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')

### 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)

### 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

## 1.2 Find the dimensions of the data set.

Information about an R [data frame](#DataFrame) is available by calling the "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 ...

## 1.4 Look at the first 5 observations.

The first 5 observations of a [data frame](#DataFrame) are available by calling the "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() 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

## 1.6 Compute summary statistics of the data set.

Summary statistics of a [data frame](#DataFrame) are available by calling the "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

## 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

## 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

## 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")

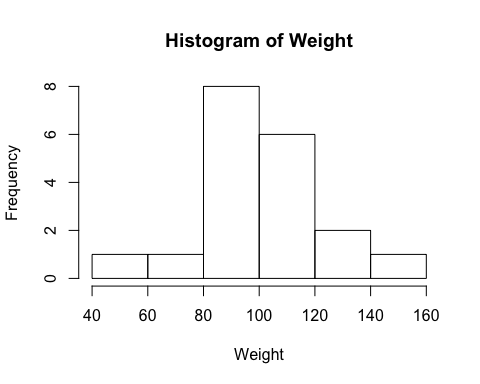
## Height Weight  
## Height 1.0000000 0.8777852  
## Weight 0.8777852 1.0000000

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

# 2 Basic Graphing and Plotting Functions

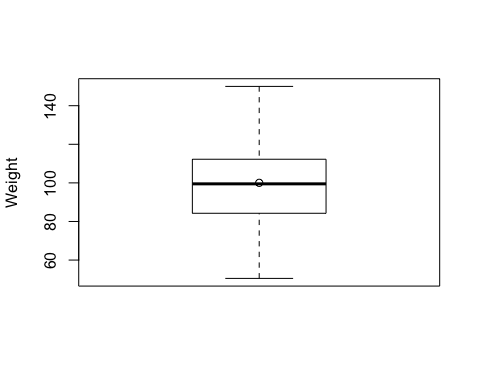
## 2.1 Visualize a single continuous variable by producing a histogram.

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



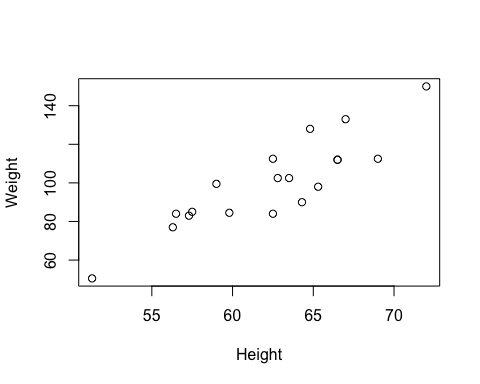
## 2.2 Visualize a single continuous variable by producing a boxplot.

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



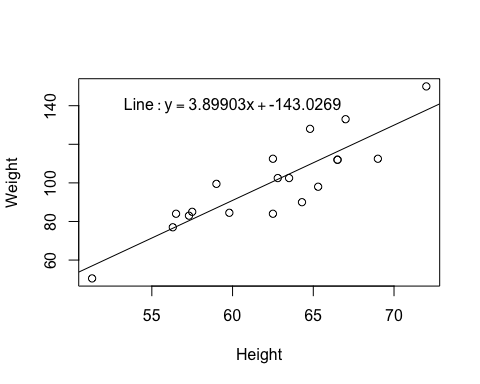
## 2.3 Visualize two continuous variables by producing a scatterplot.

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



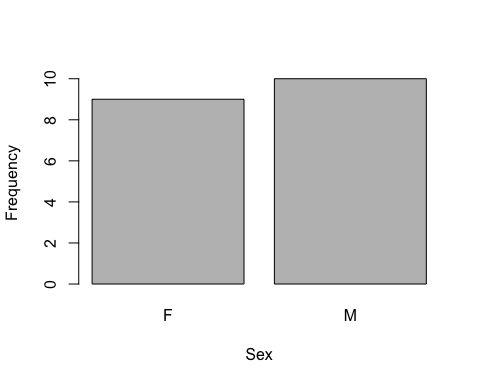
## 2.4 Visualize a relationship between two continuous variables by producing a scatterplot and a plotted line of best fit.

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)))



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

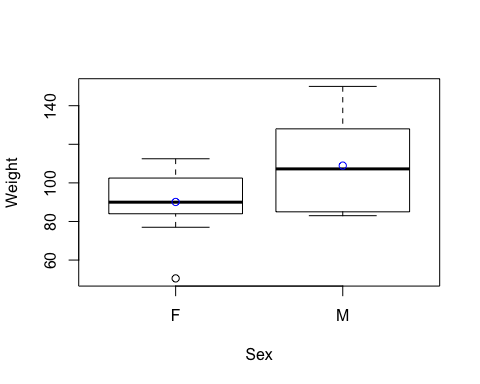
counts <- table(student$Sex)  
# beside = TRUE indicates to print the bars side by side instead of on top of   
# each other   
# names.arg indicates which names to use to label the bars   
barplot(counts, beside=TRUE, ylab= "Frequency", xlab= "Sex", names.arg=names(counts))



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

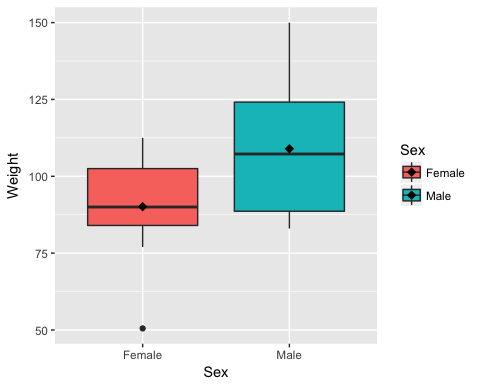
### a) Simple side-by-side boxplot without color.

# 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)

# 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() function, the exp() function, the sqrt() function, and the abs() 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

## 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

## 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

## 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

## 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

## 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

### 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

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

## 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

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

## 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 to fix the ACTUAL column (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)

# 5 Regression & Modeling

## 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")

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

## 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   
reg <- glm(greater15 ~ total\_bill, data = tips, family = "binomial"(link='logit'))  
summary(reg)

##   
## 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 on training data and assess against testing 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')  
  
# Fit a linear regression model of tip by total\_bill on the training data   
m <- lm(tip ~ total\_bill, data = train)  
  
# Predict the tip based on the total\_bill given in the testing data   
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))  
prediction$tip\_hat = predict(m, newdata = test)  
  
# Compute the squared difference between predicted tip and actual tip   
prediction$diff <- (prediction$tip\_hat - test$tip)\*\*2  
  
# Compute the mean of the squared differences (mean squared error)   
# as an assessment of the model   
mean\_sq\_error <- mean(prediction$diff)  
print(mean\_sq\_error)

## [1] 1.087594

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

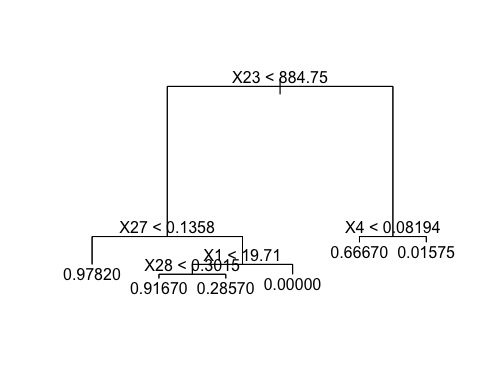
### a) Fit a decision tree model on training data and do not prune the model before assessing against testing data.

#### i. Assess the model against the 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)  
  
# Prediction on training data   
out <- predict(treeMod)  
out <- unname(out)  
  
# If the prediction probability is less than 0.5, classify this as a 0  
# and otherwise classify as a 1. This isn't the best method -- a better   
# method would be randomly assigning a 0 or 1 when a probability of 0.5   
# occurrs, but this insures that results are consistent.  
pred.response <- ifelse(out < 0.5, 0, 1)  
  
# Determine how many were correctly classified  
correct <- (train$Target == pred.response)  
table(correct)

## correct  
## FALSE TRUE   
## 12 386

# Plot the decision tree  
plot(treeMod)  
text(treeMod)



# Determine variable importance  
summary(treeMod)

##   
## Regression tree:  
## tree(formula = Target ~ ., data = train)  
## 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.

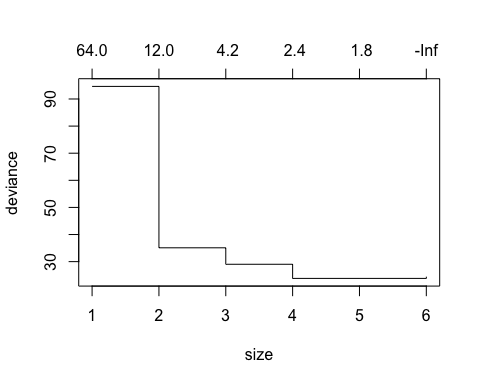
# Prediction on testing data  
out <- predict(treeMod, test)  
out <- unname(out)  
pred.response <- ifelse(out < 0.5, 0, 1)   
  
# Determine how many were correctly classified  
correct <- (test$Target == pred.response)  
table(correct)

## correct  
## FALSE TRUE   
## 12 159

### b) Fit a decision tree model on training data and prune the model before assessing against testing data.

#### i. Assess the model against the training data, plot the tree, and determine variable importance.

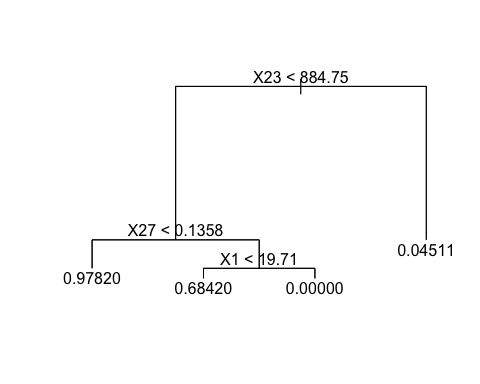
treeMod <- tree(Target ~ ., data = train)  
  
# Run the cross validation to determine where to prune  
cvTree <- cv.tree(treeMod, rand = c(1,0), FUN = prune.tree)   
  
# Plot to see where to prune  
plot(cvTree)



# Set size corresponding to lowest value in below plot  
treePrunedMod <- prune.tree(treeMod, best = 4)   
  
# Prediction on training data   
out <- predict(treePrunedMod)  
out <- unname(out)  
  
pred.response <- ifelse(out < 0.5, 0, 1)  
  
# Determine how many were correctly classified  
correct <- (train$Target == pred.response)  
table(correct)

## correct  
## FALSE TRUE   
## 17 381

# Plot the decision tree  
plot(treePrunedMod)  
text(treePrunedMod)



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

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

## correct  
## FALSE TRUE   
## 8 163

[tree](#TREE)

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

### a) Build a model, determine variable importance, and assess the model against the training data.

# Notice we are using new data sets that need to be read into the environment  
train <- read.csv('/Users/iris\_train.csv')  
test <- read.csv('/Users/iris\_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)  
  
fit <- randomForest(as.factor(Target) ~ ., data = train)  
  
# Determine variable importance  
importance(fit)

## MeanDecreaseGini  
## sepal.length..cm. 7.063303  
## sepal.width..cm. 1.680818  
## petal.length..cm. 30.229582  
## petal.width..cm. 30.288231

# Prediction on training data  
Prediction <- predict(fit, train)  
Prediction <- unname(Prediction)  
  
# Determine how many were correctly classified  
correct <- (train$Target == Prediction)  
table(correct)

## correct  
## TRUE   
## 105

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

# Prediction on testing data  
Prediction <- predict(fit, test)  
Prediction <- unname(Prediction)  
  
# Determine how many were correctly classified  
correct <- (test$Target == Prediction)  
table(correct)

## correct  
## FALSE TRUE   
## 4 41

[randomForest](#RANDOM)

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

### a) Build a model and assess the model against the 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/tips\_train.csv')  
test <- read.csv('/Users/tips\_test.csv')  
  
set.seed(29)  
  
fit <- randomForest(tip ~ total\_bill, data = train)  
  
# Prediction on training data  
Prediction <- predict(fit, train)  
Prediction <- unname(Prediction)  
  
# Determine mean squared error  
diff <- (train$tip - Prediction)\*\*2  
mean(diff)

## [1] 0.349471

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

# Prediction on testing data  
Prediction <- predict(fit, test)  
Prediction <- unname(Prediction)  
  
# Determine mean squared error  
diff <- (test$tip - Prediction)\*\*2  
mean(diff)

## [1] 1.237201

[randomForest](#RANDOM)

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

### a) Build a model and assess the model against the 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 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  
# n.minobsinnode = minimum number of observations in the trees terminal nodes  
fit <- gbm(Target ~ ., distribution = "bernoulli", data = train,   
 n.trees = 2500,   
 shrinkage = .01, n.minobsinnode = 20)  
  
# Prediction on training data  
gbmTrainPredictions <- predict(object = fit, newdata = train, type = "response",   
 n.trees = 2500)  
pred <- ifelse(gbmTrainPredictions < 0.5, 0, 1)  
  
# Determine how many were correctly classified  
correct <- (pred == train$Target)  
table(correct)

## correct  
## TRUE   
## 398

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

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

## correct  
## FALSE TRUE   
## 5 166

[gbm](#GBM)

## 5.9 Fit a support vector classification model.

### a) Build a model and assess the model against the 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)

## Warning: package 'e1071' was built under R version 3.3.2

# Fit a support vector classification model   
model <- svm(Target ~ ., train, type = 'C-classification', kernel = 'linear')  
  
# Evaluation on training data  
predictedY <- predict(model, train)  
prediction <- data.frame(matrix(ncol = 0, nrow = nrow(train)))  
prediction$predY <- unname(predictedY)  
  
# Determine how many were correctly classified  
prediction$actual <- train$Target  
prediction$correct <- (prediction$predY == prediction$actual)  
table(prediction$correct)

##   
## FALSE TRUE   
## 6 392

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

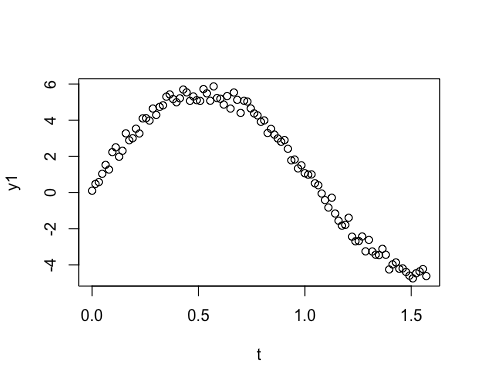
# Evaluation on testing data  
predictedY <- predict(model, test)  
prediction <- data.frame(matrix(ncol = 0, nrow = nrow(test)))  
prediction$predY <- unname(predictedY)  
  
# Determine how many were correctly classified  
prediction$actual <- test$Target  
prediction$correct <- (prediction$predY == prediction$actual)  
table(prediction$correct)

##   
## FALSE TRUE   
## 5 166

## 5.10 Fit a support vector regression model.

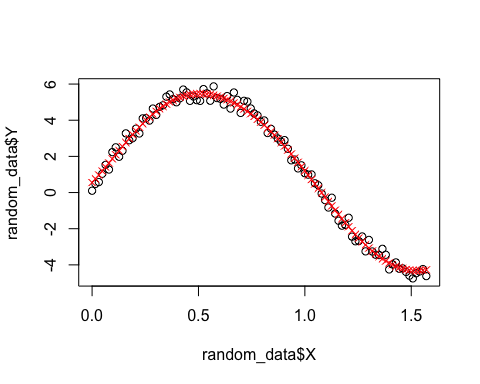
### a) Generate random data based on a sine curve.

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)

### b) Fit a support vector regression model to the data.

# 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)



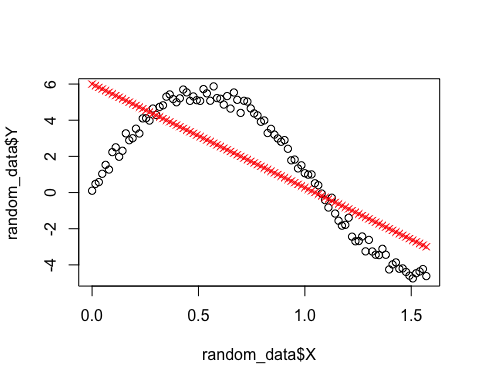
prediction = data.frame(matrix(ncol = 0, nrow = nrow(random\_data)))  
prediction$predY <- predictedY  
prediction$actual <- random\_data$Y  
prediction$sq\_diff <- (prediction$predY - prediction$actual)\*\*2  
print(mean(prediction$sq\_diff))

## [1] 0.08052554

[svm()](https://www.rdocumentation.org/packages/e1071/versions/1.6-8/topics/svm) [points()](https://www.rdocumentation.org/packages/graphics/versions/3.4.0/topics/points)

### c) Fit a linear regression model to the data.

linMod <- lm(Y ~ X, data = random\_data)  
  
pred\_lin <- predict(linMod, newdata = random\_data)  
plot.new()  
plot(random\_data$X, random\_data$Y)  
points(random\_data$X, pred\_lin, col = "red", pch = 4)



prediction = data.frame(matrix(ncol = 1, nrow = nrow(random\_data)))  
prediction$predY <- pred\_lin  
prediction$actual <- random\_data$Y  
prediction$sq\_diff <- (prediction$predY - prediction$actual)\*\*2  
print(mean(prediction$sq\_diff))

## [1] 4.904294

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

# 6 Model Evaluation & Selection

## 6.1 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/tips\_train.csv')  
test <- read.csv('/Users/tips\_test.csv')  
  
# 1. Linear Regression Model  
linMod <- lm(tip ~ ., data = train)  
  
# Evaluation on training data  
pred\_lin <- predict(linMod, newdata = train)  
  
# Determine coefficient of determination score  
r2\_lin <- 1 - ( (sum((train$tip - pred\_lin)\*\*2)) / (sum((train$tip - mean(train$tip))\*\*2)) )  
print(paste0("Linear regression model r^2 score (coefficient of determination): ", r2\_lin))

## [1] "Linear regression model r^2 score (coefficient of determination): 0.496730342166266"

--

# 2. Random Forest Regression Model  
set.seed(29)  
rfMod <- randomForest(tip ~ ., 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$tip - pred\_rf)\*\*2)) / (sum((train$tip - mean(train$tip))\*\*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): 0.534405284634815"

### b) Evaluation on testing data.

# 1. Linear Regression Model (linMod)   
  
# Evaluation on testing data  
pred\_lin <- predict(linMod, newdata = test)  
  
# Determine coefficient of determination score  
r2\_lin = 1 - ( (sum((test$tip - pred\_lin)\*\*2)) / (sum((test$tip - mean(test$tip))\*\*2)) )  
print(paste0("Linear regression model r^2 score (coefficient of determination): ", r2\_lin))

## [1] "Linear regression model r^2 score (coefficient of determination): 0.270944937190937"

--

# 2. 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$tip - pred\_rf)\*\*2)) / (sum((test$tip - mean(test$tip))\*\*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): 0.330167582821381"

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).

## 6.2 Evaluate the accuracy of classification 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/breastcancer\_train.csv')  
test <- read.csv('/Users/breastcancer\_test.csv')  
set.seed(29)  
  
# 1. Decision Tree Classification Model  
treeMod <- tree(Target ~ ., data = train)  
  
# Evaluation on training data  
out <- predict(treeMod)  
out <- unname(out)  
pred\_tree <- ifelse(out < 0.5, 0, 1)  
  
# Determine accuracy score  
accuracy\_tree <- (1/nrow(train)) \* sum(as.numeric(pred\_tree == train$Target))  
print(paste0("Decision tree model accuracy: ", accuracy\_tree))

## [1] "Decision tree model accuracy: 0.969849246231156"

--

# 2. 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"

--

# 3. Gradient Boosting Classifcation Model  
gbmMod <- gbm(Target ~ ., distribution = "bernoulli", data = train,   
 n.trees = 2500,   
 shrinkage = .01, n.minobsinnode = 20)  
  
# Evaluation on training data  
pred\_gbm <- predict(object = gbmMod, newdata = train, type = "response",   
 n.trees = 2500)  
pred\_gbm <- ifelse(pred\_gbm < 0.5, 0, 1)  
  
# Determine accuracy score  
accuracy\_gbm <- (1/nrow(train)) \* sum(as.numeric(pred\_gbm == train$Target))  
print(paste0("Gradient boosting model accuracy: ", accuracy\_gbm))

## [1] "Gradient boosting model accuracy: 1"

### b) Evaluation on testing data.

# 1. Decision Tree Classification Model (treeMod)   
  
# Evaluation on testing data  
out <- predict(treeMod, test)  
out <- unname(out)  
pred\_tree <- ifelse(out < 0.5, 0, 1)  
  
# Determine accuracy score  
accuracy\_tree <- (1/nrow(test)) \* sum(as.numeric(pred\_tree == test$Target))  
print(paste0("Decision tree model accuracy: ", accuracy\_tree))

## [1] "Decision tree model accuracy: 0.929824561403509"

--

# 2. 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: 0.970760233918129"

--

# 3. Gradient Boosting Classifcation Model (gbmMod)  
  
# Evaluation on testing data  
pred\_gbm <- predict(object = gbmMod, newdata = test, type = "response",   
 n.trees = 2500)  
pred\_gbm <- ifelse(pred\_gbm < 0.5, 0, 1)  
  
# Determine accuracy score  
accuracy\_gbm <- (1/nrow(test)) \* sum(as.numeric(pred\_gbm == test$Target))  
print(paste0("Gradient boosting model accuracy: ", accuracy\_gbm))

## [1] "Gradient boosting model accuracy: 0.970760233918129"

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).

## 6.3 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: 95.9% +/- 1.26%"

[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)

[caret](#CARET)

# Alphabetical Index

## caret

[caret](https://www.rdocumentation.org/packages/caret/versions/6.0-76) is an R programming package of tools for training and plotting classification and regression models.

## Data Frame

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

[dplyr](https://cran.r-project.org/web/packages/dplyr/index.html) is an R programming package of tools for workng with [data frame](#DataFrame) like objects.

## gbm

[gbm](https://cran.r-project.org/web/packages/gbm/index.html) is an R programming package useful for building and analyzing gradient boosting models.

## gdata

[gdata](https://cran.r-project.org/web/packages/gdata/index.html) is an R programming package of tools useful for data manipulation.

## List

An [R list](http://www.r-tutor.com/r-introduction/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

[randomForest](https://cran.r-project.org/web/packages/randomForest/index.html) is an R programming package of tools useful for building and analyzing classification and regression random forest models.

## rjson

[rjson](https://cran.r-project.org/web/packages/rjson/index.html) is an R programming package of tools useful for converting R objects into JSON objects, and JSON objects into R objects.

## Series/Array

A series is a one-dimensional data frame, which is also called an [array](https://www.tutorialspoint.com/r/r_arrays.htm) in R. 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

## tree

[tree](https://cran.r-project.org/web/packages/tree/index.html) is an R programming package of tools useful for building and analyzing classification and regression decision trees.

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