Chapter 4: Applied Exercises

Phuong Dong Le

```
library(class)
library(MASS)
library(ISLR)
library(RColorBrewer)
library(corrplot)
library(boot)
### GGplot:
library(ggplot2)
library(ggthemes)
library(tidyverse)
### Styling for tables and figures:
library(kableExtra)
library(gridExtra)
```

Exercise 10: The question involves using the `Weekly` data set which is part of the `ISLR` package.

- This question involves the Weekly data set. The data is similar in natrue to the Smarket data except that it contains 1089 weekly returns for 21 years from the beginning of 1990 to the end of 2010.

Part (a)

```
attach(Weekly)
dim(Weekly)
```

```
## [1] 1089    9
```

```
names(Weekly)
```

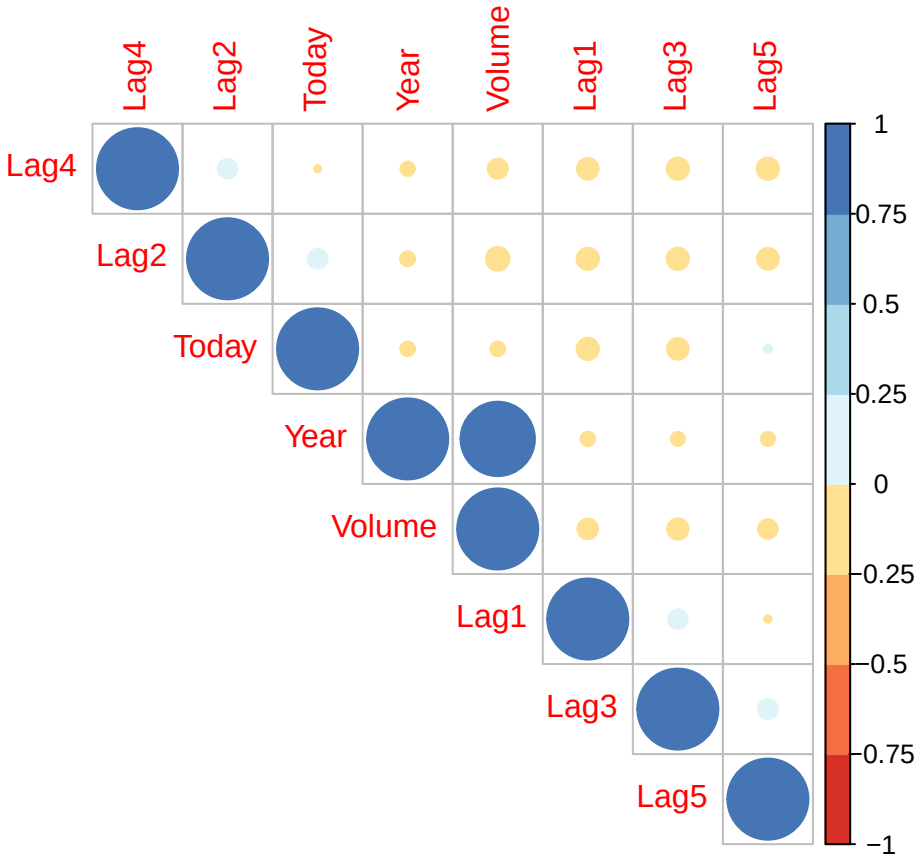
```
## [1] "Year"      "Lag1"      "Lag2"      "Lag3"      "Lag4"      "Lag5"
## [7] "Volume"    "Today"     "Direction"
```

```
summary(Weekly)
```

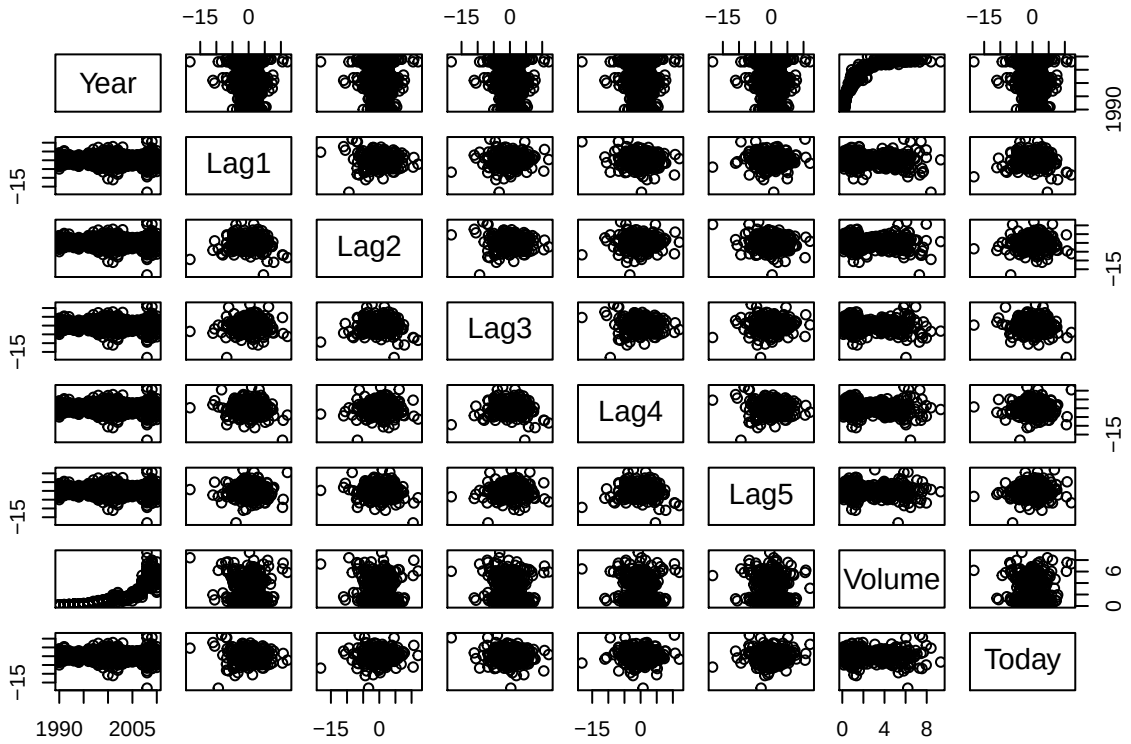
```
##       Year           Lag1               Lag2               Lag3
##  Min.   :1990   Min.   :-18.1950   Min.   :-18.1950   Min.   :-18.1950
##  1st Qu.:1995   1st Qu.: -1.1540   1st Qu.: -1.1540   1st Qu.: -1.1580
##  Median :2000   Median :  0.2410   Median :  0.2410   Median :  0.2410
##  Mean   :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
```

```
### Truncate:
X = Weekly[,-9]
M = cor(X)
corrplot(M, type  = "upper", order = "hclust",
         col = brewer.pal(n = 8, name = "RdYlBu"))
```




```
pairs(X)
```



Part (b)

- We perfom the logistic regression with Direction as the response and the five lag variables + Volume = predictors.

```
glm.fits = glm(formula = Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 + Volume,
               data = Weekly,
               family = "binomial")

summary(glm.fits)
```

```
##
## 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 coefficient of Lag2 appears to be statistically significant becaus the p values is the smallest. Another one could be possibly is Lag1 which is 0.1181.

Part (c)

- Compute the confusion matrix and overall fraction of correct predictions.

```
glm.probs = predict(glm.fits, type = "response")
## Create the vector of classifying predictions:
contrasts(Weekly$Direction) ## classify 1 for Up.
```

```
##      Up
## Down  0
## Up    1
```

```
## Create the vector of classifying:
glm.pred = rep("Down", 1089)
glm.pred[glm.probs > 0.50] = "Up"

## Compute the confusion matrix table:

table(glm.pred, Direction)
```

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

```
## Accuracry:
mean(glm.pred == Direction)
```

```
## [1] 0.5610652
```

- So our model has around 56% of accuracy overal. According to the confusion matrix, the logistic regression predicts 430 days for "Up" while in reality there are 557 days for "Up". For Decline days, it is also accurate as it predicts 54 days of declining while in reality our data has 48 of days declining.

Part (d)

- We fit the logistic regression model for a training data period from 1990 and 2008, with Lag2 as the only predictor. Compute the confusion matrix and the overall factor of correct predictions for the held out data (the data from 2009 and 2010).

```
## Split into training data set and testing:
train = (Year >= 1990 & Year <= 2008)
Weekly.Train = Weekly[train,]
Weekly.Test = Weekly[!train,]
dim(Weekly.Test)
```

```
## [1] 104   9
```

```
## Prediction data:
Direction.Pred = Direction[!train]




glm.fits = glm(Direction ~ Lag2, data = Weekly,
               subset = train,
               family= "binomial")


glm.probs = predict(object = glm.fits, Weekly.Test, type = "response")

glm.pred = rep("Down", 104)
glm.pred[glm.probs>0.5] = "Up"

table(glm.pred, Direction.Pred)
```

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

```
### Accuracy:
mean(glm.pred == Direction.Pred)
```

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

- The accuracy is overally 62% of accucracy. There are 34 days of increasing for prediction while in reality there are such 56 days. The model predicts 9 days of declining while in reality there are 5 of such day.

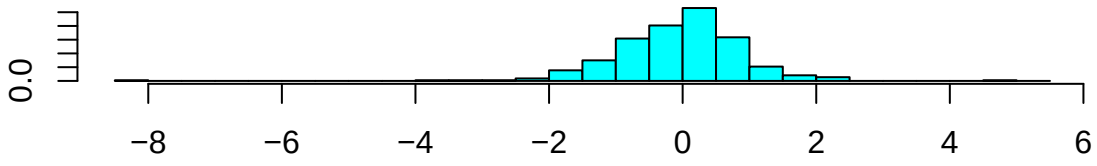
Part (e)

- Repeat part (d) using linear discriminant analysis:

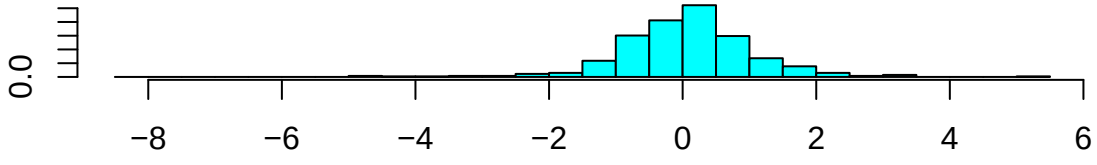
```
lda.fit = lda(Direction ~ Lag2, data = Weekly, subset = train)
lda.fit
```

```
## Call:
## lda(Direction ~ Lag2, data = Weekly, subset = 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
```

```
plot(lda.fit)
```



group Down



group Up

```
## Predict:

lda.pred = predict(object = lda.fit, Weekly.Test)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

```
lda.class = lda.pred$class
```

```
table(lda.class, Direction.Pred)
```

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

```
mean(lda.class == Direction.Pred)
```

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

- Similar result with logistic regression.

Part (f): Repeat using QDA

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

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

```
## Down -0.03568254
## Up    0.26036581
```

```
## Predict:
qda.pred = predict(object = qda.fit, Weekly.Test)
names(qda.pred)
```

```
## [1] "class"     "posterior"
```

```
qda.class = qda.pred$class
table(qda.class, Direction.Pred)
```

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

```
## Accuracy:
mean(qda.class == Direction.Pred)
```

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

- QDA does not give a good result although the accuracy rate is about 58%.

Part (g): Using KNN with K=1

```
train = (Year >= 1990 & Year <= 2008)
## Split into training and testing data set:
train.X = as.matrix(Lag2[train])
test.X = as.matrix(Lag2[!train])
### Training and testing for Y:
train.Y =as.matrix(Direction[train])
test.Y = as.matrix(Direction[!train])



set.seed(1)
### Method of K-Nearest Neighbors:
knn.fit = knn(train.X,test.X, train.Y, k = 1)
table(knn.fit, test.Y)
```

```
##        test.Y
## knn.fit Down Up
##    Down   21 30
##    Up     22 31
```

- In this case, we may conclude that the percentage of correct predictions on the test data is 50%. In other words 50% is the test error rate. We could also say that for weeks when the market goes up, the model is right 50.8196721% of the time. For weeks when the market goes down, the model is right only 48.8372093% of the time.

Exercise 11: develop a model to predict whether a given car gets high or low gass mileage based on the `Auto` data set.

Part (a)

- We create a binary variable `mpg01` containing 1 if `mpg` is a value above its median, and 0 if it is below the median.

```r
attach(Auto)


medvalue = median(Auto$mpg)
print(list(
  median_mpg = medvalue
))
```

```
## $median_mpg
## [1] 22.75
```

```r
mpg01 =rep(0, 392)
mpg01[mpg > medvalue] = 1
Auto = cbind(Auto, mpg01)
Auto = as.data.frame(Auto)
str(Auto)
```

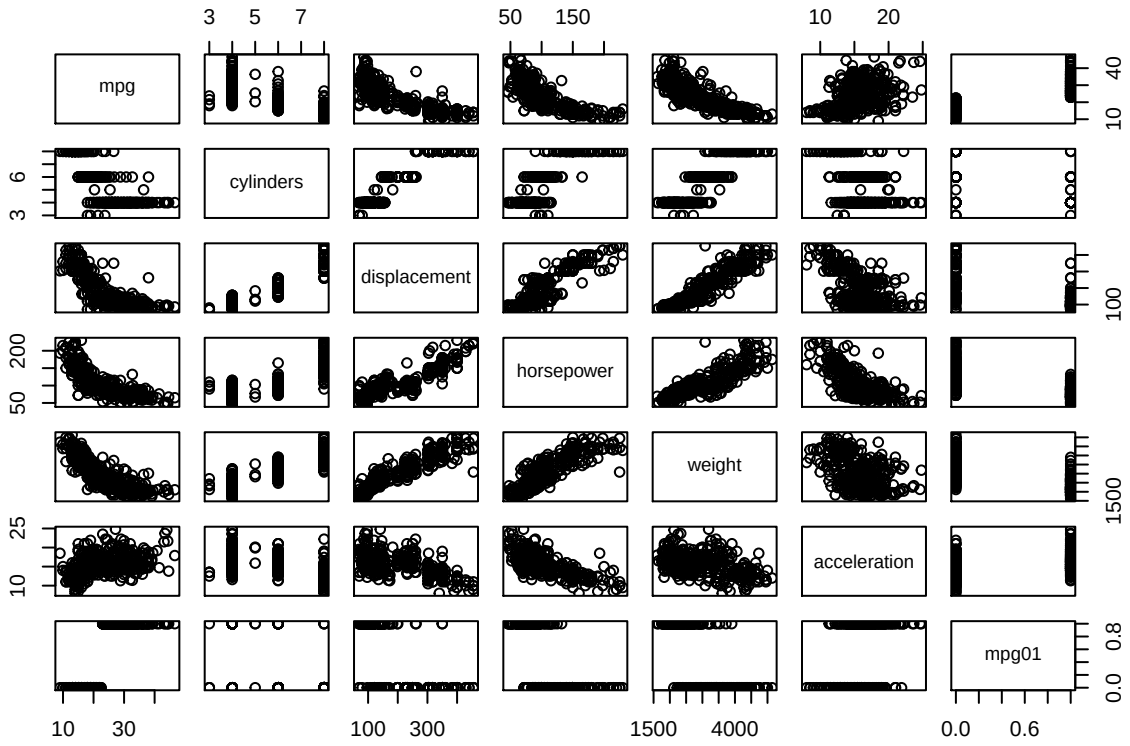
```
## 'data.frame':    392 obs. of  10 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : num  8 8 8 8 8 8 8 8 8 8 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : num  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ name        : Factor w/ 304 levels "amc ambassador brougham",..: 49 36 231 14 161 141 54 223 241 1
##  $ mpg01       : num  0 0 0 0 0 0 0 0 0 0 ...
```

Part (b)

```r
names(Auto)
```

```
##  [1] "mpg"          "cylinders"    "displacement" "horsepower"
##  [5] "weight"       "acceleration" "year"         "origin"
##  [9] "name"         "mpg01"
```

```r
pairs(Auto[,c(1,2,3,4,5,6,10)])
```



- The features seem most likely to predict `mpg01` are `horsepower`, `weight`, and `acceleration`.

Part (c)

- We split the data into training set and a test set:

```
attach(Auto)
### We split the training set into halves:
train = (year %% 2 == 0)


### Training and Testing Set:

Auto.train = Auto[train,]
Auto.test = Auto[!train,]

mpg01.test = mpg01[!train]
```

Part (d)

- We perform LDA on the training data in order to predict `mpg01` using the variables that seemed most associated with `mpg01`:

```
lda.fit = lda(mpg01 ~ weight + horsepower + acceleration, data = Auto, subset = train)
lda.fit
```

```
## Call:
## lda(mpg01 ~ weight + horsepower + acceleration, data = Auto,
##     subset = train)
##
## Prior probabilities of groups:
##         0         1
## 0.4571429 0.5428571
```

```
##
## Group means:
##     weight horsepower acceleration
## 0 3604.823  133.14583     14.47500
## 1 2314.763   77.92105     16.62895
##
## Coefficients of linear discriminants:
##                        LD1
## weight       -0.002039321
## horsepower    0.001069694
## acceleration  0.033964356
```

```
pred.lda <- predict(lda.fit, Auto.test)
lda.class = pred.lda$class

table(lda.class, mpg01.test)
```

```
##          mpg01.test
## lda.class  0  1
##         0 80  7
##         1 20 75
```

```
ErrorRate = mean( lda.class != mpg01.test   )

print(list(
  Test_Error_Rate = ErrorRate
))
```

```
## $Test_Error_Rate
## [1] 0.1483516
```

- Our test error rate is of 14.83%.

Part (e)

- We consider the Quadratic Discriminant Analysis on the training data set:

```
qda.fit = qda(mpg01 ~ horsepower + weight + acceleration,
              data = Auto,
              subset = train)

qda.fit
```

```
## Call:
## qda(mpg01 ~ horsepower + weight + acceleration, data = Auto,
##     subset = train)
##
## Prior probabilities of groups:
##         0         1
## 0.4571429 0.5428571
##
## Group means:
##   horsepower   weight acceleration
## 0  133.14583 3604.823     14.47500
## 1   77.92105 2314.763     16.62895
```

```
qda.pred = predict(qda.fit, Auto.test)
```

```
qda.class = qda.pred$class

table(qda.class, mpg01.test)
```

```
##          mpg01.test
## qda.class  0  1
##         0 82  9
##         1 18 73
```

```
ErrorRate.qda = mean(qda.class != mpg01.test)

print(list(
  Test_Error_Rate = ErrorRate.qda
))
```

```
## $Test_Error_Rate
## [1] 0.1483516
```

- The test error for quadratic discriminant analysis is still around 14.835.

Part (f)

- Now we consider the use of logistic regression:

```
glm.fit = glm(mpg01 ~ horsepower + weight + acceleration,
              data = Auto,
              subset = train,
              family = "binomial")
summary(glm.fit)
```

```
##
## Call:
## glm(formula = mpg01 ~ horsepower + weight + acceleration, family = "binomial",
##     data = Auto, subset = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.52771  -0.06568   0.07309   0.26084   2.19579
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)  18.021769   4.515914   3.991 6.59e-05 ***
## horsepower   -0.059303   0.032101  -1.847   0.0647 .
## weight       -0.004199   0.001057  -3.972 7.14e-05 ***
## acceleration -0.025583   0.195316  -0.131   0.8958
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 289.577  on 209  degrees of freedom
## Residual deviance:  90.278  on 206  degrees of freedom
## AIC: 98.278
##
## Number of Fisher Scoring iterations: 7
```

```
glm.probs = predict(glm.fit, Auto.test, type = "response")


glm.pred = rep(0, length(glm.probs))
glm.pred[glm.probs > 0.5] = 1

table(glm.pred, mpg01.test)
```

```
##         mpg01.test
## glm.pred  0  1
##        0 88 15
##        1 12 67
```

```
ErrorRate.log = mean(glm.pred != mpg01.test)

print(list(
  Test_Error_Rate = ErrorRate.log
))
```

```
## $Test_Error_Rate
## [1] 0.1483516
```

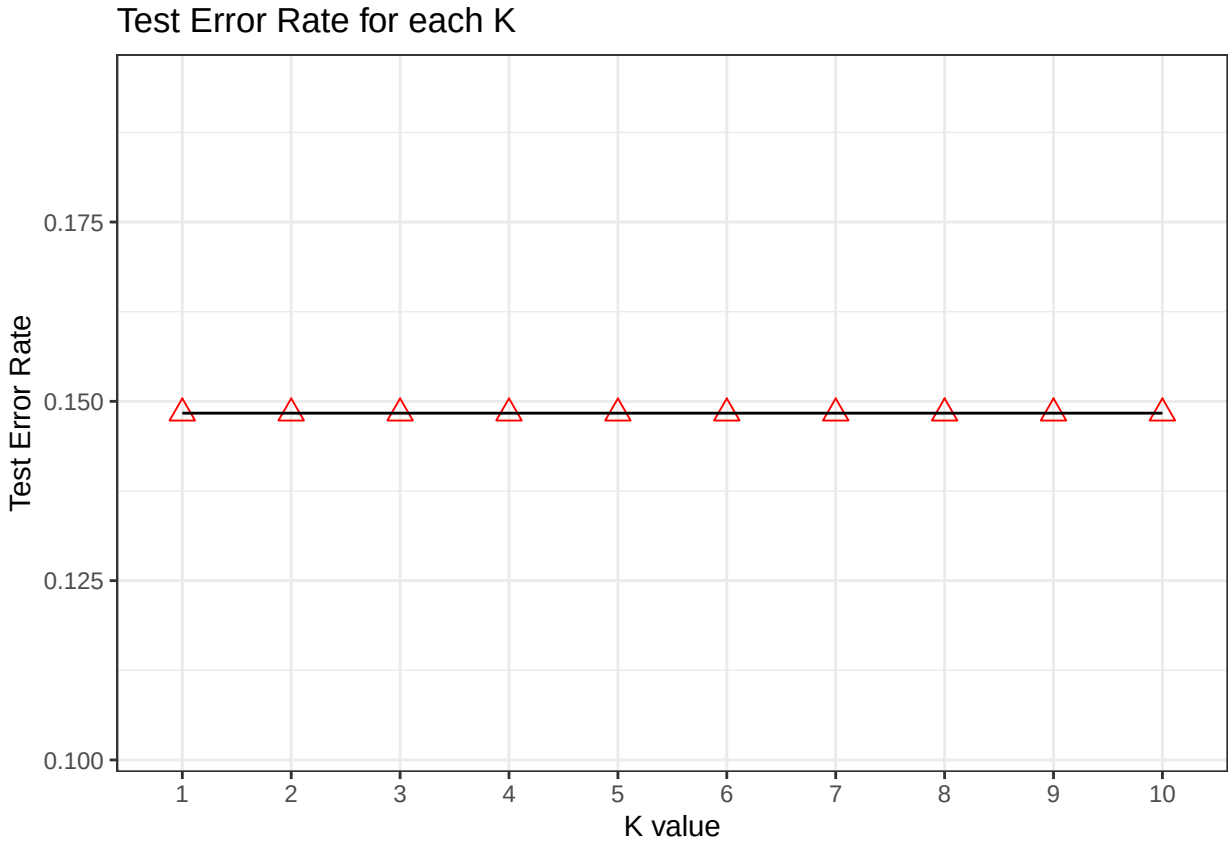
- The test error rate is about 14.83516%

Part (g)

- We now consider perfoming the K-Nearest Neighborhood Classification:

```
### Testing and Trainign set:
train.X = cbind(horsepower, weight, acceleration)[train,]
test.X = cbind(horsepower, weight, acceleration)[!train,]

train.Y = mpg01[train]
test.Y = mpg01[-train]


### We perform the knn for value k from 1 to 10.

ErrorSet = rep(NA)
Kvalue = 1:10

for (i in 1:10){
  knn.fit = knn(train = train.X,
                test = test.X,
                train.Y, k = i)
  ErrorSet[i] = mean(knn.fit != mpg01.test)
}


DataKNN = cbind(Kvalue, ErrorSet)
DataKNN = as.data.frame(DataKNN)


ErrorPlot = ggplot(data = DataKNN,
                   aes(x = factor(Kvalue),
                       y = ErrorRate)) +
```

```
  geom_point(pch  = 2, color = "red", size = 3) +
  geom_path(aes(
    x = Kvalue,
    y = ErrorRate
  )) +
  ggtitle(label = "Test Error Rate for each K") +
  xlab(label = "K value") +
  ylab(label = "Test Error Rate") +
  theme_bw()

print(ErrorPlot)
```




```
DataKNN.min = filter(DataKNN, c(ErrorSet == min(ErrorSet)))

print(DataKNN.min)
```

```
##   Kvalue  ErrorSet
## 1      8 0.1208791
```

- For the value of K from 1 to 10, the best values for K that give the minimum value of Test Error Rate is $K = 8$.

```
### K = 8:
knn.fit8 = knn(train = train.X,
               test = test.X,
               train.Y, k = 8)
Table.K.8 = table(knn.fit8, mpg01.test)
print(Table.K.8)
```

```
##         mpg01.test
## knn.fit8  0  1
##        0 85  9
##        1 15 73
```

Exercise 13: This question involves the `Boston` data set:

```
attach(Boston)
names(Boston)
```

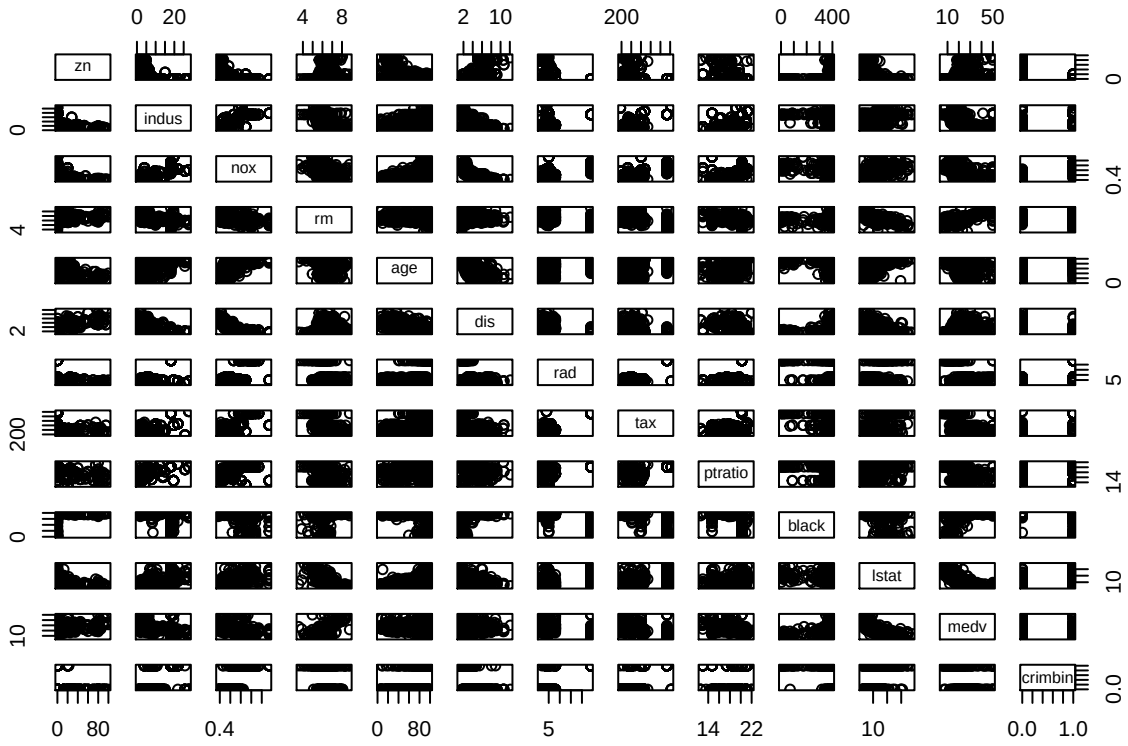
```
##  [1] "crim"    "zn"      "indus"   "chas"    "nox"     "rm"      "age"
##  [8] "dis"     "rad"     "tax"     "ptratio" "black"   "lstat"   "medv"
```

```
Boston = Boston[,1:14]

crimbin = rep(0, 506)
crimbin[crim > median(crim)] = 1

Boston = cbind(Boston, crimbin)
Boston = as.data.frame(Boston)
str(Boston)
```

```
## 'data.frame':    506 obs. of  15 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : num  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ black  : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
##  $ crimbin: num  0 0 0 0 0 0 0 0 0 0 ...
```

Part (a)

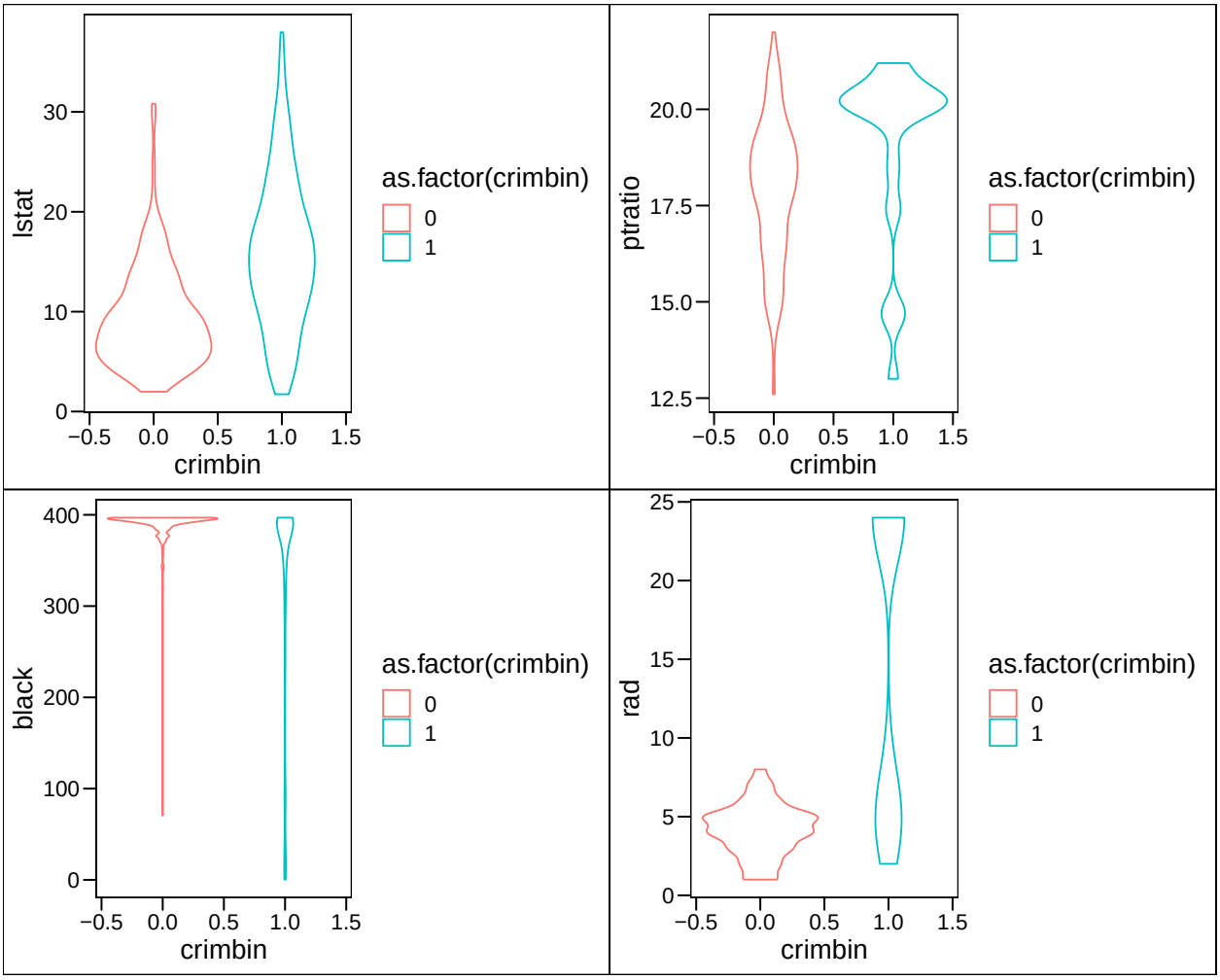
```
pairs(Boston[,c(2,3,5,6,7,8,9,10,11,12,13,14,15)])
```



- The variable features that seem likely to predict `crim01` are:
- `lstat`, `ptratio`, `black`, `rad`.

```
Plot1 = ggplot(data = Boston,
               aes(x = crimbin, y = lstat,
                   color = as.factor(crimbin))) +
  geom_violin() +
  theme_base()

Plot2 = ggplot(data = Boston,
               aes(x = crimbin, y = ptratio,
                   color = as.factor(crimbin))) +
  geom_violin() +
  theme_base()


Plot3 = ggplot(data = Boston,
               aes(x = crimbin, y = black,
                   color = as.factor(crimbin))) +
 geom_violin() +
  theme_base()


Plot4 = ggplot(data = Boston,
               aes(x = crimbin, y = rad,
                   color = as.factor(crimbin))) +
  geom_violin() +
  theme_base()
```

```
grid.arrange(Plot1,Plot2,Plot3, Plot4, ncol =2)
```



Part (c)

- We consider splitting the data into training and testing set:

```
set.seed(1)
train = sample(x = 506, size = 506*0.5)
Boston.train = Boston[train, ]
Boston.test = Boston[-train, ]
crimbin.test = crimbin[-train]
```

Part (d)

- We fit the logistic regression:

```
glm.fit = glm(formula =  crimbin ~ lstat + black + ptratio + rad,
              data = Boston,
              subset = train,
              family = "binomial")

summary(glm.fit)
```

```
##
## Call:
## glm(formula = crimbin ~ lstat + black + ptratio + rad, family = "binomial",
##     data = Boston, subset = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.34530  -0.69121  -0.24608   0.04392   2.01526
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.45340    2.82810  -0.160 0.872628
## lstat        0.10520    0.03113   3.380 0.000726 ***
## black       -0.01077    0.00601  -1.792 0.073166 .
## ptratio      0.03626    0.08643   0.419 0.674856
## rad          0.40479    0.10768   3.759 0.000171 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 350.73  on 252  degrees of freedom
## Residual deviance: 195.71  on 248  degrees of freedom
## AIC: 205.71
##
## Number of Fisher Scoring iterations: 8
```

```
glm.probs = predict(glm.fit, Boston.test, type = "response")

glm.pred = rep(0, length(glm.probs))
glm.pred[glm.probs > 0.25] = 1

table(glm.pred, crimbin.test)
```

```
##         crimbin.test
## glm.pred   0   1
##        0  79  18
##        1  47 109
```

```
print(
  mean(glm.pred != crimbin.test)
)
```

```
## [1] 0.256917
```

- The test error rate is 0.267.

Part (e)

- We consider using Linear Discriminant Analysis:

```
lda.fit = lda(crimbin ~ lstat + black + ptratio + rad,
              data = Boston, subset = train)
lda.fit
```

```
## Call:
## lda(crimbin ~ lstat + black + ptratio + rad, data = Boston, subset = train)
##
```

```
## Prior probabilities of groups:
##         0         1
## 0.5019763 0.4980237
##
## Group means:
##       lstat    black  ptratio       rad
## 0  9.814331 386.8047 17.96693  4.173228
## 1 16.192143 318.4443 19.38810 15.428571
##
## Coefficients of linear discriminants:
##                   LD1
## lstat    0.0534692448
## black   -0.0004037933
## ptratio  0.0325547925
## rad      0.1191840108
```

```
lda.pred = predict(lda.fit, Boston.test)

lda.class = lda.pred$class

table(lda.class, crimbin.test)
```

```
##          crimbin.test
## lda.class   0   1
##         0 124  60
##         1   2  67
```

```
print(
  mean(lda.class!= crimbin.test)
)
```

```
## [1] 0.2450593
```

- The test error rate for LDA is 24.5%

Part (f)

- We consider using quadratic discriminant analysis:

```
qda.fit = qda(
  crimbin ~ lstat + black + ptratio + rad,
              data = Boston, subset = train
)

qda.fit
```

```
## Call:
## qda(crimbin ~ lstat + black + ptratio + rad, data = Boston, subset = train)
##
## Prior probabilities of groups:
##         0         1
## 0.5019763 0.4980237
##
## Group means:
##       lstat    black  ptratio       rad
## 0  9.814331 386.8047 17.96693  4.173228
## 1 16.192143 318.4443 19.38810 15.428571
```

```
qda.pred = predict(qda.fit, Boston.test)

qda.class = qda.pred$class

table(qda.class, crimbin.test)
```

```
##          crimbin.test
## qda.class   0   1
##         0 124  50
##         1   2  77
```

```
print(
  mean(qda.class!= crimbin.test)
)
```

```
## [1] 0.2055336
```

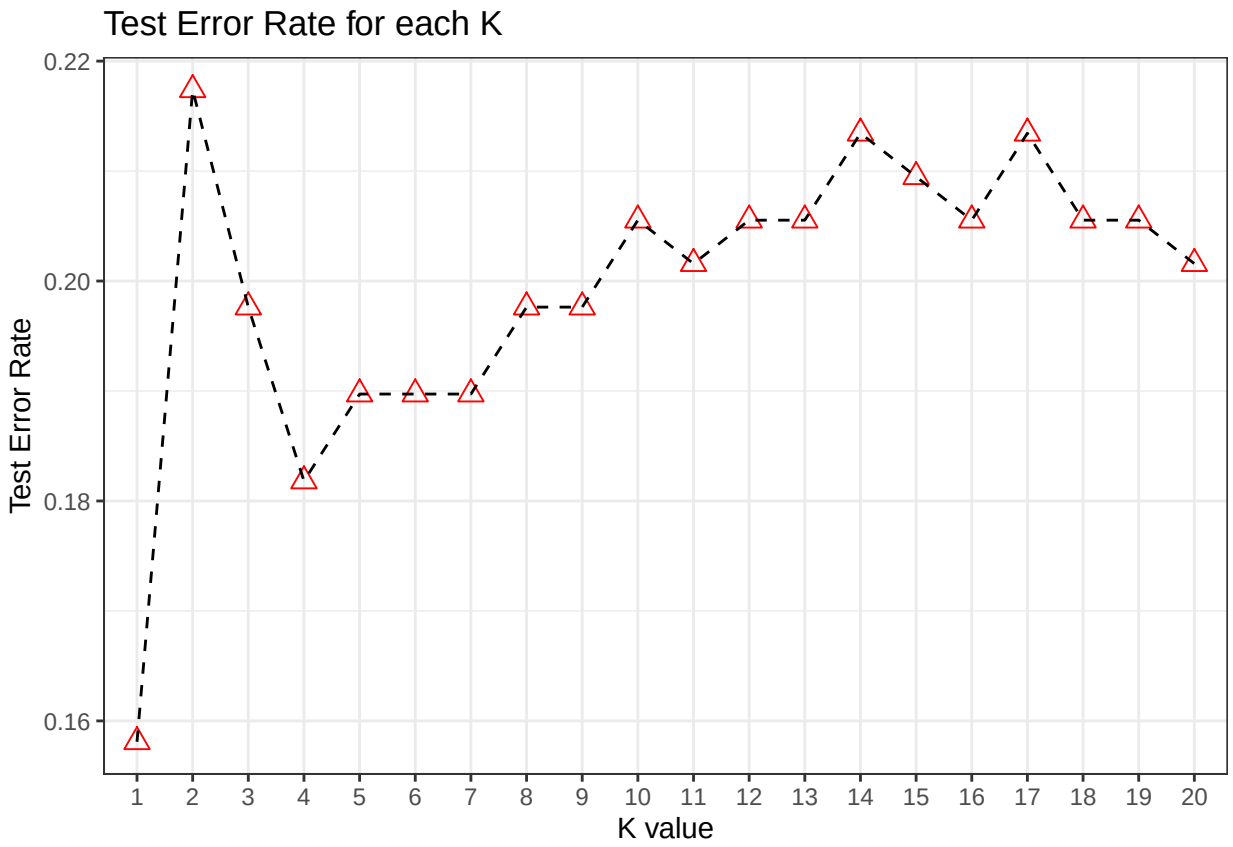
- The test error rate for QDA is 20.5%.

Part (g)

- We now consider using the K-Nearest Neighbors:

```
## Splitting into training and testing sets:
set.seed(1)
train = sample(x = 506, size = 506*0.5)
train.X = cbind(lstat, black, ptratio, rad)[train,]
test.X = cbind(lstat, black, ptratio, rad)[-train,]

train.Y = crimbin[train]
test.Y = crimbin[-train]

Kvalue = 1:20
ErrorRate = rep(NA)

for (i in Kvalue){
knn.fit = knn(train.X,  test.X,
              train.Y, k = i)
ErrorRate[i] = mean(knn.fit != test.Y)

}
DataKNN = cbind(ErrorRate, Kvalue)
DataKNN = as.data.frame(DataKNN)

ErrorPlot = ggplot(data = DataKNN,
                   aes(x = factor(Kvalue),
                       y = ErrorRate)) +
  geom_point(pch  = 2, color = "red", size = 3) +
  geom_path(aes(
    x = Kvalue,
    y = ErrorRate
  ), lty = 2) +
  ggtitle(label = "Test Error Rate for each K") +
  xlab(label = "K value") +
  ylab(label = "Test Error Rate") +
  theme_bw()
```

```
print(ErrorPlot)
```




```
DataKNNmin = filter(DataKNN, c(ErrorRate == min(ErrorRate)))
print(DataKNNmin)
```

```
##   ErrorRate Kvalue
## 1 0.1581028      1
```

- the best value $K = 1$ which has the lowest the test error rate which is 0.158.

```
knn.fit1 = knn(train.X,  test.X,
               train.Y, k = 1)
table(knn.fit1, test.Y)
```

```
##         test.Y
## knn.fit1   0   1
##        0 112  26
##        1  14 101
```