4.7 Exercises

- Exercise 10
- Exercise 11
- Exercise 12
- Exercise 13

Exercise 10

This questions should be answered using the **Weekly** data set, which is part of the **ISLR** package. This data is similar in nature to the **Smarket** data from this chapter's lab, except that it contains 1,089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.

a. Produce some numerical and graphical summaries of the **Weekly** data. Do there appear to be any patterns?

```
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.4.4
```

```
fix(Weekly)
```

```
summary(Weekly)
```

```
##       Year          Lag1               Lag2               Lag3         
##  Min.   :1990   Min.   :-18.1950   Min.   :-18.1950   Min.   :-18.1950  
##  1st Qu.:1995   1st Qu.: -1.1540   1st Qu.: -1.1540   1st Qu.: -1.1580  
##  Median :2000   Median :  0.2410   Median :  0.2410   Median :  0.2410  
##  Mean   :2000   Mean   :  0.1506   Mean   :  0.1511   Mean   :  0.1472  
##  3rd Qu.:2005   3rd Qu.:  1.4050   3rd Qu.:  1.4090   3rd Qu.:  1.4090  
##  Max.   :2010   Max.   : 12.0260   Max.   : 12.0260   Max.   : 12.0260  
##       Lag4               Lag5              Volume       
##  Min.   :-18.1950   Min.   :-18.1950   Min.   :0.08747  
##  1st Qu.: -1.1580   1st Qu.: -1.1660   1st Qu.:0.33202  
##  Median :  0.2380   Median :  0.2340   Median :1.00268  
##  Mean   :  0.1458   Mean   :  0.1399   Mean   :1.57462  
##  3rd Qu.:  1.4090   3rd Qu.:  1.4050   3rd Qu.:2.05373  
##  Max.   : 12.0260   Max.   : 12.0260   Max.   :9.32821  
##      Today          Direction 
##  Min.   :-18.1950   Down:484  
##  1st Qu.: -1.1540   Up  :605  
##  Median :  0.2410             
##  Mean   :  0.1499             
##  3rd Qu.:  1.4050             
##  Max.   : 12.0260             
```

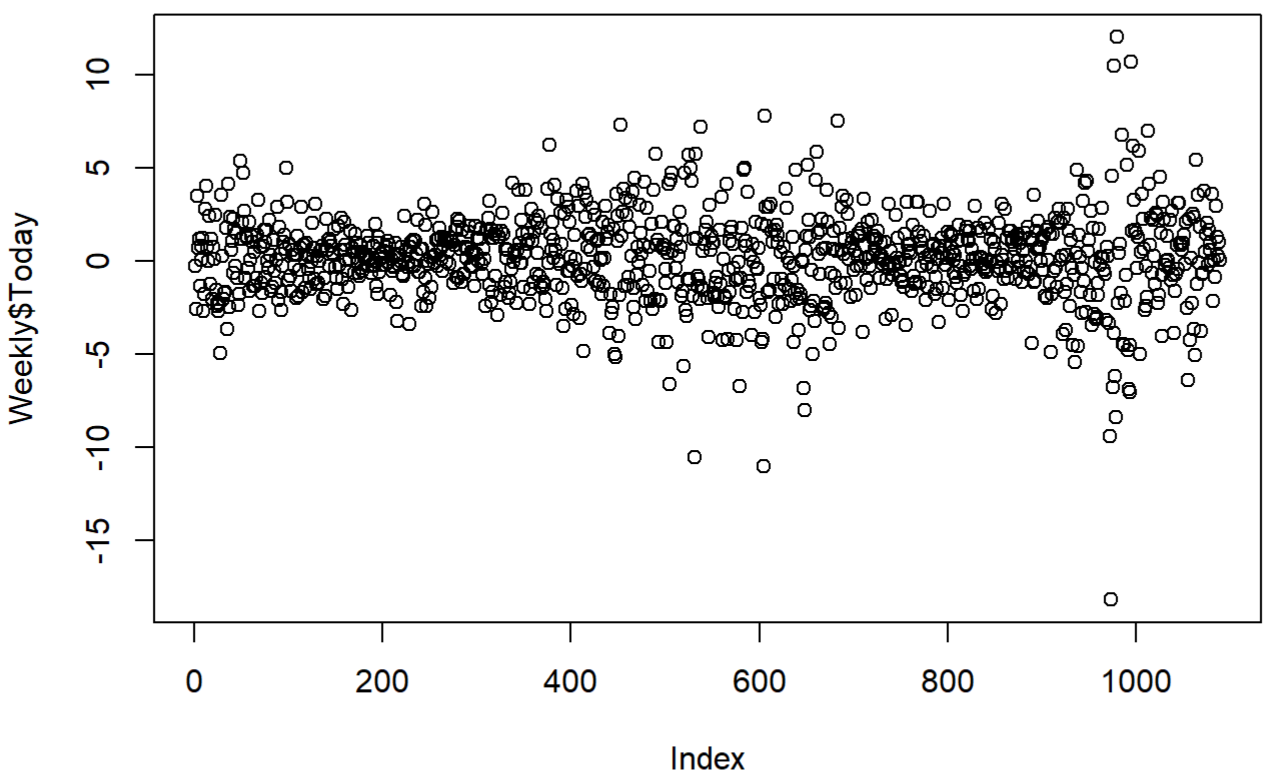
*As you would expect, the lag variables and the **Today** variable all share the same distributions.*

```
605 / (484 + 605)
```

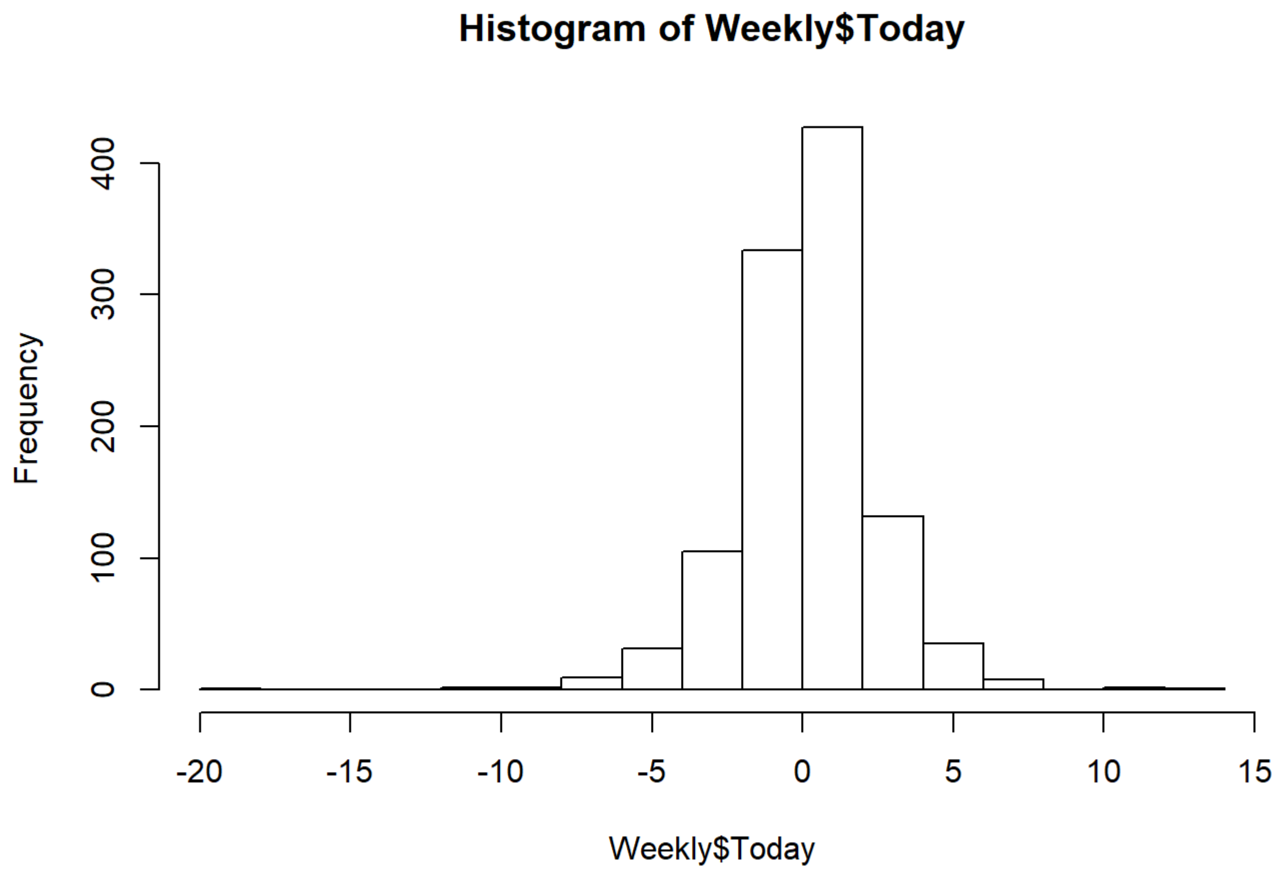
```
## [1] 0.5555556
```

There are good deal more **Up** *days than there are* **Down** *days*. 55.56% *of the day's are recorded as a positive weekly return.*

```
plot(Weekly$Today)
```

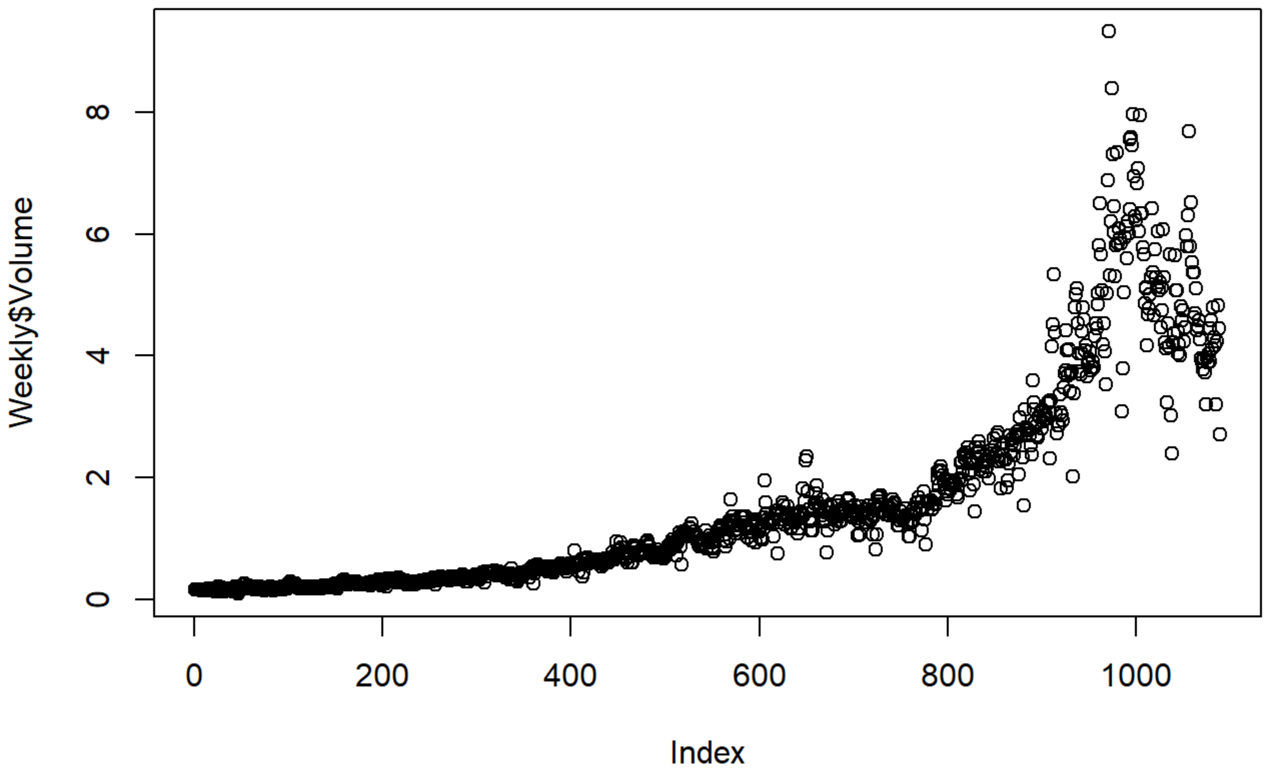



```
hist(Weekly$Today)
```



At a first glance, the weekly returns are pretty normally distributed around 0, *with no real trends over time.*

```
plot(Weekly$Volume)
```



The **Volume** *of shares traded increases in an exponential fashion over time, so there has definitely been consistent market growth.*

b. Use the full data set to perform a logistic regression with **Direction** as the response and the five lag variables plus **Volume** as predictors. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?

```
glm.fit = glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume, data = Weekly, fami
ly = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
##     Volume, family = binomial, data = Weekly)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6949  -1.2565   0.9913   1.0849   1.4579
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.26686    0.08593   3.106   0.0019 **
## Lag1        -0.04127    0.02641  -1.563   0.1181
## Lag2         0.05844    0.02686   2.175   0.0296 *
## Lag3        -0.01606    0.02666  -0.602   0.5469
## Lag4        -0.02779    0.02646  -1.050   0.2937
## Lag5        -0.01447    0.02638  -0.549   0.5833
## Volume      -0.02274    0.03690  -0.616   0.5377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1496.2  on 1088  degrees of freedom
## Residual deviance: 1486.4  on 1082  degrees of freedom
## AIC: 1500.4
##
## Number of Fisher Scoring iterations: 4
```

The only statistically significant predictor appears to be **Lag2**, *with a* p-*value of* 0.0296.

c. Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.

```
glm.prob = predict(glm.fit, type = 'response')
glm.pred = rep('Down', length(glm.prob))
glm.pred[glm.prob > .5] = 'Up'
table(glm.pred, Weekly$Direction)
```

```
##
## glm.pred Down  Up
##     Down   54  48
##     Up    430 557
```

At a probability cutoff of 0.5, *this model is very optimistic. It predicts that* 987 *of the weeks from the training data should have a positive return* (*versus the* 605 *weeks of the training data that actually have a positive return*). 987 *weeks corresponds to* 90.6% *of the data* (*versus the* 55.6% *that actually had a positive return*).

d. Now fit the logistic regression omdel using a training data preiod from 1990 to 2008, with **Lag2** as the only predictor. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data form 2009 and 2010).

```
train = Weekly$Year < 2009
summary(Weekly[train,])
```

```
##       Year           Lag1              Lag2              Lag3        
##  Min.   :1990   Min.   :-18.1950   Min.   :-18.1950   Min.   :-18.1950  
##  1st Qu.:1994   1st Qu.: -1.1540   1st Qu.: -1.1470   1st Qu.: -1.1540  
##  Median :1999   Median :  0.2310   Median :  0.2340   Median :  0.2310  
##  Mean   :1999   Mean   :  0.1245   Mean   :  0.1278   Mean   :  0.1229  
##  3rd Qu.:2004   3rd Qu.:  1.3340   3rd Qu.:  1.3370   3rd Qu.:  1.3370  
##  Max.   :2008   Max.   : 12.0260   Max.   : 12.0260   Max.   : 12.0260  
##       Lag4               Lag5             Volume            Today         
##  Min.   :-18.1950   Min.   :-18.195   Min.   :0.08747   Min.   :-18.1950  
##  1st Qu.: -1.1540   1st Qu.: -1.154   1st Qu.:0.30734   1st Qu.: -1.1540  
##  Median :  0.2300   Median :  0.230   Median :0.80485   Median :  0.2310  
##  Mean   :  0.1222   Mean   :  0.121   Mean   :1.20597   Mean   :  0.1305  
##  3rd Qu.:  1.3370   3rd Qu.:  1.337   3rd Qu.:1.51585   3rd Qu.:  1.3370  
##  Max.   : 12.0260   Max.   : 12.026   Max.   :9.32821   Max.   : 12.0260  
##  Direction 
##  Down:441  
##  Up  :544  
##            
##            
##            
## 
```

```
x = 441 + 544
x / (484 + 605)
```

```
## [1] 0.9044995
```

```
544 / x
```

```
## [1] 0.5522843
```

The training data consists of 90.4% *of the whole data set, leaving* 9.6% *of the data to use as test data. The percentage of training data with a positive weekly return is close to the overall percentage at* 55.2%.

```
summary(Weekly[!train,])
```

```
##       Year          Lag1            Lag2            Lag3
##  Min.   :2009   Min.   :-7.0350   Min.   :-7.0350   Min.   :-7.0350
##  1st Qu.:2009   1st Qu.:-1.0608   1st Qu.:-1.2143   1st Qu.:-1.2143
##  Median :2010   Median : 0.5220   Median : 0.4615   Median : 0.5220
##  Mean   :2010   Mean   : 0.3976   Mean   : 0.3714   Mean   : 0.3775
##  3rd Qu.:2010   3rd Qu.: 2.2393   3rd Qu.: 2.2393   3rd Qu.: 2.2393
##  Max.   :2010   Max.   :10.7070   Max.   :10.7070   Max.   :10.7070
##       Lag4              Lag5             Volume          Today
##  Min.   :-7.0350   Min.   :-7.0350   Min.   :2.390   Min.   :-7.0350
##  1st Qu.:-1.2143   1st Qu.:-1.3233   1st Qu.:4.234   1st Qu.:-1.0608
##  Median : 0.4615   Median : 0.4370   Median :4.851   Median : 0.4615
##  Mean   : 0.3693   Mean   : 0.3191   Mean   :5.066   Mean   : 0.3333
##  3rd Qu.: 2.2393   3rd Qu.: 2.2043   3rd Qu.:5.794   3rd Qu.: 2.2043
##  Max.   :10.7070   Max.   :10.7070   Max.   :7.963   Max.   :10.7070
##  Direction
##  Down:43
##  Up  :61
##
##
##
##
```

```
61 / (43 + 61)
```

```
## [1] 0.5865385
```

The percentage of positive weekly returns is a bit higher for the test data, at 58.7%. *Conversely, the percentage of negative weekly returns is* 41.3%.

```
glm.fit = glm(Direction ~ Lag2, data = Weekly[train,], family = binomial)
summary(glm.fit)
```

```
## 
## Call:
## glm(formula = Direction ~ Lag2, family = binomial, data = Weekly[train,
##     ])
## 
## Deviance Residuals: 
##    Min      1Q  Median      3Q     Max  
## -1.536  -1.264   1.021   1.091   1.368  
## 
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)   
## (Intercept)  0.20326    0.06428   3.162  0.00157 **
## Lag2         0.05810    0.02870   2.024  0.04298 * 
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
##     Null deviance: 1354.7  on 984  degrees of freedom
## Residual deviance: 1350.5  on 983  degrees of freedom
## AIC: 1354.5
## 
## Number of Fisher Scoring iterations: 4
```

```
glm.prob = predict(glm.fit, Weekly[!train,], type = 'response')
glm.pred = rep('Down', length(glm.prob))
glm.pred[glm.prob > .5] = 'Up'
table(glm.pred, Weekly[!train, 'Direction'])
```

```
## 
## glm.pred Down Up
##     Down    9  5
##     Up     34 56
```

This model is also quite optimistic, predicting 86.5% *of the test data to have positive weekly returns.*

```
mean(glm.pred == Weekly[!train, 'Direction'])
```

```
## [1] 0.625
```

The model is correct on the test data 62.5% *of the time, which is only a bit better than a dummy model of predicting only* **Up**. *The dummy model would have a test accuracy rate of* 58.7%.

e. Repeat (d) Using LDA.

```
library(MASS)
```

```
lda.fit = lda(Direction ~ Lag2, data = Weekly[train,])
lda.fit
```

```
## Call:
## lda(Direction ~ Lag2, data = Weekly[train, ])
##
## Prior probabilities of groups:
##      Down        Up
## 0.4477157 0.5522843
##
## Group means:
##             Lag2
## Down -0.03568254
## Up    0.26036581
##
## Coefficients of linear discriminants:
##            LD1
## Lag2 0.4414162
```

```
lda.pred  = predict(lda.fit, Weekly[!train,])
lda.class = lda.pred$class
table(lda.class, Weekly[!train, 'Direction'])
```

```
##
## lda.class Down Up
##      Down    9  5
##      Up     34 56
```

```
mean(lda.class == Weekly[!train, 'Direction'])
```

```
## [1] 0.625
```

The LDA model performs exactly the same as the logistic regression model.

f. Repeat (d) Using QDA.

```
qda.fit = qda(Direction ~ Lag2, data = Weekly[train,])
qda.fit
```

```
## Call:
## qda(Direction ~ Lag2, data = Weekly[train, ])
##
## Prior probabilities of groups:
##      Down        Up
## 0.4477157 0.5522843
##
## Group means:
##             Lag2
## Down -0.03568254
## Up    0.26036581
```

```
qda.pred  = predict(qda.fit, Weekly[!train,])
qda.class = qda.pred$class
table(qda.class, Weekly[!train, 'Direction'])
```

```
##
## qda.class Down Up
##      Down    0  0
##      Up     43 61
```

The QDA model is worse than the logistic regression and LDA models. It predicts all of the training data to be a positive weekly return, which yields a training accuracy rate of 58.7% *- the same as the dummy model of only predicting* **Up**.

g. Repeat (d) using KNN with K = 1.

```
library(class)
```

```
train.X = data.frame(Weekly$Lag2[train])
test.X  = data.frame(Weekly$Lag2[!train])
train.Direction = Weekly$Direction[train]
knn.pred = knn(train.X, test.X, train.Direction, k = 1)
table(knn.pred, Weekly$Direction[!train])
```

```
##
## knn.pred Down Up
##     Down   21 30
##     Up     22 31
```

```
mean(knn.pred == Weekly[!train, 'Direction'])
```

```
## [1] 0.5
```

The KNN classifier with $K = 1$ *performs poorly. It has a relatively high test error rate, and it does not even have better performance on predicting specifically for* **Up** *or* **Down**.

h. Which of these methods appear to provide the best results on this data?

Of these methods, the logistic regression and LDA models perform the best on this data.

i. Experiment with different combinations of predictors, including possible transformations and interactions for each of the methods. Report the variables, method, and associated confusion matrix that appears to provide the best results on the hold out data. Note that you should also experiment with values for *K* in the KNN classifier.

```
train.X = Weekly[ train, c('Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5')]
test.X  = Weekly[!train, c('Lag1', 'Lag2', 'Lag3', 'Lag4', 'Lag5')]
knn.pred = knn(train.X, test.X, train.Direction, k = 1)
table(knn.pred, Weekly$Direction[!train])
```

```
##
## knn.pred Down Up
##     Down   21 28
##     Up     22 33
```

Using all of the lag predictors with $K = 1$ KNN yields almost the exact same result as just using **Lag2**.

```
knn.pred = knn(train.X, test.X, train.Direction, k = 2)
table(knn.pred, Weekly$Direction[!train])
```

```
##
## knn.pred Down Up
##     Down   19 29
##     Up     24 32
```

$K = 2$ *is a little bit worse, but still very similar.*

```
for (k in 3:8) {
  knn.pred = knn(train.X, test.X, train.Direction, k = k)
  print(c(k, mean(knn.pred == Weekly[!train, 'Direction'])))
}
```

```
## [1] 3.0000000 0.5576923
## [1] 4.0000000 0.5288462
## [1] 5.0000000 0.5480769
## [1] 6.0000000 0.5673077
## [1] 7.0000000 0.5673077
## [1] 8.0000000 0.5769231
```

KNN with $K = 4$ seems to be pretty decent with 60.6% *test accuracy. However, this performance is still worse than the LDA using* **Lag2**.

```
table(knn(train.X, test.X, train.Direction, k = 4), Weekly[!train, 'Direction'])
```

```
##
##        Down Up
##   Down   18 29
##   Up     25 32
```

Like the other KNN results we have looked at, this model is not as optimistic as the logistic regression, LDA, and QDA. However, it does still predict most of the data to be a positive weekly return.

```
glm.fit = glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, data = Weekly[train,], famil
y = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5, family = binomial,
##     data = Weekly[train, ])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8534  -1.2491   0.9941   1.0886   1.5126
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.220141   0.065069   3.383 0.000716 ***
## Lag1        -0.055438   0.029051  -1.908 0.056357 .
## Lag2         0.053290   0.029348   1.816 0.069401 .
## Lag3        -0.009292   0.029265  -0.318 0.750859
## Lag4        -0.024484   0.028970  -0.845 0.398036
## Lag5        -0.031544   0.029005  -1.088 0.276796
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1354.7  on 984  degrees of freedom
## Residual deviance: 1345.1  on 979  degrees of freedom
## AIC: 1357.1
##
## Number of Fisher Scoring iterations: 4
```

```
glm.prob = predict(glm.fit, Weekly[!train,], type = 'response')
glm.pred = rep('Down', sum(!train))
glm.pred[glm.prob > .5] = 'Up'
table(glm.pred, Weekly[!train, 'Direction'])
```

```
##
## glm.pred Down Up
##     Down   10 14
##     Up     33 47
```

```
mean(glm.pred == Weekly[!train, 'Direction'])
```

```
## [1] 0.5480769
```

The logistic regression using all of the lag predictors doesn’t have great performance. It is has a higher test error rate than the dummy model.

```
glm.fit = glm(Direction ~ Lag2 + Volume, data = Weekly[train,], family = binomial)
glm.prob = predict(glm.fit, Weekly[!train,], type = 'response')
glm.pred = rep('Down', sum(!train))
glm.pred[glm.prob > .5] = 'Up'
mean(glm.pred == Weekly[!train, 'Direction'])
```

```
## [1] 0.5384615
```

```
glm.fit = glm(Direction ~ I(Lag2^2), data = Weekly[train,], family = binomial)
glm.prob = predict(glm.fit, Weekly[!train,], type = 'response')
glm.pred = rep('Down', sum(!train))
glm.pred[glm.prob > .5] = 'Up'
mean(glm.pred == Weekly[!train, 'Direction'])
```

```
## [1] 0.5865385
```

I can't find anything with a better performance on the test data than the original logistic regression. Note that using transformations like **log()** *or* **sqrt()** *are not available on the lag predictors since they contain negative values.*

Exercise 11

In this problem, you will develop a model to predict whether a given car gets high or low gas mileage based on the **Auto** data set.

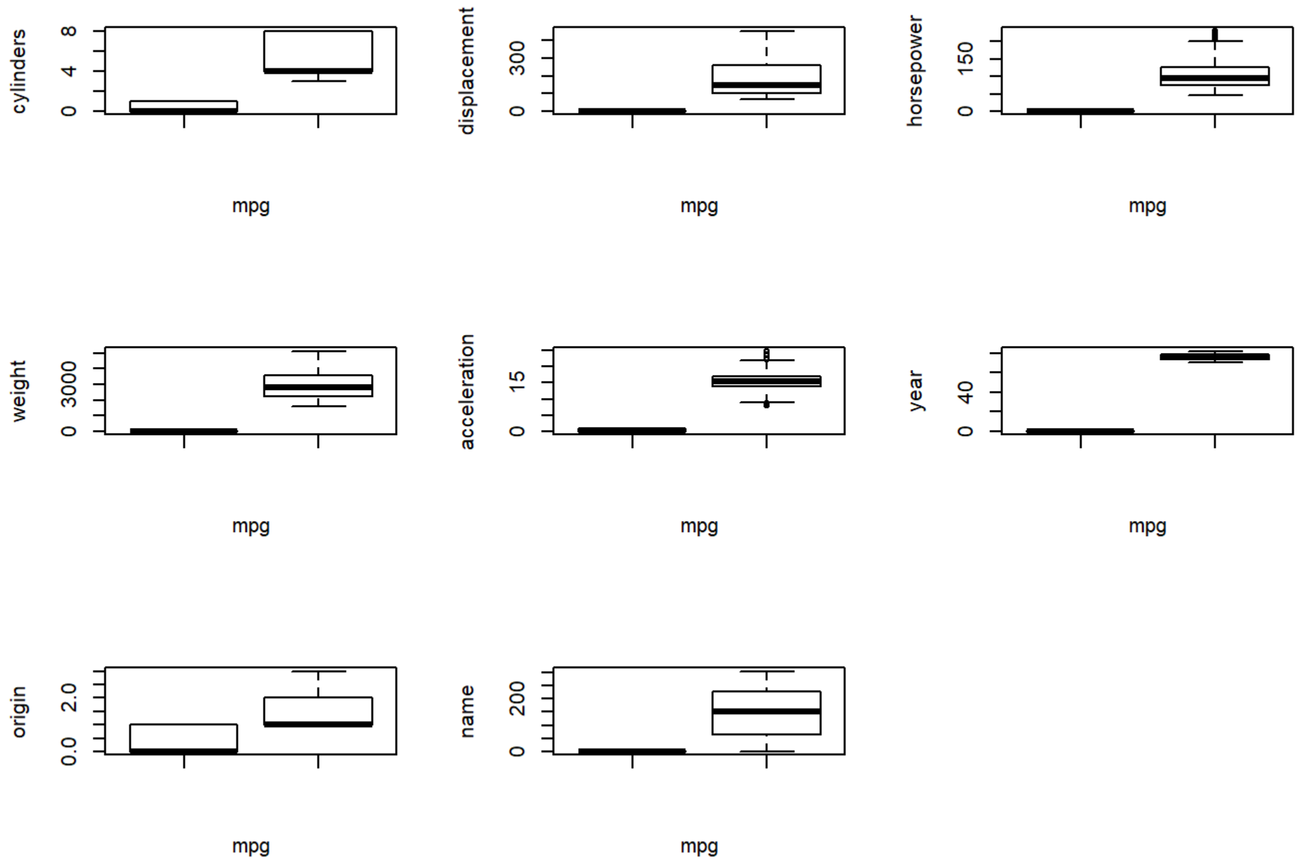
```
fh = 'D:/GoogleDrive/Introduction to Statistical Learning with Applications in R/data-se
ts/Auto.csv'
Auto = read.csv(file = fh, header = TRUE, na.strings = '?')
```

a. Create a binary variable, **mpg01**, that contains a 1 if **mpg** contains a value above its median, and a 0 if **mpg** contains a value below its median. You can compute the median using the **median()** function. NOte you may find it helpful to use the **data.frame()** function to create a single data set containing both **mpg01** and the other **Auto** variables.

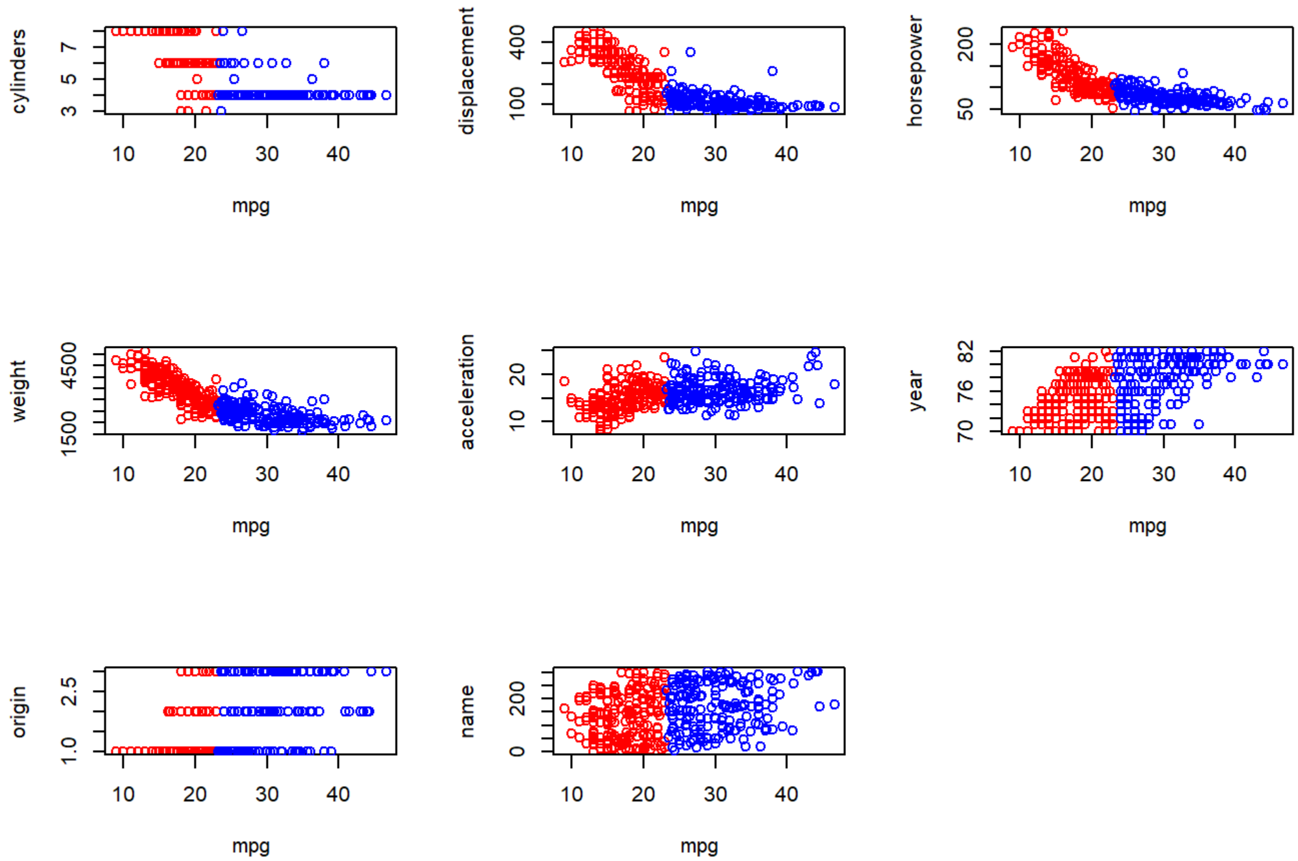
```
Auto$mpg01 = 0
Auto$mpg01[Auto$mpg > median(Auto$mpg)] = 1
```

b. Explore the data graphically in order to investigate the association between **mpg01** and the other features. Which of the other features seem most likely to be useful in predicting **mpg01**? Scatterplots and boxplots may be useful tools to answer this question. Describe your findings.

```
par(mfrow=c(3,3))
for (c in names(Auto)[-c(1,10)]) {
  boxplot(Auto$mpg01, Auto[,c], xlab = 'mpg', ylab = c)
}
```




```
idx0 = Auto$mpg01 == 0
idx1 = !idx0
par(mfrow = c(3,3))
xlim = c(min(Auto$mpg), max(Auto$mpg))
for (c in names(Auto)[-c(1,10)]) {
  if (c == 'name') {
    ylim = c(0, length(unique(Auto[,c])))
  } else {
    ylim = c(min(Auto[,c], na.rm = TRUE), max(Auto[,c], na.rm = TRUE))
  }
  plot(Auto$mpg[idx0], Auto[idx0,c], xlab = 'mpg', ylab = c, col = 'red', xlim = xlim, ylim = ylim)
  points(Auto$mpg[idx1], Auto[idx1,c], col = 'blue')
}
```



It looks like the number of cylinders, the engine displacement, the engine horsepower, and the vehicle weight are all important predictors. The 0 to 60 acceleration and model year of the vehicle may also be useful predictors.

c. Split the data into a training set and a test set.

```
set.seed(1)
Auto.less = Auto[Auto$mpg01 == 0,]
Auto.more = Auto[Auto$mpg01 == 1,]
sample.index = function(df, percent) {
  len = dim(df)[1]
  idx = rep(FALSE, len)
  for (i in sample(len, size = floor(percent * len), replace = FALSE)) idx[i] = TRUE
  return(idx)
}
less.test = sample.index(Auto.less, .1)
more.test = sample.index(Auto.more, .1)

Auto.test  = rbind(Auto.less[ less.test,], Auto.more[ more.test,])
Auto.train = rbind(Auto.less[!less.test,], Auto.more[!more.test,])
```

We'll just take 10% of the data pseudo-randomly. We'll randomly draw 10% of the observations with **mpg** *greater than it's median and another random draw for 10% of the observations with* **mpg** *less than it's median. That data set will be the test set. The rest of the data will be the training set.*

d. Perform LDA on the training data in order to predict **mpg01** using the variables that seemed most associated with **mpg01** in (b). What is the test error of the model obtained?

```
form = as.formula(mpg01 ~ cylinders + displacement + horsepower + weight + acceleration
+ year)
lda.fit = lda(form, data = Auto.train)
lda.fit
```

```
## Call:
## lda(form, data = Auto.train)
##
## Prior probabilities of groups:
##         0         1
## 0.5240793 0.4759207
##
## Group means:
##   cylinders displacement horsepower   weight acceleration     year
## 0  6.670270     268.0378  128.66486 3590.832     14.66811 74.31351
## 1  4.154762     113.3006   78.05357 2312.208     16.53452 77.78571
##
## Coefficients of linear discriminants:
##                        LD1
## cylinders    -0.3386864685
## displacement -0.0004135396
## horsepower    0.0128313961
## weight       -0.0014176477
## acceleration  0.0145067025
## year          0.1496640155
```

```
lda.class = predict(lda.fit, Auto.test)$class
table(lda.class, Auto.test$mpg01)
```

```
##
## lda.class  0  1
##         0 18  0
##         1  2 19
```

```
mean(lda.class == Auto.test$mpg01)
```

```
## [1] 0.9487179
```

The test error rate obtained is 5.13%. *It seems the selected predictors work well for the LDA method here.*

e. Repeat (d) using QDA.

```
qda.fit = qda(form, data = Auto.train)
qda.fit
```

```
## Call:
## qda(form, data = Auto.train)
##
## Prior probabilities of groups:
##         0         1
## 0.5240793 0.4759207
##
## Group means:
##   cylinders displacement horsepower   weight acceleration     year
## 0  6.670270     268.0378  128.66486 3590.832     14.66811 74.31351
## 1  4.154762     113.3006   78.05357 2312.208     16.53452 77.78571
```

```
qda.class = predict(qda.fit, Auto.test)$class
table(qda.class, Auto.test$mpg01)
```

```
##
## qda.class  0  1
##         0 20  2
##         1  0 17
```

```
mean(qda.class == Auto.test$mpg01)
```

```
## [1] 0.9487179
```

Interestingly, the QDA method has the same test error rate as LDA. However, it tends to prefer to classify **mpg01** *as* 0, *whereas the LDA result tends to classify* **mpg01** *as* 1.

f. Repeat (d) using logistic regression.

```
glm.fit = glm(form, data = Auto.train, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = form, family = binomial, data = Auto.train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.10644  -0.09711  -0.00041   0.19275   2.62892
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -21.458428   6.073500  -3.533 0.000411 ***
## cylinders      0.293116   0.449983   0.651 0.514793
## displacement  -0.011044   0.011701  -0.944 0.345259
## horsepower    -0.035038   0.026160  -1.339 0.180441
## weight        -0.004362   0.001201  -3.631 0.000282 ***
## acceleration   0.013488   0.148354   0.091 0.927556
## year           0.484263   0.080591   6.009 1.87e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 488.54  on 352  degrees of freedom
## Residual deviance: 138.12  on 346  degrees of freedom
##   (5 observations deleted due to missingness)
## AIC: 152.12
##
## Number of Fisher Scoring iterations: 8
```

```
glm.prob = predict(glm.fit, Auto.test, type = 'response')
glm.pred = rep(0, dim(Auto.test)[1])
glm.pred[glm.prob > .5] = 1
table(glm.pred, Auto.test$mpg01)
```

```
##
## glm.pred  0  1
##        0 19  1
##        1  1 18
```

```
mean(glm.pred == Auto.test$mpg01)
```

```
## [1] 0.9487179
```

The logistic regression method yields, again, the same test error rate. This time though, the incorrect classifications are split between **mpg01** *of* 0 *and* 1.

g. Repeat (d) using KNN with multiple values for *K*. Which value of *K* seems to perform the best on this data set?

```
acc.rate = 1:10
x.cols = c('cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'year')
train.X = scale(Auto.train[,x.cols])
test.X  = scale(Auto.test[ ,x.cols])

# knn does not allow for any NaN's in the data
idx = complete.cases(train.X)

for (k in 1:10) {
  knn.pred = knn(train.X[idx,], test.X, Auto.train$mpg01[idx], k = k)
  acc.rate[k] = mean(knn.pred == Auto.test$mpg01)
}
df = data.frame(k = 1:10, accuracy = acc.rate, error = 1 - acc.rate)
df
```

```
##     k  accuracy      error
## 1   1 0.9487179 0.05128205
## 2   2 0.8974359 0.10256410
## 3   3 0.9487179 0.05128205
## 4   4 0.8974359 0.10256410
## 5   5 0.9487179 0.05128205
## 6   6 1.0000000 0.00000000
## 7   7 0.9743590 0.02564103
## 8   8 0.9743590 0.02564103
## 9   9 0.9743590 0.02564103
## 10 10 0.9743590 0.02564103
```

With $K = 6$, the KNN method perfectly predicts the test data.

Exercise 12

This problem involves writing functions.

a. Write a function, **Power()**, that prints out the result of raising 2 to the 3rd power. In other words, your function should compute 2^3 and print out the results. *Hint: Recall that* **x^a** *raises* **x** *to the power* **a**. *Use the* **print()** *function to output the result.*

```
Power = function () {
  print(2^3)
}
Power()
```

```
## [1] 8
```

b. Create a new function, **Power2()**, that allows you to pass *any* two numbers, **x** and **a**, and prints out the value of **x^a**. You can do this by beginning your function with the line `Power2=function(x,a){` You should be able to call your function by entering, for instance, `Power2(3,8)` on the command line. This should output the value of 3^8, namely, 6,561.

```
Power2 = function (x,a) {
  print(x^a)
}
Power2(3,8)
```

```
## [1] 6561
```

c. Using the **Power2()** function that you just wrote, compute 10^3, 8^{17}, and 131^3.

```
Power2(10,3)
```

```
## [1] 1000
```

```
Power2(8,17)
```

```
## [1] 2.2518e+15
```

```
Power2(131,3)
```

```
## [1] 2248091
```

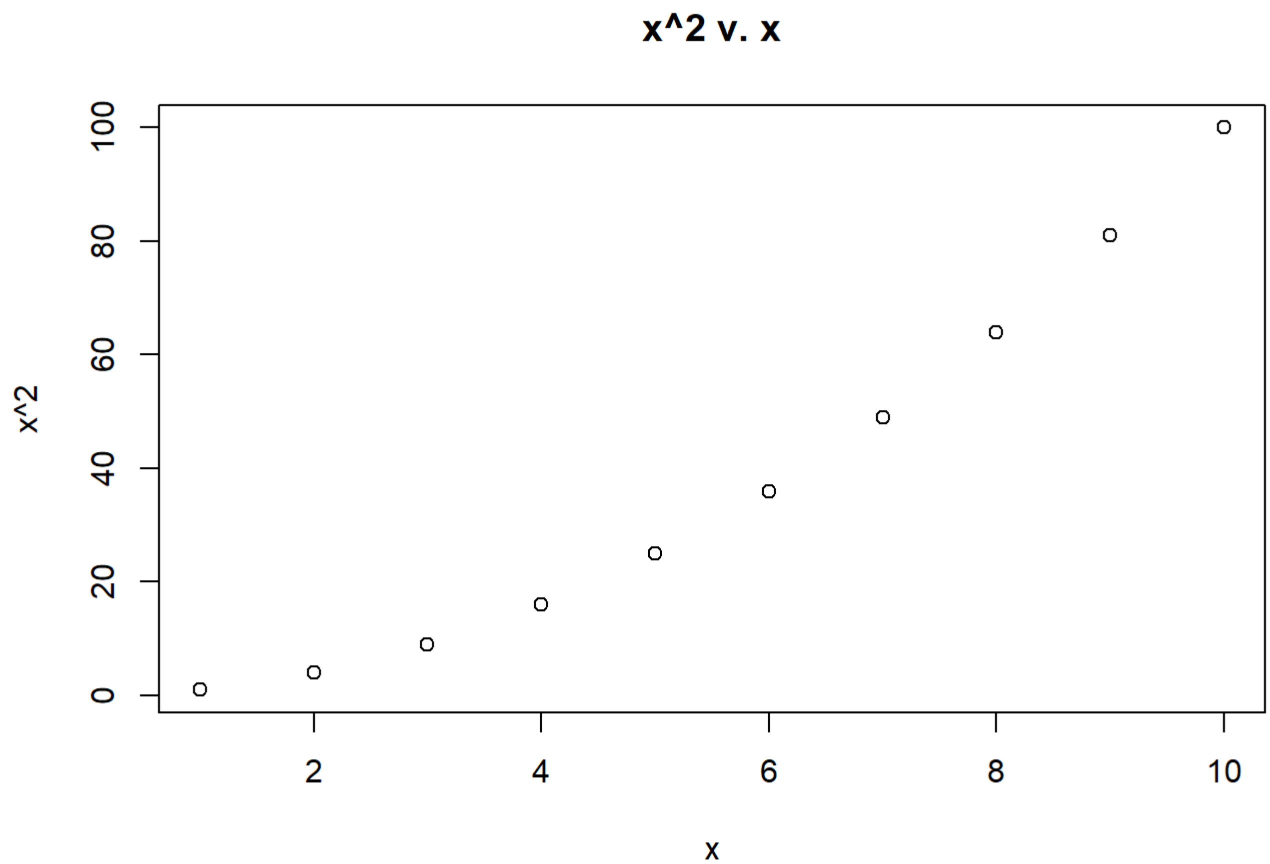
d. Now create a new function, **Power3()**, that actually *returns* the result **x^a** as an R object, rather than simply printing it to the screen. That is, if you store the value **x^a** in an object called **result** within your function, then you can simply **return()** this result, using the following line: `return(result)` The line above should be the last line in your function, before the **}** symbol.

```
Power3 = function (x,a) return(x^a)
result = Power3(2,3)
result
```

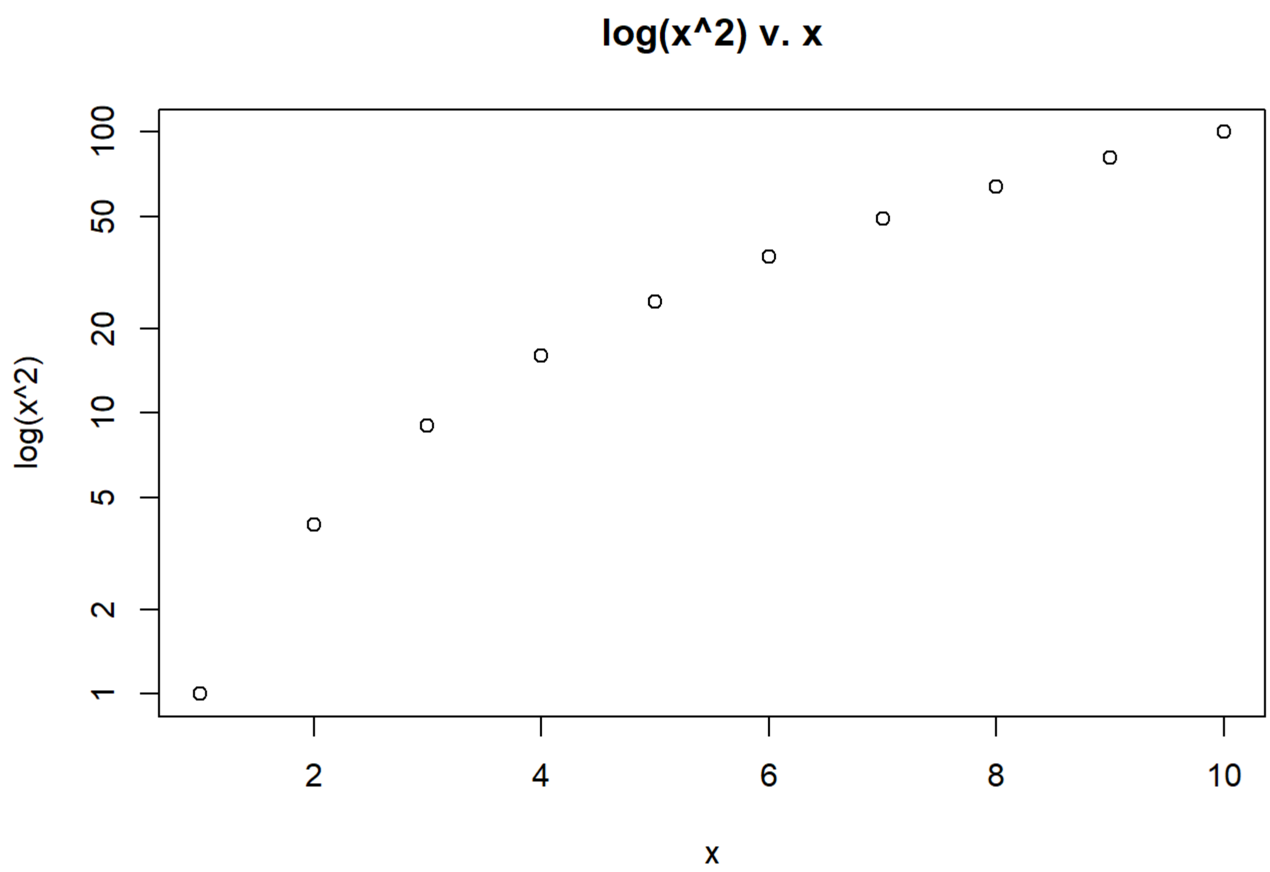
```
## [1] 8
```

e. Now using the **Power3()** function, create a plot of $f(x) = x^2$. The *x*-axis should display a range of integers from 1 to 10, and the *y*-axis should display x^2. Label the axes appropriately, and use an appropriate title for the figure. Consider displaying either the *x*-axis, the *y*-axis, or both on the log-scale. You can do this by using **log='x'**, **log='y'**, or **log='xy'** as arguments to the **plot()** function.

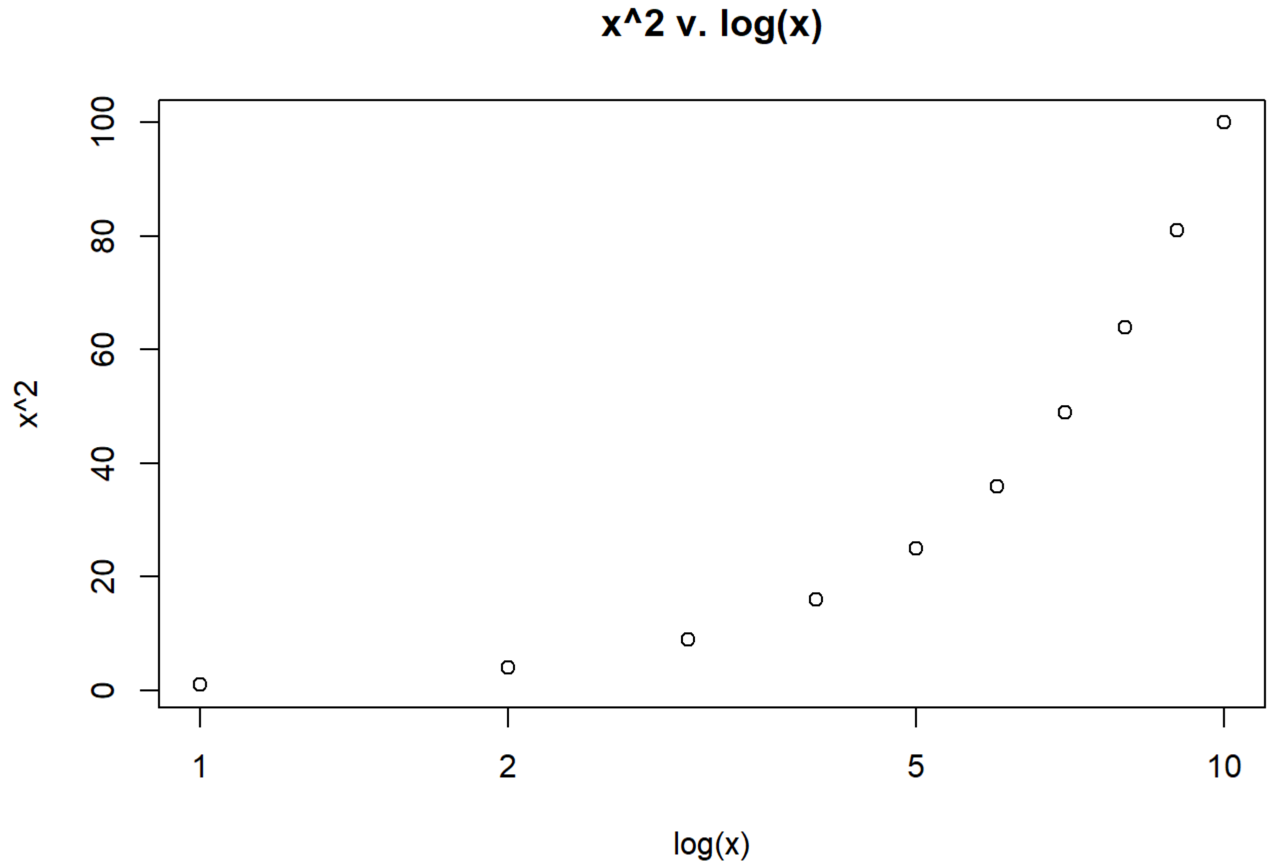
```
x = 1:10
plot(x, Power3(x,2), main = 'x^2 v. x', xlab = 'x', ylab = 'x^2')
```



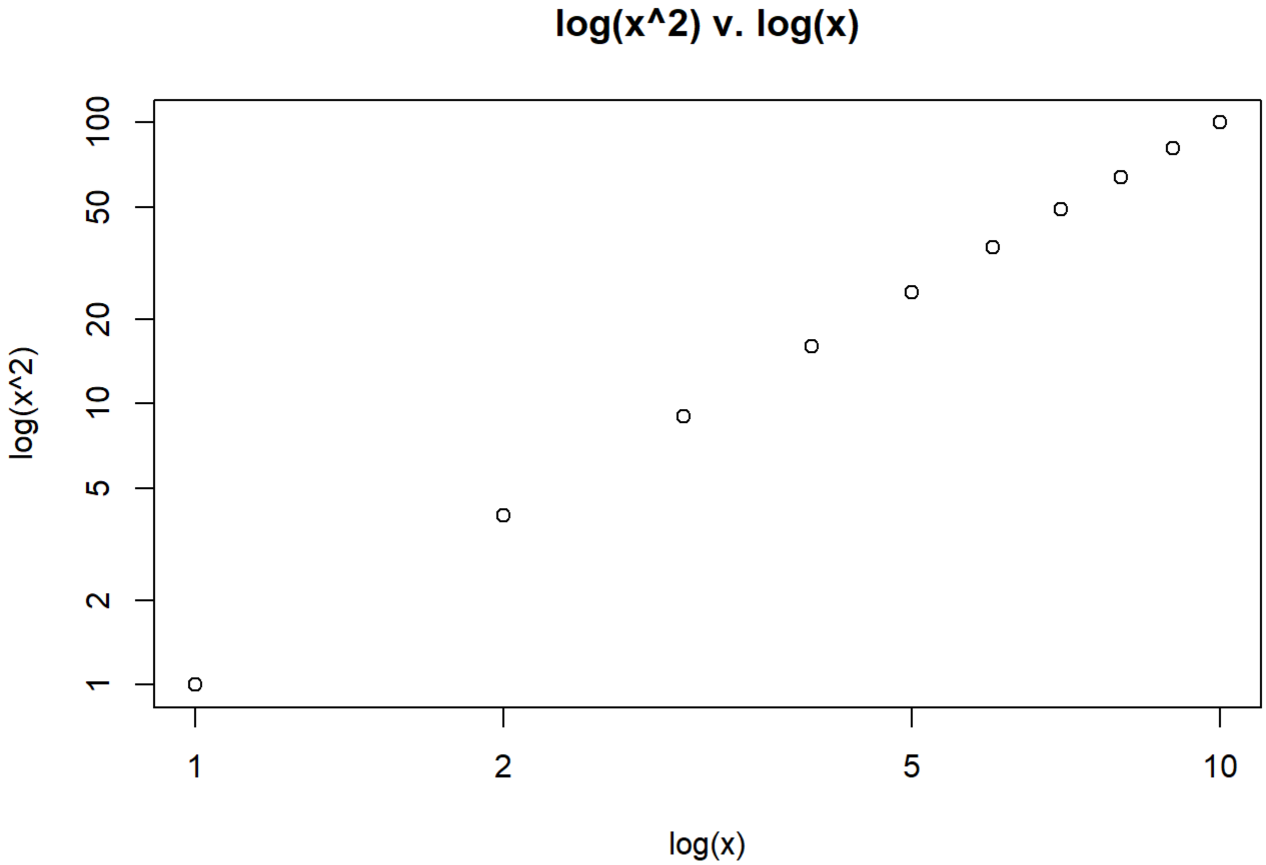

```
x = 1:10
plot(x, Power3(x,2), main = 'log(x^2) v. x', xlab = 'x', ylab = 'log(x^2)', log = 'y')
```




```
x = 1:10
plot(x, Power3(x,2), main = 'x^2 v. log(x)', xlab = 'log(x)', ylab = 'x^2', log = 'x')
```

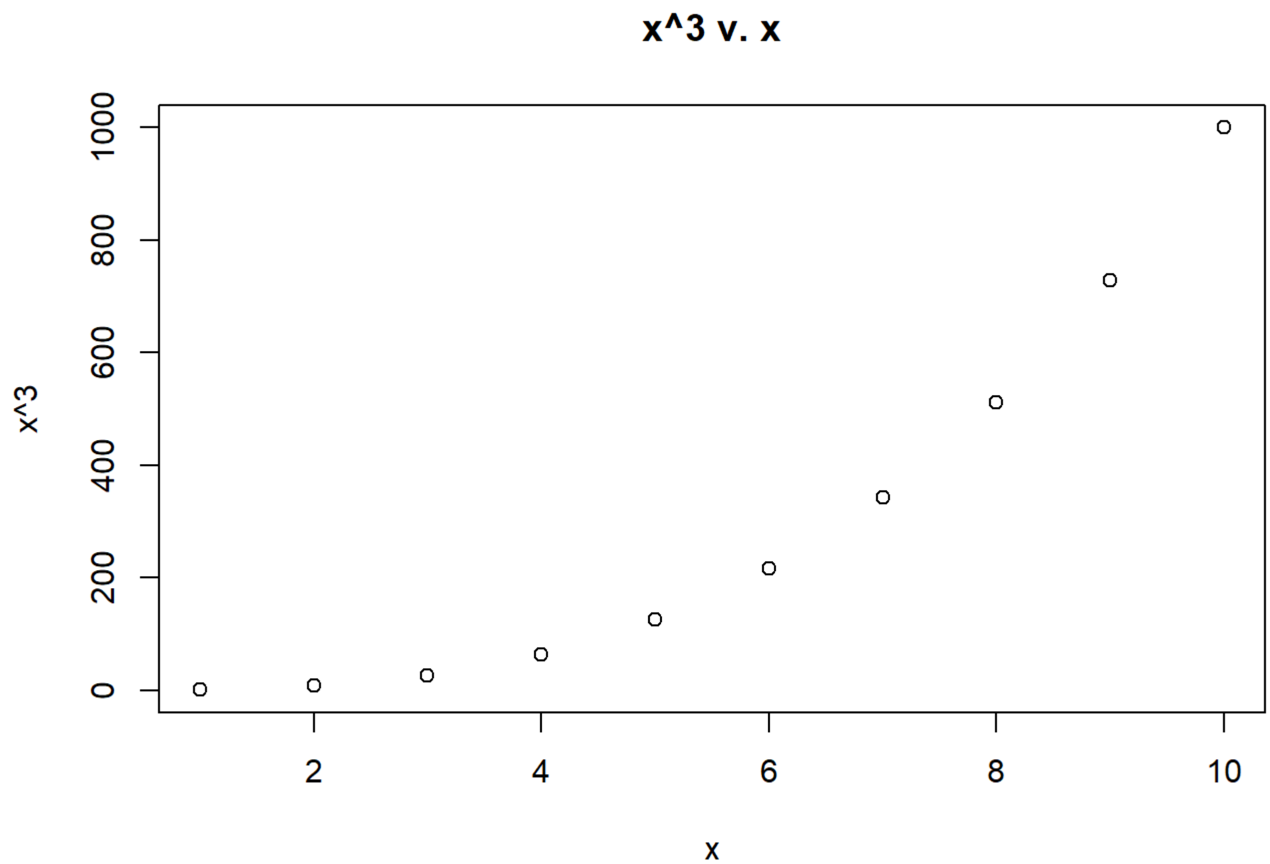



```
x = 1:10
plot(x, Power3(x,2), main = 'log(x^2) v. log(x)', xlab = 'log(x)', ylab = 'log(x^2)', lo
g = 'xy')
```



f. Create a function, **PlotPower()**, that allows you to create a plot of **x** against **x^a** for a fixed **a** and for a range of values **x**. For instance, if you call `PlotPower(1:10,3)` then a plot should be created with an *x*-axis taking on values 1,2,…,10, and a *y*-axis taking on values $1^3, 2^3, \ldots, 10^3$.

```
PlotPower = function (x,a) plot(x, x^a, main = sprintf('x^%i v. x', a), xlab = 'x', ylab
= sprintf('x^%i', a))
PlotPower(1:10, 3)
```



Exercise 13

Using the **Boston** data set, fit classification models in order to predict whether a given suburb has a crime rate above or below the median. Explore logistic regression, LDA, and KNN models using various subsets of the predictors. Describe your findings.

```
Boston$crim01 = 0
Boston$crim01[Boston$crim > median(Boston$crim)] = 1
```

First we make a new variable in the Boston data set that indicates whether the crime rate is above or below the median. We will call this variable **crim01**. *A value of* 0 *will indicate that the suburb's crime rate is less than or equal to the median. A value of* 1 *will indicate that the suburb's crime rate is above the median.*

```
glm.fit = glm(crim01 ~ ., data = Boston[,-1], family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = crim01 ~ ., family = binomial, data = Boston[,
##     -1])
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3946  -0.1585  -0.0004   0.0023   3.4239
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -34.103704   6.530014  -5.223 1.76e-07 ***
## zn           -0.079918   0.033731  -2.369  0.01782 *
## indus        -0.059389   0.043722  -1.358  0.17436
## chas          0.785327   0.728930   1.077  0.28132
## nox          48.523782   7.396497   6.560 5.37e-11 ***
## rm           -0.425596   0.701104  -0.607  0.54383
## age           0.022172   0.012221   1.814  0.06963 .
## dis           0.691400   0.218308   3.167  0.00154 **
## rad           0.656465   0.152452   4.306 1.66e-05 ***
## tax          -0.006412   0.002689  -2.385  0.01709 *
## ptratio       0.368716   0.122136   3.019  0.00254 **
## black        -0.013524   0.006536  -2.069  0.03853 *
## lstat         0.043862   0.048981   0.895  0.37052
## medv          0.167130   0.066940   2.497  0.01254 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 701.46  on 505  degrees of freedom
## Residual deviance: 211.93  on 492  degrees of freedom
## AIC: 239.93
##
## Number of Fisher Scoring iterations: 9
```

Next we will explore a logistic regression using all of the predictors, since R will provide statistical significance for each predictor. Of course, we want to exclude **crim** *from the fit. If you already have access to the crime rate itself, then "predicting" whether or not the crime rate is above the median is trivial.*

We can see from the fit that the proportion of residential land zoned (**zn**), *the nitrogen oxides concentration* (**nox**), *the weighted mean of distances to five Boston employment centres* (**dis**), *the index of accessibility to radial highways* (**rad**), *the full-value property-tax rate* (**tax**), *the pupil-teacher ratio* (**ptratio**), *the proportion of blacks by town* (**black**), *and the median value of owner occupied homes* (**medv**) *all have a p-value less than* 5%, *so these are the predictors that we will use for our models.*

```
set.seed(1)
cols = c('crim01', 'zn', 'nox', 'dis', 'rad', 'tax', 'ptratio', 'black', 'medv')
Boston.less = Boston[Boston$crim01 == 0,]
Boston.more = Boston[Boston$crim01 == 1,]

less.test = sample.index(Boston.less, .1)
more.test = sample.index(Boston.more, .1)

Boston.test  = rbind(Boston.less[ less.test, cols], Boston.more[ more.test, cols])
Boston.train = rbind(Boston.less[!less.test, cols], Boston.more[!more.test, cols])
```

Using the function we defined in Exercise 11c, *we split the Boston data set into a test set, consisting of* 10% *of the below-median-data and* 10% *of the above-median-data, and a training set, consisting of the rest of the data.*

```
glm.fit = glm(crim01 ~ ., data = Boston.train, family = binomial)
glm.prob = predict(glm.fit, Boston.test, type = 'response')
glm.pred = rep(0, dim(Boston.test)[1])
glm.pred[glm.prob > .5] = 1
table(glm.pred, Boston.test$crim01)
```

```
##
## glm.pred  0  1
##        0 23  1
##        1  2 24
```

```
mean(glm.pred == Boston.test$crim01)
```

```
## [1] 0.94
```

The resulting logistic regression has a 6% *test error rate - not bad.*

```
lda.fit = lda(crim01 ~ ., data = Boston.train)
lda.pred = predict(lda.fit, Boston.test)$class
table(lda.pred, Boston.test$crim01)
```

```
##
## lda.pred  0  1
##        0 25  3
##        1  0 22
```

```
mean(lda.pred == Boston.test$crim01)
```

```
## [1] 0.94
```

The linear discriminant analysis also has a 6% *test error rate. Seems like an amusing coincidence that these two methods yield the same test error rate in both this exercise and in Exercise* 11.

```
qda.fit = qda(crim01 ~ ., data = Boston.train)
qda.pred = predict(qda.fit, Boston.test)$class
table(qda.pred, Boston.test$crim01)
```

```
##
## qda.pred  0  1
##        0 25  3
##        1  0 22
```

```
mean(qda.pred == Boston.test$crim01)
```

```
## [1] 0.94
```

Again, a test error rate of 6%. *And again, amusing. In this case the confusion matrix is the same for both LDA and QDA. It makes you wonder if the same observations were misclassified.*

```
which(lda.pred != Boston.test$crim01)
```

```
## [1] 26 27 34
```

```
which(qda.pred != Boston.test$crim01)
```

```
## [1] 26 28 34
```

LDA and QDA yield almost the exact same results, but not quite.

```
acc.rate = rep(0,10)
for (k in 1:10) {
  knn.pred = knn(scale(Boston.train[,-1]), scale(Boston.test[,-1]), Boston.train$crim01,
k = k)
  acc.rate[k] = mean(knn.pred == Boston.test$crim01)
}
data.frame(k = 1:10, accuracy = acc.rate, error = 1 - acc.rate)
```

```
##     k accuracy error
## 1   1     0.96  0.04
## 2   2     0.98  0.02
## 3   3     0.96  0.04
## 4   4     0.98  0.02
## 5   5     0.98  0.02
## 6   6     1.00  0.00
## 7   7     0.98  0.02
## 8   8     0.98  0.02
## 9   9     1.00  0.00
## 10 10     0.96  0.04
```

With a value of 6*or* 9 *for* K, *KNN has a test error rate of* 0%, *perfectly predicting the test data.*