Data Science Using Python, SAS, & R:

A Rosetta Stone for Analytical Languages

Table of Contents

R Tutorial 5

1 Reading in Data and Basic Statistical Functions 5

1.1 Read in the data. 5

a) Read the data in as a .csv file. 5

b) Read the data in as a .xls file. 5

c) Read the data in as a .json file. 5

1.2 Find the dimensions of the data set. 6

1.3 Find basic information about the data set. 6

1.4 Look at the first 5 (last 5) observations. 6

1.5 Calculate means of numeric variables. 7

1.6 Compute summary statistics of the data set. 7

1.7 Descriptive statistics functions applied to columns of the data set. 7

1.8 Produce a one-way table to describe the frequency of a variable. 8

a) Produce a one-way table of a discrete variable. 8

b) Produce a one-way table of a categorical variable. 8

1.9 Produce a two-way table to visualize the frequency of two categorical (or discrete) variables. 8

1.10 Select a subset of the data that meets a certain criterion. 8

1.11 Determine the correlation between two continuous variables. 9

2 Basic Graphing and Plotting Functions 9

2.1 Visualize a single continuous variable by producing a histogram. 9

2.2 Visualize a single continuous variable by producing a boxplot. 10

2.3 Visualize two continuous variables by producing a scatterplot. 10

2.4 Visualize a relationship between two continuous variables by producing a scatterplot and a plotted line of best fit. 11

2.5 Visualize a categorical variable by producing a bar chart. 12

2.6 Visualize a continuous variable, grouped by a categorical variable, using side-by-side boxplots. 13

a) Simple side-by-side boxplot without color. 13

b) More advanced side-by-side boxplot with color. 14
3 Basic Data Wrangling and Manipulation 15
3.1 Create a new variable in a data set as a function of existing variables in the data set. 15
3.2 Create a new variable in a data set using if/else logic of existing variables in the data set. 16
3.3 Create a new variable in a data set using mathemtical functions applied to existing variables in the data set. 16
3.4 Drop variables from a data set. 16
3.5 Sort a data set by a variable. 17
a) Sort data set by a continuous variable. 17
b) Sort data set by a categorical variable. 17
3.6 Compute descriptive statistics of continuous variables, grouped by a categorical variable. 17
3.7 Add a new row to the bottom of a data set. 17
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. 18
4 More Advanced Data Wrangling 19
4.1 Drop observations with missing information. 19
4.2 Merge two data sets together on a common variable. 19
a) First, select specific columns of a data set to create two smaller data sets. 19
b) Second, we want to merge the two smaller data sets on the common variable. 20
c) Finally, we want to check to see if the merged data set is the same as the original data set. 20
4.3 Merge two data sets together by index number only. 20
a) First, select specific columns of a data set to create two smaller data sets. 20
b) Second, we want to join the two smaller data sets. 21
c) Finally, we want to check to see if the joined data set is the same as the original data set. 21
4.4 Create a pivot table to summarize information about a data set. 21
4.5 Return all unique values from a text variable. 22
5 Preparation & Basic Regression 22
5.1 Pre-process a data set using principal component analysis. 22
5.2 Split data into training and testing data and export as a .csv file. 23
5.3 Fit a logistic regression model. 23
5.4 Fit a linear regression model. 24

6 Supervised Machine Learning 24
6.1 Fit a logistic regression model on training data and assess against testing data. 24
a) Fit a logistic regression model on training data. 24
b) Assess the model against the testing data. 25
6.2 Fit a linear regression model on training data and assess against testing data. 26
a) Fit a linear regression model on training data. 26
b) Assess the model against the testing data. 27
6.3 Fit a decision tree model on training data and assess against testing data. 27
a) Fit a decision tree classification model. 27
b) Fit a decision tree regression model. 29
6.4 Fit a random forest model on training data and assess against testing data. 30
a) Fit a random forest classification model. 30
b) Fit a random forest regression model. 31
6.5 Fit a gradient boosting model on training data and assess against testing data. 32
a) Fit a gradient boosting classification model. 32
b) Fit a gradient boosting regression model. 34
6.6 Fit an extreme gradient boosting model on training data and assess against testing data. 35
a) Fit an extreme gradient boosting classification model. 35
b) Fit an extreme gradient boosting regression model. 36
6.7 Fit a support vector model on training data and assess against testing data. 37
a) Fit a support vector classification model. 37
b) Fit a support vector regression model. 37
6.8 Fit a neural network model on training data and assess against testing data. 38
a) Fit a neural network classification model. 38
b) Fit a neural network regression model. 39
7 Unsupervised Machine Learning 39
7.1 KMeans Clustering 39
7.2 Spectral Clustering 40
7.3 Ward Hierarchical Clustering 40
7.4 DBSCAN 40
7.5 Self-organized map 41
8 Forecasting 42
8.1 Fit an ARIMA model to a timeseries. 42

a) Plot the timeseries. 42
b) Fit an ARIMA (0, 1, 1) model and predict 2 years (24 months). 42
8.2 Fit a Simple Exponential Smoothing model to a timeseries. 43
a) Plot the timeseries. 43
b) Fit a Simple Exponential Smoothing model, predict 2 years (24 months) out and plot predictions. 44
8.3 Fit a Holt-Winters model to a timeseries. 45
a) Plot the timeseries. 45
b) Fit a Holt-Winters additive model, predict 2 years (24 months) out and plot predictions. 46
8.4 Fit a Facebook Prophet forecasting model to a timeseries. 47
9 Model Evaluation & Selection 48
9.1 Evaluate the accuracy of regression models. 48
a) Evaluation on training data. 48
b) Evaluation on testing data. 49
9.2 Evaluate the accuracy of classification models. 49
a) Evaluation on training data. 49
b) Evaluation on testing data. 50
9.3 Evaluation with cross validation. 50
a) KFold 50
b) ShuffleSplit 51
10 Text Analytics 52
11 Deep Learning 52
Appendix 52
1 Built-in R Objects 52
2 R packages used in this tutorial 53
Alphabetical Index 54
Array 54
Data Frame 54
Dictionary 54
List 55
Vector 55

R Tutorial

Welcome to the R tutorial version of "Data Science Using Python, SAS, & R: A Rosetta Stone for Analytical Languages". This tutorial includes examples of common data science tasks, organized in the same way across 3 data science languages. Before beginning this tutorial, please check to make sure you have R 3.3.1 installed (this is not required, but this is the release used to generate the following examples). Also, the following R packages are used throughout this tutorial. You may not need all of the following packages to fit your specific needs, but they are listed below, and also in Appendix Section 2 with more detail:

gdata | rjson | ggplot2 | dplyr | tree |randomForest | gbm | xgboost | e1071 | RSNNS | caret | kernlab | dbscan |forecast

To install R packages you need to run the following in the R console:

```
install.packages("name_of_package")
```

Note: In R, comments are indicated in code with a "#" character, and arrays and matrices begin with index 1. Also, "<-" and "=" can be used interchangeably.

Now let's get started!

---

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()

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

```
# call the gdata package
library(gdata)

student_xls <- read.xls('/Users/class.xls', 1)
```

gdata | read.xls()

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

There is more code involved in reading a .json file into R so it becomes a proper data frame. Also, this code is specific for a certain .json format, so you may have to change it to fix your needs.

```
# call the rjson package
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
```

rjson | fromJSON()

1.2 Find the dimensions of the data set.

The shape of an R data frame 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.

Information about an R data frame is available by calling the str() function, with the data name as an argument.

```
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 (last 5) observations.

The first 5 observations of a data frame 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 (*n*=). 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 means 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 data set to return
# only numeric variables

# Then we can use the colMeans() function to return the means of
# column variables
colMeans(student[sapply(student, is.numeric)])
```

```
##       Age    Height    Weight
##  13.31579  62.33684 100.02632
```

colMeans() | sapply() | is.numeric

1.6 Compute summary statistics of the data set.

Summary statistics of a data frame 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
```

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()

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()

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
```

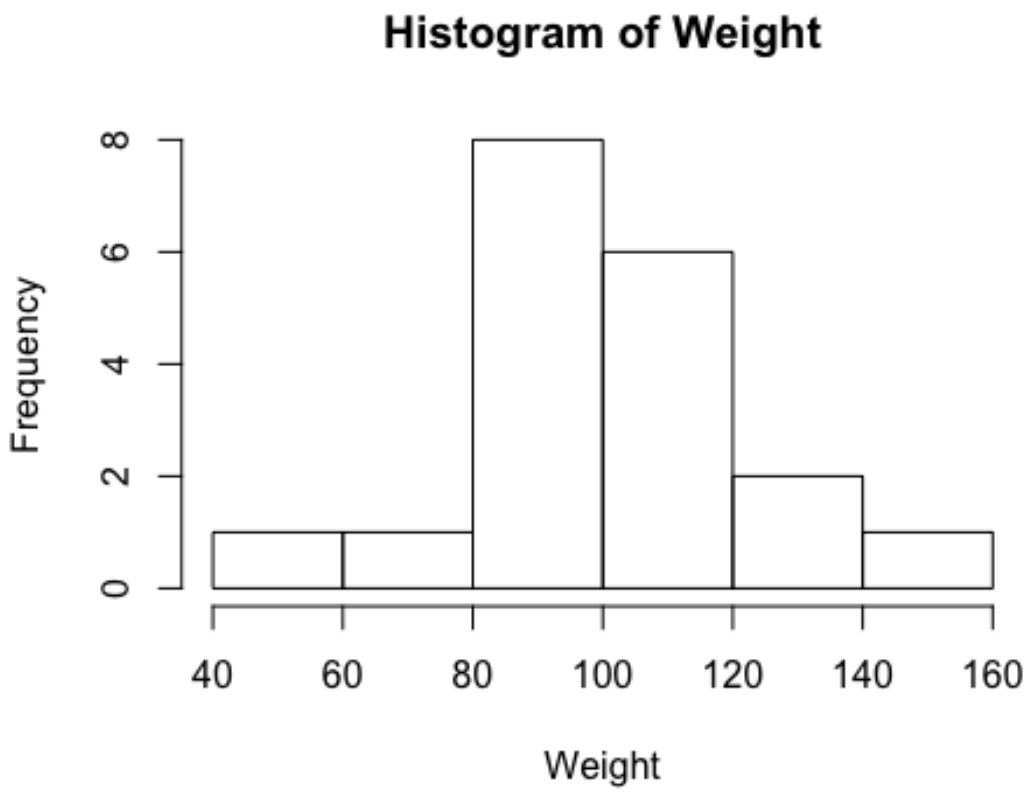
subset() | cor()

---

2 Basic Graphing and Plotting Functions

2.1 Visualize a single continuous variable by producing a histogram.

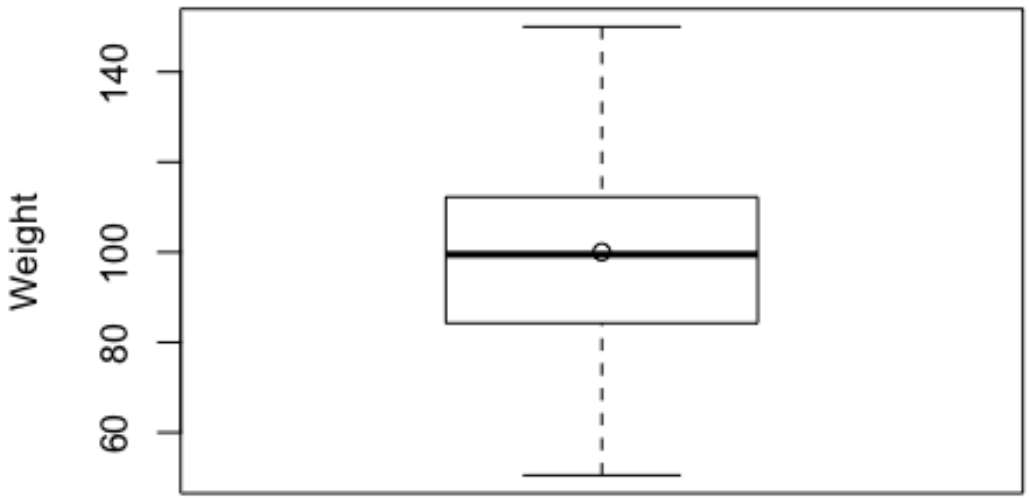
```
# Setting student$Weight to a new variable "Weight" cleans up the labeling of
# the histogram
Weight <- student$Weight
hist(Weight)
```



hist()

2.2 Visualize a single continuous variable by producing a boxplot.

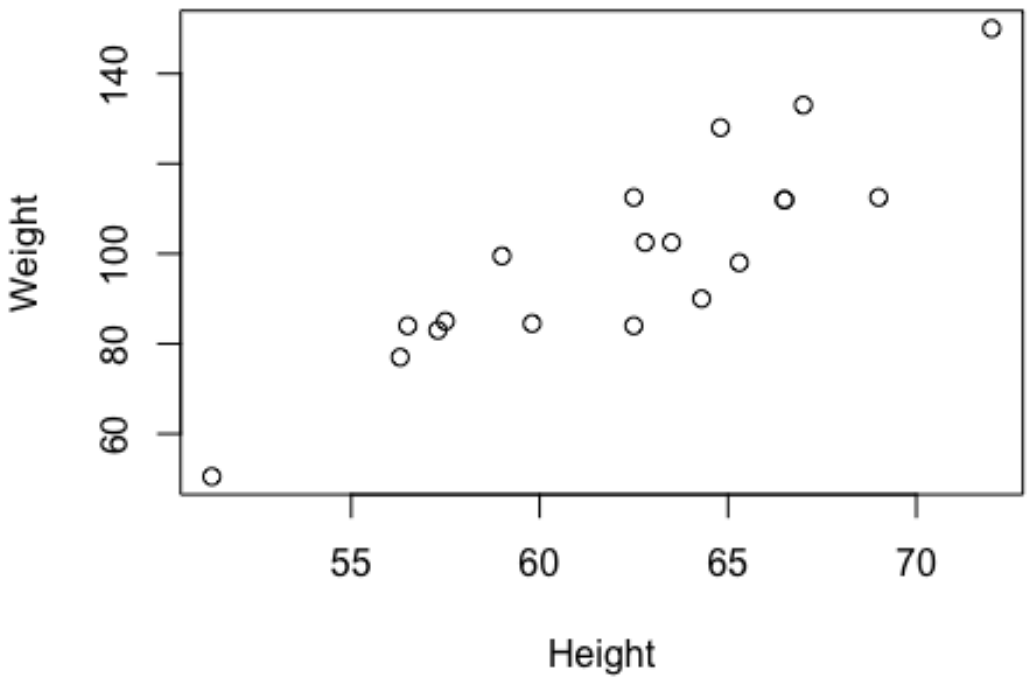
```
# points(mean(Weight)) tells R to plot the mean on the boxplot
boxplot(Weight, ylab="Weight")
points(mean(Weight))
```



boxplot() | points()

2.3 Visualize two continuous variables by producing a scatterplot.

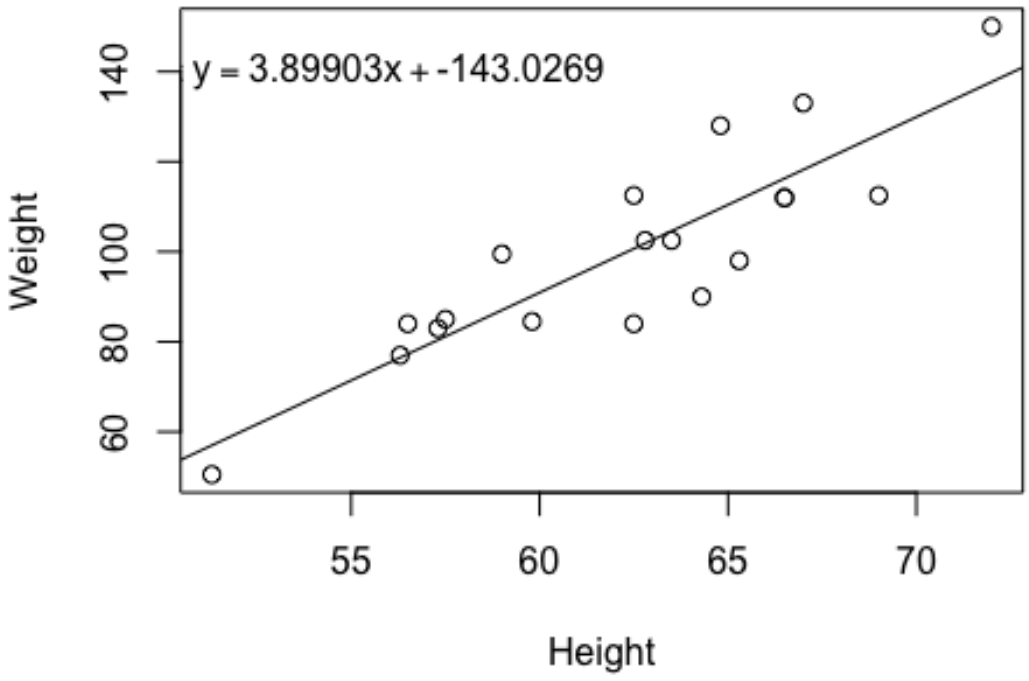
```
Height <- student$Height
# Notice here you specify the x variable, followed by the y variable
plot(Height, Weight)
```



plot()

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

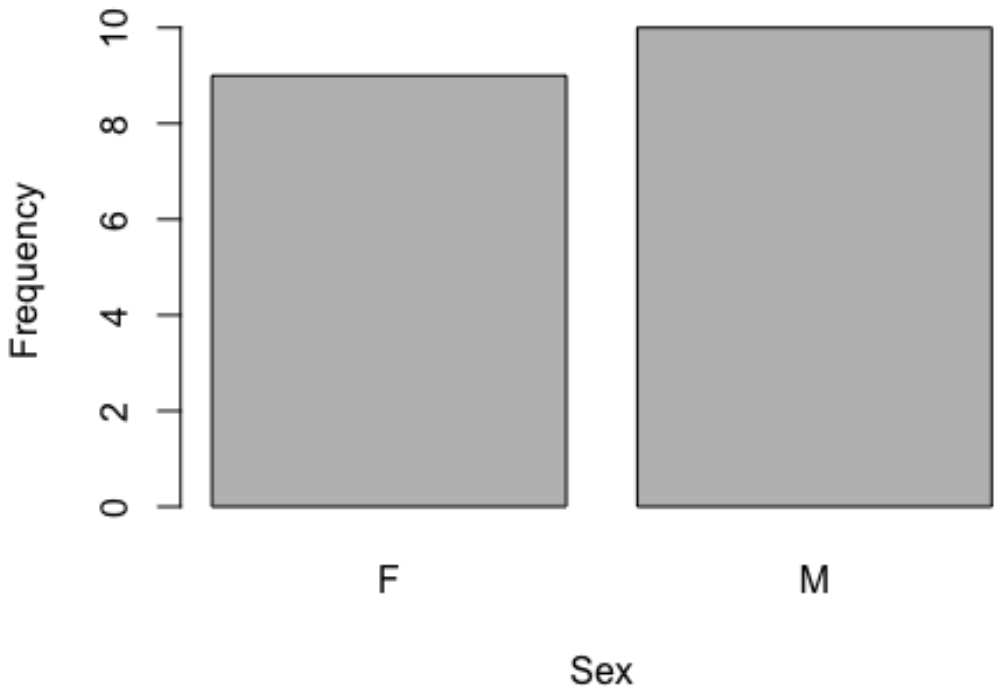
```
plot(Height, Weight)

# 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(55, 140, bquote(Line: y == .(slope) * x + .(intercept)))
```



lm() | coef() | as.matrix() | abline() | text() | 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))
```

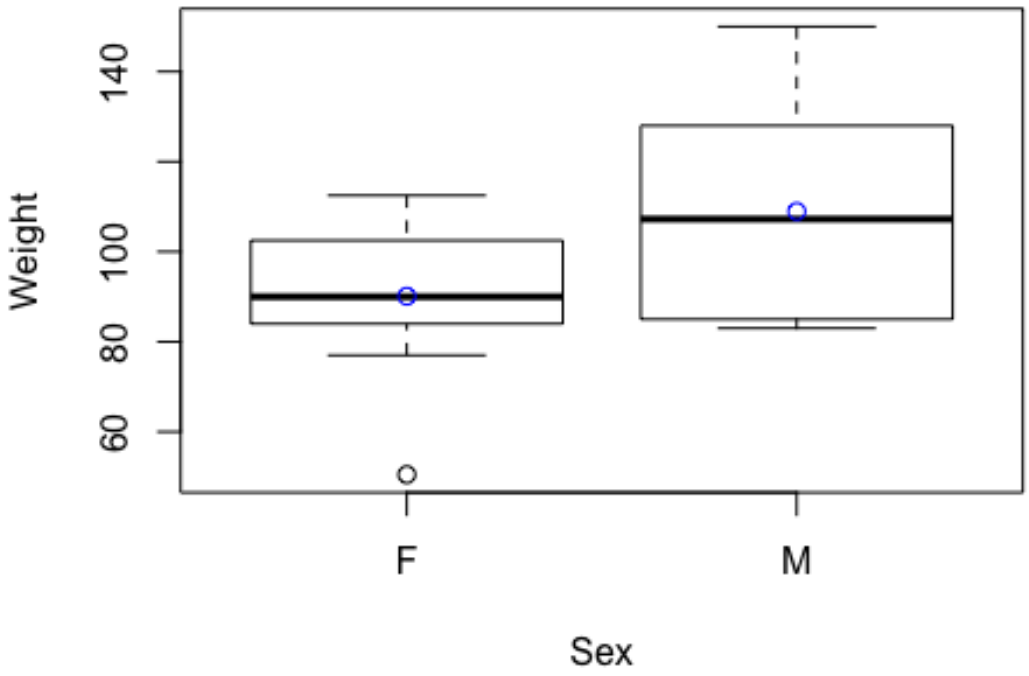


barplot() | names()

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

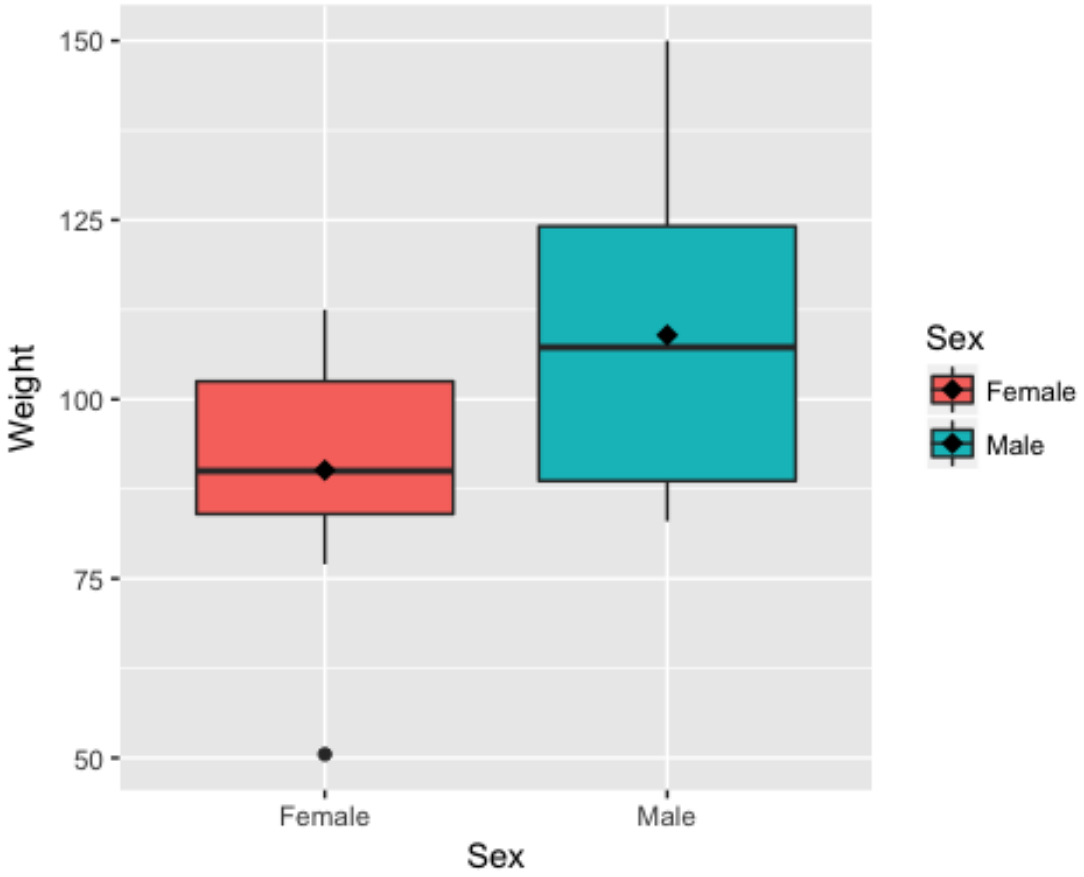
a) Simple side-by-side boxplot without color.

```
# 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.

```
# call the ggplot2 package
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 | factor() | c() | aes() | geom_boxplot() | stat_summary()

---

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 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()

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

Using the log(), exp(), sqrt(), ifelse() and abs() functions.

```
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
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
```

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() | 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() | rbind.data.frame()

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
```

user-defined functions

---

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()

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()

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()

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')

# call the dplyr package
library(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 | group_by | 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()

In the following sections, several data set will be used more than once for prediction and modeling. Often, they will be re-read into the environment so we are always going back to the original, raw data.

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
## SepalLength  0.5223716 -0.37231836  0.7210168  0.2619956
## SepalWidth  -0.2633549 -0.92555649 -0.2420329 -0.1241348
## PetalLength  0.5812540 -0.02109478 -0.1408923 -0.8011543
## PetalWidth   0.5656110 -0.06541577 -0.6338014  0.5235463
```

prcomp()

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_R.csv")
write.csv(test, file = "/Users/iris_test_R.csv")
```

floor() | nrow() | set.seed() | sample() | seq_len() | write.csv()

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
# You could fit the model of greater15 as a function of all
# other variables with "greater15 ~ ."
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()

5.4 Fit a linear regression model.

```
# 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
```

lm()

6 Supervised Machine Learning

Many of the following models will make use of the 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')
```

```r
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)

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.

```r
# Prediction on testing data
predictions <- predict(logreg, test, type = 'response')
predY <- ifelse(predictions < 0.5, 0, 1)

# 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

# Determine how many were correctly classified
Results <- ifelse(predY == test$greater15, "Correct", "Wrong")
table(Results)
```

```
## Results
## Correct   Wrong
##      34      15
```

glm()

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')

# Fit a linear regression model
# The "." character tells the model to use all variables except the response
# variabe (Target)
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 on testing data
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))
prediction$predY = predict(linreg, newdata = test)

# Compute the squared difference between predicted tip and actual tip
prediction$sq_diff <- (prediction$predY - 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
```

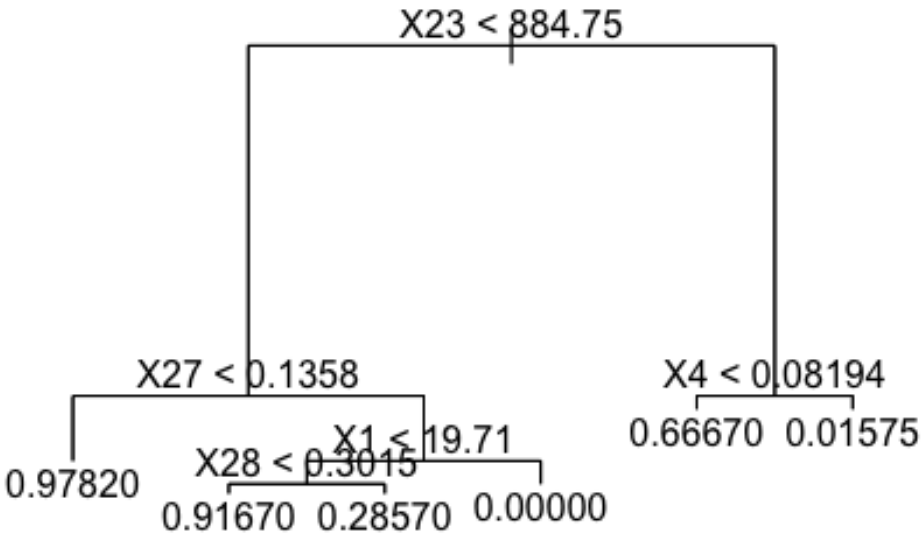
lm()

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 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')

# call the tree package
library(tree)

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)
predY <- ifelse(out < 0.5, 0, 1)

# Determine how many were correctly classified
Results <- ifelse(test$Target == predY, "Correct", "Wrong")
table(Results)
```

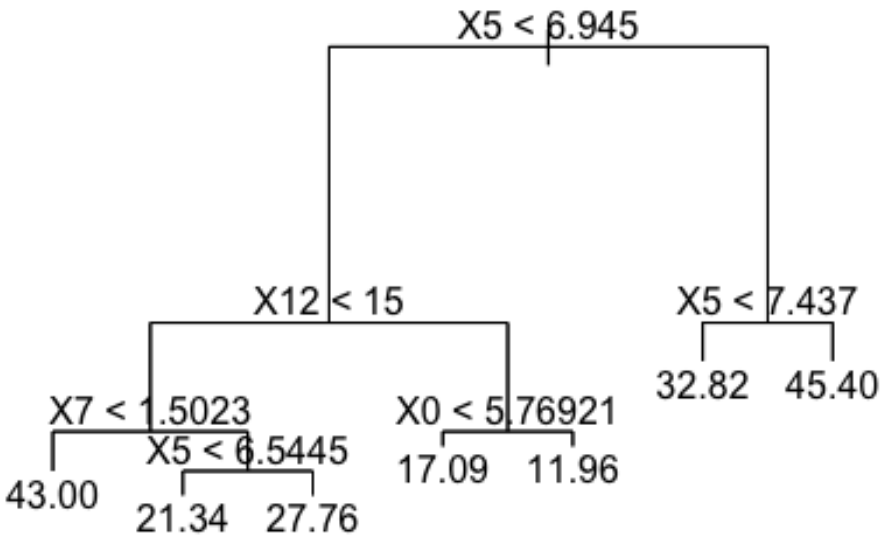
```
## Results
## Correct   Wrong
##     159      12
```

tree

b) Fit a decision tree regression model.

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

```
train <- read.csv('/Users/boston_train.csv')
test <- read.csv('/Users/boston_test.csv')

treeMod <- tree(Target ~ ., data = train)

# 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] "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.

```
# Prediction on testing data
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))
prediction$predY = predict(treeMod, newdata = test)

# Determine mean squared error
prediction$sq_diff <- (prediction$predY - test$Target)**2
mean_sq_error <- mean(prediction$sq_diff)
print(mean_sq_error)
```

```
## [1] 25.12126
```

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.

```
train <- read.csv('/Users/breastcancer_train.csv')
test <- read.csv('/Users/breastcancer_test.csv')

# call the randomForest package
library(randomForest)
set.seed(29)

# as.factor() since classification model
rfMod <- randomForest(as.factor(Target) ~ ., data = train)

# Determine variable importance
var_import <- importance(rfMod)
var_import <- data.frame(sort(var_import, decreasing = TRUE,
                              index.return = TRUE))
var_import$MeanDecreaseGini <- var_import$x
var_import$X <- var_import$ix - 1
var_import <- subset(var_import, select = -c(ix, x))
head(var_import, n=5)
```

```
##   MeanDecreaseGini  X
## 1         22.88452 27
## 2         21.64371  7
## 3         21.56885 22
## 4         20.13103 23
## 5         18.90989 20
```

ii) Assess the model against the testing data.

```
# Prediction on testing data
predY <- predict(rfMod, test)

# Determine how many were correctly classified
Results <- ifelse(test$Target == predY, "Correct", "Wrong")
table(Results)
```

```
## Results
## Correct   Wrong
##     166       5
```

randomForest | as.factor()

b) Fit a random forest regression model.

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

```
train <- read.csv('/Users/boston_train.csv')
test <- read.csv('/Users/boston_test.csv')

# call the randomForest package
library(randomForest)
set.seed(29)

rfMod <- randomForest(Target ~ ., data = train)

# Determine variable importance
var_import <- importance(rfMod)
var_import <- data.frame(sort(var_import, decreasing = TRUE,
                              index.return = TRUE))
var_import$MeanDecreaseGini <- var_import$x
var_import$X <- var_import$ix - 1
var_import <- subset(var_import, select = -c(ix, x))
head(var_import, n=5)
```

```
##   MeanDecreaseGini  X
## 1         8662.298 12
## 2         8451.836  5
## 3         2147.288  0
## 4         2105.072  7
## 5         1915.570  2
```

ii) Assess the model against the testing data.

```
# Prediction on testing data
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))
prediction$predY = predict(rfMod, newdata = test)

# Determine mean squared error
prediction$sq_diff <- (prediction$predY - test$Target)**2
mean_sq_error <- mean(prediction$sq_diff)
print(mean_sq_error)

## [1] 9.028163
```

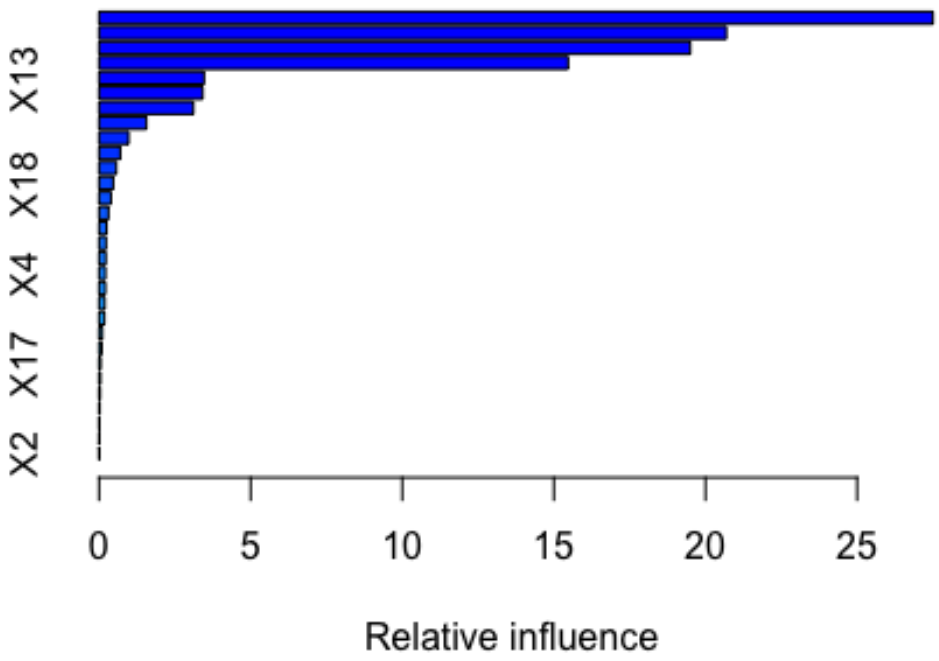
randomForest

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.

```
train <- read.csv('/Users/breastcancer_train.csv')
test <- read.csv('/Users/breastcancer_test.csv')

# call the gbm package
library(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
var_import <- summary(gbMod)
```




```
head(var_import, n=5)
```

```
##     var  rel.inf
## X27 X27 27.50103
## X7   X7 20.68575
## X23 X23 19.49976
## X22 X22 15.46766
## X13 X13  3.46162
```

ii) Assess the model against the testing data.

```
# Prediction on testing data
out <- predict(object = gbMod, newdata = test,
                          type = "response", n.trees = 2500)
predY <- ifelse(out < 0.5, 0, 1)

# Determine how many were correctly classified
Results <- ifelse(test$Target == predY, "Correct", "Wrong")
table(Results)
```

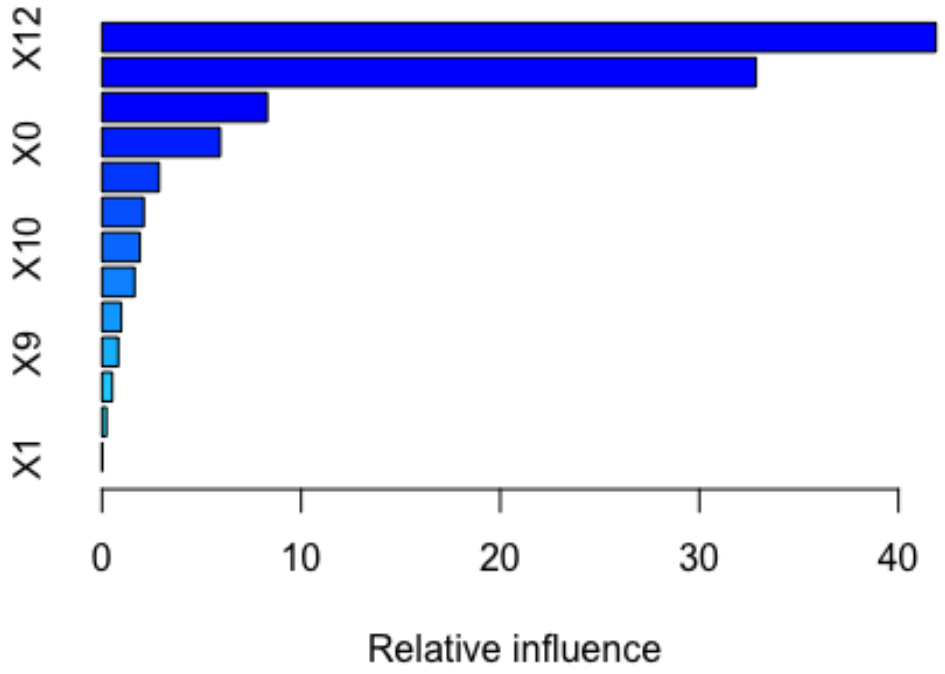
```
## Results
## Correct   Wrong
##     167       4
```

gbm

b) Fit a gradient boosting regression model.

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

```
train <- read.csv('/Users/boston_train.csv')
test <- read.csv('/Users/boston_test.csv')

# call the gbm package
library(gbm)
set.seed(29)

gbMod <- gbm(Target ~ ., data = train, distribution = "gaussian",
             n.trees = 2500, shrinkage = .01)

# Determine variable importance
var_import <- summary(gbMod)
```




```
head(var_import, n=5)
```

```
##     var   rel.inf
## X12 X12 41.882259
## X5   X5 32.846422
## X7   X7  8.298140
```

```
## X0   X0  5.937391
## X4   X4  2.867877
```

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$predY <- predict(object = gbMod, newdata = test,
                            type = "response", n.trees = 2500)

# Compute mean squared error
prediction$sq_diff <- (prediction$predY - test$Target)**2
mean_sq_error <- mean(prediction$sq_diff)
print(mean_sq_error)

## [1] 11.88728
```

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.

```
train <- read.csv('/Users/breastcancer_train.csv')
test <- read.csv('/Users/breastcancer_test.csv')

# call the xgboost package
library(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))))
predY <- ifelse(predictions < 0.5, 0, 1)

# Determine how many were correctly classified
```

```
Results <- ifelse(test$Target == predY, "Correct", "Wrong")
table(Results)
```

```
## Results
## Correct   Wrong
##     165       6
```

xgboost

b) Fit an extreme gradient boosting regression model.

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

```
train <- read.csv('/Users/boston_train.csv')
test <- read.csv('/Users/boston_test.csv')

# call the xgboost package
library(xgboost)
set.seed(29)

# Fit the model
xgbMod <- xgboost(data.matrix(subset(train, select = -c(Target))),
                  data.matrix(train$Target), max_depth = 3, nrounds = 10,
                  n_estimators = 2500, shrinkage = .01)
```

```
## [1]  train-rmse:17.131615
## [2]  train-rmse:12.419768
## [3]  train-rmse:9.116973
## [4]  train-rmse:6.777830
## [5]  train-rmse:5.182819
## [6]  train-rmse:4.113659
## [7]  train-rmse:3.403357
## [8]  train-rmse:2.955893
## [9]  train-rmse:2.677797
## [10] train-rmse:2.485887
```

ii) Assess the model against the testing data.

```
# Prediction on testing
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))
prediction$predY <- predict(xgbMod,
                    data.matrix(subset(test, select = -c(Target))))

# Compute the squared difference between predicted tip and actual tip
prediction$sq_diff <- (prediction$predY - 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] 14.27491
```

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.

```
train <- read.csv('/Users/breastcancer_train.csv')
test <- read.csv('/Users/breastcancer_test.csv')

# call the e1071 package
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
predY <- unname(predict(svMod, subset(test, select = -c(Target))))

# Determine how many were correctly classified
Results <- ifelse(test$Target == predY, "Correct", "Wrong")
table(Results)
```

```
## Results
## Correct   Wrong
##     166       5
```

e1071 | svm()

b) Fit a support vector regression model.

i) Fit a support vector regression model on training data.

```
train <- read.csv('/Users/boston_train.csv')
test <- read.csv('/Users/boston_test.csv')

# call the e1071 package
library(e1071)

svMod <- svm(Target ~ ., train)
```

ii) Assess the model against the testing data.

```
# Prediction on testing data
prediction = data.frame(matrix(ncol = 0, nrow = nrow(test)))
prediction$predY <- unname(predict(svMod, test))
prediction$sq_diff <- (prediction$predY - test$Target)**2
print(mean(prediction$sq_diff))
```

```
## [1] 11.83309
```

e1071 | 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 using new data sets
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))

# call the RSNNS package
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.

```
# Prediction on testing data
predictions <- predict(nnMod, testInputs)

# Determine how many were correctly classified
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
```

RSNNS | confusionMatrix()

b) Fit a neural network regression model.

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

```
train <- read.csv('/Users/boston_train.csv')
test <- read.csv('/Users/boston_test.csv')

# call the RSNNS package
library(RSNNS)
set.seed(29)

# 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()

```
# Assess against testing data, remembering to multiply by 50
preds = data.frame(matrix(ncol = 0, nrow = nrow(test)))
preds$predY <- predict(nnMod, scaled_test)*50
preds$sq_error <- (preds$predY - test$Target)**2
print(mean(preds$sq_error))

## [1] 20.27705
```

RSNNS

---

7 Unsupervised Machine Learning

7.1 KMeans Clustering

```
iris = read.csv('/Users/iris.csv')

iris$Species = ifelse(iris$Target == 0, "Setosa",
                      ifelse(iris$Target == 1, "Versicolor", "Virginica"))

features <- as.matrix(subset(iris, select = c(PetalLength, PetalWidth,
                                              SepalLength, SepalWidth)))

set.seed(29)

kmeans <- kmeans(features, 3)

table(iris$Species, kmeans$cluster)
```

```
##
##                1  2  3
##   Setosa      50  0  0
##   Versicolor   0 48  2
##   Virginica    0 14 36
```

kmeans()

7.2 Spectral Clustering

```
# call the kernlab package
library(kernlab)

set.seed(29)

spectral <- specc(features, centers = 3, iterations = 10, nystrom.red = TRUE)

labels <- as.data.frame(spectral)

table(iris$Species, labels$spectral)

##
##                1  2  3
##   Setosa      50  0  0
##   Versicolor   0 47  3
##   Virginica    0  3 47
```

kernlab | specc()

7.3 Ward Hierarchical Clustering

```
set.seed(29)

hclust <- hclust(dist(features), method = "ward.D2")

table(iris$Species, cutree(hclust, 3))

##
##                1  2  3
##   Setosa      50  0  0
##   Versicolor   0 49  1
##   Virginica    0 15 35
```

Hierarchical Clustering in R | hclust()

7.4 DBSCAN

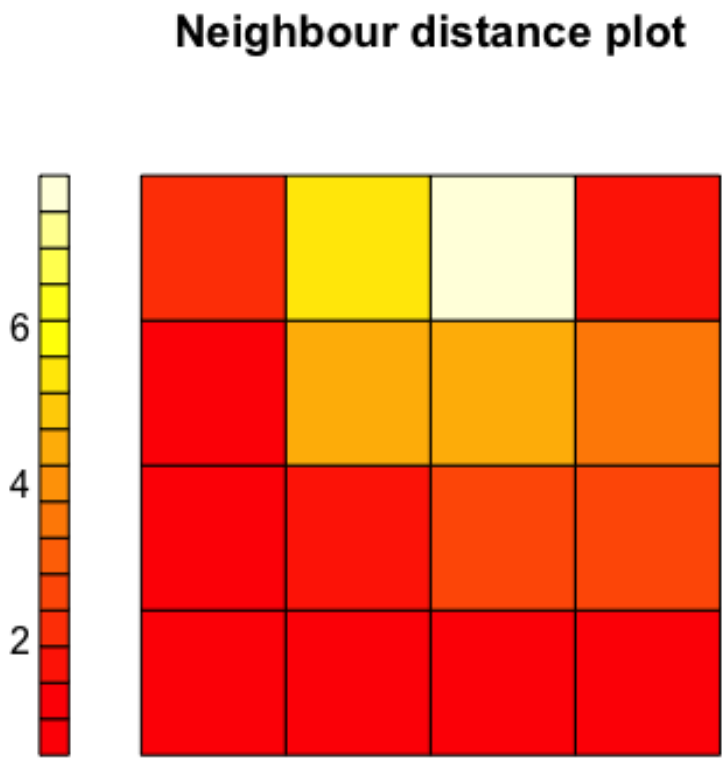
```
# call the dbscan package
library(dbscan)

set.seed(29)
```

```r
# eps = 0.5 is default in Python
dbscan <- dbscan(features, eps = 0.5)

table(iris$Species, dbscan$cluster)

##
##                0  1  2
##   Setosa       1 49  0
##   Versicolor   6  0 44
##   Virginica   10  0 40
```

dbscan

7.5 Self-organized map

```r
# call the kohonen package
library(kohonen)

# Seed chosen to match SAS and R results
set.seed(5)

fit <- som(features, mode = "online", somgrid(4, 4, "rectangular"))

plot(fit, type = "dist.neighbour", shape = "straight")
```

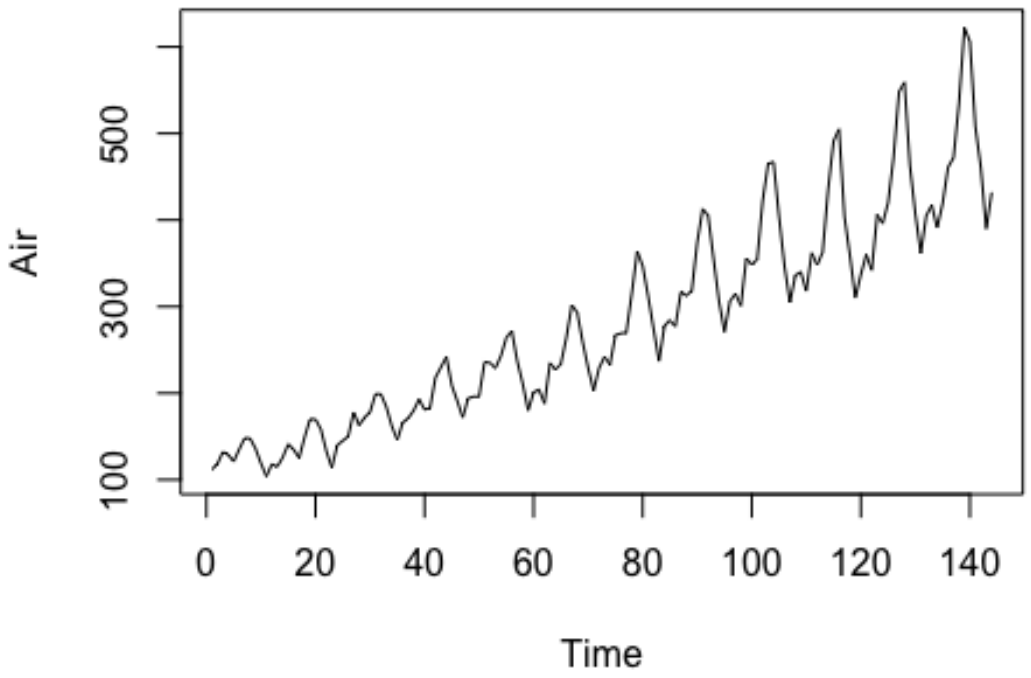


8 Forecasting

8.1 Fit an ARIMA model to a timeseries.

a) Plot the timeseries.

```
# Read in new data set
air <- read.csv('/Users/air.csv')

air_series <- air$AIR

plot.ts(air_series, ylab="Air")
```

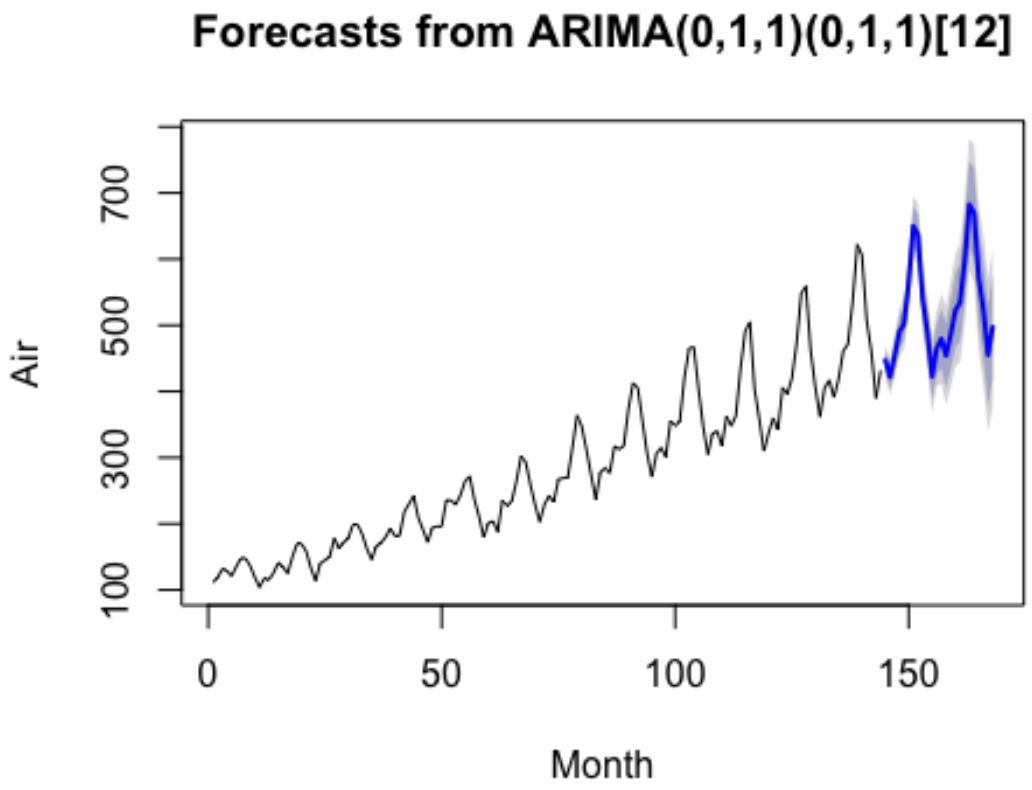


plot.ts()

b) Fit an ARIMA (0, 1, 1) model and predict 2 years (24 months).

```
a_fit <- arima(air_series, order = c(0,1,1),
               seasonal = list(order = c(0,1,1), period = 12),
```

```
                method = "ML")

# call the forecast package
library(forecast)

a_forecast <- forecast(a_fit, 24)

plot(a_forecast, xlab = "Month", ylab = "Air")
```

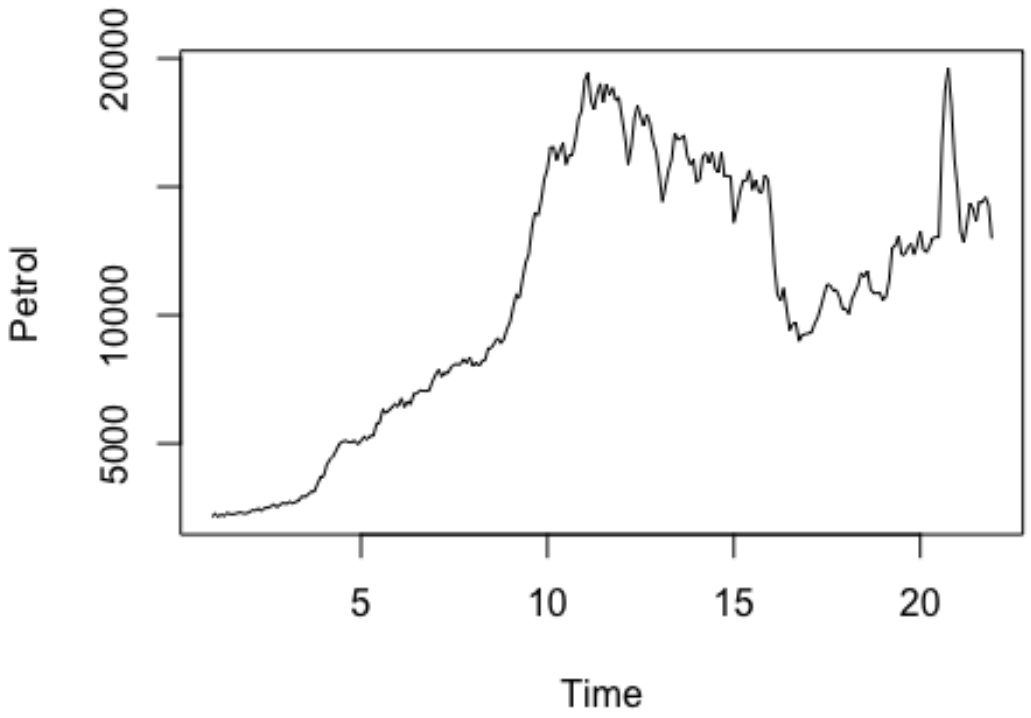


arima() | forecast

8.2 Fit a Simple Exponential Smoothing model to a timeseries.

a) Plot the timeseries.

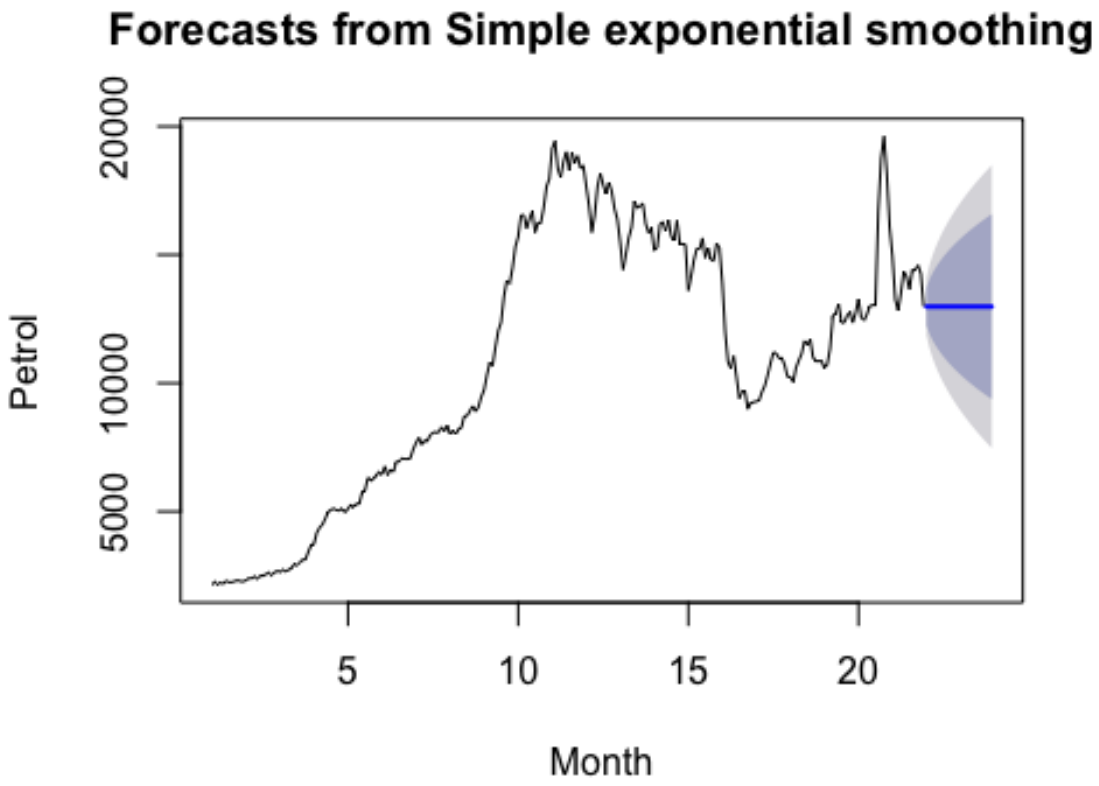
```
# Read in new data set
usecon <- read.csv('/Users/usecon.csv')

petrol_series <- usecon$PETROL

petrol <- ts(petrol_series, frequency = 12)

plot.ts(petrol, ylab="Petrol")
```



ts() | plot.ts()

b) Fit a Simple Exponential Smoothing model, predict 2 years (24 months) out and plot predictions.

```
# call the forecast package
library(forecast)

ses_fit <- ses(petrol, h=24, alpha = 0.9999)

plot(ses_fit, xlab = "Month", ylab = "Petrol")
```

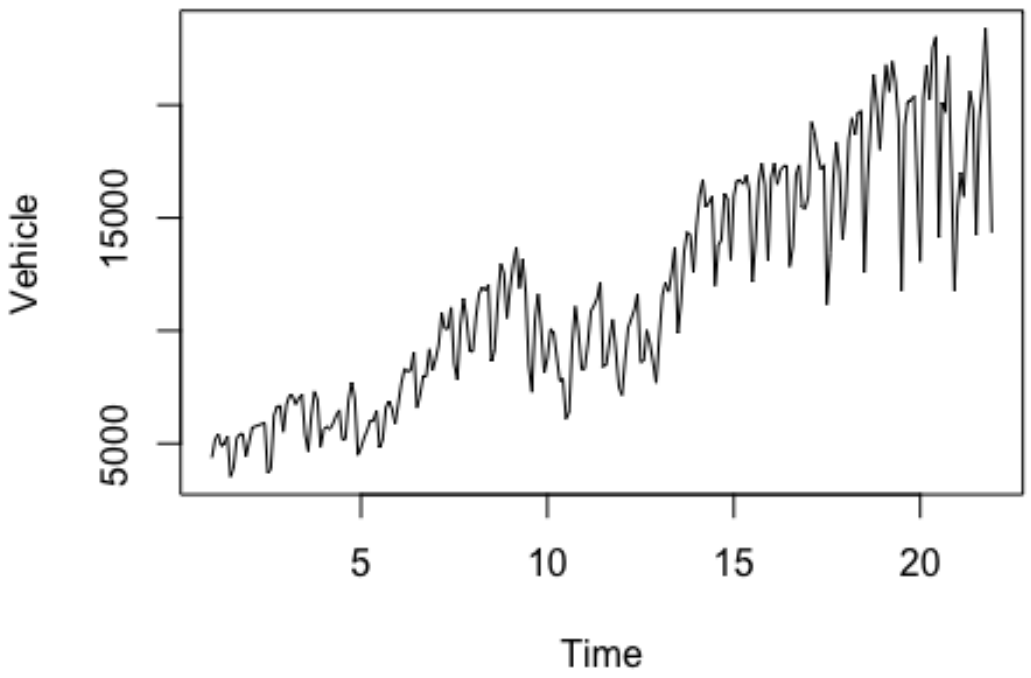


forecast

8.3 Fit a Holt-Winters model to a timeseries.

a) Plot the timeseries.

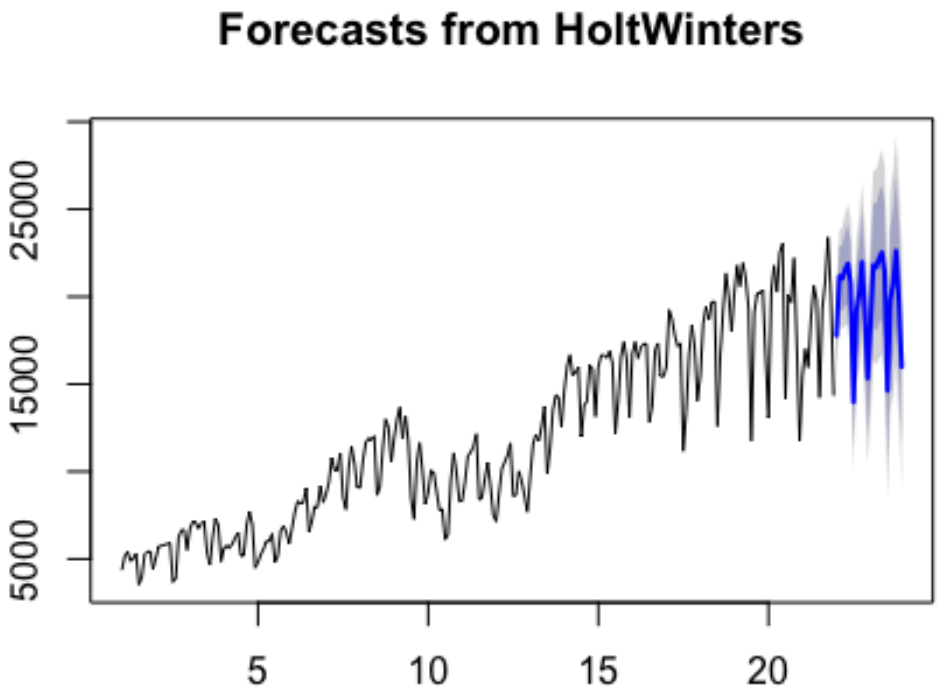
```
vehicle_series <- usecon$VEHICLES

vehicle <- ts(vehicle_series, frequency = 12)

plot.ts(vehicle, ylab="Vehicle")
```



ts() | plot.ts()

b) Fit a Holt-Winters additive model, predict 2 years (24 months) out and plot predictions.

```
# call the forecast package
library(forecast)

add_fit <- HoltWinters(vehicle, seasonal = "additive")

add_forecast <- forecast(add_fit, 24)

plot(add_forecast)
```

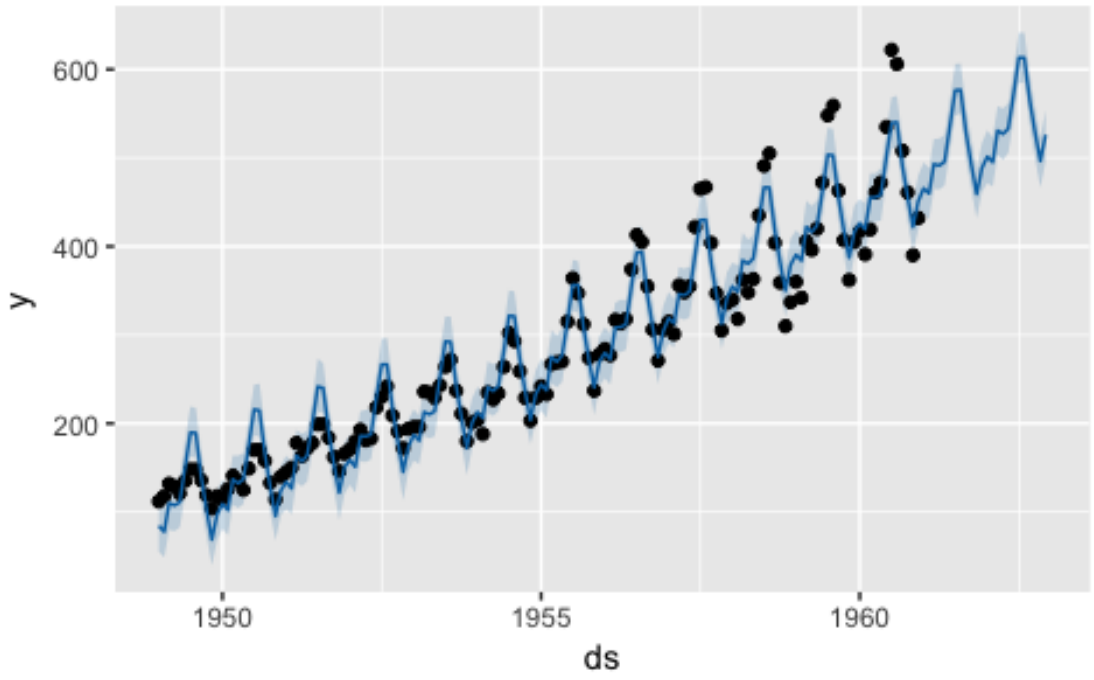


forecast

8.4 Fit a Facebook Prophet forecasting model to a timeseries.

```
air <- read.csv('/Users/air.csv')

# call the prophet & dplyr packages
library(prophet)
library(dplyr)

air_df <- data.frame(matrix(ncol = 0, nrow = nrow(air)))

air_df$ds <- as.Date(air$DATE, format = "%m/%d/%Y")
air_df$y <- air$AIR

m <- prophet(air_df, yearly.seasonality = TRUE, weekly.seasonality = FALSE)

## Initial log joint probability = -2.46502
## Optimization terminated normally:
##   Convergence detected: absolute parameter change was below tolerance

future <- make_future_dataframe(m, periods = 24, freq = "month")

forecast <- predict(m, future)
```

```
plot(m, forecast)
```



Facebook Prophet R API

---

9 Model Evaluation & Selection

9.1 Evaluate the accuracy of regression models.

a) Evaluation on training data.

```
train <- read.csv('/Users/boston_train.csv')
test <- read.csv('/Users/boston_test.csv')

set.seed(29)

# Random Forest Regression Model
# call the randomForest package
library(randomForest)
```

```
rfMod <- randomForest(Target ~ ., data = train)

# Evaluation on training data
predY <- predict(rfMod, train)
predY <- unname(predY)

# Determine coefficient of determination score
r2_rf <- 1 - ( (sum((train$Target -
                       predY)**2)) / (sum((train$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"
```

b) Evaluation on testing data.

```
# Random Forest Regression Model (rfMod)

# Evaluation on testing data
predY <- predict(rfMod, test)
predY <- unname(predY)

# Determine coefficient of determination score
r2_rf = 1 - ( (sum((test$Target -
                      predY)**2)) / (sum((test$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"
```

randomForest | predict() | unname()

The formula used here for the coefficient of determination score is based off the Python skearn formula for r2_score. For more information about model assessment in R, please review information about the R package caret.

9.2 Evaluate the accuracy of classification models.

a) Evaluation on training data.

```
train <- read.csv('/Users/digits_train.csv')
test <- read.csv('/Users/digits_test.csv')

set.seed(29)

# Random Forest Classification Model
# call the randomForest package
```

```
library(randomForest)

rfMod <- randomForest(as.factor(Target) ~ ., data = train)

# Evaluation on training data
predY <- predict(rfMod, train)
predY <- unname(predY)

# Determine accuracy score
accuracy_rf <- (1/nrow(train)) * sum(as.numeric(predY == 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
predY <- predict(rfMod, test)
predY <- unname(predY)

# Determine accuracy score
accuracy_rf <- (1/nrow(test)) * sum(as.numeric(predY == test$Target))
print(paste0("Random forest model accuracy: ", accuracy_rf))
```

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

randomForest | predict() | unname()

The formula used here for the accuracy score is based off the Python skearn formula for accuracy_score. For more information about model assessment in R, please review information about the R package caret.

9.3 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')

# call the caret and randomForest packages
library(caret)
library(randomForest)

set.seed(29)

# Create the 5 cross validation folds
train_control <- trainControl(method = "cv", number = 5,
                              savePredictions = TRUE)
```

```r
# 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: 95.77% +/- 1.68%"
```

caret | randomForest

b) ShuffleSplit

```r
# call the caret and randomForest packages
library(caret)
library(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],])
}
```

```r
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: 96.24% +/- 0.53%"
```

caret | randomForest | createDataPartition

---

10 Text Analytics

11 Deep Learning

Appendix

1 Built-in R Objects

Vectors

- Logical
- Numeric
- Integer
- Complex
- Character
- Raw

Lists

Matrics

Arrays

Factors

Data Frames

2 R packages used in this tutorial

gdata

Data manipulation

rjson

Converting R objects into JSON objects, and JSON objects into R objects

ggplot2

Visualizations and graphics

dplyr

Working with data frame like objects

tree

Decision trees models

randomForest

Random forest models

gbm

Gradient boosting models

xgboost

Extreme gradient boosting models

e1071

Support vector machine models

RSNNS

Neural network models

kernlab

Spectral clustering

dbscan

DBSCAN clustering

kohonen

Supervised and unsupervised self-organizing maps

forecast

Displaying and analyzing time series for forecasting

caret

Training and plotting classification and regression models

Alphabetical Index

Array

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
```

Data Frame

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. Please see the following example of named list creation and access:

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

```
## James
##    12
```

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:

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

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

```r
print(list1[1])
```

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

Vector

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

For more information on R packages and functions, along with helpful examples, please see R.