight BMI  
## 1 Alfred Male 14 69.0 112.5 16.61153  
## 2 Alice Female 13 56.5 84.0 18.49855  
## 3 Barbara Female 13 65.3 98.0 16.15679  
## 4 Carol Female 14 62.8 102.5 18.27090  
## 5 Henry Male 14 63.5 102.5 17.87030

## 3.2 Create a new variable in a data set using if/else logic of existing variables in the data set.

# Notice the use of the ifelse() function for a single if condition  
student$BMI\_Class <- ifelse(student$BMI<19.0, "Underweight", "Healthy")  
head(student, n=5)

## Name Sex Age Height Weight BMI BMI\_Class  
## 1 Alfred Male 14 69.0 112.5 16.61153 Underweight  
## 2 Alice Female 13 56.5 84.0 18.49855 Underweight  
## 3 Barbara Female 13 65.3 98.0 16.15679 Underweight  
## 4 Carol Female 14 62.8 102.5 18.27090 Underweight  
## 5 Henry Male 14 63.5 102.5 17.87030 Underweight

[ifelse()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/ifelse)

## 3.3 Create a new variable in a data set using mathemtical functions applied to existing variables in the data set.

Using the [log()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/log) function, the [exp()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/log) function, the [sqrt()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/MathFun) function, and the [abs()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/MathFun) function.

student$LogWeight <- log(student$Weight)  
student$ExpAge <- exp(student$Age)  
student$SqrtHeight <- sqrt(student$Height)  
student$BMI\_Neg <- ifelse(student$BMI < 19.0, -student$BMI, student$BMI)  
student$BMI\_Pos <- abs(student$BMI\_Neg)  
  
# Create a boolean variable  
student$BMI\_Check <- (student$BMI == student$BMI\_Pos)  
head(student, n=5)

## Name Sex Age Height Weight BMI BMI\_Class LogWeight  
## 1 Alfred Male 14 69.0 112.5 16.61153 Underweight 4.722953  
## 2 Alice Female 13 56.5 84.0 18.49855 Underweight 4.430817  
## 3 Barbara Female 13 65.3 98.0 16.15679 Underweight 4.584967  
## 4 Carol Female 14 62.8 102.5 18.27090 Underweight 4.629863  
## 5 Henry Male 14 63.5 102.5 17.87030 Underweight 4.629863  
## ExpAge SqrtHeight BMI\_Neg BMI\_Pos BMI\_Check  
## 1 1202604.3 8.306624 -16.61153 16.61153 TRUE  
## 2 442413.4 7.516648 -18.49855 18.49855 TRUE  
## 3 442413.4 8.080842 -16.15679 16.15679 TRUE  
## 4 1202604.3 7.924645 -18.27090 18.27090 TRUE  
## 5 1202604.3 7.968689 -17.87030 17.87030 TRUE

## 3.4 Drop variables from a data set.

# -c() function tells R not to select the columns listed 1  
student <- subset(student, select = -c(LogWeight, ExpAge, SqrtHeight, BMI\_Neg,   
 BMI\_Pos, BMI\_Check))  
head(student, n=5)

## Name Sex Age Height Weight BMI BMI\_Class  
## 1 Alfred Male 14 69.0 112.5 16.61153 Underweight  
## 2 Alice Female 13 56.5 84.0 18.49855 Underweight  
## 3 Barbara Female 13 65.3 98.0 16.15679 Underweight  
## 4 Carol Female 14 62.8 102.5 18.27090 Underweight  
## 5 Henry Male 14 63.5 102.5 17.87030 Underweight

[subset()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/subset)

## 3.5 Sort a data set by a variable.

### a) Sort data set by a continuous variable.

student <- student[order(student$Age), ]  
# Notice that R uses a stable sorting algorithm by default  
head(student, n=5)

## Name Sex Age Height Weight BMI BMI\_Class  
## 11 Joyce Female 11 51.3 50.5 13.49000 Underweight  
## 18 Thomas Male 11 57.5 85.0 18.07335 Underweight  
## 6 James Male 12 57.3 83.0 17.77150 Underweight  
## 7 Jane Female 12 59.8 84.5 16.61153 Underweight  
## 10 John Male 12 59.0 99.5 20.09437 Healthy

### b) Sort data set by a categorical variable.

student <- student[order(student$Sex), ]  
# Notice that the data is now sorted first by Sex and then within Sex by Age   
head(student, n=5)

## Name Sex Age Height Weight BMI BMI\_Class  
## 11 Joyce Female 11 51.3 50.5 13.49000 Underweight  
## 7 Jane Female 12 59.8 84.5 16.61153 Underweight  
## 13 Louise Female 12 56.3 77.0 17.07770 Underweight  
## 2 Alice Female 13 56.5 84.0 18.49855 Underweight  
## 3 Barbara Female 13 65.3 98.0 16.15679 Underweight

[order()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/order)

## 3.6 Compute descriptive statistics of continuous variables, grouped by a categorical variable.

# Notice the syntax of Age, Height, Weight, and BMI as a function of Sex   
aggregate(cbind(Age, Height, Weight, BMI) ~ Sex, student, mean)

## Sex Age Height Weight BMI  
## 1 Female 13.22222 60.58889 90.11111 17.05104  
## 2 Male 13.40000 63.91000 108.95000 18.59424

[aggregate()](https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/aggregate) [cbind()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/cbind)

## 3.7 Add a new row to the bottom of a data set.

# Look at the tail of the data currently  
tail(student, n=5)

## Name Sex Age Height Weight BMI BMI\_Class  
## 1 Alfred Male 14 69.0 112.5 16.61153 Underweight  
## 5 Henry Male 14 63.5 102.5 17.87030 Underweight  
## 17 Ronald Male 15 67.0 133.0 20.82847 Healthy  
## 19 William Male 15 66.5 112.0 17.80451 Underweight  
## 15 Philip Male 16 72.0 150.0 20.34144 Healthy

# rbind.data.frame() function binds two data frames together by rows   
student <- rbind.data.frame(student, data.frame(Name='Jane', Sex = 'F', Age = 14,   
 Height = 56.3, Weight = 77.0,   
 BMI = 17.077695,   
 BMI\_Class = 'Underweight'))  
tail(student, n=5)

## Name Sex Age Height Weight BMI BMI\_Class  
## 5 Henry Male 14 63.5 102.5 17.87030 Underweight  
## 17 Ronald Male 15 67.0 133.0 20.82847 Healthy  
## 19 William Male 15 66.5 112.0 17.80451 Underweight  
## 15 Philip Male 16 72.0 150.0 20.34144 Healthy  
## 110 Jane F 14 56.3 77.0 17.07769 Underweight

[data.frame()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/data.frame) [rbind.data.frame()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/cbind)

## 3.8 Create a user defined function and apply it to a variable in the data set to create a new variable in the data set.

toKG <- function(lb) {  
 return(0.45359237 \* lb)  
}  
  
student$Weight\_KG <- toKG(student$Weight)  
head(student, n=5)

## Name Sex Age Height Weight BMI BMI\_Class Weight\_KG  
## 11 Joyce Female 11 51.3 50.5 13.49000 Underweight 22.90641  
## 7 Jane Female 12 59.8 84.5 16.61153 Underweight 38.32856  
## 13 Louise Female 12 56.3 77.0 17.07770 Underweight 34.92661  
## 2 Alice Female 13 56.5 84.0 18.49855 Underweight 38.10176  
## 3 Barbara Female 13 65.3 98.0 16.15679 Underweight 44.45205

# 4 More Advanced Data Wrangling

## 4.1 Drop observations with missing information.

# Notice the use of the fish data set because it has some missing   
# observations   
fish <- read.csv('/Users/fish.csv')  
  
# First sort by Weight, requesting those with NA for Weight first   
fish <- fish[order(fish$Weight, na.last=FALSE), ]  
head(fish, n=5)

## Species Weight Length1 Length2 Length3 Height Width  
## 14 Bream NA 29.5 32.0 37.3 13.9129 5.0728  
## 41 Roach 0.0 19.0 20.5 22.8 6.4752 3.3516  
## 73 Perch 5.9 7.5 8.4 8.8 2.1120 1.4080  
## 146 Smelt 6.7 9.3 9.8 10.8 1.7388 1.0476  
## 148 Smelt 7.0 10.1 10.6 11.6 1.7284 1.1484

new\_fish <- na.omit(fish)  
head(new\_fish, n=5)

## Species Weight Length1 Length2 Length3 Height Width  
## 41 Roach 0.0 19.0 20.5 22.8 6.4752 3.3516  
## 73 Perch 5.9 7.5 8.4 8.8 2.1120 1.4080  
## 146 Smelt 6.7 9.3 9.8 10.8 1.7388 1.0476  
## 148 Smelt 7.0 10.1 10.6 11.6 1.7284 1.1484  
## 147 Smelt 7.5 10.0 10.5 11.6 1.9720 1.1600

[na.omit()](https://www.rdocumentation.org/packages/data.table/versions/1.10.4/topics/na.omit.data.table)

## 4.2 Merge two data sets together on a common variable.

### a) First, select specific columns of a data set to create two smaller data sets.

# Notice the use of the student data set again, however we want to reload   
# it without the changes we've made previously   
student <- read.csv('/Users/class.csv')  
student1 <- subset(student, select=c(Name, Sex, Age))  
head(student1, n=5)

## Name Sex Age  
## 1 Alfred M 14  
## 2 Alice F 13  
## 3 Barbara F 13  
## 4 Carol F 14  
## 5 Henry M 14

student2 <- subset(student, select=c(Name, Height, Weight))  
head(student2, n=5)

## Name Height Weight  
## 1 Alfred 69.0 112.5  
## 2 Alice 56.5 84.0  
## 3 Barbara 65.3 98.0  
## 4 Carol 62.8 102.5  
## 5 Henry 63.5 102.5

### b) Second, we want to merge the two smaller data sets on the common variable.

new <- merge(student1, student2)  
head(new, n=5)

## Name Sex Age Height Weight  
## 1 Alfred M 14 69.0 112.5  
## 2 Alice F 13 56.5 84.0  
## 3 Barbara F 13 65.3 98.0  
## 4 Carol F 14 62.8 102.5  
## 5 Henry M 14 63.5 102.5

[merge()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/merge)

### c) Finally, we want to check to see if the merged data set is the same as the original data set.

all.equal(student, new)

## [1] TRUE

[all.equal()](https://www.rdocumentation.org/packages/data.table/versions/1.10.4/topics/all.equal)

## 4.3 Merge two data sets together by index number only.

### a) First, select specific columns of a data set to create two smaller data sets.

newstudent1 <- subset(student, select=c(Name, Sex, Age))  
head(newstudent1, n=5)

## Name Sex Age  
## 1 Alfred M 14  
## 2 Alice F 13  
## 3 Barbara F 13  
## 4 Carol F 14  
## 5 Henry M 14

newstudent2 <- subset(student, select=c(Height, Weight))  
head(newstudent2, n=5)

## Height Weight  
## 1 69.0 112.5  
## 2 56.5 84.0  
## 3 65.3 98.0  
## 4 62.8 102.5  
## 5 63.5 102.5

### b) Second, we want to join the two smaller data sets.

new2 <- cbind(newstudent1, newstudent2)  
head(new2, n=5)

## Name Sex Age Height Weight  
## 1 Alfred M 14 69.0 112.5  
## 2 Alice F 13 56.5 84.0  
## 3 Barbara F 13 65.3 98.0  
## 4 Carol F 14 62.8 102.5  
## 5 Henry M 14 63.5 102.5

### c) Finally, we want to check to see if the joined data set is the same as the original data set.

all.equal(student, new2)

## [1] TRUE

## 4.4 Create a pivot table to summarize information about a data set.

# Notice we are using a new data set that needs to be read into the   
# environment  
price <- read.csv('/Users/price.csv')  
  
# The package we need (dplyr) must first be  
# installed and then called to use  
require(dplyr)  
  
# The following code is used to remove the "," and "$" characters from the   
# ACTUAL column so that values can be summed   
price$ACTUAL <- gsub('[$]', '', price$ACTUAL)  
price$ACTUAL <- as.numeric(gsub(',', '', price$ACTUAL))  
  
filtered = group\_by(price, COUNTRY, STATE, PRODTYPE, PRODUCT)  
basic\_sum = summarise(filtered, REVENUE = sum(ACTUAL))  
head(basic\_sum, n=5)

## Source: local data frame [5 x 5]  
## Groups: COUNTRY, STATE, PRODTYPE [3]  
##   
## COUNTRY STATE PRODTYPE PRODUCT REVENUE  
## <fctr> <fctr> <fctr> <fctr> <dbl>  
## 1 Canada British Columbia FURNITURE BED 197706.6  
## 2 Canada British Columbia FURNITURE SOFA 216282.6  
## 3 Canada British Columbia OFFICE CHAIR 200905.2  
## 4 Canada British Columbia OFFICE DESK 186262.2  
## 5 Canada Ontario FURNITURE BED 194493.6

[dplyr](#DPLYR) [group\_by](https://www.rdocumentation.org/packages/dplyr/versions/0.5.0/topics/group_by) [summarise()](https://www.rdocumentation.org/packages/dplyr/versions/0.5.0/topics/summarise)

## 4.5 Return all unique values from a text variable.

print(unique(price$STATE))

## [1] California Colorado Florida   
## [4] Illinois New York North Carolina   
## [7] Texas Washington Baja California Norte  
## [10] Campeche Michoacan Nuevo Leon   
## [13] British Columbia Ontario Quebec   
## [16] Saskatchewan   
## 16 Levels: Baja California Norte British Columbia California ... Washington

[unique()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/unique)

# 5 Preparation & Basic Regression

## 5.1 Pre-process a data set using principal component analysis.

# Notice we are using a new data set that needs to be read into the   
# environment  
iris <- read.csv('/Users/iris.csv')  
features <- subset(iris, select = -c(Target))  
  
pca <- prcomp(x = features, scale = TRUE)  
print(pca)

## Standard deviations:  
## [1] 1.7061120 0.9598025 0.3838662 0.1435538  
##   
## Rotation:  
## PC1 PC2 PC3 PC4  
## sepal.length..cm. 0.5223716 -0.37231836 0.7210168 0.2619956  
## sepal.width..cm. -0.2633549 -0.92555649 -0.2420329 -0.1241348  
## petal.length..cm. 0.5812540 -0.02109478 -0.1408923 -0.8011543  
## petal.width..cm. 0.5656110 -0.06541577 -0.6338014 0.5235463

[prcomp()](https://stat.ethz.ch/R-manual/R-devel/library/stats/html/prcomp.html)

## 5.2 Split data into training and testing data and export as a .csv file.

# Set the sample size of the training data  
smp\_size <- floor(0.7 \* nrow(iris))  
  
# set.seed() is used to specify a seed for a random integer so that the   
# results are reproducible  
set.seed(29)  
train\_ind <- sample(seq\_len(nrow(iris)), size = smp\_size)  
  
train <- iris[train\_ind, ]  
test <- iris[-train\_ind, ]  
  
write.csv(train, file = "/Users/iris\_train.csv")  
write.csv(test, file = "/Users/iris\_test.csv")

[floor()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/Round) [nrow()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/nrow) [set.seed()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/Random) [sample()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/sample) [seq\_len()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/seq) [write.csv()](https://www.rdocumentation.org/packages/utils/versions/3.4.0/topics/write.table)

## 5.3 Fit a logistic regression model.

# Notice we are using a new data set that needs to be read into the   
# environment  
tips <- read.csv('/Users/tips.csv')  
  
# The following code is used to determine if the individual left more   
# than a 15% tip   
tips$fifteen <- 0.15 \* tips$total\_bill  
tips$greater15 <- ifelse(tips$tip > tips$fifteen, 1, 0)  
  
# Notice the syntax of greater15 as a function of total\_bill   
logreg <- glm(greater15 ~ total\_bill, data = tips, family = "binomial"(link='logit'))  
summary(logreg)

##   
## Call:  
## glm(formula = greater15 ~ total\_bill, family = binomial(link = "logit"),   
## data = tips)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.6757 -1.1766 0.8145 1.0145 2.0774   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 1.64772 0.35467 4.646 3.39e-06 \*\*\*  
## total\_bill -0.07248 0.01678 -4.319 1.57e-05 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 335.48 on 243 degrees of freedom  
## Residual deviance: 313.74 on 242 degrees of freedom  
## AIC: 317.74  
##   
## Number of Fisher Scoring iterations: 4

[glm()](https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/glm)

## 5.4 Fit a linear regression model.

# Notice we are using the same data set, but it needs to be read in again  
# without the previous changes we made  
tips <- read.csv('/Users/tips.csv')  
  
# Notice the syntax of tip as function of total\_bill  
linreg <- lm(tip ~ total\_bill, data = tips)  
summary(linreg)

##   
## Call:  
## lm(formula = tip ~ total\_bill, data = tips)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -3.1982 -0.5652 -0.0974 0.4863 3.7434   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 0.920270 0.159735 5.761 2.53e-08 \*\*\*  
## total\_bill 0.105025 0.007365 14.260 < 2e-16 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.022 on 242 degrees of freedom  
## Multiple R-squared: 0.4566, Adjusted R-squared: 0.4544   
## F-statistic: 203.4 on 1 and 242 DF, p-value: < 2.2e-16

# 6 Regression & Machine Learning: Modeling & Prediction

Many of the following models will make use of the [predict()](https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/predict) function.

## 6.1 Fit a logistic regression model on training data and assess against testing data.

### a) Fit a logistic regression model on training data.

# Notice we are using new data sets that need to be read into the environment  
train <- read.csv('/Users/tips\_train.csv')  
test <- read.csv('/Users/tips\_test.csv')  
  
# The following code is used to determine if the individual left more   
# than a 15% tip   
train$fifteen <- 0.15 \* train$total\_bill  
train$greater15 <- ifelse(train$tip > train$fifteen, 1, 0)  
test$fifteen <- 0.15 \* test$total\_bill  
test$greater15 <- ifelse(test$tip > test$fifteen, 1, 0)  
  
# Notice the syntax of greater15 as a function of total\_bill   
logreg <- glm(greater15 ~ total\_bill, data = train, family = "binomial"(link='logit'))  
summary(logreg)

##   
## Call:  
## glm(formula = greater15 ~ total\_bill, family = binomial(link = "logit"),   
## data = train)  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -1.6409 -1.1929 0.8144 1.0027 2.0381   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 1.64613 0.39459 4.172 3.02e-05 \*\*\*  
## total\_bill -0.07064 0.01849 -3.820 0.000134 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 267.61 on 194 degrees of freedom  
## Residual deviance: 250.58 on 193 degrees of freedom  
## AIC: 254.58  
##   
## Number of Fisher Scoring iterations: 4

### b) Assess the model against the testing data.

# Prediction on testing data  
predictions <- predict(logreg, test, type = 'response')  
predY <- ifelse(predictions < 0.5, 0, 1)  
  
# Determine how many were correctly classified  
Results <- ifelse(predY == test$greater15, "Correct", "Wrong")  
table(Results)

## Results  
## Correct Wrong   
## 34 15

## 6.2 Fit a linear regression model on training data and assess against testing data.

### a) Fit a linear regression model on training data.

# Notice we are using new data sets that need to be read into the environment  
train <- read.csv('/Users/boston\_train.csv')  
test <- read.csv('/Users/boston\_test.csv')  
  
# Notice the syntax of tip as function of total\_bill  
linreg <- lm(Target ~ ., data = train)  
summary(linreg)

##   
## Call:  
## lm(formula = Target ~ ., data = train)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -15.6466 -2.8461 -0.5395 1.7077 26.2160   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 36.108196 6.504968 5.551 5.73e-08 \*\*\*  
## X0 -0.085634 0.042774 -2.002 0.046077 \*   
## X1 0.046034 0.017150 2.684 0.007626 \*\*   
## X2 0.036413 0.076006 0.479 0.632186   
## X3 3.247961 1.074138 3.024 0.002686 \*\*   
## X4 -14.872938 4.636090 -3.208 0.001463 \*\*   
## X5 3.576869 0.536993 6.661 1.10e-10 \*\*\*  
## X6 -0.008703 0.016853 -0.516 0.605890   
## X7 -1.368905 0.252960 -5.412 1.18e-07 \*\*\*  
## X8 0.313120 0.082366 3.802 0.000170 \*\*\*  
## X9 -0.012882 0.004599 -2.801 0.005383 \*\*   
## X10 -0.976900 0.170996 -5.713 2.43e-08 \*\*\*  
## X11 0.011326 0.003359 3.372 0.000832 \*\*\*  
## X12 -0.526715 0.062563 -8.419 1.08e-15 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 4.988 on 340 degrees of freedom  
## Multiple R-squared: 0.7236, Adjusted R-squared: 0.7131   
## F-statistic: 68.48 on 13 and 340 DF, p-value: < 2.2e-16

### b) Assess the model against the testing data.

# Predict the tip based on the total\_bill given in the testing data   
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))  
prediction$target\_hat = predict(linreg, newdata = test)  
  
# Compute the squared difference between predicted tip and actual tip   
prediction$sq\_diff <- (prediction$target\_hat - test$Target)\*\*2  
  
# Compute the mean of the squared differences (mean squared error)   
# as an assessment of the model   
mean\_sq\_error <- mean(prediction$sq\_diff)  
print(mean\_sq\_error)

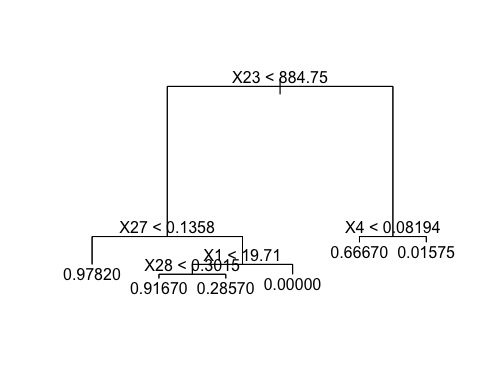
## [1] 17.77131

## 6.3 Fit a decision tree model on training data and assess against testing data.

### a) Fit a decision tree classification model.

#### i) Fit a decision tree classification model on training data, plot the tree, and determine variable importance.

# Notice we are using new data sets that need to be read into the environment  
train <- read.csv('/Users/breastcancer\_train.csv')  
test <- read.csv('/Users/breastcancer\_test.csv')  
  
# The package we need to fit a tree model (tree) must first be  
# installed and then called to use  
library(tree)  
  
# The "." character tells the model to use all variables except the response   
# variabe (Target)  
treeMod <- tree(Target ~ ., data = train, method = "class")  
  
# Plot the decision tree  
plot(treeMod)  
text(treeMod)



# Determine variable importance  
summary(treeMod)

##   
## Regression tree:  
## tree(formula = Target ~ ., data = train, method = "class")  
## Variables actually used in tree construction:  
## [1] "X23" "X27" "X1" "X28" "X4"   
## Number of terminal nodes: 6   
## Residual mean deviance: 0.02688 = 10.54 / 392   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -0.97820 -0.01575 0.02183 0.00000 0.02183 0.98430

#### ii) Assess the model against the testing data.

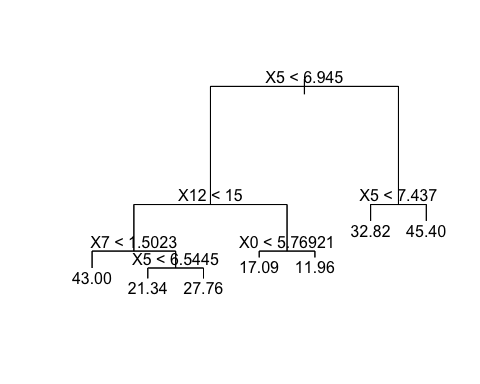
# Prediction on testing data  
out <- predict(treeMod, test)  
out <- unname(out)  
pred.response <- ifelse(out < 0.5, 0, 1)   
  
# Determine how many were correctly classified  
Results <- ifelse(test$Target == pred.response, "Correct", "Wrong")  
table(Results)

## Results  
## Correct Wrong   
## 159 12

### b) Fit a decision tree regression model.

#### i) Fit a decision tree regression model on training data, plot the tree, and determine variable importance.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/boston\_train.csv')  
test <- read.csv('/Users/boston\_test.csv')  
  
treeMod <- tree(Target ~ ., data = train, method = "anova")  
  
# Plot the decision tree  
plot(treeMod)  
text(treeMod)



# Determine variable importance  
summary(treeMod)

##   
## Regression tree:  
## tree(formula = Target ~ ., data = train, method = "anova")  
## Variables actually used in tree construction:  
## [1] "X5" "X12" "X7" "X0"   
## Number of terminal nodes: 7   
## Residual mean deviance: 14.67 = 5091 / 347   
## Distribution of residuals:  
## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -28.0000 -1.8070 0.3264 0.0000 2.2320 10.0100

#### ii) Assess the model against the testing data.

# Predict the tip based on the total\_bill given in the testing data   
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))  
prediction$target\_hat = predict(treeMod, newdata = test)  
  
# Compute the squared difference between predicted tip and actual tip   
prediction$sq\_diff <- (prediction$target\_hat - test$Target)\*\*2  
  
# Compute the mean of the squared differences (mean squared error)   
# as an assessment of the model   
mean\_sq\_error <- mean(prediction$sq\_diff)  
print(mean\_sq\_error)

## [1] 25.12126

[tree](#TREE)

## 6.4 Fit a random forest model on training data and assess against testing data.

### a) Fit a random forest classification model.

#### i) Fit a random forest classification model on training data and determine variable importance.

# Notice we are using new data sets that need to be read into the environment  
train <- read.csv('/Users/breastcancer\_train.csv')  
test <- read.csv('/Users/breastcancer\_test.csv')  
  
# The package we need to fit a random forest model (randomForest) must   
# first be installed and then called to use  
require(randomForest)  
set.seed(29)  
  
rfMod <- randomForest(as.factor(Target) ~ ., data = train)  
  
# Determine variable importance  
print(importance(rfMod))

## MeanDecreaseGini  
## X0 7.4350001  
## X1 3.2158122  
## X2 8.1031917  
## X3 10.0537757  
## X4 1.6386749  
## X5 2.0837671  
## X6 9.6116813  
## X7 21.6437079  
## X8 0.7307710  
## X9 0.6775099  
## X10 2.6091324  
## X11 1.4562832  
## X12 3.2163635  
## X13 7.1262562  
## X14 0.9956042  
## X15 0.7235749  
## X16 0.9234519  
## X17 0.9404855  
## X18 0.7108068  
## X19 0.9439451  
## X20 18.9098892  
## X21 3.9869086  
## X22 21.5688509  
## X23 20.1310343  
## X24 2.6803628  
## X25 2.3652153  
## X26 8.0234990  
## X27 22.8845187  
## X28 2.2314638  
## X29 1.4673965

#### ii) Assess the model against the testing data.

# Prediction on testing data  
pred.response <- predict(rfMod, test)  
  
# Determine how many were correctly classified  
Results <- ifelse(test$Target == pred.response, "Correct", "Wrong")  
table(Results)

## Results  
## Correct Wrong   
## 166 5

### b) Fit a random forest regression model.

#### i) Fit a random forest regression model on training data and determine variable importance.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/boston\_train.csv')  
test <- read.csv('/Users/boston\_test.csv')  
  
# The package we need to fit a random forest model (randomForest) must   
# first be installed and then called to use  
require(randomForest)  
set.seed(29)  
  
rfMod <- randomForest(Target ~ ., data = train)  
  
# Determine variable importance  
print(importance(rfMod))

## IncNodePurity  
## X0 2147.2876  
## X1 246.8517  
## X2 1915.5704  
## X3 230.8659  
## X4 1776.4803  
## X5 8451.8355  
## X6 850.6732  
## X7 2105.0723  
## X8 298.5322  
## X9 939.6180  
## X10 1739.6930  
## X11 651.7589  
## X12 8662.2980

#### ii) Assess the model against the testing data.

# Predict the Target in the testing data   
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))  
prediction$target\_hat = predict(rfMod, newdata = test)  
  
# Compute the squared difference between predicted tip and actual tip   
prediction$sq\_diff <- (prediction$target\_hat - test$Target)\*\*2  
  
# Compute the mean of the squared differences (mean squared error)   
# as an assessment of the model   
mean\_sq\_error <- mean(prediction$sq\_diff)  
print(mean\_sq\_error)

## [1] 9.028163

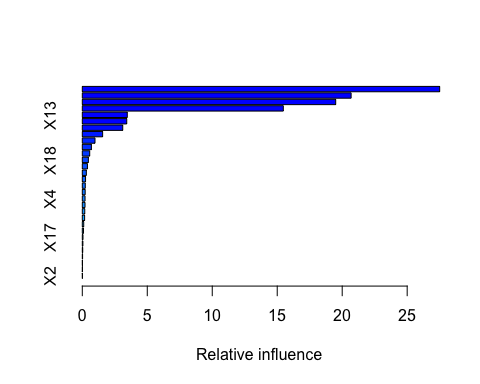
[randomForest](#RANDOM)

## 6.5 Fit a gradient boosting model on training data and assess against testing data.

### a) Fit a gradient boosting classification model.

#### i) Fit a gradient boosting classification model on training data and determine variable importance.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/breastcancer\_train.csv')  
test <- read.csv('/Users/breastcancer\_test.csv')  
  
# The package we need to fit a gradient boosting model (gbm) must first   
# be installed and then called to use  
require(gbm)  
set.seed(29)  
  
# distribution = "bernoulli" is appropriate when there are only 2   
# unique values  
# n.trees = total number of trees to fit which is analogous to the number   
# of iterations  
# shrinkage = learning rate or step-size reduction, whereas a lower   
# learning rate requires more iterations  
gbMod <- gbm(Target ~ ., distribution = "bernoulli", data = train, n.trees = 2500, shrinkage = .01)  
  
# Determine variable importance  
summary(gbMod)



## var rel.inf  
## X27 X27 27.501030040  
## X7 X7 20.685752752  
## X23 X23 19.499763247  
## X22 X22 15.467662358  
## X13 X13 3.461620283  
## X20 X20 3.420135928  
## X21 X21 3.112590280  
## X1 X1 1.565270103  
## X26 X26 0.977012668  
## X24 X24 0.699708897  
## X28 X28 0.570461255  
## X18 X18 0.467509741  
## X11 X11 0.395889286  
## X15 X15 0.310587247  
## X29 X29 0.250279911  
## X3 X3 0.228744132  
## X25 X25 0.221418596  
## X4 X4 0.211240553  
## X14 X14 0.209380216  
## X6 X6 0.190119477  
## X9 X9 0.178909422  
## X19 X19 0.109630024  
## X10 X10 0.091983934  
## X17 X17 0.053485704  
## X8 X8 0.045784794  
## X12 X12 0.033088702  
## X5 X5 0.021833418  
## X16 X16 0.008407155  
## X0 X0 0.008278750  
## X2 X2 0.002421124

#### ii) Assess the model against the testing data.

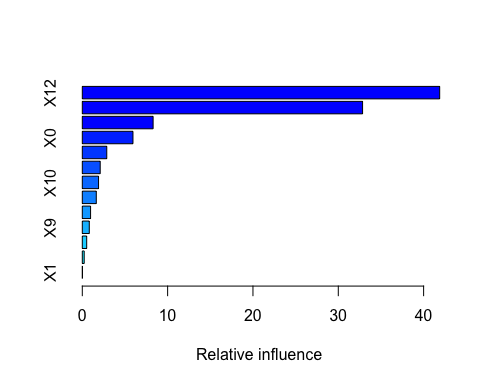
# Prediction on testing data  
out <- predict(object = gbMod, newdata = test,   
 type = "response", n.trees = 2500)  
pred.response <- ifelse(out < 0.5, 0, 1)  
  
# Determine how many were correctly classified  
Results <- ifelse(test$Target == pred.response, "Correct", "Wrong")  
table(Results)

## Results  
## Correct Wrong   
## 167 4

### b) Fit a gradient boosting regression model.

#### i) Fit a gradient boosting regression model on training data and determine variable importance.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/boston\_train.csv')  
test <- read.csv('/Users/boston\_test.csv')  
  
# The package we need to fit a gradient boosting model (gbm) must first   
# be installed and then called to use  
require(gbm)  
set.seed(29)  
  
# distribution = "bernoulli" is appropriate when there are only 2   
# unique values  
# n.trees = total number of trees to fit which is analogous to the number   
# of iterations  
# shrinkage = learning rate or step-size reduction, whereas a lower   
# learning rate requires more iterations  
  
# Fit the model, dividing target by 50 for scaling  
gbMod <- gbm(Target / 50 ~ ., data = train, distribution = "gaussian", n.trees = 2500, shrinkage = .01)  
  
# Determine variable importance  
summary(gbMod)



## var rel.inf  
## X12 X12 41.8822593  
## X5 X5 32.8464221  
## X7 X7 8.2981404  
## X0 X0 5.9373914  
## X4 X4 2.8678768  
## X11 X11 2.0983248  
## X10 X10 1.9043257  
## X3 X3 1.6390562  
## X6 X6 0.9700331  
## X9 X9 0.8179346  
## X8 X8 0.5157516  
## X2 X2 0.2224838  
## X1 X1 0.0000000

#### ii) Assess the model against the testing data.

# Predict the Target in the testing data, remembeing to multiply by 50  
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))  
prediction$target\_hat <- predict(object = gbMod, newdata = test,   
 type = "response", n.trees = 2500)\*50  
  
# Compute the squared difference between predicted tip and actual tip   
prediction$sq\_diff <- (prediction$target\_hat - test$Target)\*\*2  
  
# Compute the mean of the squared differences (mean squared error)   
# as an assessment of the model   
mean\_sq\_error <- mean(prediction$sq\_diff)  
print(mean\_sq\_error)

## [1] 11.88728

[gbm](#GBM)

## 6.6 Fit an extreme gradient boosting model on training data and assess against testing data.

### a) Fit an extreme gradient boosting classification model.

#### i) Fit an extreme gradient boosting classification model on training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/breastcancer\_train.csv')  
test <- read.csv('/Users/breastcancer\_test.csv')  
  
# The package we need to fit an extreme gradient boosting model (xgboost)   
# must first be installed and then called to use  
require(xgboost)  
set.seed(29)  
  
# Fit the model  
xgbMod <- xgboost(data.matrix(subset(train, select = -c(Target))),   
 data.matrix(train$Target), max\_depth = 3, nrounds = 2,  
 objective = "binary:logistic", n\_estimators = 2500,   
 shrinkage = .01)

## [1] train-error:0.037688   
## [2] train-error:0.020101

#### ii) Assess the model against the testing data.

# Prediction on testing data  
predictions <- predict(xgbMod, data.matrix(subset(test, select = -c(Target))))  
pred.response <- ifelse(predictions < 0.5, 0, 1)  
  
# Determine how many were correctly classified  
Results <- ifelse(test$Target == pred.response, "Correct", "Wrong")  
table(Results)

## Results  
## Correct Wrong   
## 165 6

### b) Fit an extreme gradient boosting regression model.

#### i) Fit an extreme gradient boosting regression model on training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/boston\_train.csv')  
test <- read.csv('/Users/boston\_test.csv')  
  
# The package we need to fit an extreme gradient boosting model (xgboost)   
# must first be installed and then called to use  
require(xgboost)  
set.seed(29)  
  
# Fit the model, dividing target by 50 for scaling  
xgbMod <- xgboost(data.matrix(subset(train, select = -c(Target))),   
 data.matrix(train$Target / 50), max\_depth = 3, nrounds = 2,  
 n\_estimators = 2500, shrinkage = .01)

## [1] train-rmse:0.146609   
## [2] train-rmse:0.114851

#### ii) Assess the model against the testing data.

# Predict the target in the testing data, remembering to   
# multiply by 50   
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))  
prediction$target\_hat <- predict(xgbMod,   
 data.matrix(subset(test, select = -c(Target))))\*50  
  
# Compute the squared difference between predicted tip and actual tip   
prediction$sq\_diff <- (prediction$target\_hat - test$Target)\*\*2  
  
# Compute the mean of the squared differences (mean squared error)   
# as an assessment of the model   
mean\_sq\_error <- mean(prediction$sq\_diff)  
print(mean\_sq\_error)

## [1] 36.13079

[xgboost](#XGBOOST)

## 6.7 Fit a support vector model on training data and assess against testing data.

### a) Fit a support vector classification model.

#### i) Fit a support vector classification model on training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/breastcancer\_train.csv')  
test <- read.csv('/Users/breastcancer\_test.csv')  
  
# The package we need to fit an svm model (e1071) must first   
# be installed and then called to use  
library(e1071)  
  
# Fit a support vector classification model   
svMod <- svm(Target ~ ., train, type = 'C-classification', kernel = 'linear')

#### ii) Assess the model against the testing data.

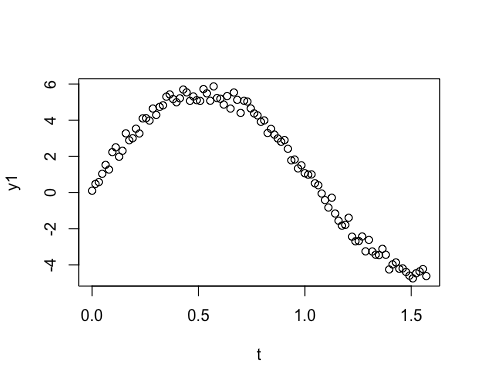
# Prediction on testing data  
pred.response <- unname(predict(svMod, test))  
  
# Determine how many were correctly classified  
Results <- ifelse(test$Target == pred.response, "Correct", "Wrong")  
table(Results)

## Results  
## Correct Wrong   
## 166 5

### b) Fit a support vector regression model.

#### i) Generate random data based on a sine curve.

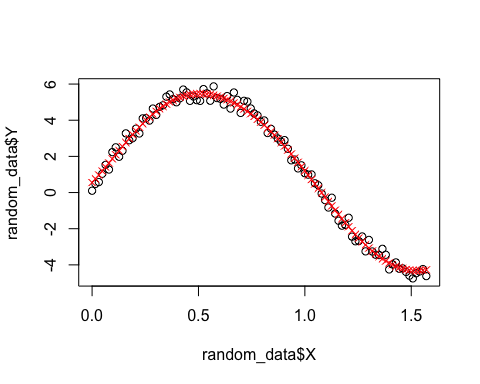
set.seed(29)  
# Generate the time variable  
t <- seq(from = 0, to = 0.5\*pi,length.out=100)  
  
# Generate the sine curve with uniform noise  
y1 <- 5\*sin(3\*t) + runif(100)  
  
# Create a data frame for the generated data  
random\_data <- data.frame(matrix(ncol=0, nrow = 100))  
random\_data$X <- t  
random\_data$Y <- y1  
  
# Plot the generated data  
plot(t,y1)



[seq()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/seq) [runif()](https://www.rdocumentation.org/packages/stats/versions/3.4.0/topics/Uniform)

#### ii) Fit a support vector regression model to the data and assess the predictions.

# The package we need to fit an svm model (e1071) must first   
# be installed and then called to use  
library(e1071)  
  
model <- svm(Y ~ X, random\_data)  
predictedY <- predict(model, random\_data)  
plot.new()  
plot(random\_data$X, random\_data$Y)  
points(random\_data$X, predictedY, col = "red", pch = 4)



prediction = data.frame(matrix(ncol = 0, nrow = nrow(random\_data)))  
prediction$predY <- predictedY  
prediction$actual <- random\_data$Y  
prediction$sq\_diff <- (prediction$predY - prediction$actual)\*\*2  
print(mean(prediction$sq\_diff))

## [1] 0.08052554

[e1071](https://cran.r-project.org/web/packages/e1071/e1071.pdf) [svm()](https://www.rdocumentation.org/packages/e1071/versions/1.6-8/topics/svm)

## 6.8 Fit a neural network model on training data and assess against testing data.

### a) Fit a neural network classification model.

#### i) Fit a neural network classification model on training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/digits\_train.csv')  
test <- read.csv('/Users/digits\_test.csv')  
  
trainInputs <- subset(train, select = -c(Target))  
testInputs <- subset(test, select = -c(Target))  
  
# The package we need to fit a neural network model (RSNNS) must   
# first be installed and then called to use  
library(RSNNS)  
  
set.seed(29)  
  
trainTarget <- decodeClassLabels(train$Target)  
testTarget <- decodeClassLabels(test$Target)  
  
nnMod <- mlp(trainInputs, trainTarget, inputsTest=testInputs, targetsTest=testTarget, size = 100, maxit = 200)

#### ii) Assess the model against the testing data.

predictions <- predict(nnMod, testInputs)  
confusionMatrix(testTarget, predictions)

## predictions  
## targets 1 2 3 4 5 6 7 8 9 10  
## 1 55 0 0 0 1 0 1 1 0 0  
## 2 0 54 2 0 0 0 1 0 1 0  
## 3 0 0 58 0 0 0 0 0 0 0  
## 4 0 0 0 56 0 1 0 2 0 0  
## 5 0 0 0 0 53 0 0 0 1 0  
## 6 0 0 0 0 0 58 1 0 0 0  
## 7 0 0 0 0 0 0 41 0 0 0  
## 8 0 1 0 0 0 0 0 49 0 1  
## 9 1 3 0 3 0 0 0 0 36 2  
## 10 0 1 0 0 0 1 0 2 1 52

[confusionMatrix()](https://www.rdocumentation.org/packages/RSNNS/versions/0.4-9/topics/confusionMatrix)

### b) Fit a neural network regression model.

#### i) Fit a neural network regression model on training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/boston\_train.csv')  
test <- read.csv('/Users/boston\_test.csv')  
  
# Scale input data  
scaled\_train <- data.frame(scale(subset(train, select = -c(Target))))  
scaled\_test <- data.frame(scale(subset(test, select = -c(Target))))  
  
# Fit neural network regression model, dividing target by 50 for scaling  
nnMod <- mlp(scaled\_train, train$Target / 50, inputsTest=scaled\_test, targetsTest=test$Target / 50, maxit = 1000)

[scale()](https://www.rdocumentation.org/packages/raster/versions/2.5-8/topics/scale)

# Assess against testing data, remembering to multiply by 50  
preds = data.frame(matrix(ncol = 0, nrow = nrow(test)))  
preds$predY <- predict(nnMod, scaled\_test)\*50  
preds$target <- test$Target  
preds$sq\_error <- (preds$predY - preds$target)\*\*2  
print(mean(preds$sq\_error))

## [1] 24.9167

[RSNNS](#RSNNS)

# 7 Model Evaluation & Selection

## 7.1 Evaluate the accuracy of regression models.

### a) Evaluation on training data.

# Notice we are re-using data sets but it is good to re-read the original   
# version back into the environment  
train <- read.csv('/Users/boston\_train.csv')  
test <- read.csv('/Users/boston\_test.csv')  
  
# Random Forest Regression Model  
set.seed(29)  
rfMod <- randomForest(Target ~ ., data = train)  
  
# Evaluation on training data  
pred\_rf <- predict(rfMod, train)  
pred\_rf <- unname(pred\_rf)  
  
# Determine coefficient of determination score  
r2\_rf <- 1 - ( (sum((train$Target - pred\_rf)\*\*2)) / (sum((train$Target - mean(train$Target))\*\*2)) )  
print(paste0("Random forest regression model r^2 score (coefficient of determination): ", r2\_rf))

## [1] "Random forest regression model r^2 score (coefficient of determination): 0.972080769152132"

[unname()](https://www.rdocumentation.org/packages/base/versions/3.4.0/topics/unname)

### b) Evaluation on testing data.

# Random Forest Regression Model (rfMod)   
  
# Evaluation on testing data  
pred\_rf <- predict(rfMod, test)  
pred\_rf <- unname(pred\_rf)  
  
# Determine coefficient of determination score  
r2\_rf = 1 - ( (sum((test$Target - pred\_rf)\*\*2)) / (sum((test$Target - mean(test$Target))\*\*2)) )  
print(paste0("Random forest regression model r^2 score (coefficient of determination): ", r2\_rf))

## [1] "Random forest regression model r^2 score (coefficient of determination): 0.886681832095677"

The formula used here for the coefficient score is based off the Python skearn formula for [r2\_score](http://scikit-learn.org/stable/modules/model_evaluation.html#r2-score-the-coefficient-of-determination). For more information about model assessment in R, please review information about the R package [caret](https://cran.r-project.org/web/packages/caret/index.html).

## 7.2 Evaluate the accuracy of classification models.

### a) Evaluation on training data.

# Notice we are using new data sets that need to be read into   
# the environment  
train <- read.csv('/Users/digits\_train.csv')  
test <- read.csv('/Users/digits\_test.csv')  
set.seed(29)  
  
# Random Forest Classification Model  
  
rfMod <- randomForest(as.factor(Target) ~ ., data = train)  
  
# Evaluation on training data  
pred\_rf <- predict(rfMod, train)  
pred\_rf <- unname(pred\_rf)  
  
# Determine accuracy score  
accuracy\_rf <- (1/nrow(train)) \* sum(as.numeric(pred\_rf == train$Target))  
print(paste0("Random forest model accuracy: ", accuracy\_rf))

## [1] "Random forest model accuracy: 1"

### b) Evaluation on testing data.

# Random Forest Classification Model (rfMod)  
  
# Evaluation on testing data  
pred\_rf <- predict(rfMod, test)  
pred\_rf <- unname(pred\_rf)  
  
# Determine accuracy score  
accuracy\_rf <- (1/nrow(test)) \* sum(as.numeric(pred\_rf == test$Target))  
print(paste0("Random forest model accuracy: ", accuracy\_rf))

## [1] "Random forest model accuracy: 0.974074074074074"

The formula used here for the accuracy score is based off the Python skearn formula for [accuracy\_score](http://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score). For more information about model assessment in R, please review information about the R package [caret](https://cran.r-project.org/web/packages/caret/index.html).

## 7.3 Evaluation with cross validation.

### a) KFold

# Notice we are using a new data set that needs to be read into the   
# environment   
breastcancer = read.csv('/Users/breastcancer.csv')  
  
# The packages we need (caret & randomForest) must first   
# be installed and then called to use  
library(caret)  
library(randomForest)  
  
# Create the 5 cross validation folds  
train\_control <- trainControl(method = "cv", number = 5, savePredictions = TRUE)  
  
# Convert Target into a factor variable for the random forest model  
breastcancer$Target <- factor(breastcancer$Target, levels = c(1,0),   
 labels = c(1, 0))  
  
# Train the model, using the 5 cross validation folds  
model <- train(Target~., data = breastcancer, trControl = train\_control, method = "rf")  
  
# Assess the accuracy of the model  
tab <- model$pred  
tab$correct <- (tab$pred == tab$obs)  
tab$correct\_num <- ifelse(tab$correct=="TRUE", 1, 0)  
aggdata <- unname(as.matrix(aggregate(correct\_num ~ Resample, tab, sum)))  
aggdata <- as.numeric(aggdata[,2])  
counts <- unname(table(tab$Resample))  
accuracy <- c(0,0,0,0,0)  
for (i in 1:5) {  
 accuracy[i] <- aggdata[i]/counts[i]  
}  
  
print(paste0("Accuracy: ", round(mean(accuracy)\*100, digits=2), "% +/- ", round(sd(accuracy)\*100, digits=2), "%"))

## [1] "Accuracy: 95.84% +/- 1.22%"

[caret](#CARET)

### b) ShuffleSplit

# Notice we are using a new data set that needs to be read into the   
# environment   
breastcancer = read.csv('/Users/breastcancer.csv')  
  
# The package we need to create a data partition (caret) must first   
# be installed and then called to use  
require(caret)  
require(randomForest)  
set.seed(29)  
  
X = subset(breastcancer, select = -c(Target))  
Y = breastcancer$Target  
  
# Create the data partition  
trainIndex <- createDataPartition(Y, times = 5, p = 0.7, list = FALSE)  
accuracy <- c(0, 0, 0, 0, 0)  
  
for (i in 1:5) {  
 nam <- paste("data\_train", i, sep ="")  
 assign(nam, breastcancer[trainIndex[,i],])  
 nam <- paste("data\_test", i, sep ="")  
 assign(nam, breastcancer[-trainIndex[,i],])  
}  
  
data\_train <- list(data\_train1, data\_train2, data\_train3, data\_train4, data\_train5)  
data\_test <- list(data\_test1, data\_test2, data\_test3, data\_test4, data\_test5)  
  
# Train the model and assess the accuracy  
for (i in 1:5) {  
 fit <- randomForest(as.factor(Target) ~ ., data = data\_train[[i]])  
 Prediction <- predict(fit, data\_test[[i]])  
 Prediction <- unname(Prediction)  
 correct <- (data\_test[[i]]$Target == Prediction)  
 counts <- unname(table(correct))  
 accuracy[i] <- counts[2] / sum(counts)  
}  
  
print(paste0("Accuracy: ", round(mean(accuracy)\*100, digits=2), "% +/- ", round(sd(accuracy)\*100, digits=2), "%"))

## [1] "Accuracy: 94.24% +/- 1.05%"

[createDataPartition](https://www.rdocumentation.org/packages/caret/versions/6.0-76/topics/createDataPartition)

# Appendix

## 1 Built-in R-Objects

### [Vectors](#vectors)

* [Logical](#logical)
* [Numeric](#numeric)
* [Integer](#int)
* [Complex](#complex)
* [Character](#char)
* [Raw](#raw)

### [Lists](#list)

### [Matrics](#matrix)

### [Arrays](#array)

### [Factors](#factors)

### [Data Frames](#DataFrame)

# Alphabetical Index

## [Array](https://www.tutorialspoint.com/r/r_arrays.htm)

A one-dimensional data frame. Please see the following example of array creation and access:

my\_array <- c(1, 3, 5, 9)  
print(my\_array)

## [1] 1 3 5 9

print(my\_array[1])

## [1] 1

## [caret](https://www.rdocumentation.org/packages/caret/versions/6.0-76)

An R programming package of tools for training and plotting classification and regression models.

## [Data Frame](https://stat.ethz.ch/R-manual/R-devel/library/base/html/data.frame.html)

An R Data Frame is a two-dimensional tabular structure with labeled axes (rows and columns), where data observations are represented by rows and data variables are represented by columns.

## Dictionary

A dictionary is an associative array which is indexed by keys which map to values. Therefore, a dictionary is an unordered set of key:value pairs where each key is unique. In R, a dictionary can be implemented using a [named list](http://www.r-tutor.com/r-introduction/list/named-list-members). Please see the following example of named list creation and access:

student <- read.csv('/Users/class.csv')  
values <- student$Age  
names(values) <- student$Name  
print(values["James"])

## James   
## 12

## [dplyr](https://cran.r-project.org/web/packages/dplyr/dplyr.pdf)

An R programming package of tools for workng with [data frame](#DataFrame) like objects.

## [gbm](https://cran.r-project.org/web/packages/gbm/gbm.pdf)

An R programming package useful for building and analyzing gradient boosting models.

## [gdata](https://cran.r-project.org/web/packages/gdata/gdata.pdf)

An R programming package of tools useful for data manipulation.

## [List](http://www.r-tutor.com/r-introduction/list)

An R list is a sequence of comma-separated objects that need not be of the same type. Please see the following example of list creation and access:

list1 <- list('item1', 102)  
print(list1)

## [[1]]  
## [1] "item1"  
##   
## [[2]]  
## [1] 102

print(list1[1])

## [[1]]  
## [1] "item1"

## [randomForest](https://cran.r-project.org/web/packages/randomForest/randomForest.pdf)

An R programming package of tools useful for building and analyzing classification and regression random forest models.

## [RSNNS](https://cran.r-project.org/web/packages/RSNNS/RSNNS.pdf) [#RSNNS]

An R programming package of tools useful for building and analyzing classification and regression neural network models.

## [rjson](https://cran.r-project.org/web/packages/rjson/rjson.pdf)

An R programming package of tools useful for converting R objects into JSON objects, and JSON objects into R objects.

## [tree](https://cran.r-project.org/web/packages/tree/index.html)

An R programming package of tools useful for building and analyzing classification and regression decision trees.

## [Vector](https://www.tutorialspoint.com/r/r_data_types.htm)

A one-dimensional data structure which is able to hold different classes of elements, but only one class per vector.

## [xgboost](https://cran.r-project.org/web/packages/xgboost/xgboost.pdf)

An R programming package of tools useful for building and analyzing classification and regression extreme gradient boosting models.

For more information on R packages and functions, along with helpful examples, please see [R](https://www.rdocumentation.org/).