Homework 4

Naga Kartheek Peddisetty, 50538422

11/17/2023

Question 1) Adopted from ISLR2: Consider the Hitters data in the ISLR2 package. In this exercise, we want to predict Salary.

```
library(ISLR2)
data("Hitters")
head(Hitters)
```

```
##                   AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Andy Allanson      293   66     1   30  29    14     1    293    66      1
## -Alan Ashby         315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis        479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson       496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga   321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin    594  169     4   74  51    35    11   4408  1133     19
##                   CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Andy Allanson       30   29     14      A        E     446      33     20
## -Alan Ashby         321  414    375      N        W     632      43     10
## -Alvin Davis        224  266    263      A        W     880      82     14
## -Andre Dawson       828  838    354      N        E     200      11      3
## -Andres Galarraga    48   46     33      N        E     805      40      4
## -Alfredo Griffin    501  336    194      A        W     282     421     25
##                   Salary NewLeague
## -Andy Allanson        NA         A
## -Alan Ashby        475.0         N
## -Alvin Davis       480.0         A
## -Andre Dawson      500.0         N
## -Andres Galarraga   91.5         N
## -Alfredo Griffin   750.0         A
```

```
dim(Hitters)
```

```
## [1] 322  20
```

```
sum(is.na(Hitters$Salary))
```

```
## [1] 59
```

```
Hitter <- na.omit(Hitters)
Hitter$Salary <- log(Hitter$Salary)
dim(Hitter)
```

```
## [1] 263  20
```

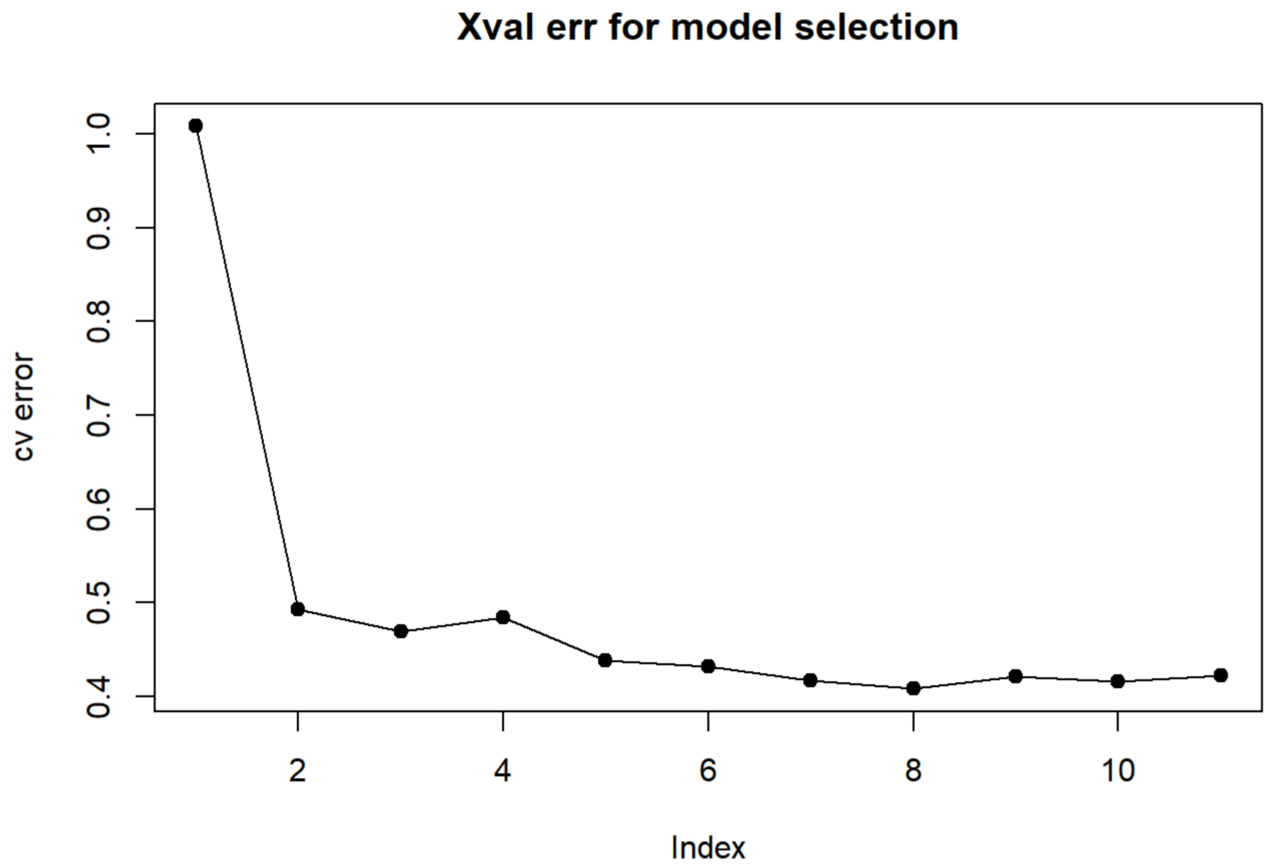
```
head(Hitter)
```

```
##                    AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun
## -Alan Ashby          315   81     7   24  38    39    14   3449   835     69
## -Alvin Davis         479  130    18   66  72    76     3   1624   457     63
## -Andre Dawson        496  141    20   65  78    37    11   5628  1575    225
## -Andres Galarraga    321   87    10   39  42    30     2    396   101     12
## -Alfredo Griffin     594  169     4   74  51    35    11   4408  1133     19
## -Al Newman           185   37     1   23   8    21     2    214    42      1
##                    CRuns CRBI CWalks League Division PutOuts Assists Errors
## -Alan Ashby          321  414    375      N        W     632      43     10
## -Alvin Davis         224  266    263      A        W     880      82     14
## -Andre Dawson        828  838    354      N        E     200      11      3
## -Andres Galarraga     48   46     33      N        E     805      40      4
## -Alfredo Griffin     501  336    194      A        W     282     421     25
## -Al Newman            30    9     24      N        E      76     127      7
##                      Salary NewLeague
## -Alan Ashby        6.163315         N
## -Alvin Davis       6.173786         A
## -Andre Dawson      6.214608         N
## -Andres Galarraga  4.516339         N
## -Alfredo Griffin   6.620073         A
## -Al Newman         4.248495         A
```

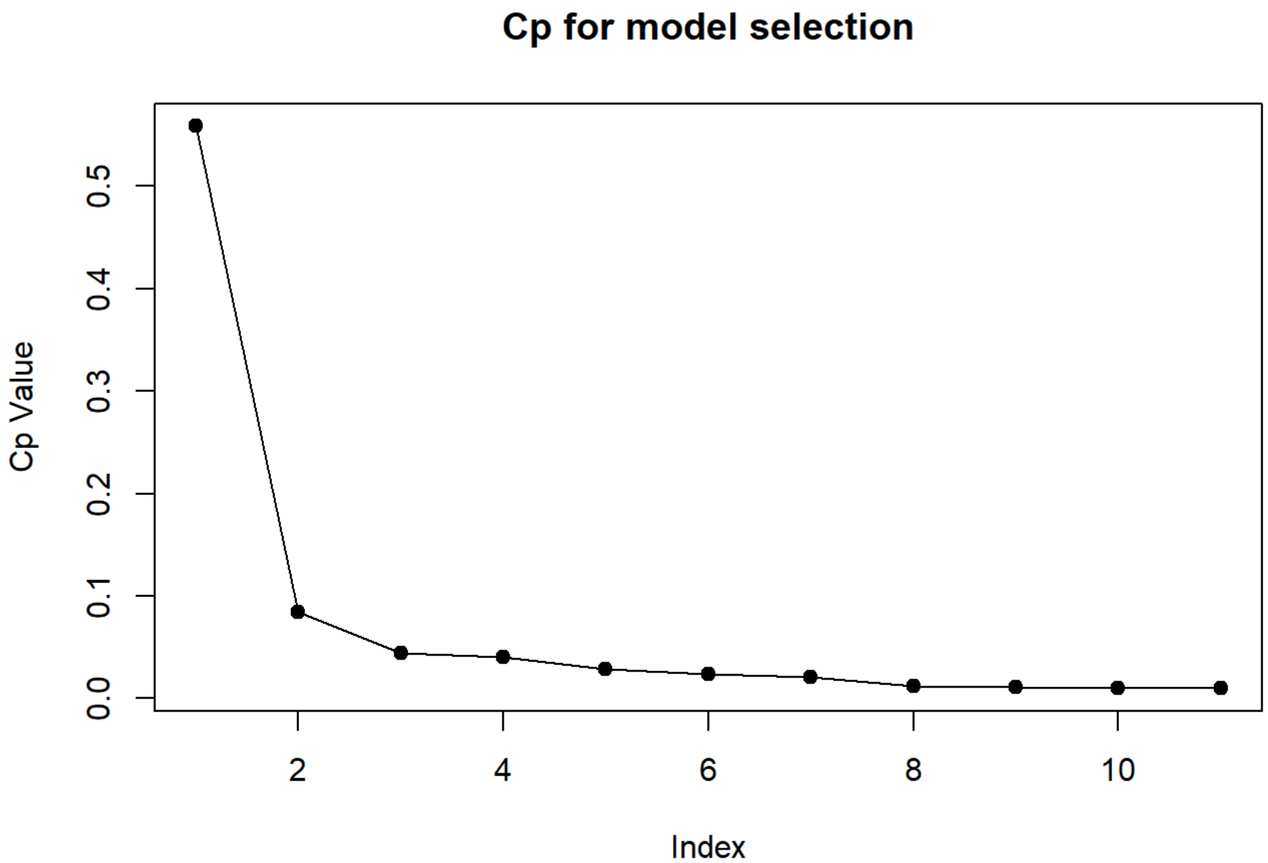
```
### Creating a training and test data

set.seed(123)

indis <- sample(1:nrow(Hitter),size = round(0.7 * nrow(Hitter)))

train_data <- Hitter[indis, ]
test_data <- Hitter[-indis, ]

X_train <- train_data[, -19]
Y_train <- train_data[, 19]

X_test <- test_data[, -19]
Y_test <- test_data[,19]
```

a) Apply CART to this dataset. Show the trees before and after pruning and interpret the results, report the error rate.

```
library(rpart)
library(rpart.plot)

model.controls <- rpart.control(minbucket = 2, minsplit = 5, xval = 10, cp = 0.01)
fit.Hitters <- rpart(Salary~., data = train_data, control = model.controls)

plot(fit.Hitters$cptable[,4], main = "Xval err for model selection", ylab = "cv error",pch=1
9,type='o')
```




```
plot(fit.Hitters$cptable[,1], main = "Cp for model selection", ylab = "Cp Value",pch=19,type
='o')
```



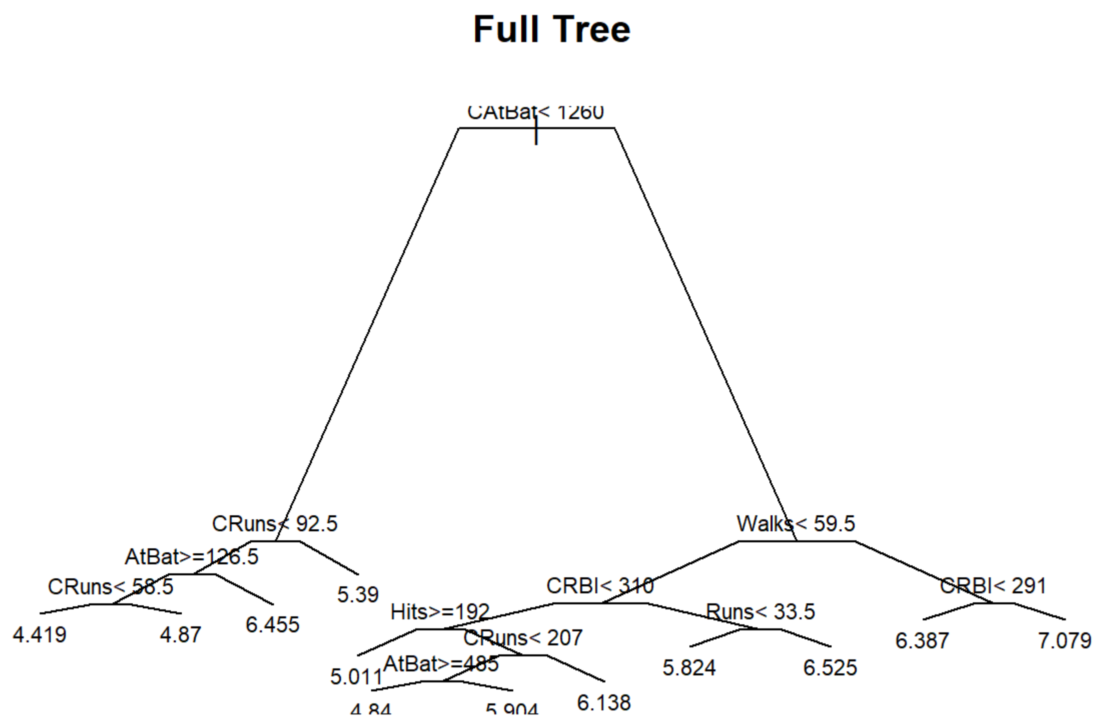
```
min_cp <- which.min(fit.Hitters$cptable[,4])
min_cp
```

```
## 8
## 8
```

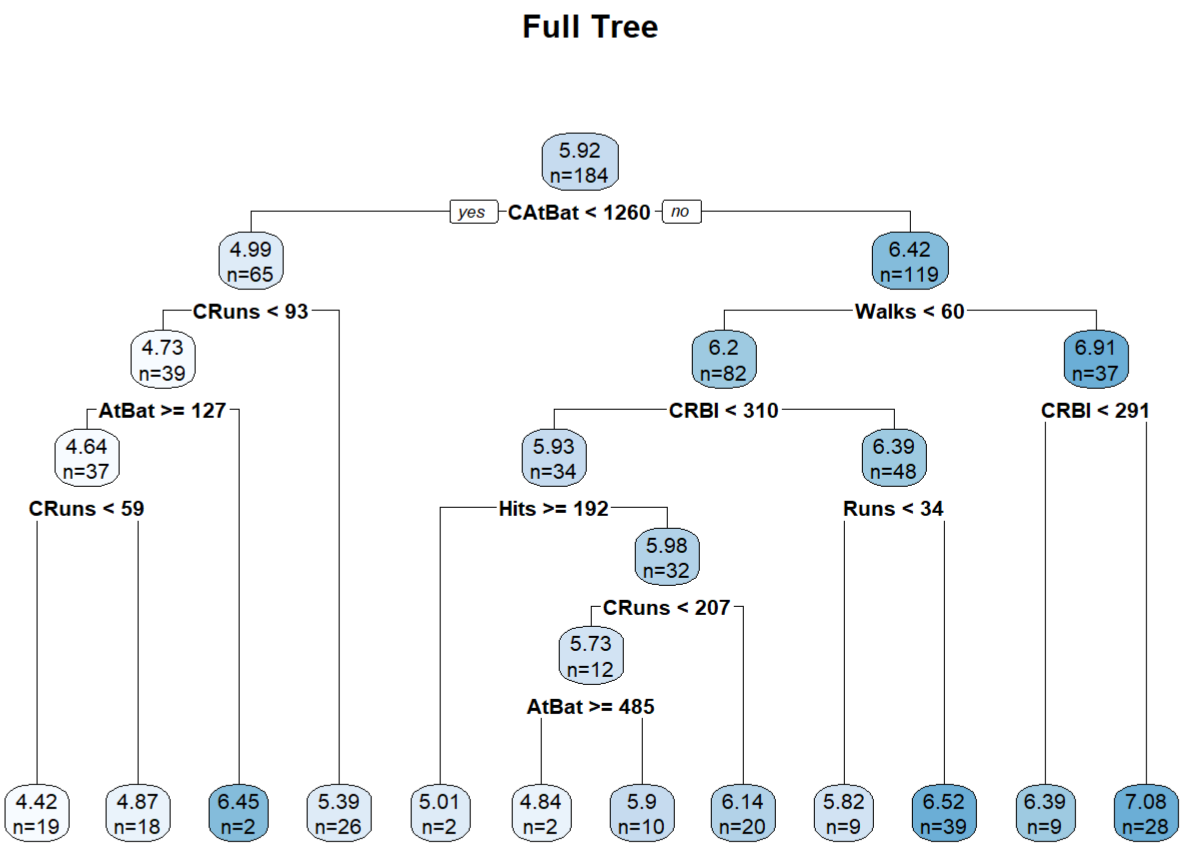
```
fit.Hitters$variable.importance
```

```
##       CRuns      CAtBat       CHits      CWalks        CRBI       Years
## 101.9840073  99.2254735  98.1706913  86.2907783  84.8702509  55.7585737
##       Walks        Runs       AtBat        Hits      CHmRun         RBI
##  21.1522582  19.4383245  13.3180067  11.5248882   7.2160866   5.1243238
##       HmRun     Assists      Errors     PutOuts
##   2.7888113   1.5948515   0.9433441   0.3171719
```

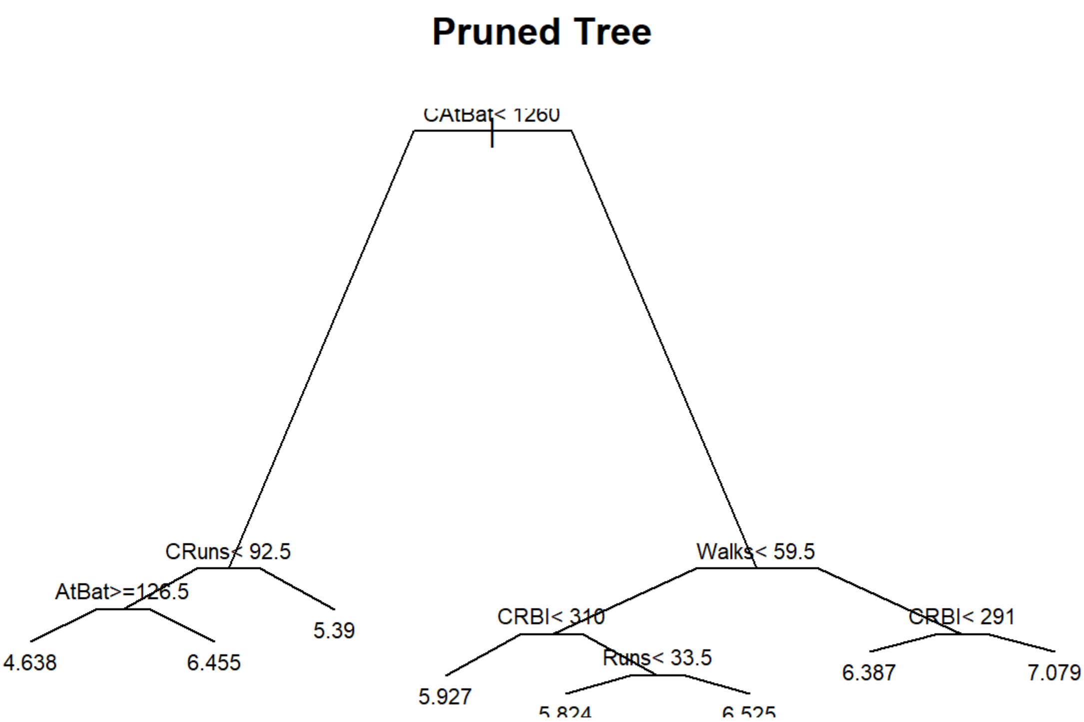
```
pruned.fit.Hitters <- prune(fit.Hitters, cp = fit.Hitters$cptable[min_cp, 1])


plot(fit.Hitters, branch = .3, compress = T, main = "Full Tree")
text(fit.Hitters, cex = 0.7)
```



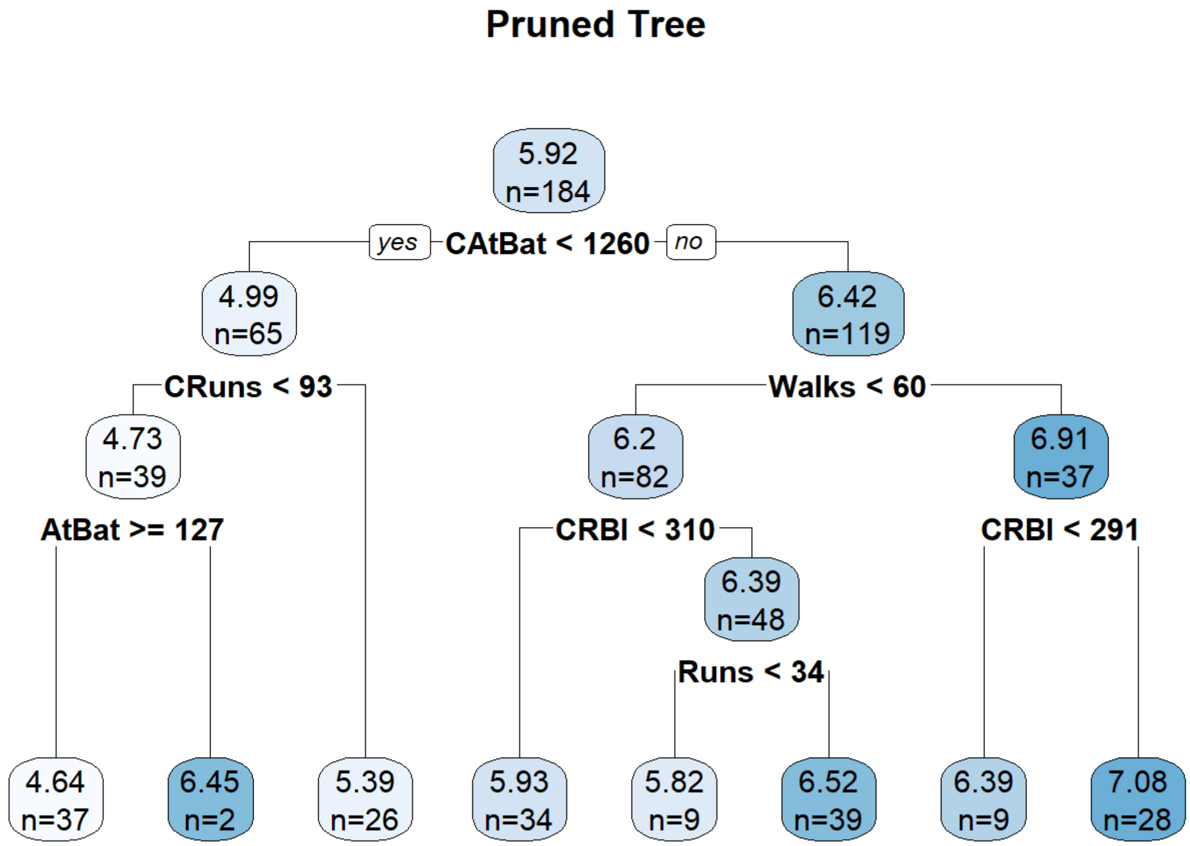

```
rpart.plot(fit.Hitters,digits = 3, extra = 1,main = "Full Tree")
```




```
plot(pruned.fit.Hitters, branch = .3, compress = T, main = "Pruned Tree")
text(pruned.fit.Hitters, cex = 0.7)
```



```
rpart.plot(pruned.fit.Hitters,digits = 3, extra = 1,main = "Pruned Tree")
```




```
pruned.fit.Hitters$variable.importance
```

```
##     CRuns    CAtBat     CHits    CWalks      CRBI     Years     Walks      Runs
## 98.841335 96.922138 95.867356 84.514449 83.829149 55.229954 21.152258 15.775081
##     AtBat      Hits    CHmRun       RBI     HmRun   Assists
##  9.648412  7.855293  7.216087  3.237636  2.788811  1.594851
```

```
# make a prediction
pred_train <- predict(pruned.fit.Hitters, newdata = train_data)
train_mse <- mean((pred_train - Y_train)^2)
train_mse
```

```
## [1] 0.1656272
```

```
pred_test <- predict(pruned.fit.Hitters, newdata = test_data)
test_mse <- mean((pred_test - Y_test)^2)
test_mse
```

```
## [1] 0.2170922
```

b) Apply bagging to this dataset and report the error rate.

```r
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.3.2
```

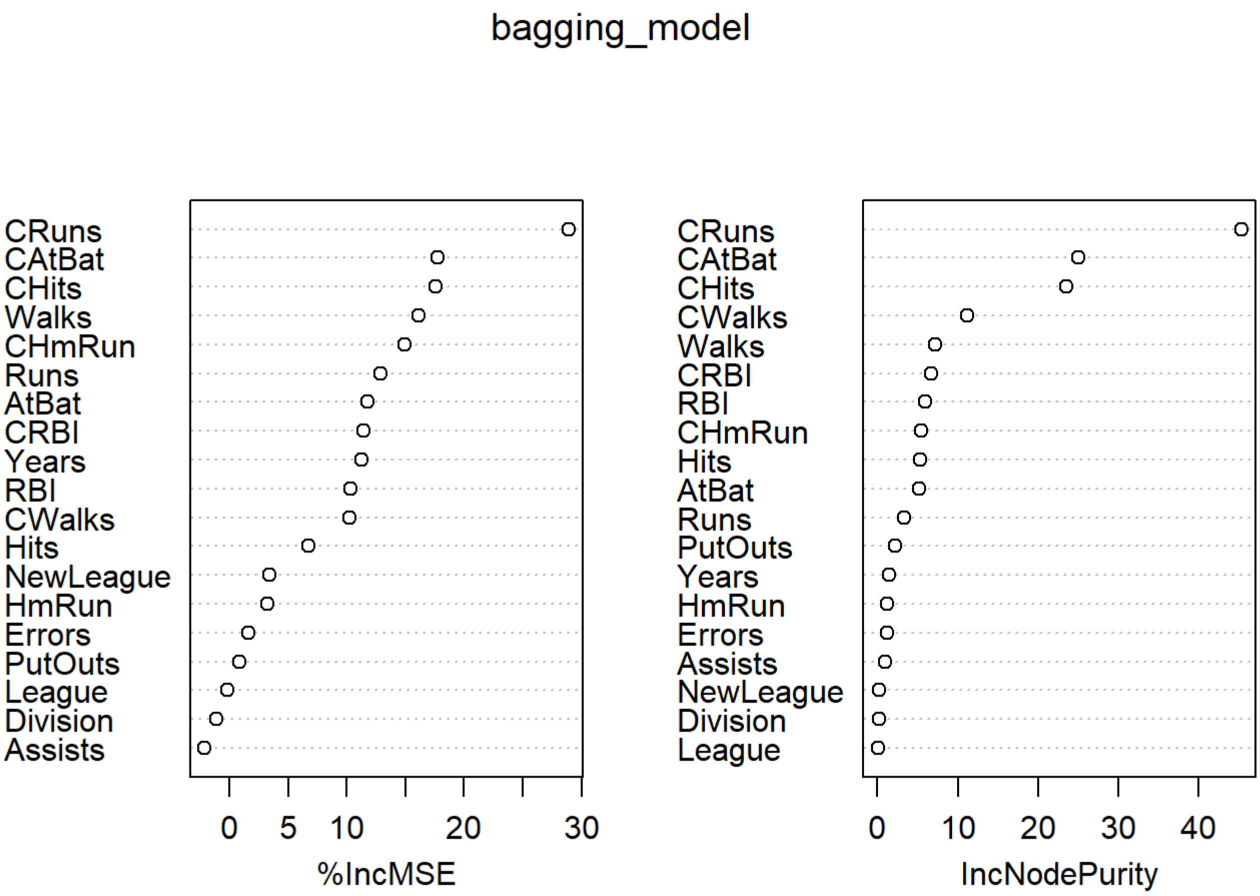
```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
set.seed(123)

bagging_model <- randomForest(Salary ~ ., data = train_data, mtry = 19, ntree=1000, importanc
e = TRUE)

pred_bagging <- predict(bagging_model, newdata = test_data, type='response')
bagging_mse <- mean((pred_bagging - Y_test)^2)
bagging_mse
```

```
## [1] 0.1968603
```

```r
varImpPlot(bagging_model)
```




```r
importance(bagging_model)
```

```
##              %IncMSE IncNodePurity
## AtBat     11.8069008     5.1899710
## Hits       6.7422374     5.3510989
## HmRun      3.2679218     1.2025907
## Runs      12.9179507     3.2609587
## RBI       10.2882152     5.8910500
## Walks     16.0958740     7.1298268
## Years     11.2435504     1.4525297
## CAtBat    17.7696602    24.9746561
## CHits     17.6045840    23.5678221
## CHmRun    14.9359593     5.4574027
## CRuns     28.8886159    45.2944879
## CRBI      11.4563024     6.6632223
## CWalks    10.2465680    11.2062530
## League    -0.1809110     0.1074265
## Division  -1.1220714     0.1493949
## PutOuts    0.8962148     2.2002255
## Assists   -2.0927097     0.9683162
## Errors     1.6470294     1.1533738
## NewLeague  3.3881294     0.2474779
```

c) Create a plot displaying the test error resulting from random forests on this data set for a more comprehensive range of values for mtry and ntree. You can model your plot after Figure 8.10. Describe the results obtained.

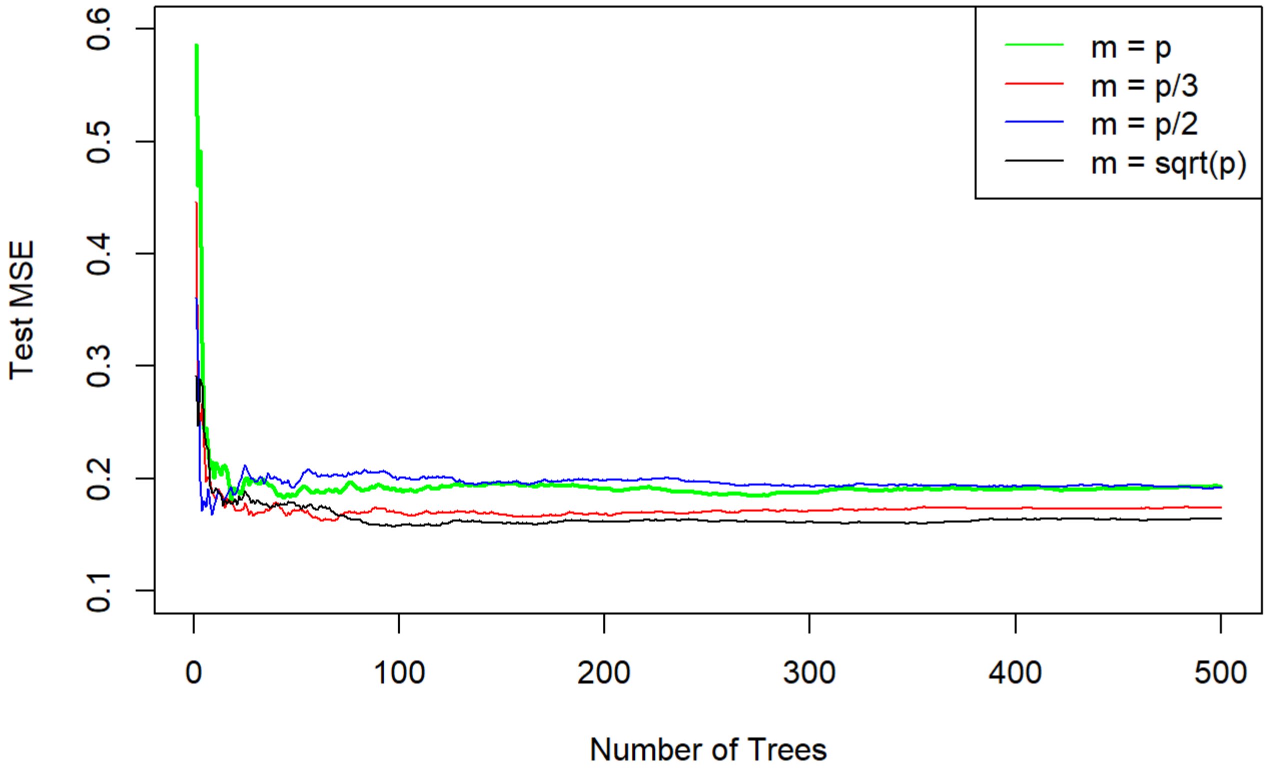
```
set.seed(123)

rf.Hitter1 <- randomForest(x = X_train , y = Y_train, xtest = X_test, ytest = Y_test , mtry = ncol(Hitter) - 1, ntree = 500)
rf.Hitter2 <- randomForest(x = X_train , y = Y_train, xtest = X_test, ytest = Y_test , mtry = (ncol(Hitter) - 1) / 3, ntree = 500)
rf.Hitter3 <- randomForest(x = X_train , y = Y_train, xtest = X_test, ytest = Y_test , mtry = (ncol(Hitter) - 1) / 2, ntree = 500)
rf.Hitter4 <- randomForest(x = X_train , y = Y_train, xtest = X_test, ytest = Y_test , mtry = sqrt(ncol(Hitter) - 1), ntree = 500)

names(rf.Hitter1)
```

```
##  [1] "call"            "type"            "predicted"       "mse"
##  [5] "rsq"             "oob.times"       "importance"      "importanceSD"
##  [9] "localImportance" "proximity"       "ntree"           "mtry"
## [13] "forest"          "coefs"           "y"               "test"
## [17] "inbag"
```

```
names(rf.Hitter1$test)
```

```
## [1] "predicted" "mse"       "rsq"       "proximity"
```

```
plot(1:500, rf.Hitter1$test$mse, col = "green", type = "l", xlab = "Number of Trees", ylab =
"Test MSE",ylim = c(0.1, 0.6),lwd=2)
lines(1:500, rf.Hitter2$test$mse, col = "red", type = "l")
lines(1:500, rf.Hitter3$test$mse, col = "blue", type = "l")
lines(1:500, rf.Hitter4$test$mse, col = "black", type = "l")
legend("topright", c("m = p", "m = p/3","m = p/2", "m = sqrt(p)"), col = c("green", "red", "b
lue","black"), cex = 1, lty = 1)
```

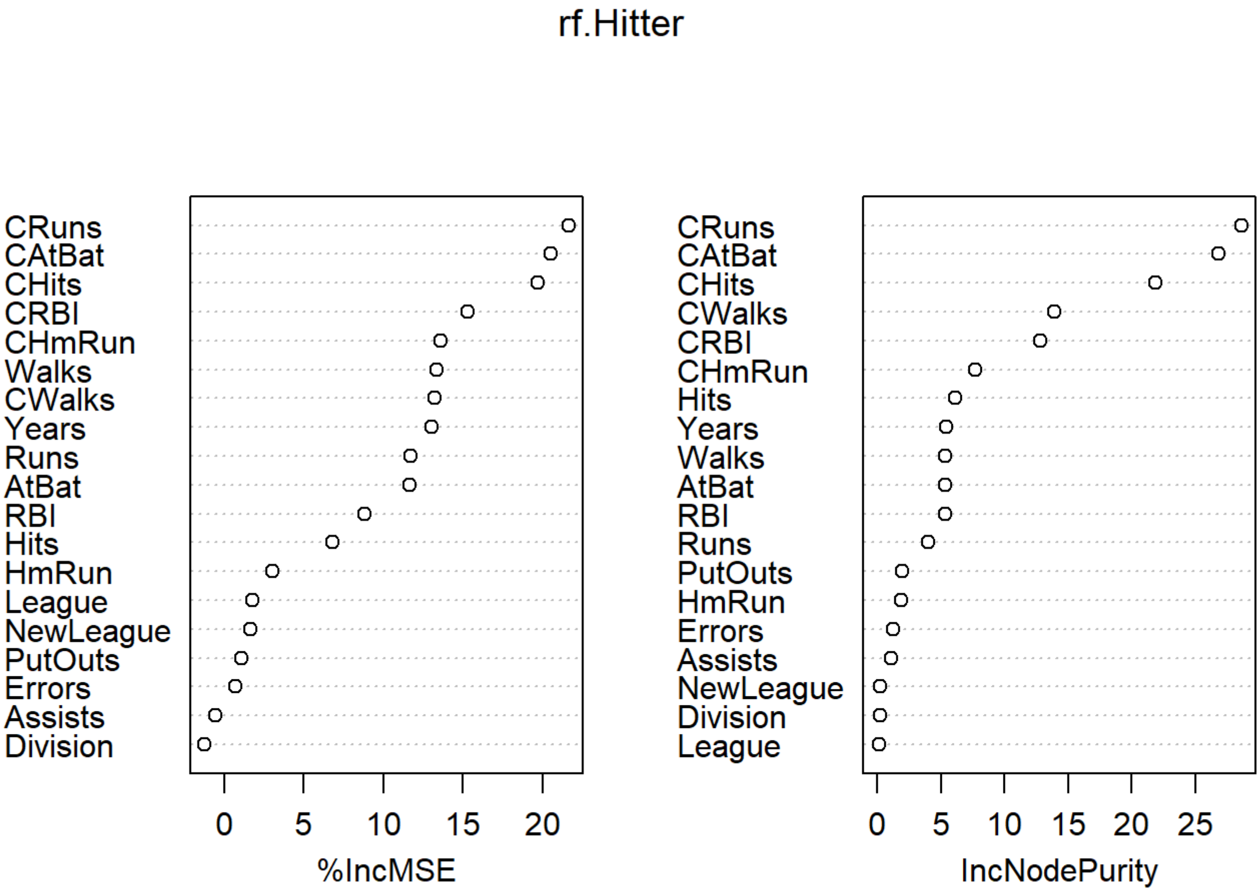


The test MSE for a single tree is very high, as the number of trees increases the test MSE decreases.

Also the test MSE for m = p is slightly higher compared to m=p/3, m=p/2 and m = sqrt(p).

By default, randomForest() uses p/3 variables when building a random forest of regression trees, and √p variables when building a random forest of classification trees. Here we can use mtry = 6.

```
set.seed (123)

rf.Hitter <- randomForest(Salary ~ ., data = train_data , mtry = 6, ntree = 1000, importance
= TRUE)

varImpPlot(rf.Hitter)
```




```
importance(rf.Hitter)
```

```
##              %IncMSE IncNodePurity
## AtBat     11.6615862     5.2973305
## Hits       6.8071536     6.1447558
## HmRun      3.0050653     1.8583437
## Runs      11.7039652     4.0173762
## RBI        8.8098141     5.2855021
## Walks     13.3510926     5.3233732
## Years     13.0490245     5.3983070
## CAtBat    20.5065129    26.7826115
## CHits     19.6787875    21.8778756
## CHmRun    13.6090957     7.7176299
## CRuns     21.6073293    28.5899146
## CRBI      15.2809455    12.8065136
## CWalks    13.2149229    13.9278007
## League     1.7551600     0.1624662
## Division  -1.2258269     0.1815606
## PutOuts    1.0919156     1.9426675
## Assists   -0.5817058     1.0545429
## Errors     0.7060671     1.1945702
## NewLeague  1.6278445     0.2296988
```

```
rf_pred <- predict(rf.Hitter, newdata = test_data)
rf_mse <- mean ((rf_pred - Y_test)^2)
rf_mse
```

```
## [1] 0.1797395
```

d) Apply boosting to this data using different shrinkage parameters. Tune the model and report the test error.

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 4.3.2
```

```
## Loaded gbm 2.1.8.1
```

```
set.seed(123)

shrink <- c(0.001,0.01,0.1, 0.4, 0.6, 0.8)
store_error <- numeric(length(shrink))
for (i in 1:length(shrink)){
    boost.fit <- gbm(Salary ~., data = train_data, n.trees = 1000, shrinkage = shrink[i], int
eraction.depth = 3, distribution = "gaussian")
    y_hat <- predict(boost.fit, newdata = test_data, n.trees = 1000)
    store_error[i] <- mean((y_hat-Y_test)^2)
}

print(data.frame(Shrinkage = shrink, Error_Rate = store_error))
```

```
##   Shrinkage Error_Rate
## 1     0.001  0.2727657
## 2     0.010  0.2078916
## 3     0.100  0.2434072
## 4     0.400  0.3752155
## 5     0.600  0.2659010
## 6     0.800  0.5635252
```

```
boost_mse <- store_error[which.min(store_error)]
boost_mse
```

```
## [1] 0.2078916
```

By using λ = 0.01 we got the lower test MSE = 0.2078916

e) Compare and contrast your results from A-D.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:randomForest':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
data <- data.frame(Methods = c("Trees","Bagging","Random forest","Boosting"), Test_MSE_percen
tage = c(round(test_mse * 100 , 2), round(bagging_mse * 100 , 2),round(rf_mse * 100 , 2), rou
nd(boost_mse * 100 , 2)))

arrange(data,Test_MSE_percentage)
```

```
##         Methods Test_MSE_percentage
## 1 Random forest               17.97
## 2       Bagging               19.69
## 3      Boosting               20.79
## 4         Trees               21.71
```

On our test data predictions Random forest performs well with low test MSE, Bagging and boosting also performs well but slightly low compared to Random forests.

And the tree model (Trees) has a higher Test_MSE, indicating that ensemble methods like Random Forest or Bagging might be the best choice for improving predictive performance.

Question 2) Access the wine data from the UCI machine learning repository (https://archive.ics.uci.edu/ml/datasets/wine (https://archive.ics.uci.edu/ml/datasets/wine)). These data are the results of a chemical analysis of 178 wines grown over the decade 1970-1979 in the same region of Italy, but derived from three different cultivars (Barolo, Grignolino, Barbera). The Babera wines were predominately from a period that was much later than that of the Barolo and Grignolino wines. The analysis determined the quantities MalicAcid, Ash, AlcAsh, Mg, Phenols, Proa, Color, Hue, OD, and Proline. There are 58 Barolo wines, 71 Grignolino wines, and 48 Barbera wines. Construct the appropriate-size classification tree for this dataset. Apply an ensemble technique (e.g., random forests or boosting). Compare the performance.

```
library(rpart)
library(rpart.plot)

setwd('D:/Buffalo/files/wine')

set.seed(123)
X <- read.csv('wine.data')
dim(X)
```

```
## [1] 177  14
```

```
unique(X$X1)
```

```
## [1] 1 2 3
```

```
colnames(X) <- c('wines','Alcohol','Malicacid','Ash','Alcalinity_of_ash','Magnesium','Total_p
henols','Flavanoids','Nonflavanoid_phenols','Proanthocyanins','Color_intensity','Hue','dilute
d_wines','Proline')
head(X)
```

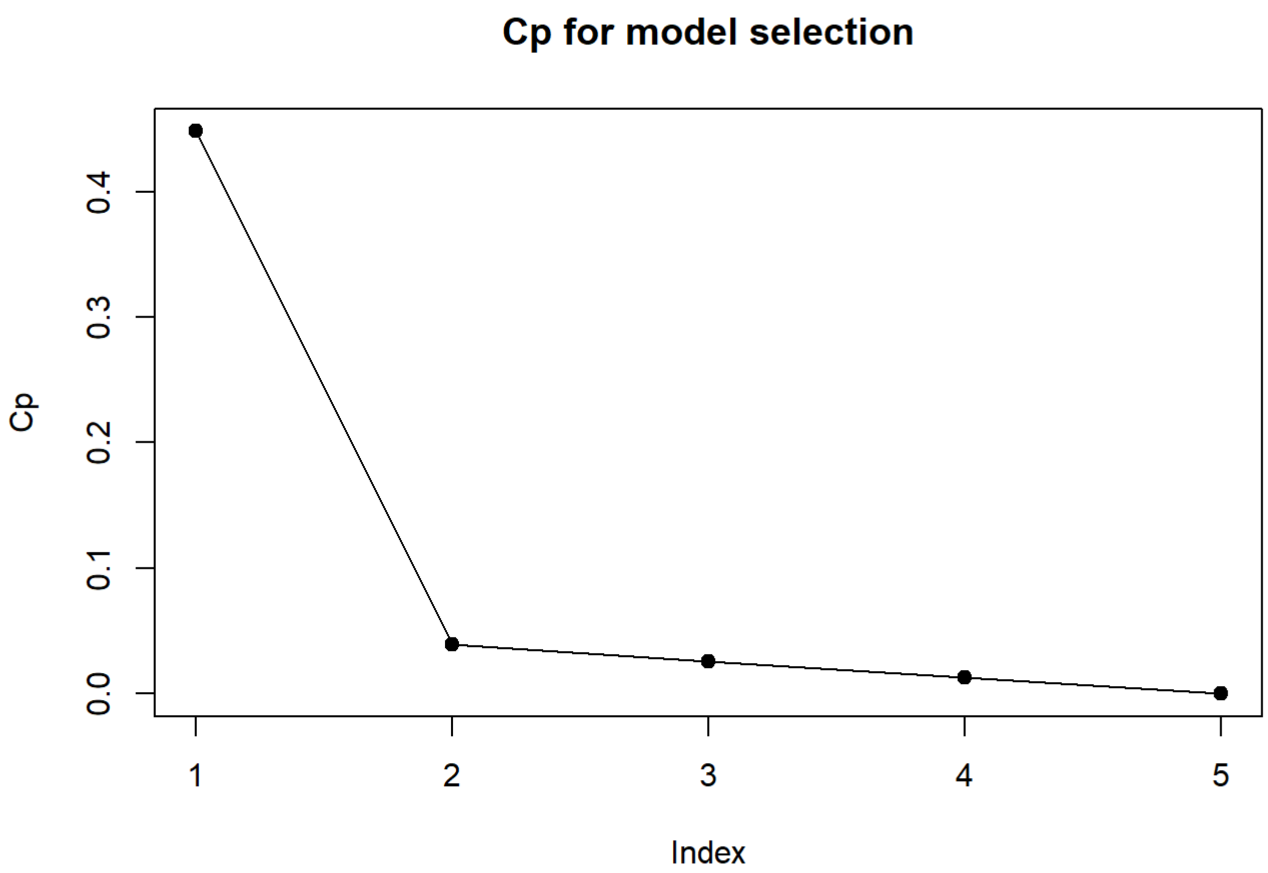
```
##   wines Alcohol Malicacid  Ash Alcalinity_of_ash Magnesium Total_phenols
## 1     1   13.20      1.78 2.14              11.2       100          2.65
## 2     1   13.16      2.36 2.67              18.6       101          2.80
## 3     1   14.37      1.95 2.50              16.8       113          3.85
## 4     1   13.24      2.59 2.87              21.0       118          2.80
## 5     1   14.20      1.76 2.45              15.2       112          3.27
## 6     1   14.39      1.87 2.45              14.6        96          2.50
##   Flavanoids Nonflavanoid_phenols Proanthocyanins Color_intensity  Hue
## 1       2.76                 0.26            1.28            4.38 1.05
## 2       3.24                 0.30            2.81            5.68 1.03
## 3       3.49                 0.24            2.18            7.80 0.86
## 4       2.69                 0.39            1.82            4.32 1.04
## 5       3.39                 0.34            1.97            6.75 1.05
## 6       2.52                 0.30            1.98            5.25 1.02
##   diluted_wines Proline
## 1          3.40    1050
## 2          3.17    1185
## 3          3.45    1480
## 4          2.93     735
## 5          2.85    1450
## 6          3.58    1290
```

```
indis <- sample(1:nrow(X),size = round(0.7*nrow(X)))
train_data <- X[indis, ]
test_data <- X[-indis, ]

X_train <- train_data[, -1]
Y_train <- train_data[, 1]

X_test <- test_data[, -1]
Y_test <- test_data[,1]

model.control <- rpart.control(minsplit = 3, xval = 10, cp = 0)
fit.X <- rpart(wines~., data = train_data, method = "class", control = model.control)

names(fit.X)
```

```
##  [1] "frame"               "where"               "call"
##  [4] "terms"               "cptable"             "method"
##  [7] "parms"               "control"             "functions"
## [10] "numresp"             "splits"              "variable.importance"
## [13] "y"                   "ordered"
```

```
plot(fit.X$cptable[,4], main = "Xval error for model selection", ylab = "xerr",pch=19,type
='o')
```

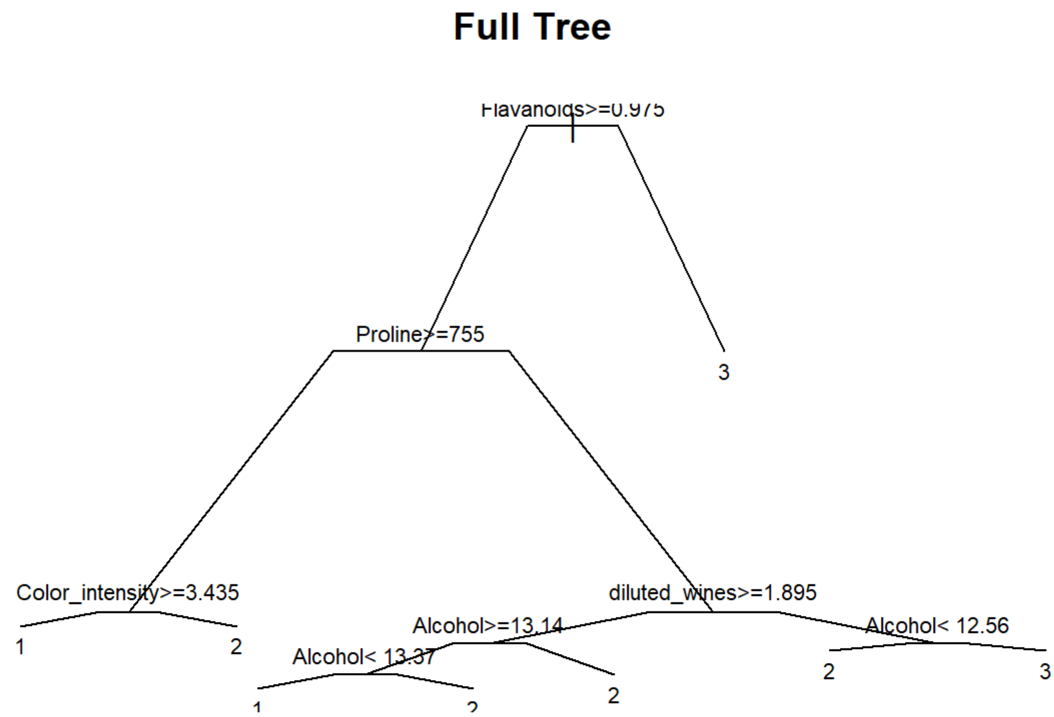

```
plot(fit.X$cptable[,1], main = "Cp for model selection", ylab = "Cp",pch=19,type='o')
```



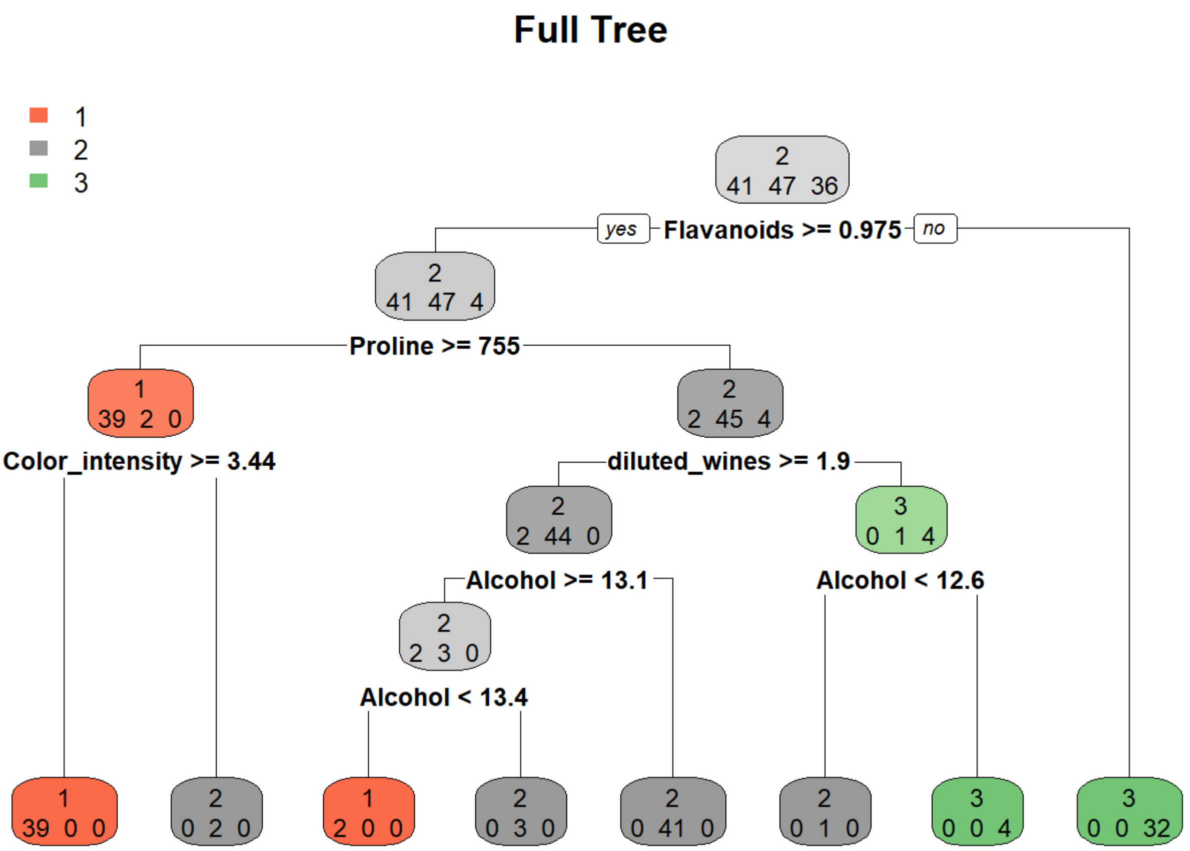

```
min_cp <- which.min(fit.X$cptable[,4])
min_cp
```

```
## 4
## 4
```

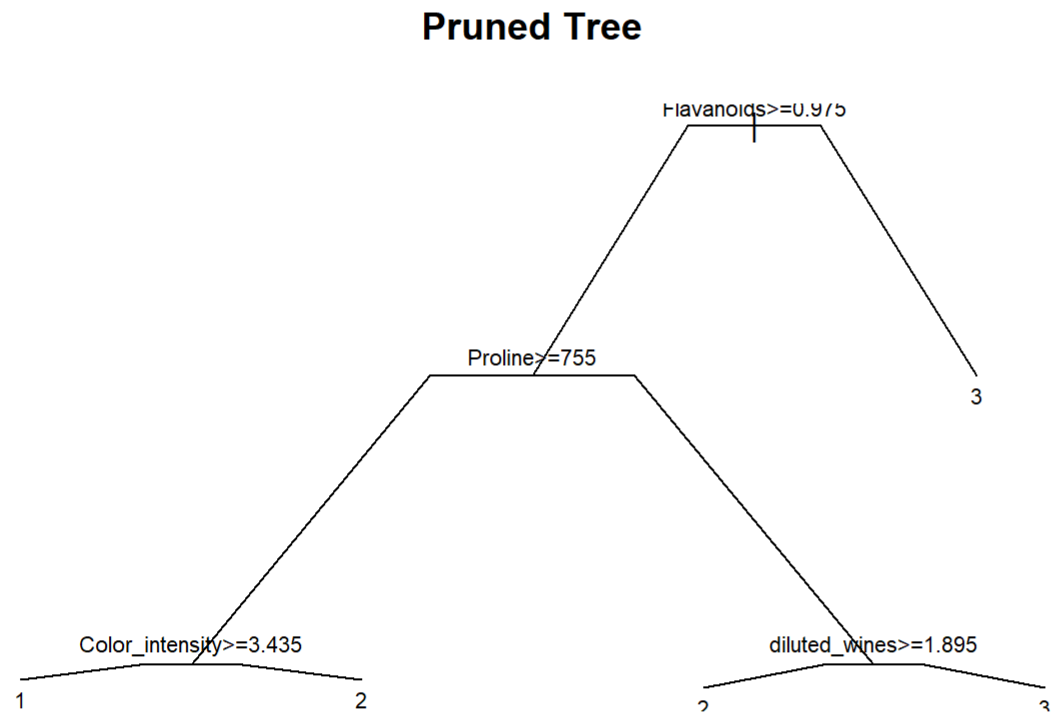
```
pruned_fit_X <- prune(fit.X, cp = fit.X$cptable[min_cp, 1])

plot(fit.X, branch = .3, compress = T, main = "Full Tree")
text(fit.X, cex = .7)
```




```
rpart.plot(fit.X,digits = 3, extra = 1,main = "Full Tree")
```

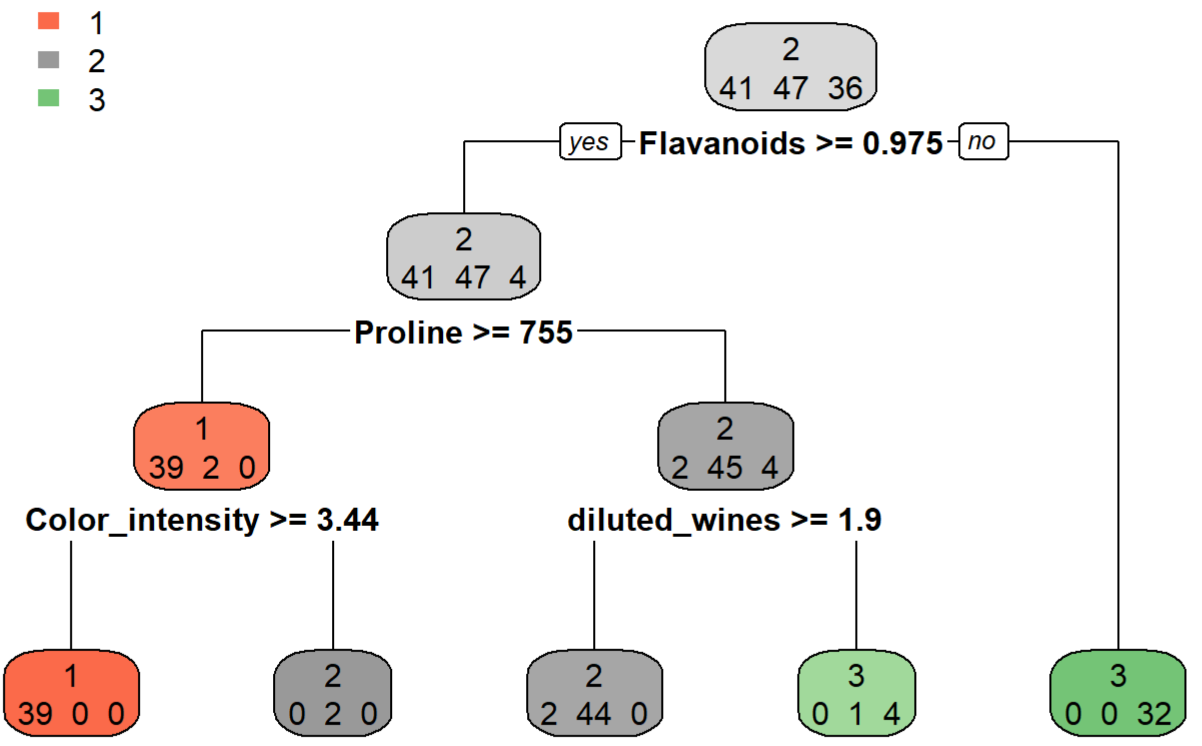


```
plot(pruned_fit_X, branch = .3, compress = T, main = "Pruned Tree")
text(pruned_fit_X, cex = .7)
```




```
rpart.plot(pruned_fit_X,digits = 3, extra = 1,main = "Pruned Tree")
```

Pruned Tree




```
pred_test <- predict(pruned_fit_X, newdata = test_data, type = 'class')
table(pred_test,Y_test)
```

```
##           Y_test
## pred_test  1  2  3
##          1 17  0  0
##          2  0 22  1
##          3  0  2 11
```

```
acc <- sum(pred_test == Y_test) / nrow(test_data)
cat("Classification Tree Accuracy:", acc * 100, "\n")
```

```
## Classification Tree Accuracy: 94.33962
```

Random forest :

```
library(randomForest)
set.seed(123)

train_data$wines <- as.factor(train_data$wines)
test_data$wines <- as.factor(test_data$wines)

rf_model <- randomForest(wines ~ ., data = train_data, ntree = 1000)
rf_pred <- predict(rf_model, newdata = test_data, type = 'class')

table(rf_pred,Y_test)
```

```
##        Y_test
## rf_pred  1  2  3
##       1 17  0  0
##       2  0 23  0
##       3  0  1 12
```

```
rf_acc <- sum(rf_pred == Y_test) / nrow(test_data)
cat("Random Forest Accuracy:", rf_acc * 100, "\n")
```

```
## Random Forest Accuracy: 98.11321
```

Boosting :

```
library(adabag)
```

```
## Warning: package 'adabag' was built under R version 4.3.2
```

```
## Loading required package: caret
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
## Loading required package: lattice
```

```
## Loading required package: foreach
```

```
## Loading required package: doParallel
```

```
## Warning: package 'doParallel' was built under R version 4.3.2
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
set.seed(123)

train_data$wines <- as.factor(train_data$wines)
test_data$wines <- as.factor(test_data$wines)

boost_model <- boosting(wines ~ ., data = train_data, mfinal = 500)

boost_pred <- predict(boost_model, newdata = test_data, type = 'class')

names(boost_pred)
```

```
## [1] "formula"   "votes"     "prob"      "class"     "confusion" "error"
```

```
boost_pred$confusion
```

```
##                Observed Class
## Predicted Class  1  2  3
##               1 17  0  0
##               2  0 23  1
##               3  0  1 11
```

```
boost_acc <- sum(boost_pred$class == Y_test) / nrow(test_data)
cat("Boosting Accuracy:", boost_acc * 100, "\n")
```

```
## Boosting Accuracy: 96.22642
```

```
library(dplyr)

data <- data.frame(Methods = c("Trees" ,"Random forest","Boosting"), Accuracy = c(acc * 100,r
f_acc * 100,boost_acc * 100))

arrange(data,desc(Accuracy))
```

```
##         Methods Accuracy
## 1 Random forest 98.11321
## 2      Boosting 96.22642
## 3         Trees 94.33962
```

Random forest accuracy is more than the other two methods Boosting and trees.

So, we can say Random forest performs better compared to the other models.

Question 3) Adopted from ISLR2: This problem involves the OJ data set in the ISLR2 package.We are interested in the prediction of “Purchase”. Divide the data into test and training.

```r
library(ISLR2)
library(e1071)
set.seed(123)

data("OJ")
dim(OJ)  ### 1070 * 18
```

```
## [1] 1070   18
```

```r
head(OJ)
```

```
##   Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1       CH            237       1    1.75    1.99   0.00    0.0         0
## 2       CH            239       1    1.75    1.99   0.00    0.3         0
## 3       CH            245       1    1.86    2.09   0.17    0.0         0
## 4       MM            227       1    1.69    1.69   0.00    0.0         0
## 5       CH            228       7    1.69    1.69   0.00    0.0         0
## 6       CH            230       7    1.69    1.99   0.00    0.0         0
##   SpecialMM  LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM
## 1         0 0.500000        1.99        1.75      0.24     No  0.000000
## 2         1 0.600000        1.69        1.75     -0.06     No  0.150754
## 3         0 0.680000        2.09        1.69      0.40     No  0.000000
## 4         0 0.400000        1.69        1.69      0.00     No  0.000000
## 5         0 0.956535        1.69        1.69      0.00    Yes  0.000000
## 6         1 0.965228        1.99        1.69      0.30    Yes  0.000000
##   PctDiscCH ListPriceDiff STORE
## 1  0.000000          0.24     1
## 2  0.000000          0.24     1
## 3  0.091398          0.23     1
## 4  0.000000          0.00     1
## 5  0.000000          0.00     0
## 6  0.000000          0.30     0
```

```r
unique(OJ$Purchase)
```

```
## [1] CH MM
## Levels: CH MM
```

```r
indis <- sample(1:nrow(OJ),size = round(0.7 * nrow(OJ)))

train_data <- OJ[indis, ]
test_data <- OJ[-indis, ]

X_train <- train_data[, -1]
Y_train <- train_data[, 1]

X_test <- test_data[, -1]
Y_test <- test_data[, 1]
```

a)) Fit a support vector classifier with varying cost parameters over the range [0.01,10]. Plot the training and test error across this spectrum of cost parameters and determine the optimal cost.

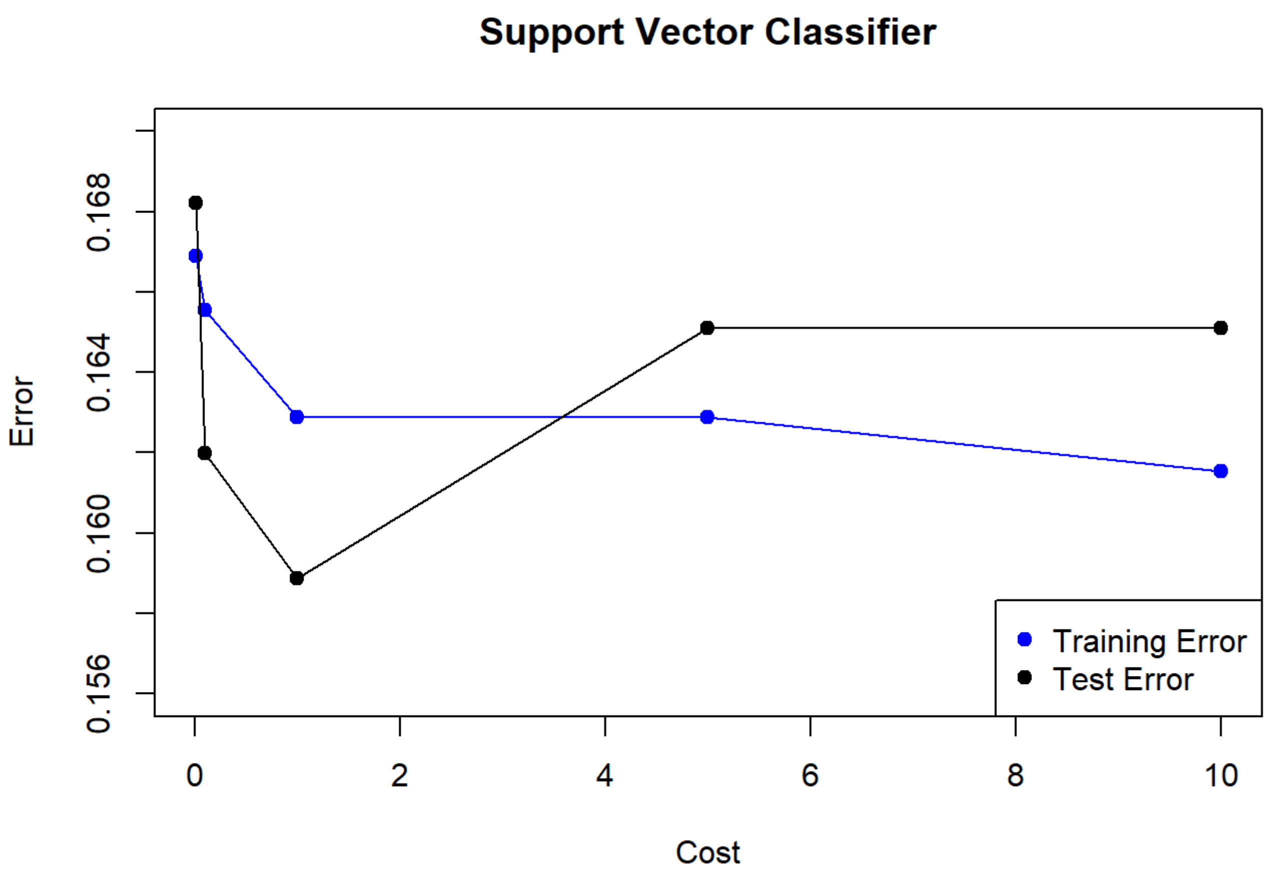
```
set.seed(123)

cost_values <- c(0.01,0.1,1,5,10)
train_error <- rep(NA,length(cost_values))
test_error <- rep(NA,length(cost_values))

for (i in 1:length(cost_values)){
  fit <- svm(Purchase~., data = train_data, kernel = "linear", cost = cost_values[i], type = 
'C-classification')
  train_pred <- predict(fit, train_data)
  test_pred <- predict(fit, test_data)

  train_error[i] <- sum(train_pred != Y_train)/length(Y_train)
  test_error[i] <- sum(test_pred != Y_test)/length(Y_test)
}

train_error
```

```
## [1] 0.1668892 0.1655541 0.1628838 0.1628838 0.1615487
```

```
test_error
```

```
## [1] 0.1682243 0.1619938 0.1588785 0.1651090 0.1651090
```

```
plot(cost_values, train_error, type = "o", col = "blue", xlab = "Cost", ylab = "Error", main 
= "Support Vector Classifier",ylim = c(0.156,0.170),pch = 19)
lines(cost_values, test_error, type = "o", col = "black",pch = 19)
legend("bottomright", legend = c("Training Error", "Test Error"), col = c("blue", "black"), p
ch = 19)
```




```
optimal_cost <- cost_values[which.min(test_error)]

cat("Optimal cost for linear SVM is :", optimal_cost)
```

```
## Optimal cost for linear SVM is : 1
```

b) Repeat the exercise in (A) for a support vector machine with a radial kernel. (Use the default parameter for gamma). Repeat the exercise again for a support vector machine with a polynomial kernel of degree=2. Reflect on the performance of the SVM with different kernels, and the support vector classifier, i.e., SVM with a linear kernel.

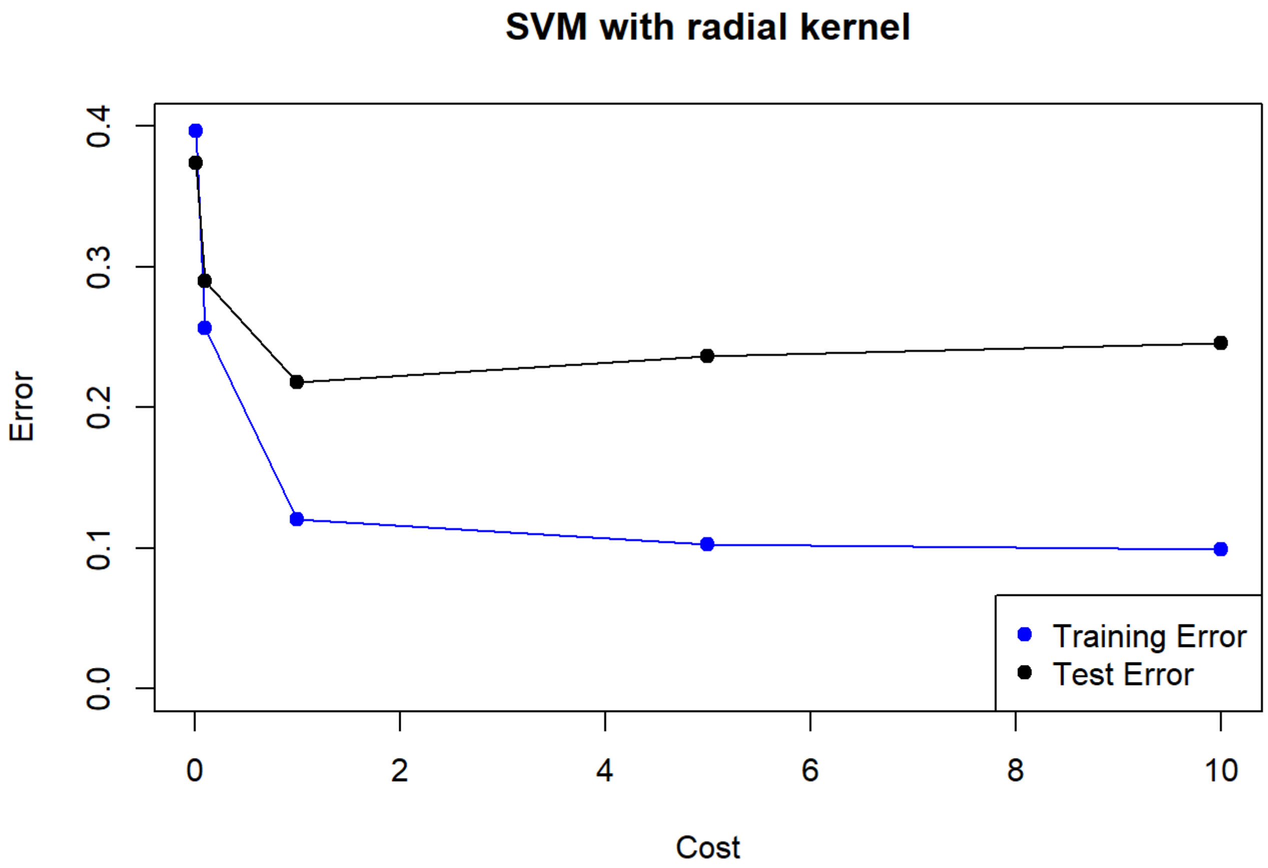
```r
set.seed(123)

cost_values <- c(0.01,0.1,1,5,10)
train_error1 <- rep(NA,length(cost_values))
test_error1 <- rep(NA,length(cost_values))

for (i in 1:length(cost_values)){
  fit1 <- svm(Purchase~., data = train_data, kernel = "radial", cost = cost_values[i], type =
'C-classification',gamma = 0.5)
  train_pred1 <- predict(fit1, train_data)
  test_pred1 <- predict(fit1, test_data)

  train_error1[i] <- sum(train_pred1 != Y_train)/length(Y_train)
  test_error1[i] <- sum(test_pred1 != Y_test)/length(Y_test)
}

train_error1
```

```
## [1] 0.3965287 0.2563418 0.1201602 0.1028037 0.0987984
```

```r
test_error1
```

```
## [1] 0.3738318 0.2897196 0.2180685 0.2367601 0.2461059
```

```r
plot(cost_values, train_error1, type = "o", col = "blue", xlab = "Cost", ylab = "Error", main
= "SVM with radial kernel",ylim = c(0,0.40),pch = 19)
lines(cost_values, test_error1, type = "o", col = "black",pch = 19)
legend("bottomright", legend = c("Training Error", "Test Error"), col = c("blue", "black"), p
ch = 19)
```




```
optimal_cost1 <- cost_values[which.min(test_error1)]

cat("Optimal cost for SVM with radial kernal is :", optimal_cost1)
```

```
## Optimal cost for SVM with radial kernal is : 1
```

SVM with polynomial kernel of degree = 2.

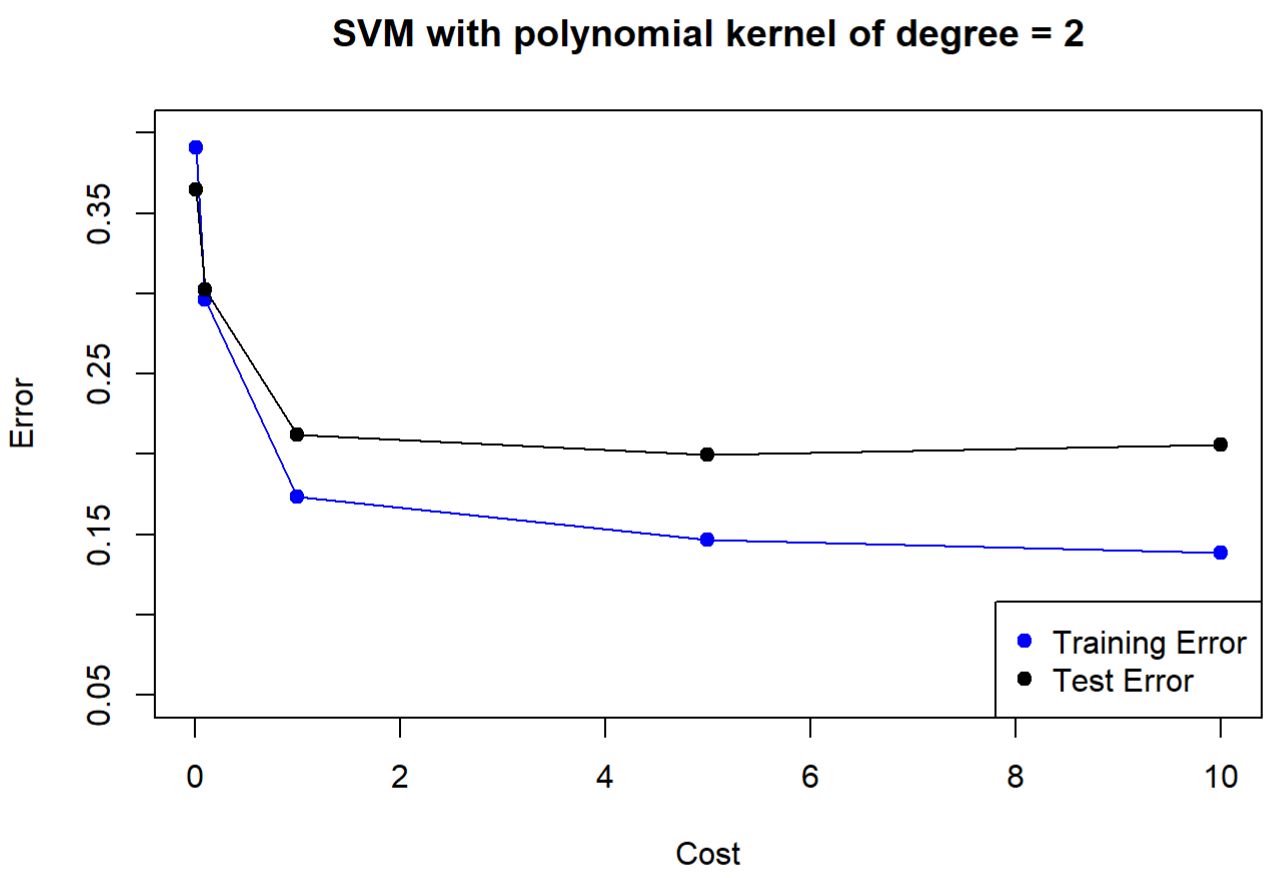
```
set.seed(123)

train_error2 <- rep(NA,length(cost_values))
test_error2 <- rep(NA,length(cost_values))

for (i in 1:length(cost_values)){
  fit2 <- svm(Purchase~., data = train_data, kernel = "polynomial", cost = cost_values[i], de
gree = 2, type = 'C-classification')
  train_pred2 <- predict(fit2, train_data)
  test_pred2 <- predict(fit2, test_data)

  train_error2[i] <- sum(train_pred2 != Y_train)/length(Y_train)
  test_error2[i] <- sum(test_pred2 != Y_test)/length(Y_test)
}

train_error2
```

```
## [1] 0.3911883 0.2963952 0.1735648 0.1468625 0.1388518
```

```
test_error2
```

```
## [1] 0.3644860 0.3021807 0.2118380 0.1993769 0.2056075
```

```
plot(cost_values, train_error2, type = "o", col = "blue", xlab = "Cost", ylab = "Error", main
= "SVM with polynomial kernel of degree = 2",ylim = c(0.05,0.40),pch = 19)
lines(cost_values, test_error2, type = "o", col = "black",pch = 19)
legend("bottomright", legend = c("Training Error", "Test Error"), col = c("blue", "black"), p
ch = 19)
```




```
optimal_cost2 <- cost_values[which.min(test_error2)]

cat("Optimal cost for SVM with polynomial kernel of degree = 2 :", optimal_cost2)
```

```
## Optimal cost for SVM with polynomial kernel of degree = 2 : 5
```

```
library(dplyr)

data <- data.frame(Methods = c("SVM with linear kernel","SVM with radial kernel","SVM with po
lynomial kernel of degree = 2"), Optimal_cost = c(optimal_cost,optimal_cost1,optimal_cost2) ,
Test_MSE_percentage = c(round(min(test_error) * 100 , 2), round(min(test_error1) * 100 , 2),r
ound(min(test_error2) * 100 , 2)))

arrange(data,Test_MSE_percentage)
```

```
##                                     Methods Optimal_cost Test_MSE_percentage
## 1                   SVM with linear kernel            1               15.89
## 2 SVM with polynomial kernel of degree = 2            5               19.94
## 3                   SVM with radial kernel            1               21.81
```

SVM with linear kernel performs well with low test error compared to the other two models.

And second SVM with polynomial kernel of degree = 2 performs better compared to SVM with radial kernel.

Question 4) 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.

```
library(ISLR2)
data("Auto")
dim(Auto)    ### 392 * 9
```

```
## [1] 392   9
```

```
head(Auto)
```

```
##   mpg cylinders displacement horsepower weight acceleration year origin
## 1  18         8          307        130   3504         12.0   70      1
## 2  15         8          350        165   3693         11.5   70      1
## 3  18         8          318        150   3436         11.0   70      1
## 4  16         8          304        150   3433         12.0   70      1
## 5  17         8          302        140   3449         10.5   70      1
## 6  15         8          429        198   4341         10.0   70      1
##                        name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

```
A <- Auto
```

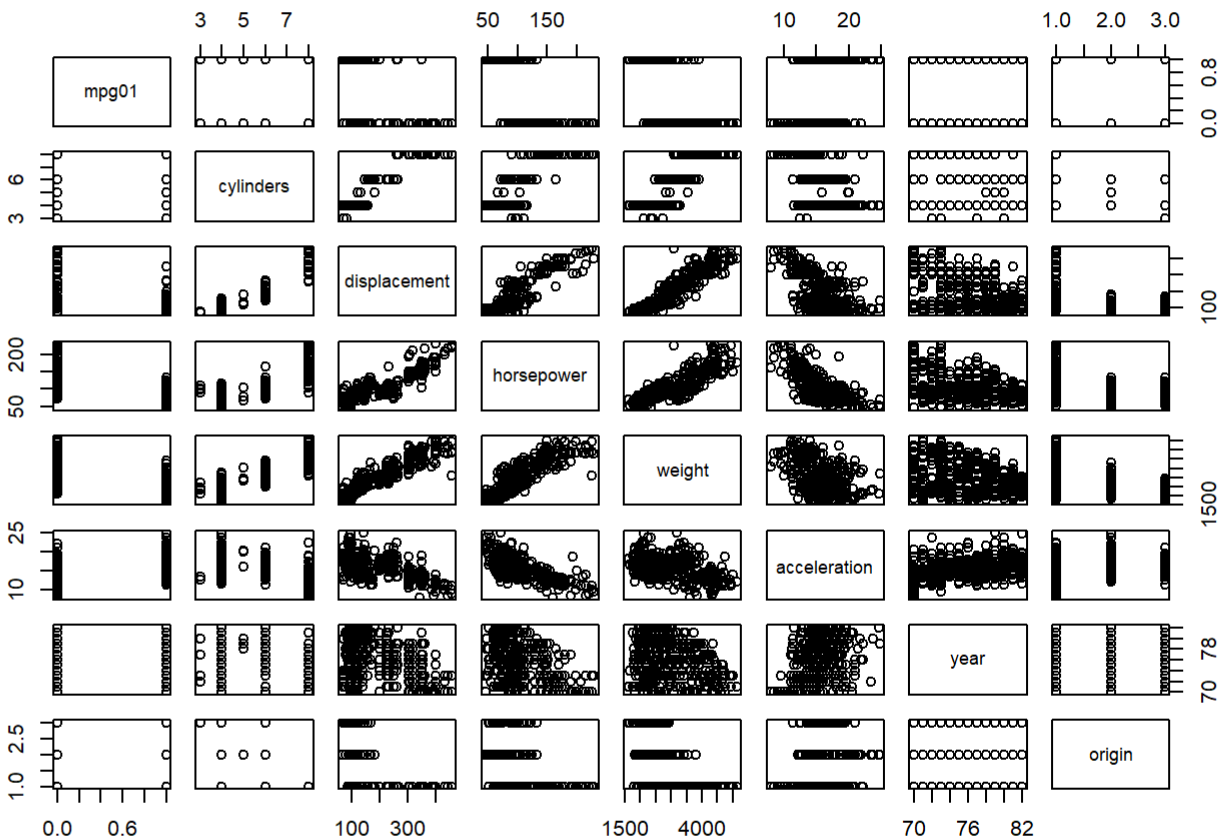
a) Create a binary variable, mpg01, that contains a 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median. You can compute the median using the median() function. Note you may fnd it helpful to use the data.frame() function to create a single data set containing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

```
mpg01 <- ifelse(A$mpg > median(A$mpg), 1 , 0)
my_auto <- data.frame(mpg01 , A[,-1])
head(my_auto)
```

```
##   mpg01 cylinders displacement horsepower weight acceleration year origin
## 1     0         8          307        130   3504         12.0   70      1
## 2     0         8          350        165   3693         11.5   70      1
## 3     0         8          318        150   3436         11.0   70      1
## 4     0         8          304        150   3433         12.0   70      1
## 5     0         8          302        140   3449         10.5   70      1
## 6     0         8          429        198   4341         10.0   70      1
##                        name
## 1 chevrolet chevelle malibu
## 2         buick skylark 320
## 3        plymouth satellite
## 4             amc rebel sst
## 5               ford torino
## 6          ford galaxie 500
```

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.

```
pairs(my_auto[, -9])
```



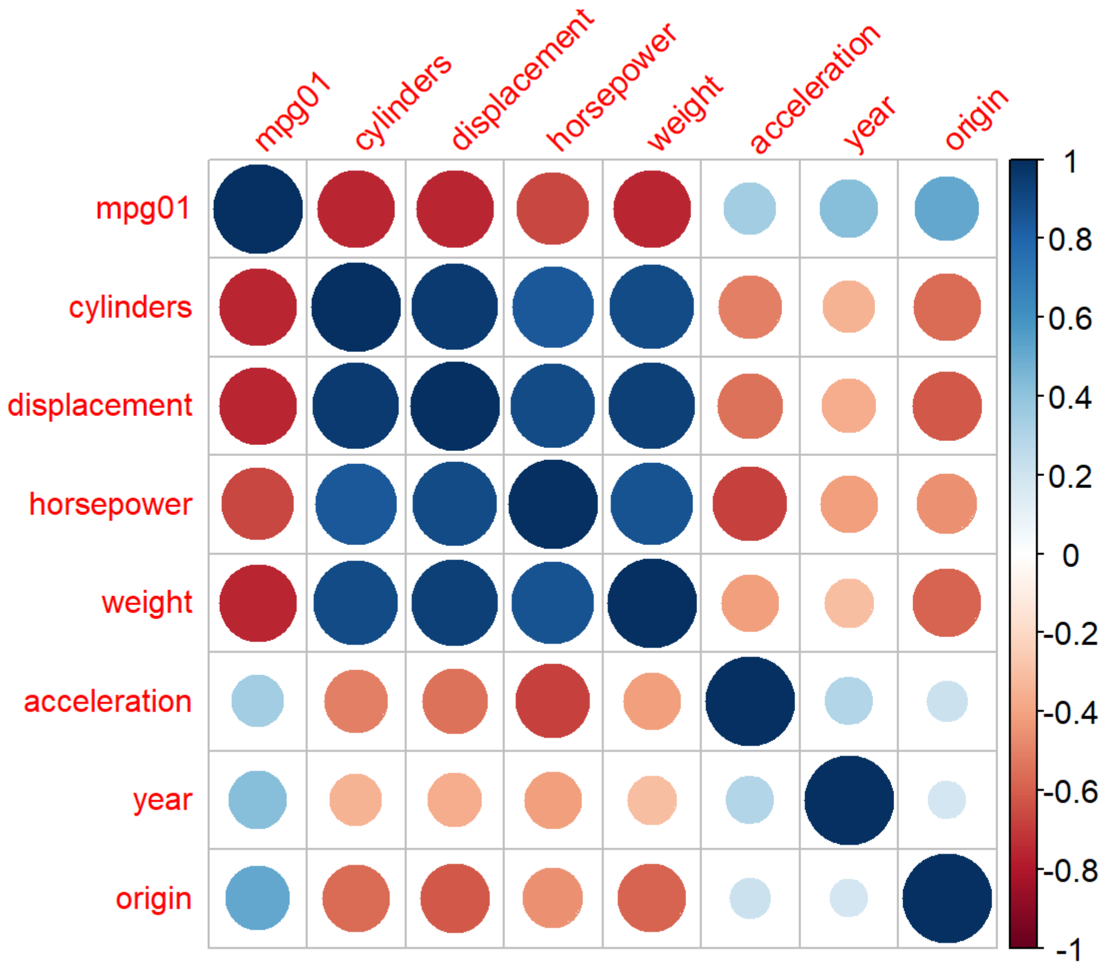

```
cor(my_auto[, -9])
```

```
##                    mpg01  cylinders displacement horsepower     weight
## mpg01          1.0000000 -0.7591939   -0.7534766 -0.6670526 -0.7577566
## cylinders     -0.7591939  1.0000000    0.9508233  0.8429834  0.8975273
## displacement -0.7534766  0.9508233    1.0000000  0.8972570  0.9329944
## horsepower    -0.6670526  0.8429834    0.8972570  1.0000000  0.8645377
## weight        -0.7577566  0.8975273    0.9329944  0.8645377  1.0000000
## acceleration  0.3468215 -0.5046834   -0.5438005 -0.6891955 -0.4168392
## year           0.4299042 -0.3456474   -0.3698552 -0.4163615 -0.3091199
## origin         0.5136984 -0.5689316   -0.6145351 -0.4551715 -0.5850054
##              acceleration       year     origin
## mpg01           0.3468215  0.4299042  0.5136984
## cylinders      -0.5046834 -0.3456474 -0.5689316
## displacement   -0.5438005 -0.3698552 -0.6145351
## horsepower     -0.6891955 -0.4163615 -0.4551715
## weight         -0.4168392 -0.3091199 -0.5850054
## acceleration    1.0000000  0.2903161  0.2127458
## year            0.2903161  1.0000000  0.1815277
## origin          0.2127458  0.1815277  1.0000000
```

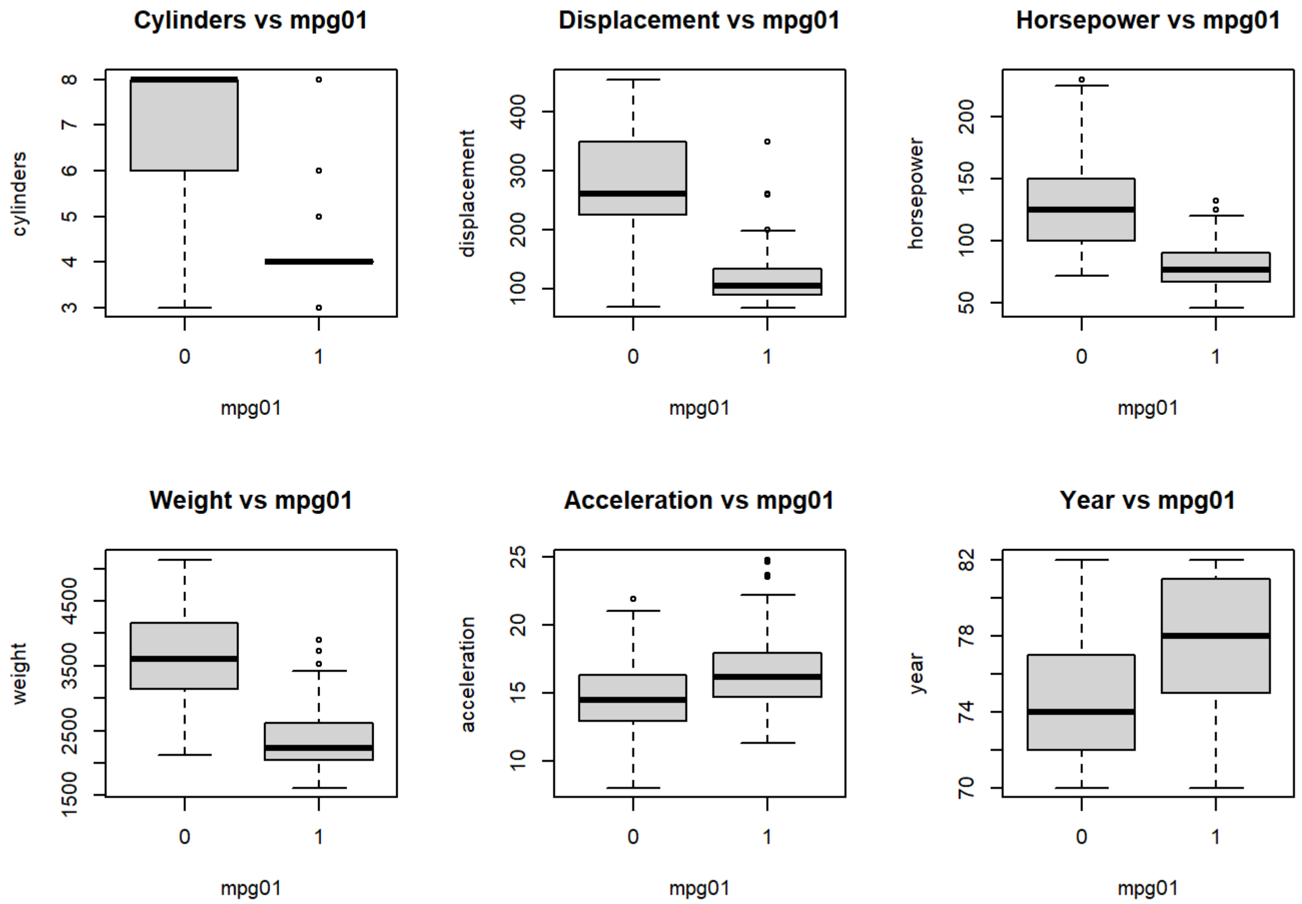
```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
corrplot(cor(my_auto[, -9]), tl.col = "red", tl.srt = 45, tl.cex = 1, cl.cex = 1)
```



```
par(mfrow=c(2,3))
boxplot(cylinders ~ mpg01, data = my_auto, main = "Cylinders vs mpg01")
boxplot(displacement ~ mpg01, data = my_auto, main = "Displacement vs mpg01")
boxplot(horsepower ~ mpg01, data = my_auto, main = "Horsepower vs mpg01")
boxplot(weight ~ mpg01, data = my_auto, main = "Weight vs mpg01")
boxplot(acceleration ~ mpg01, data = my_auto, main = "Acceleration vs mpg01")
boxplot(year ~ mpg01, data = my_auto, main = "Year vs mpg01")
```



As seen in the above scatterplots, boxplots and the correlation values there is some association between “mpg01” and “cylinders”, “displacement”, “horsepower” and “weight”.

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

```
set.seed(123)

indis <- sample(1:nrow(my_auto),size = round(0.7 * nrow(my_auto)))

train_data <- my_auto[indis, ]
test_data <- my_auto[-indis, ]

X_train <- train_data[, -1]
Y_train <- train_data[, 1]

X_test <- test_data[, -1]
Y_test <- test_data[, 1]
```

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?

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
## The following object is masked from 'package:ISLR2':
##
##     Boston
```

```r
set.seed(123)

lda_model <- lda(mpg01 ~ cylinders + displacement + horsepower + weight, data = train_data)
lda_model
```

```
## Call:
## lda(mpg01 ~ cylinders + displacement + horsepower + weight, data = train_data)
##
## Prior probabilities of groups:
##         0         1
## 0.4963504 0.5036496
##
## Group means:
##   cylinders displacement horsepower   weight
## 0  6.786765     275.2941  130.96324 3641.022
## 1  4.188406     114.5290   78.00725 2314.000
##
## Coefficients of linear discriminants:
##                        LD1
## cylinders    -0.3974647924
## displacement -0.0029615583
## horsepower    0.0049004106
## weight       -0.0009670704
```

```r
lda_pred <- predict(lda_model, newdata = test_data)
names(lda_pred)
```

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

```r
table(lda_pred$class, Y_test)
```

```
##    Y_test
##      0  1
##   0 50  3
##   1 10 55
```

```
lda_test_mse <- mean(lda_pred$class != Y_test)

cat("LDA test error rate is :", round(lda_test_mse * 100, 2), "%")
```

```
## LDA test error rate is : 11.02 %
```

e) Perform QDA 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?

```
set.seed(123)

qda_model <- qda(mpg01 ~ cylinders + displacement + horsepower + weight, data = train_data)
qda_model
```

```
## Call:
## qda(mpg01 ~ cylinders + displacement + horsepower + weight, data = train_data)
##
## Prior probabilities of groups:
##         0         1
## 0.4963504 0.5036496
##
## Group means:
##   cylinders displacement horsepower   weight
## 0  6.786765     275.2941  130.96324 3641.022
## 1  4.188406     114.5290   78.00725 2314.000
```

```
qda_pred <- predict(qda_model, newdata = test_data)
names(qda_pred)
```

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

```
table(qda_pred$class, Y_test)
```

```
##    Y_test
##      0  1
##   0 53  5
##   1  7 53
```

```
qda_test_mse <- mean(qda_pred$class != Y_test)

cat("QDA test error rate is :", round(qda_test_mse * 100, 2), "%")
```

```
## QDA test error rate is : 10.17 %
```

f) Perform logistic regression 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?

```
set.seed(123)

glm_model <- glm(mpg01 ~ cylinders + displacement + horsepower + weight, data = train_data, f
amily = "binomial")
summary(glm_model)
```

```
##
## Call:
## glm(formula = mpg01 ~ cylinders + displacement + horsepower +
##     weight, family = "binomial", data = train_data)
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)  11.8103006  2.0819718   5.673 1.41e-08 ***
## cylinders     0.1869071  0.3972245   0.471  0.63797
## displacement -0.0164493  0.0095899  -1.715  0.08629 .
## horsepower   -0.0443408  0.0172072  -2.577  0.00997 **
## weight       -0.0020251  0.0008573  -2.362  0.01817 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 379.83  on 273  degrees of freedom
## Residual deviance: 138.27  on 269  degrees of freedom
## AIC: 148.27
##
## Number of Fisher Scoring iterations: 7
```

```
glm_pred <- round(predict(glm_model, newdata = test_data, type = "response"))

table(glm_pred,Y_test)
```

```
##         Y_test
## glm_pred  0  1
##        0 53  6
##        1  7 52
```

```
glm_test_mse <- mean(glm_pred != Y_test)

cat("logistic regression test error rate is :", round(glm_test_mse * 100, 2), "%")
```

```
## logistic regression test error rate is : 11.02 %
```

g) Perform KNN on the training data, with several values of K, in order to predict mpg01. Use only the variables that seemed most associated with mpg01 in (b). What test errors do you obtain? Which value of K seems to perform the best on this data set?

```r
library(class)

set.seed(123)

k_value <- seq(from = 1, to = 10, by = 1)
knn_error <- numeric(length(k_value))
xtrain <- train_data[ ,c("cylinders", "displacement","horsepower","weight")]
xtest <- test_data[ ,c("cylinders", "displacement","horsepower","weight")]

for (k in k_value) {
  knn_pred <- knn(xtrain, xtest, Y_train, k = k)
  knn_error[k] <- mean(knn_pred != Y_test)
}

knn_error
```

```
##  [1] 0.16949153 0.14406780 0.12711864 0.09322034 0.11016949 0.10169492
##  [7] 0.10169492 0.10169492 0.11016949 0.10169492
```

```r
table(knn_pred,Y_test)
```

```
##         Y_test
## knn_pred  0  1
##        0 52  4
##        1  8 54
```

```r
best_k <- k_value[which.min(knn_error)]
knn_test_mse <- min(knn_error)

cat("knn test error rate is :", round(knn_test_mse * 100 , 2), "% for K = ",best_k)
```

```
## knn test error rate is : 9.32 % for K =  4
```

```r
library(dplyr)

data <- data.frame(Methods = c("LDA","QDA","logistic regression","knn for k = 4"), Test_MSE_p
ercentage = c(round(lda_test_mse * 100, 2),round(qda_test_mse * 100, 2),round(glm_test_mse *
100, 2),round(knn_test_mse * 100 , 2)))

arrange(data,Test_MSE_percentage)
```

```
##               Methods Test_MSE_percentage
## 1       knn for k = 4                9.32
## 2                 QDA               10.17
## 3                 LDA               11.02
## 4 logistic regression               11.02
```

On this “Auto” data set KNN model performs well compared to the other methods with low test MSE.

And QDA also performs well compared to LDA and Logistic Regression.