MIS40970 Data Mining - Assignment 3 Classification

Shruti Goyal (16200726)

```
#rm() is used to remove other objects from the environment
rm(list=ls())

#To check the working directory
getwd()
```

```
## [1] "G:/R_programs_git/R_Progams/Classification/Classification"
```

```
#A specific working drectory needs to be set for the loading of dataset
setwd("G:/R_programs_git/R_Progams/Classification/Classification")
```

Q1 Compare and contrast classification and clustering.

Clustering is a process of grouping objects together in a way that objects with similar features will be together and dissimilar objects will be together. It is used for data analysis.

Classification is a method of categorization where objects are recognised, differentiated on basis of training dataset.

1. **Supervision** : Classification - Supervised learning Clustering - Unsupervised learning
2. **Training set** : Classification - Find simmilarities using training set Clustering - No training set is used
3. **Process** : Classification - Categorise data as per observations in training set Clustering - Use statistical concepts and datasets are split with similar and dissimilar features
4. **Labels** : Classification - Use of labels Clustering - no labels
5. **Data Mining Methods** : Classification - It is a method of predicting instances from labeled instances Clustering - This methods identifies natural grouping of instances from unlabeled data

Examples : Classification - Random Forest, rpart, Ctree, Tree Clustering - kmeans, PAM, hierarchial

Both the methods are similar to each other as both divide the data into subsets and yet two different learning methods to get relevant information from raw data.

Q2 Describe what this piece of R code is doing and why it is an important starting point for running classification algorithm. ##> set.seed(1234) ##> dataPartition <- sample(2,nrow(data),replace=TRUE,prob=c(0.7,0.3)) ##> trainData <- data[dataPartition ==1,] ##> testData <- [dataPartition ==2,]

Classification needs supervised learning where we need to partition the data which can be used to train dataset. To explain the above set of commands, data has been imported from college.csv file as shown below:

```
#Importing library readr to load data from csv file
library(readr)
```

```
## Warning: package 'readr' was built under R version 3.3.2
```

```
#Importing data from CSV in a variable college
college <- read_csv("G:/R_programs_git/R_Progams/Classification/Classification/college.csv")
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
## Parsed with column specification:
## cols(
##   X1 = col_integer(),
##   name = col_character(),
##   accept_rate = col_double(),
##   Outstate = col_integer(),
##   Enroll = col_integer(),
##   Grad.Rate = col_integer(),
##   Private = col_character(),
##   isElite = col_character()
## )
```

```
set.seed(1234)
dataPartition <- sample(2,nrow(college),replace=TRUE,prob=c(0.7,0.3))
trainCollege <- college[dataPartition ==1,]
testCollege <- college[dataPartition ==2,]
```

For the first statement(**set.seed(1234)**), we use seed number as a starting point that is used to generate a sequence of pseudo random numbers. This function is important if there is need that results should be reproducible and debuggable easily.

Second statement(**dataPartition<-sample(2,nrow(data),replace=TRUE,prob=c(0.7,0.3))**) represents the properties of the partitions that need to be taken. Sample function takes a specified size from 2(an integer vector that can take one or more elements) so that function can generate random permutation of elements of vector(1:vector). If vector is 4, then the random permutation sequence will take numbers between 1 and 4 numbers. nrow(data) represents the size giving the numbers of items to choose and nrow is the last row of the college dataset. replace represents whether the sampling should be done with replacement or without replacement. Replace=TRUE is set to do the sampling with replacement. And prob will take probability weights for obtaining sampled elements. So it is expected that 1 will appear more than 2 times as weight is 0.7 than 0.3. If we write prob=c(0.3,0.5,0.2) then 1 is appeared to be less times than 2 but more than 3.

trainCollege <- college[dataPartition ==1,] and testCollege <- college[dataPartition ==2,] split the college dataset into 2 datasets i.e. Test data and Train data, where test data is used for testing.

All these commands will help in creating training dataset that can be changed continuously that help in formulation of classification algorithms.

Q3 What is the role of the M parameter in the Weka implementation of C4.5 algorithm? Which part of the DTL induction process does this parameter affect?

While implementing C4.5 algorithm in Weka software, M parameter sets a minimum instances per leaf that effects the separation of data on the decision tree. Minimum of the instances have atleast two branches will be at each split.As a result, there will be effect on the total number of tree formulated.

```
#Installing Weka package

#install.packages("RWeka")
```

Q4. Install R package "C50". Import customer churn dataset (churn) using data() function. Examine the churnTrain dataset. Using R run a decision-tree classification algorithm of your choice constructing a full unpruned tree and a pruned tree. Compare classification results of the pruned and unpruned trees generated.

```r
#install.packages('C50')

library(C50)
```

```
## Warning: package 'C50' was built under R version 3.3.3
```

```r
#To examine datasets in package C50
data()
```

After examining data function, it has been found out that there are two datasets in 'Customer Churn Data' (C50 package), which are churnTest and churnTrain. To import datasets in C50 package, use command "data(churn)".

```r
data(churn)
print("Details of attributes in churnTrain")
```

```
## [1] "Details of attributes in churnTrain"
```

```r
print("____________________________________")
```

```
## [1] "____________________________________"
```

```r
str(churnTrain)
```

```
## 'data.frame':    3333 obs. of  20 variables:
##  $ state                        : Factor w/ 51 levels "AK","AL","AR",..: 17 36 32 36 37 2 20 25 19 50
##  $ account_length               : int  128 107 137 84 75 118 121 147 117 141 ...
##  $ area_code                    : Factor w/ 3 levels "area_code_408",..: 2 2 2 1 2 3 3 2 1 2 ...
##  $ international_plan           : Factor w/ 2 levels "no","yes": 1 1 1 2 2 2 1 2 1 2 ...
##  $ voice_mail_plan              : Factor w/ 2 levels "no","yes": 2 2 1 1 1 1 2 1 1 2 ...
##  $ number_vmail_messages        : int  25 26 0 0 0 0 24 0 0 37 ...
##  $ total_day_minutes            : num  265 162 243 299 167 ...
##  $ total_day_calls              : int  110 123 114 71 113 98 88 79 97 84 ...
##  $ total_day_charge             : num  45.1 27.5 41.4 50.9 28.3 ...
##  $ total_eve_minutes            : num  197.4 195.5 121.2 61.9 148.3 ...
##  $ total_eve_calls              : int  99 103 110 88 122 101 108 94 80 111 ...
##  $ total_eve_charge             : num  16.78 16.62 10.3 5.26 12.61 ...
##  $ total_night_minutes          : num  245 254 163 197 187 ...
##  $ total_night_calls            : int  91 103 104 89 121 118 118 96 90 97 ...
##  $ total_night_charge           : num  11.01 11.45 7.32 8.86 8.41 ...
##  $ total_intl_minutes           : num  10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 8.7 11.2 ...
##  $ total_intl_calls             : int  3 3 5 7 3 6 7 6 4 5 ...
##  $ total_intl_charge            : num  2.7 3.7 3.29 1.78 2.73 1.7 2.03 1.92 2.35 3.02 ...
##  $ number_customer_service_calls: int  1 1 0 2 3 0 3 0 1 0 ...
##  $ churn                        : Factor w/ 2 levels "yes","no": 2 2 2 2 2 2 2 2 2 2 ...
```

```r
print("---------------------------Summary Table :churnTrain---------------------")
```

```
## [1] "---------------------------Summary Table :churnTrain---------------------"
```

```r
#To understand dataset churnTrain
summary(churnTrain)
```

```
##      state      account_length          area_code    international_plan
##  WV     : 106   Min.   :  1.0   area_code_408: 838   no :3010
##  MN     :  84   1st Qu.: 74.0   area_code_415:1655   yes: 323
##  NY     :  83   Median :101.0   area_code_510: 840
##  AL     :  80   Mean   :101.1
##  OH     :  78   3rd Qu.:127.0
##  OR     :  78   Max.   :243.0
##  (Other):2824
##  voice_mail_plan number_vmail_messages total_day_minutes total_day_calls
##  no :2411        Min.   : 0.000        Min.   :  0.0     Min.   :  0.0
##  yes: 922        1st Qu.: 0.000        1st Qu.:143.7     1st Qu.: 87.0
##                  Median : 0.000        Median :179.4     Median :101.0
##                  Mean   : 8.099        Mean   :179.8     Mean   :100.4
##                  3rd Qu.:20.000        3rd Qu.:216.4     3rd Qu.:114.0
##                  Max.   :51.000        Max.   :350.8     Max.   :165.0
##
##  total_day_charge total_eve_minutes total_eve_calls total_eve_charge
##  Min.   : 0.00    Min.   :  0.0     Min.   :  0.0   Min.   : 0.00
##  1st Qu.:24.43    1st Qu.:166.6     1st Qu.: 87.0   1st Qu.:14.16
##  Median :30.50    Median :201.4     Median :100.0   Median :17.12
##  Mean   :30.56    Mean   :201.0     Mean   :100.1   Mean   :17.08
##  3rd Qu.:36.79    3rd Qu.:235.3     3rd Qu.:114.0   3rd Qu.:20.00
##  Max.   :59.64    Max.   :363.7     Max.   :170.0   Max.   :30.91
##
##  total_night_minutes total_night_calls total_night_charge
##  Min.   : 23.2       Min.   : 33.0     Min.   : 1.040
##  1st Qu.:167.0       1st Qu.: 87.0     1st Qu.: 7.520
##  Median :201.2       Median :100.0     Median : 9.050
##  Mean   :200.9       Mean   :100.1     Mean   : 9.039
##  3rd Qu.:235.3       3rd Qu.:113.0     3rd Qu.:10.590
##  Max.   :395.0       Max.   :175.0     Max.   :17.770
##
##  total_intl_minutes total_intl_calls total_intl_charge
##  Min.   : 0.00      Min.   : 0.000   Min.   :0.000
##  1st Qu.: 8.50      1st Qu.: 3.000   1st Qu.:2.300
##  Median :10.30      Median : 4.000   Median :2.780
##  Mean   :10.24      Mean   : 4.479   Mean   :2.765
##  3rd Qu.:12.10      3rd Qu.: 6.000   3rd Qu.:3.270
##  Max.   :20.00      Max.   :20.000   Max.   :5.400
##
##  number_customer_service_calls churn
##  Min.   :0.000                 yes: 483
##  1st Qu.:1.000                 no :2850
##  Median :1.000
##  Mean   :1.563
##  3rd Qu.:2.000
##  Max.   :9.000
##
```

```r
churnTrain <- churnTrain[,c(-1,-4)]
print("Details of attributes in churnTrain after selecting few columns")
```

```
## [1] "Details of attributes in churnTrain after selecting few columns"
```

```r
print("______________________________________________________________")
```

```
## [1] "______________________________________________________________"
```

```r
str(churnTrain)
```

```
## 'data.frame':    3333 obs. of  18 variables:
##  $ account_length               : int  128 107 137 84 75 118 121 147 117 141 ...
##  $ area_code                    : Factor w/ 3 levels "area_code_408",..: 2 2 2 1 2 3 3 2 1 2 ...
##  $ voice_mail_plan              : Factor w/ 2 levels "no","yes": 2 2 1 1 1 1 2 1 1 2 ...
##  $ number_vmail_messages        : int  25 26 0 0 0 0 24 0 0 37 ...
##  $ total_day_minutes            : num  265 162 243 299 167 ...
##  $ total_day_calls              : int  110 123 114 71 113 98 88 79 97 84 ...
##  $ total_day_charge             : num  45.1 27.5 41.4 50.9 28.3 ...
##  $ total_eve_minutes            : num  197.4 195.5 121.2 61.9 148.3 ...
##  $ total_eve_calls              : int  99 103 110 88 122 101 108 94 80 111 ...
##  $ total_eve_charge             : num  16.78 16.62 10.3 5.26 12.61 ...
##  $ total_night_minutes          : num  245 254 163 197 187 ...
##  $ total_night_calls            : int  91 103 104 89 121 118 118 96 90 97 ...
##  $ total_night_charge           : num  11.01 11.45 7.32 8.86 8.41 ...
##  $ total_intl_minutes           : num  10 13.7 12.2 6.6 10.1 6.3 7.5 7.1 8.7 11.2 ...
##  $ total_intl_calls             : int  3 3 5 7 3 6 7 6 4 5 ...
##  $ total_intl_charge            : num  2.7 3.7 3.29 1.78 2.73 1.7 2.03 1.92 2.35 3.02 ...
##  $ number_customer_service_calls: int  1 1 0 2 3 0 3 0 1 0 ...
##  $ churn                        : Factor w/ 2 levels "yes","no": 2 2 2 2 2 2 2 2 2 2 ...
```

```r
set.seed(34)
dataPart <- sample(2, nrow(churnTrain),replace = TRUE,prob = c(0.7,0.3))
traindata <- churnTrain[dataPart ==1,]
testdata <- churnTrain[dataPart ==2,]

#Installing package 'tree' for Decision Tree

#install.packages("tree")

library(tree)
```

```
## Warning: package 'tree' was built under R version 3.3.3
```

```r
#Grow a tree using tree to predict customer churn for all other independent variables

fit <- tree(traindata$churn ~ .,traindata)

#To print detailed summary of splits
summary(fit)
```

```
##
## Classification tree:
## tree(formula = traindata$churn ~ ., data = traindata)
## Variables actually used in tree construction:
## [1] "total_day_minutes"             "number_customer_service_calls"
## [3] "total_intl_calls"              "total_intl_minutes"
## [5] "total_eve_minutes"             "voice_mail_plan"
## [7] "number_vmail_messages"
## Number of terminal nodes:  11
```

```
## Residual mean deviance:  0.501 = 1163 / 2322
## Misclassification error rate: 0.08487 = 198 / 2333
```

```r
cat("\n \n")
```

```r
#Prediction using predict function for both traindata and testdata
traindata_C50 = predict(fit,traindata,type="class")
testdata_C50 = predict(fit,testdata,type="class")

#Printing prediction result using table
print("Prediction of traindata")
```

```
## [1] "Prediction of traindata"
```

```r
cat("_______________________\n")
```

```
## _______________________
```

```r
table(traindata_C50,traindata$churn)
```

```
##
## traindata_C50  yes   no
##           yes  169   26
##           no   172 1966
```

```r
cat("\n")
```

```r
print("Prediction of testdata")
```

```
## [1] "Prediction of testdata"
```

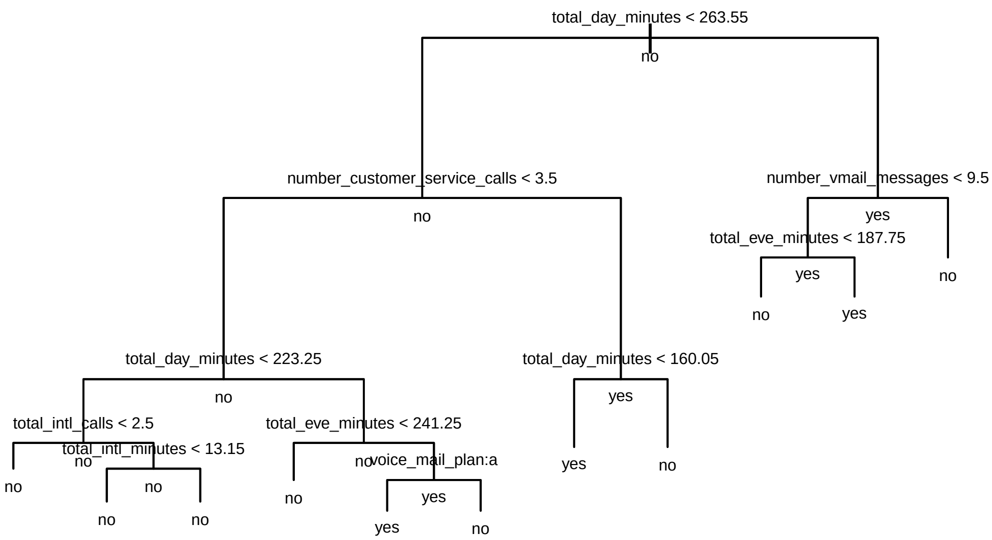
```r
cat("______________________\n")
```

```
## ______________________
```

```r
table(testdata_C50,testdata$churn)
```

```
##
## testdata_C50 yes  no
##          yes  72  16
##          no   70 842
```

```r
#To plot the tree
plot(fit)
text(fit, all = TRUE, cex = 0.5)
```




```r
#Pruning the tree
pfit = cv.tree(fit,FUN = prune.misclass)
print("Details of prune.misclass")
```

```
## [1] "Details of prune.misclass"
```

```r
print("_________________________")
```

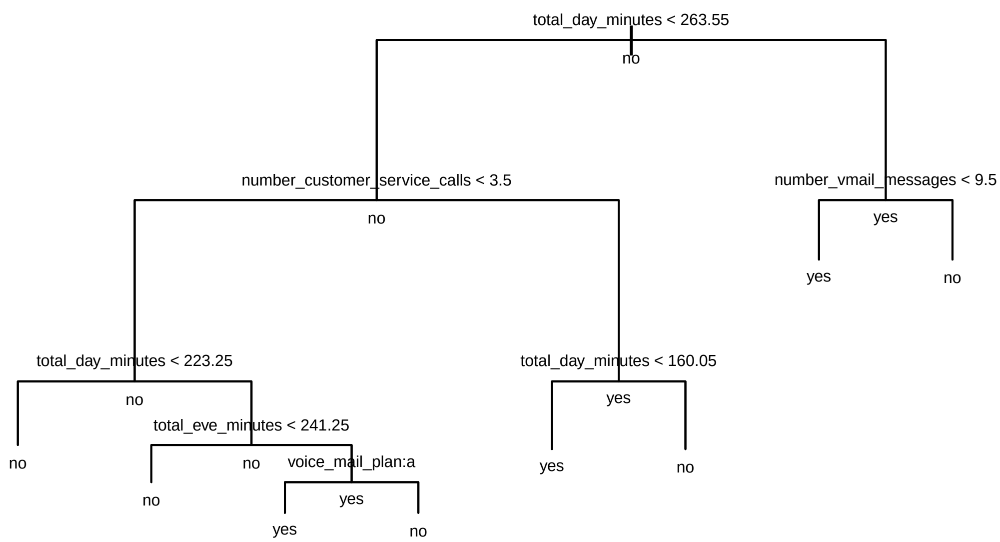
```
## [1] "_________________________"
```

```r
cat("\n")
```

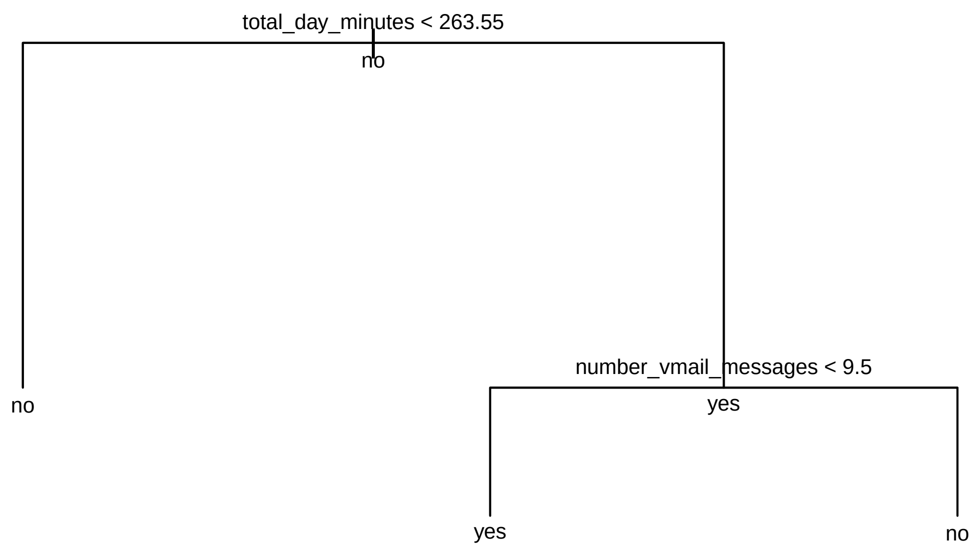
```r
pfit
```

```
## $size
## [1] 11  9  8  5  3  1
##
## $dev
## [1] 232 235 224 238 351 347
##
## $k
## [1]      -Inf  0.000000  4.000000  6.666667 28.500000 31.000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
pruneData = prune.misclass(fit,best=6)
#Plot the pruned tree
plot(pruneData)
text(pruneData,all = TRUE, cex = 0.5)
```




```
pruneData2 = prune.misclass(fit,best=2)
#Plot the pruned tree
plot(pruneData2)
text(pruneData2,all = TRUE, cex = 0.7)
```



Pruning the decision tree will help to avoid overfitting the data. It will minimize the cross validation error (uing xerror) and select complexity parameter that is associated with least error. Results shows that unpruned tree is larger because the algorithm is implemented as is. While in pruned tree, there is an additional step which analyse which nodes or branches to be removed that will not affect the performance of decision tree.

Q5 Compare generalisation performance of the pruned and unpruned tree from Q4. Output relevant summaries and confusion matrices. Describe the results.

```r
print("Summary of pruned tree with best 6")
```

```
## [1] "Summary of pruned tree with best 6"
```

```r
cat("\n")
```

```r
summary(pruneData)
```

```
##
## Classification tree:
## snip.tree(tree = fit, nodes = c(8L, 6L))
## Variables actually used in tree construction:
## [1] "total_day_minutes"             "number_customer_service_calls"
## [3] "total_eve_minutes"             "voice_mail_plan"
## [5] "number_vmail_messages"
## Number of terminal nodes:  8
## Residual mean deviance:  0.5472 = 1272 / 2325
## Misclassification error rate: 0.08658 = 202 / 2333
```

```r
cat("\n___________________________________________________________________")
```

```
##
## ___________________________________________________________________
```

```r
cat("\n")
```

```r
print("Summary of pruned tree with best 2")
```

```
## [1] "Summary of pruned tree with best 2"
```

```r
cat("\n")
```

```r
summary(pruneData2)
```

```
##
## Classification tree:
## snip.tree(tree = fit, nodes = c(6L, 2L))
## Variables actually used in tree construction:
## [1] "total_day_minutes"     "number_vmail_messages"
## Number of terminal nodes:  3
## Residual mean deviance:  0.7281 = 1696 / 2330
## Misclassification error rate: 0.1196 = 279 / 2333
```

```r
cat("\n___________________________________________________________________")
```

```
##
## ___________________________________________________________________
```

```r
cat("\n")
```

```r
print("Summary of unpruned tree")
```

```
## [1] "Summary of unpruned tree"
```

```r
cat("\n")
```

```r
summary(fit)
```

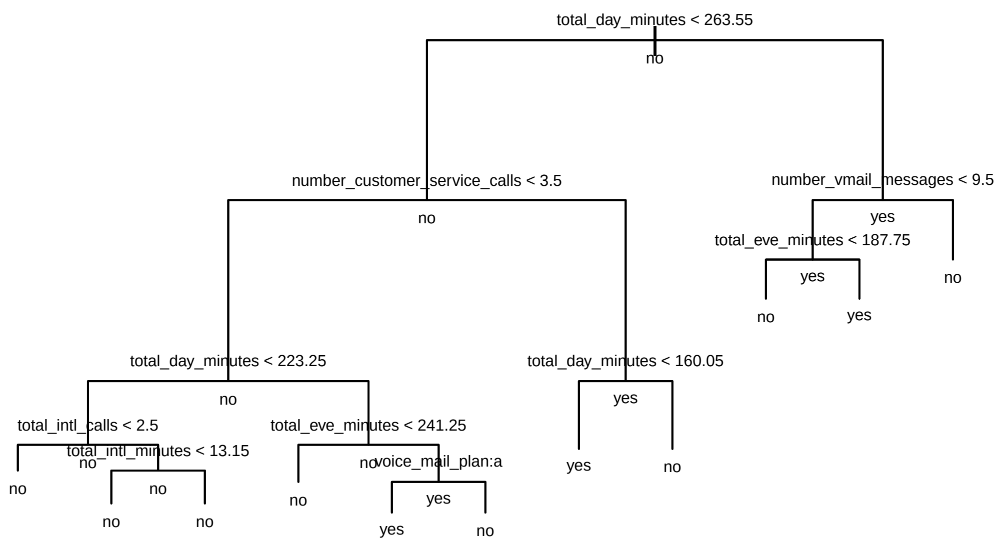
```
##
## Classification tree:
## tree(formula = traindata$churn ~ ., data = traindata)
## Variables actually used in tree construction:
## [1] "total_day_minutes"             "number_customer_service_calls"
## [3] "total_intl_calls"              "total_intl_minutes"
## [5] "total_eve_minutes"             "voice_mail_plan"
## [7] "number_vmail_messages"
## Number of terminal nodes:  11
## Residual mean deviance:  0.501 = 1163 / 2322
## Misclassification error rate: 0.08487 = 198 / 2333
```

Classification on C50 dataset has been done using **tree** package to predict whether the cutomer will churn or not. From the above summary data **Misclassification error rate** for pruned decision tree with best 2 and for pruned decision tree with best 6 are 42% and 12% bigger than unpruned decision tree respectively. Whereas **residual mean deviance** for pruned decision tree with best 2 and pruned decision tree with best 6 are 30% and 43%lower than unpruned decision tree respectively. This suggests that unpruned decision tree classification is better for training data. Also, as pruned decision tree is easy to understand because it has less risk of overfitting data. If we increase the best parameter to 13 then the pruned tree will be similar to that of unpruned classification.

```r
prune13 = prune.misclass(fit, best = 13)
```

```
## Warning in prune.tree(tree = fit, best = 13, method = "misclass"): best is
## bigger than tree size
```

```r
plot(prune13)
text(prune13,all = TRUE, cex = 0.5)
```




```r
summary(prune13)
```

```
##
## Classification tree:
## tree(formula = traindata$churn ~ ., data = traindata)
## Variables actually used in tree construction:
## [1] "total_day_minutes"             "number_customer_service_calls"
## [3] "total_intl_calls"              "total_intl_minutes"
## [5] "total_eve_minutes"             "voice_mail_plan"
## [7] "number_vmail_messages"
## Number of terminal nodes:  11
## Residual mean deviance:  0.501 = 1163 / 2322
## Misclassification error rate: 0.08487 = 198 / 2333
```

Confusion Matrix counts the number of times predicted variable has been mapped to other true variables.

```r
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.3.3
```

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

```
## Loading required package: ggplot2
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```r
print("Confusion Matrix of Unpruned decision tree of training dataset")
```

```
## [1] "Confusion Matrix of Unpruned decision tree of training dataset"
```

```r
cat("\n")
```

```r
p1 = predict(fit,churnTrain,type="class")
unp_table = table(p1,churnTrain$churn)
cat("\n")
```

```r
confusionMatrix(unp_table)
```

```
## Confusion Matrix and Statistics
##
##
## p1      yes   no
##   yes   241   42
##   no    242 2808
##
##                Accuracy : 0.9148
##                  95% CI : (0.9048, 0.9241)
##     No Information Rate : 0.8551
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5848
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.49896
##             Specificity : 0.98526
##          Pos Pred Value : 0.85159
##          Neg Pred Value : 0.92066
##              Prevalence : 0.14491
##          Detection Rate : 0.07231
##    Detection Prevalence : 0.08491
##       Balanced Accuracy : 0.74211
##
##        'Positive' Class : yes
##
```

```r
print("Confusion Matrix of Unpruned decision tree of test dataset")
```

```
## [1] "Confusion Matrix of Unpruned decision tree of test dataset"
```

```r
cat("\n")
```

```r
p2 = predict(fit,churnTest,type="class")
unp2_table = table(p2,churnTest$churn)
cat("\n")
```

```r
confusionMatrix(unp2_table)
```

```
## Confusion Matrix and Statistics
##
##
## p2      yes   no
```

```
##   yes   97   12
##   no   127 1431
##
##                Accuracy : 0.9166
##                  95% CI : (0.9023, 0.9294)
##     No Information Rate : 0.8656
##     P-Value [Acc > NIR] : 5.599e-11
##
##                   Kappa : 0.5423
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.43304
##             Specificity : 0.99168
##          Pos Pred Value : 0.88991
##          Neg Pred Value : 0.91849
##              Prevalence : 0.13437
##          Detection Rate : 0.05819
##    Detection Prevalence : 0.06539
##       Balanced Accuracy : 0.71236
##
##        'Positive' Class : yes
##
```

```r
print("Confusion Matrix of pruned decision tree of training dataset")
```

```
## [1] "Confusion Matrix of pruned decision tree of training dataset"
```

```r
cat("\n")
```

```r
p3 = predict(pruneData,churnTrain,type="class")
p_table = table(p3,churnTrain$churn)
cat("\n")
```

```r
confusionMatrix(p_table)
```

```
## Confusion Matrix and Statistics
##
##
## p3      yes   no
##   yes  267   76
##   no   216 2774
##
##                Accuracy : 0.9124
##                  95% CI : (0.9023, 0.9218)
##     No Information Rate : 0.8551
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5981
##  Mcnemar's Test P-Value : 4.141e-16
##
##             Sensitivity : 0.55280
##             Specificity : 0.97333
##          Pos Pred Value : 0.77843
##          Neg Pred Value : 0.92776
##              Prevalence : 0.14491
##          Detection Rate : 0.08011
```

```
##    Detection Prevalence : 0.10291
##       Balanced Accuracy : 0.76306
##
##        'Positive' Class : yes
##
```

```r
print("Confusion Matrix of pruned decision tree of test dataset")
```

```
## [1] "Confusion Matrix of pruned decision tree of test dataset"
```

```r
cat("\n")
```

```r
p4 = predict(pruneData,churnTest,type="class")
p2_table = table(p4,churnTest$churn)
cat("\n")
```

```r
confusionMatrix(p2_table)
```

```
## Confusion Matrix and Statistics
##
##
## p4     yes   no
##   yes  116   36
##   no   108 1407
##
##                Accuracy : 0.9136
##                  95% CI : (0.8991, 0.9267)
##     No Information Rate : 0.8656
##     P-Value [Acc > NIR] : 7.643e-10
##
##                   Kappa : 0.5703
##  Mcnemar's Test P-Value : 3.285e-09
##
##             Sensitivity : 0.51786
##             Specificity : 0.97505
##          Pos Pred Value : 0.76316
##          Neg Pred Value : 0.92871
##              Prevalence : 0.13437
##          Detection Rate : 0.06959
##    Detection Prevalence : 0.09118
##       Balanced Accuracy : 0.74645
##
##        'Positive' Class : yes
##
```

```r
treeC50=tree(traindata$churn~.,traindata)
Train=predict(treeC50,traindata ,type="class")
Test=predict(treeC50,testdata, type="class")
table(Train,traindata$churn)
```

```
##
## Train  yes   no
##   yes  169   26
##   no   172 1966
```

```r
table(Test,testdata$churn)
```

```
##
```

```
## Test  yes  no
##   yes  72  16
##   no   70 842
```

From the above results we can see that there are 23 misidentified train data for YES and 73 for NO. If data will be more than an accurate model can be created.

Q6 Install R package "caret". Import German credit rating dataset (GermanCredit). Examine the data. Use the data to build a classification model to predict "Good" or "Bad" customer credit rating. Pay attention to the model's generalisation and its' ability to correctly predict both classes. Interpret the results.

```r
#Installing Caret package
#install.packages("caret")

library(caret)
```

```r
#Importing dataset GermanCredit
data("GermanCredit")
print("Summary of German Credit")
```

```
## [1] "Summary of German Credit"
```

```r
cat("\n")
```

```r
str(GermanCredit)
```

```
## 'data.frame':    1000 obs. of  62 variables:
##  $ Duration                              : int  6 48 12 42 24 36 24 36 12 30 ...
##  $ Amount                                : int  1169 5951 2096 7882 4870 9055 2835 6948 3059 5234 ..
##  $ InstallmentRatePercentage             : int  4 2 2 2 3 2 3 2 2 4 ...
##  $ ResidenceDuration                     : int  4 2 3 4 4 4 4 2 4 2 ...
##  $ Age                                   : int  67 22 49 45 53 35 53 35 61 28 ...
##  $ NumberExistingCredits                 : int  2 1 1 1 2 1 1 1 1 2 ...
##  $ NumberPeopleMaintenance               : int  1 1 2 2 2 2 1 1 1 1 ...
##  $ Telephone                             : num  0 1 1 1 1 0 1 0 1 1 ...
##  $ ForeignWorker                         : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Class                                 : Factor w/ 2 levels "Bad","Good": 2 1 2 2 1 2 2 2 2 1 ...
##  $ CheckingAccountStatus.lt.0            : num  1 0 0 1 1 0 0 0 0 0 ...
##  $ CheckingAccountStatus.0.to.200        : num  0 1 0 0 0 0 0 1 0 1 ...
##  $ CheckingAccountStatus.gt.200          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CheckingAccountStatus.none            : num  0 0 1 0 0 1 1 0 1 0 ...
##  $ CreditHistory.NoCredit.AllPaid        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CreditHistory.ThisBank.AllPaid        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ CreditHistory.PaidDuly                : num  0 1 0 1 0 1 1 1 1 0 ...
##  $ CreditHistory.Delay                   : num  0 0 0 0 1 0 0 0 0 0 ...
##  $ CreditHistory.Critical                : num  1 0 1 0 0 0 0 0 0 1 ...
##  $ Purpose.NewCar                        : num  0 0 0 0 1 0 0 0 0 1 ...
##  $ Purpose.UsedCar                       : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ Purpose.Furniture.Equipment           : num  0 0 0 1 0 0 1 0 0 0 ...
##  $ Purpose.Radio.Television              : num  1 1 0 0 0 0 0 0 1 0 ...
##  $ Purpose.DomesticAppliance             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Repairs                       : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ Purpose.Education                    : num  0 0 1 0 0 1 0 0 0 0 ...
##  $ Purpose.Vacation                     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Retraining                   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Business                     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Purpose.Other                        : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ SavingsAccountBonds.lt.100           : num  0 1 1 1 1 0 0 1 0 1 ...
##  $ SavingsAccountBonds.100.to.500       : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ SavingsAccountBonds.500.to.1000      : num  0 0 0 0 0 0 1 0 0 0 ...
##  $ SavingsAccountBonds.gt.1000          : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ SavingsAccountBonds.Unknown          : num  1 0 0 0 0 1 0 0 0 0 ...
##  $ EmploymentDuration.lt.1              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ EmploymentDuration.1.to.4            : num  0 1 0 0 1 1 0 1 0 0 ...
##  $ EmploymentDuration.4.to.7            : num  0 0 1 1 0 0 0 0 1 0 ...
##  $ EmploymentDuration.gt.7              : num  1 0 0 0 0 0 1 0 0 0 ...
##  $ EmploymentDuration.Unemployed        : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ Personal.Male.Divorced.Seperated     : num  0 0 0 0 0 0 0 0 1 0 ...
##  $ Personal.Female.NotSingle            : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ Personal.Male.Single                 : num  1 0 1 1 1 1 1 1 0 0 ...
##  $ Personal.Male.Married.Widowed        : num  0 0 0 0 0 0 0 0 0 1 ...
##  $ Personal.Female.Single               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherDebtorsGuarantors.None          : num  1 1 1 0 1 1 1 1 1 1 ...
##  $ OtherDebtorsGuarantors.CoApplicant   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherDebtorsGuarantors.Guarantor     : num  0 0 0 1 0 0 0 0 0 0 ...
##  $ Property.RealEstate                  : num  1 1 1 0 0 0 0 0 1 0 ...
##  $ Property.Insurance                   : num  0 0 0 1 0 0 1 0 0 0 ...
##  $ Property.CarOther                    : num  0 0 0 0 0 0 0 1 0 1 ...
##  $ Property.Unknown                     : num  0 0 0 0 1 1 0 0 0 0 ...
##  $ OtherInstallmentPlans.Bank           : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherInstallmentPlans.Stores         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ OtherInstallmentPlans.None           : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Housing.Rent                         : num  0 0 0 0 0 0 0 1 0 0 ...
##  $ Housing.Own                          : num  1 1 1 0 0 0 1 0 1 1 ...
##  $ Housing.ForFree                      : num  0 0 0 1 1 1 0 0 0 0 ...
##  $ Job.UnemployedUnskilled              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Job.UnskilledResident                : num  0 0 1 0 0 1 0 0 1 0 ...
##  $ Job.SkilledEmployee                  : num  1 1 0 1 1 0 1 0 0 0 ...
##  $ Job.Management.SelfEmp.HighlyQualified: num  0 0 0 0 0 0 0 1 0 1 ...
```

```r
#Creating Trained and test dataset

set.seed(123)

GermanCredit <- GermanCredit[,!names(GermanCredit) %in% c("Duration","Amount","account_length")]

partition <- sample(2,nrow(GermanCredit),replace = TRUE,prob = c(0.7,0.3))
trainGC <- GermanCredit[partition ==1,]
testGC <- GermanCredit[partition ==2,]

cat("\n \n \n")
```

```r
print("Summary of dataset")
```

```
## [1] "Summary of dataset"
```

```r
cat("\n")
```

```r
summary(GermanCredit)
```

```
##  InstallmentRatePercentage ResidenceDuration      Age
##  Min.   :1.000             Min.   :1.000     Min.   :19.00
##  1st Qu.:2.000             1st Qu.:2.000     1st Qu.:27.00
##  Median :3.000             Median :3.000     Median :33.00
##  Mean   :2.973             Mean   :2.845     Mean   :35.55
##  3rd Qu.:4.000             3rd Qu.:4.000     3rd Qu.:42.00
##  Max.   :4.000             Max.   :4.000     Max.   :75.00
##  NumberExistingCredits NumberPeopleMaintenance   Telephone
##  Min.   :1.000         Min.   :1.000           Min.   :0.000
##  1st Qu.:1.000         1st Qu.:1.000           1st Qu.:0.000
##  Median :1.000         Median :1.000           Median :1.000
##  Mean   :1.407         Mean   :1.155           Mean   :0.596
##  3rd Qu.:2.000         3rd Qu.:1.000           3rd Qu.:1.000
##  Max.   :4.000         Max.   :2.000           Max.   :1.000
##  ForeignWorker    Class     CheckingAccountStatus.lt.0
##  Min.   :0.000   Bad :300   Min.   :0.000
##  1st Qu.:1.000   Good:700   1st Qu.:0.000
##  Median :1.000              Median :0.000
##  Mean   :0.963              Mean   :0.274
##  3rd Qu.:1.000              3rd Qu.:1.000
##  Max.   :1.000              Max.   :1.000
##  CheckingAccountStatus.0.to.200 CheckingAccountStatus.gt.200
##  Min.   :0.000                  Min.   :0.000
##  1st Qu.:0.000                  1st Qu.:0.000
##  Median :0.000                  Median :0.000
##  Mean   :0.269                  Mean   :0.063
##  3rd Qu.:1.000                  3rd Qu.:0.000
##  Max.   :1.000                  Max.   :1.000
##  CheckingAccountStatus.none CreditHistory.NoCredit.AllPaid
##  Min.   :0.000              Min.   :0.00
##  1st Qu.:0.000              1st Qu.:0.00
##  Median :0.000              Median :0.00
##  Mean   :0.394              Mean   :0.04
##  3rd Qu.:1.000              3rd Qu.:0.00
##  Max.   :1.000              Max.   :1.00
##  CreditHistory.ThisBank.AllPaid CreditHistory.PaidDuly CreditHistory.Delay
##  Min.   :0.000                  Min.   :0.00           Min.   :0.000
##  1st Qu.:0.000                  1st Qu.:0.00           1st Qu.:0.000
##  Median :0.000                  Median :1.00           Median :0.000
##  Mean   :0.049                  Mean   :0.53           Mean   :0.088
##  3rd Qu.:0.000                  3rd Qu.:1.00           3rd Qu.:0.000
##  Max.   :1.000                  Max.   :1.00           Max.   :1.000
##  CreditHistory.Critical Purpose.NewCar  Purpose.UsedCar
##  Min.   :0.000          Min.   :0.000   Min.   :0.000
##  1st Qu.:0.000          1st Qu.:0.000   1st Qu.:0.000
##  Median :0.000          Median :0.000   Median :0.000
##  Mean   :0.293          Mean   :0.234   Mean   :0.103
##  3rd Qu.:1.000          3rd Qu.:0.000   3rd Qu.:0.000
##  Max.   :1.000          Max.   :1.000   Max.   :1.000
##  Purpose.Furniture.Equipment Purpose.Radio.Television
```

```
##  Min.   :0.000              Min.   :0.00
##  1st Qu.:0.000              1st Qu.:0.00
##  Median :0.000              Median :0.00
##  Mean   :0.181              Mean   :0.28
##  3rd Qu.:0.000              3rd Qu.:1.00
##  Max.   :1.000              Max.   :1.00
##  Purpose.DomesticAppliance Purpose.Repairs Purpose.Education
##  Min.   :0.000            Min.   :0.000   Min.   :0.00
##  1st Qu.:0.000            1st Qu.:0.000   1st Qu.:0.00
##  Median :0.000            Median :0.000   Median :0.00
##  Mean   :0.012            Mean   :0.022   Mean   :0.05
##  3rd Qu.:0.000            3rd Qu.:0.000   3rd Qu.:0.00
##  Max.   :1.000            Max.   :1.000   Max.   :1.00
##  Purpose.Vacation Purpose.Retraining Purpose.Business Purpose.Other
##  Min.   :0        Min.   :0.000      Min.   :0.000    Min.   :0.000
##  1st Qu.:0        1st Qu.:0.000      1st Qu.:0.000    1st Qu.:0.000
##  Median :0        Median :0.000      Median :0.000    Median :0.000
##  Mean   :0        Mean   :0.009      Mean   :0.097    Mean   :0.012
##  3rd Qu.:0        3rd Qu.:0.000      3rd Qu.:0.000    3rd Qu.:0.000
##  Max.   :0        Max.   :1.000      Max.   :1.000    Max.   :1.000
##  SavingsAccountBonds.lt.100 SavingsAccountBonds.100.to.500
##  Min.   :0.000             Min.   :0.000
##  1st Qu.:0.000             1st Qu.:0.000
##  Median :1.000             Median :0.000
##  Mean   :0.603             Mean   :0.103
##  3rd Qu.:1.000             3rd Qu.:0.000
##  Max.   :1.000             Max.   :1.000
##  SavingsAccountBonds.500.to.1000 SavingsAccountBonds.gt.1000
##  Min.   :0.000                  Min.   :0.000
##  1st Qu.:0.000                  1st Qu.:0.000
##  Median :0.000                  Median :0.000
##  Mean   :0.063                  Mean   :0.048
##  3rd Qu.:0.000                  3rd Qu.:0.000
##  Max.   :1.000                  Max.   :1.000
##  SavingsAccountBonds.Unknown EmploymentDuration.lt.1
##  Min.   :0.000              Min.   :0.000
##  1st Qu.:0.000              1st Qu.:0.000
##  Median :0.000              Median :0.000
##  Mean   :0.183              Mean   :0.172
##  3rd Qu.:0.000              3rd Qu.:0.000
##  Max.   :1.000              Max.   :1.000
##  EmploymentDuration.1.to.4 EmploymentDuration.4.to.7
##  Min.   :0.000            Min.   :0.000
##  1st Qu.:0.000            1st Qu.:0.000
##  Median :0.000            Median :0.000
##  Mean   :0.339            Mean   :0.174
##  3rd Qu.:1.000            3rd Qu.:0.000
##  Max.   :1.000            Max.   :1.000
##  EmploymentDuration.gt.7 EmploymentDuration.Unemployed
##  Min.   :0.000          Min.   :0.000
##  1st Qu.:0.000          1st Qu.:0.000
##  Median :0.000          Median :0.000
##  Mean   :0.253          Mean   :0.062
##  3rd Qu.:1.000          3rd Qu.:0.000
```

```
##  Max.   :1.000           Max.   :1.000
##  Personal.Male.Divorced.Seperated Personal.Female.NotSingle
##  Min.   :0.00                     Min.   :0.00
##  1st Qu.:0.00                     1st Qu.:0.00
##  Median :0.00                     Median :0.00
##  Mean   :0.05                     Mean   :0.31
##  3rd Qu.:0.00                     3rd Qu.:1.00
##  Max.   :1.00                     Max.   :1.00
##  Personal.Male.Single Personal.Male.Married.Widowed Personal.Female.Single
##  Min.   :0.000        Min.   :0.000                 Min.   :0
##  1st Qu.:0.000        1st Qu.:0.000                 1st Qu.:0
##  Median :1.000        Median :0.000                 Median :0
##  Mean   :0.548        Mean   :0.092                 Mean   :0
##  3rd Qu.:1.000        3rd Qu.:0.000                 3rd Qu.:0
##  Max.   :1.000        Max.   :1.000                 Max.   :0
##  OtherDebtorsGuarantors.None OtherDebtorsGuarantors.CoApplicant
##  Min.   :0.000               Min.   :0.000
##  1st Qu.:1.000               1st Qu.:0.000
##  Median :1.000               Median :0.000
##  Mean   :0.907               Mean   :0.041
##  3rd Qu.:1.000               3rd Qu.:0.000
##  Max.   :1.000               Max.   :1.000
##  OtherDebtorsGuarantors.Guarantor Property.RealEstate Property.Insurance
##  Min.   :0.000                    Min.   :0.000       Min.   :0.000
##  1st Qu.:0.000                    1st Qu.:0.000       1st Qu.:0.000
##  Median :0.000                    Median :0.000       Median :0.000
##  Mean   :0.052                    Mean   :0.282       Mean   :0.232
##  3rd Qu.:0.000                    3rd Qu.:1.000       3rd Qu.:0.000
##  Max.   :1.000                    Max.   :1.000       Max.   :1.000
##  Property.CarOther Property.Unknown OtherInstallmentPlans.Bank
##  Min.   :0.000     Min.   :0.000    Min.   :0.000
##  1st Qu.:0.000     1st Qu.:0.000    1st Qu.:0.000
##  Median :0.000     Median :0.000    Median :0.000
##  Mean   :0.332     Mean   :0.154    Mean   :0.139
##  3rd Qu.:1.000     3rd Qu.:0.000    3rd Qu.:0.000
##  Max.   :1.000     Max.   :1.000    Max.   :1.000
##  OtherInstallmentPlans.Stores OtherInstallmentPlans.None  Housing.Rent
##  Min.   :0.000                Min.   :0.000              Min.   :0.000
##  1st Qu.:0.000                1st Qu.:1.000              1st Qu.:0.000
##  Median :0.000                Median :1.000              Median :0.000
##  Mean   :0.047                Mean   :0.814              Mean   :0.179
##  3rd Qu.:0.000                3rd Qu.:1.000              3rd Qu.:0.000
##  Max.   :1.000                Max.   :1.000              Max.   :1.000
##   Housing.Own    Housing.ForFree Job.UnemployedUnskilled
##  Min.   :0.000   Min.   :0.000   Min.   :0.000
##  1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000
##  Median :1.000   Median :0.000   Median :0.000
##  Mean   :0.713   Mean   :0.108   Mean   :0.022
##  3rd Qu.:1.000   3rd Qu.:0.000   3rd Qu.:0.000
##  Max.   :1.000   Max.   :1.000   Max.   :1.000
##  Job.UnskilledResident Job.SkilledEmployee
##  Min.   :0.0           Min.   :0.00
##  1st Qu.:0.0           1st Qu.:0.00
##  Median :0.0           Median :1.00
```

```
##  Mean   :0.2            Mean   :0.63
##  3rd Qu.:0.0            3rd Qu.:1.00
##  Max.   :1.0            Max.   :1.00
##  Job.Management.SelfEmp.HighlyQualified
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.148
##  3rd Qu.:0.000
##  Max.   :1.000
```

```r
print("Printing dimensions of Trained and Test dataset created")
```

```
## [1] "Printing dimensions of Trained and Test dataset created"
```

```r
print("_________________________Train Dataset_________________")
```

```
## [1] "_________________________Train Dataset_________________"
```

```r
dim(trainGC)
```

```
## [1] 705  60
```

```r
print("_________________________Test  Dataset_________________")
```

```
## [1] "_________________________Test  Dataset_________________"
```

```r
dim(testGC)
```

```
## [1] 295  60
```

```r
library(tree)

#Unprunned dataset of GermanCredit

unp_fit = tree(trainGC$Class~.,trainGC)

print("Summary of classification of German Credit")
```

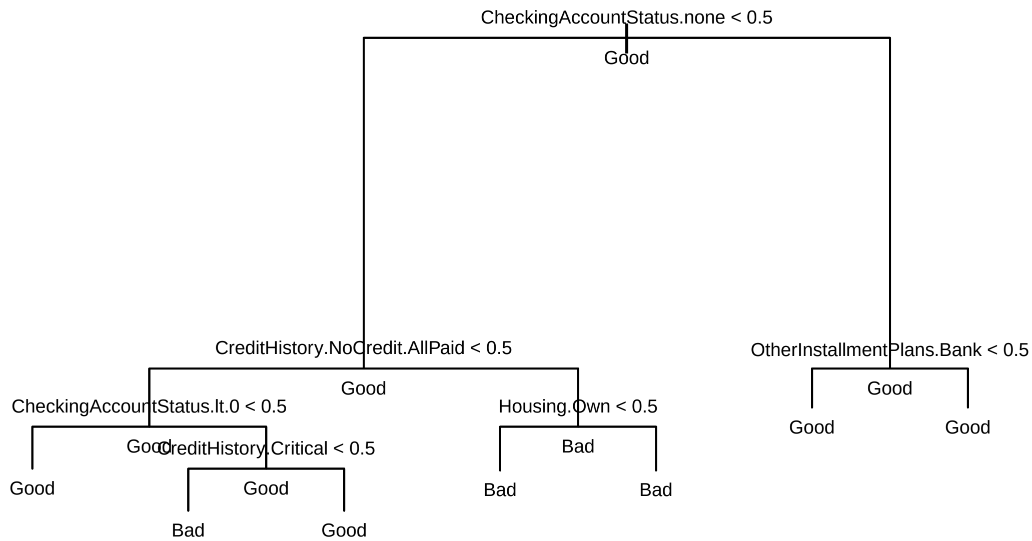
```
## [1] "Summary of classification of German Credit"
```

```r
cat("\n")
```

```r
summary(unp_fit)
```

```
##
## Classification tree:
## tree(formula = trainGC$Class ~ ., data = trainGC)
## Variables actually used in tree construction:
## [1] "CheckingAccountStatus.none"     "CreditHistory.NoCredit.AllPaid"
## [3] "CheckingAccountStatus.lt.0"     "CreditHistory.Critical"
## [5] "Housing.Own"                    "OtherInstallmentPlans.Bank"
## Number of terminal nodes:  7
## Residual mean deviance:  1.017 = 709.7 / 698
## Misclassification error rate: 0.2511 = 177 / 705
```

```r
plot(unp_fit)
text(unp_fit,all=TRUE,cex = 0.6)
```




```r
#Confusion Matrix
pGC = predict(unp_fit,trainGC,type = "class")
p_table = table(pGC, trainGC$Class)
print(p_table)
```

```
##
## pGC    Bad Good
##   Bad   92   62
##   Good 115  436
```

```r
print("Confusion Matrix for trained German Credit dataset")
```

```
## [1] "Confusion Matrix for trained German Credit dataset"
```

```r
cat("\n")
```

```r
confusionMatrix(p_table)
```

```
## Confusion Matrix and Statistics
##
##
## pGC    Bad Good
##   Bad   92   62
##   Good 115  436
##
##                Accuracy : 0.7489
##                  95% CI : (0.7152, 0.7806)
##     No Information Rate : 0.7064
```

```
##     P-Value [Acc > NIR] : 0.006747
##
##                   Kappa : 0.3458
##  Mcnemar's Test P-Value : 9.285e-05
##
##             Sensitivity : 0.4444
##             Specificity : 0.8755
##          Pos Pred Value : 0.5974
##          Neg Pred Value : 0.7913
##              Prevalence : 0.2936
##          Detection Rate : 0.1305
##    Detection Prevalence : 0.2184
##       Balanced Accuracy : 0.6600
##
##        'Positive' Class : Bad
##
```

```r
pGC1 = predict(unp_fit,testGC,type = "class")
p1_table = table(pGC1, testGC$Class)
print(p1_table)
```

```
##
## pGC1   Bad Good
##   Bad   38   35
##   Good  55  167
```

```r
print("Confusion Matrix for test German Credit dataset")
```

```
## [1] "Confusion Matrix for test German Credit dataset"
```

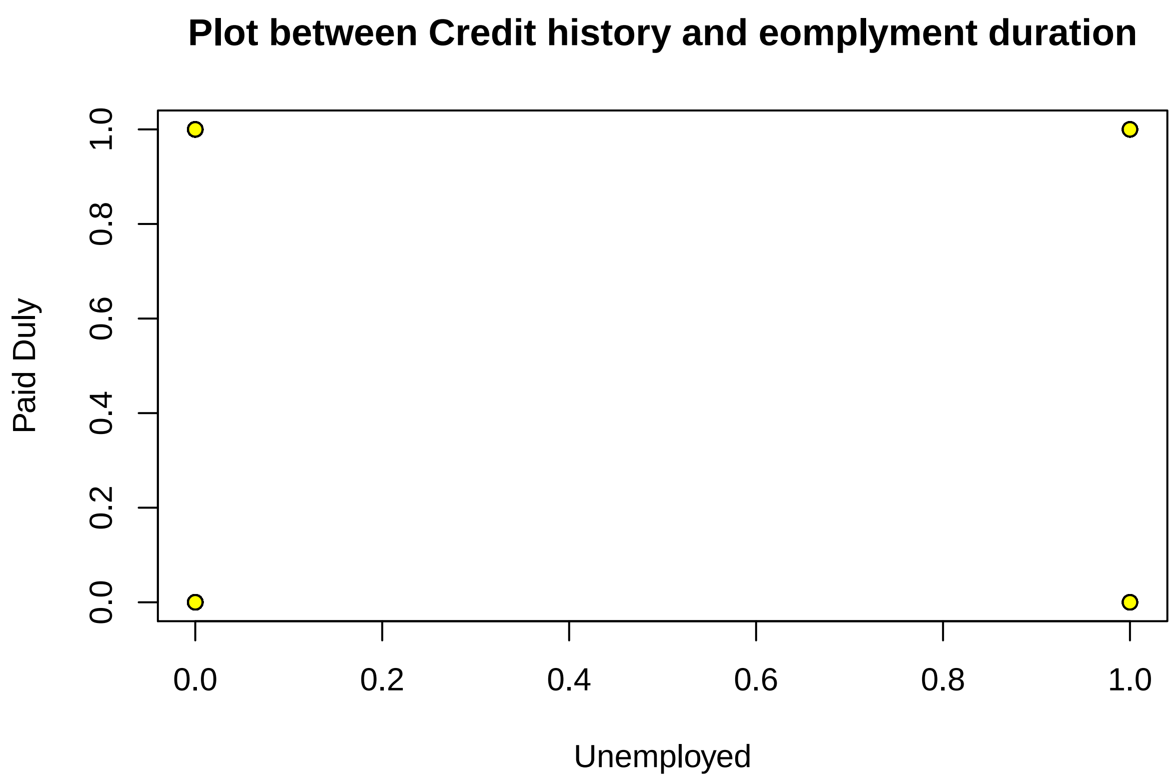
```r
cat("\n")
```

```r
confusionMatrix(p1_table)
```

```
## Confusion Matrix and Statistics
##
##
## pGC1   Bad Good
##   Bad   38   35
##   Good  55  167
##
##                Accuracy : 0.6949
##                  95% CI : (0.6389, 0.747)
##     No Information Rate : 0.6847
##     P-Value [Acc > NIR] : 0.3797
##
##                   Kappa : 0.2498
##  Mcnemar's Test P-Value : 0.0452
##
##             Sensitivity : 0.4086
##             Specificity : 0.8267
##          Pos Pred Value : 0.5205
##          Neg Pred Value : 0.7523
##              Prevalence : 0.3153
##          Detection Rate : 0.1288
##    Detection Prevalence : 0.2475
```

```
##        Balanced Accuracy : 0.6177
##
##         'Positive' Class : Bad
##
```

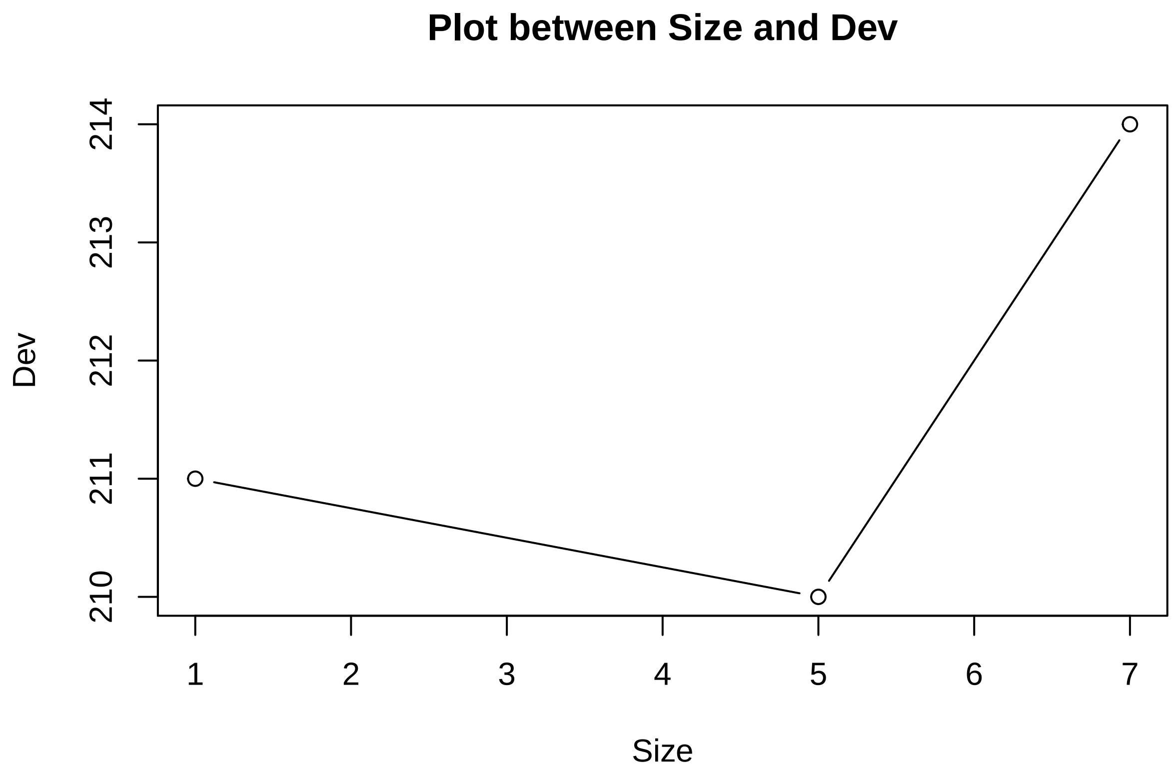
```
plot(trainGC$CreditHistory.PaidDuly~trainGC$EmploymentDuration.Unemployed,pch=21,main="Plot between Cred
```




```
#Pruned tree
cv_German=cv.tree(unp_fit,FUN=prune.misclass)
print(cv_German)
```

```
## $size
## [1] 7 5 1
##
## $dev
## [1] 214 210 211
##
## $k
## [1] -Inf  0.0  7.5
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

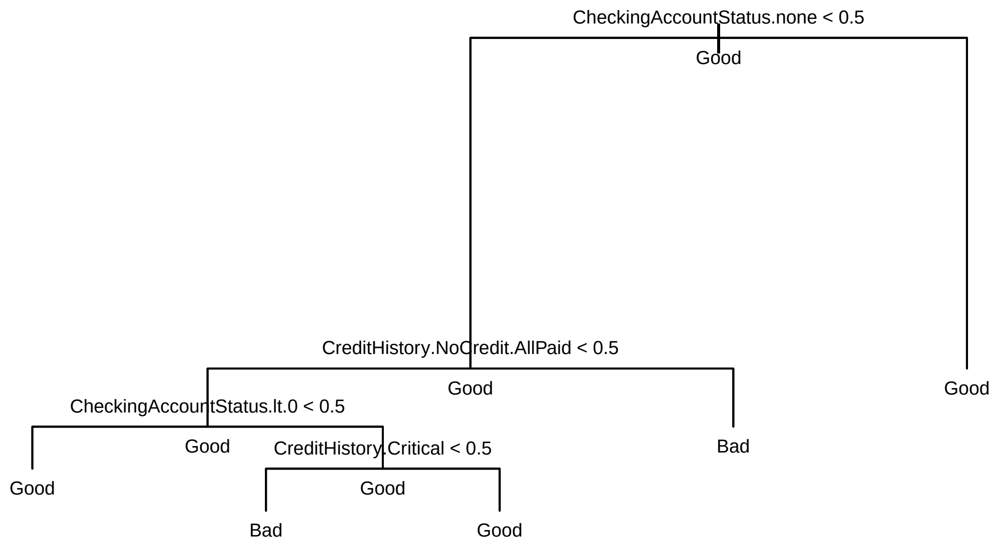
```
 plot(cv_German$size,cv_German$dev,type="b",xlab = "Size",ylab = "Dev",main = "Plot between Size and Dev
```




```r
prune_GC = prune.misclass(unp_fit,best=2)
summary(prune_GC)
```

```
##
## Classification tree:
## snip.tree(tree = unp_fit, nodes = c(3L, 5L))
## Variables actually used in tree construction:
## [1] "CheckingAccountStatus.none"     "CreditHistory.NoCredit.AllPaid"
## [3] "CheckingAccountStatus.lt.0"     "CreditHistory.Critical"
## Number of terminal nodes:  5
## Residual mean deviance:  1.044 = 731.1 / 700
## Misclassification error rate: 0.2511 = 177 / 705
```

```r
plot(prune_GC)
text(prune_GC,all=TRUE,cex = 0.6)
```



As from the summary results of pruned and unpruned decision tree, we can see that Misclassification error rate is same for both the tree types. However, residual mean deviance is larger and unpruned tree is better fit and helps in understanding more than pruned tree.

Q7 Load file college.csv provided on Blackboard. Explore the data. Prepare the data for analysis. Use three different classification algorithms to classify colleges into two classes based on the label ("Not Elite", "Elite") (at least one algorithm of the type decision tree). 1. Describe what you have learned about the dataset and classification results. 2. What classification algorithm(s) did you use, with what parameter settings and how these settings affected the algorithm(s) performance? 3. Exclude the class labels from the data and explore this dataset with clustering. Compare clustering results with results of classification.

```
library(readr)

cdata <- read_csv("G:/R_programs_git/R_Progams/Classification/Classification/college.csv", col_types = c
```

```
## Warning: Missing column names filled in: 'X1' [1]
```

```
View(cdata)
```

Now we will ue descriptive analytics to understand insights about dataset College (cdata).

```
cat("\n \n")
```

```r
print("Summary of dataset")
```

```
## [1] "Summary of dataset"
```

```r
cat("\n")
```

```r
summary(cdata)
```

```
##        X1            name            accept_rate       Outstate    
##  Min.   :  1   Length:777         Min.   :0.1545   Min.   : 2340  
##  1st Qu.:195   Class :character   1st Qu.:0.6756   1st Qu.: 7320  
##  Median :389   Mode  :character   Median :0.7788   Median : 9990  
##  Mean   :389                      Mean   :0.7469   Mean   :10441  
##  3rd Qu.:583                      3rd Qu.:0.8485   3rd Qu.:12925  
##  Max.   :777                      Max.   :1.0000   Max.   :21700  
##      Enroll       Grad.Rate      Private        isElite   
##  Min.   :  35   Min.   : 10.00   Yes:565   Elite    : 78  
##  1st Qu.: 242   1st Qu.: 53.00   No :212   Not Elite:699  
##  Median : 434   Median : 65.00                            
##  Mean   : 780   Mean   : 65.46                            
##  3rd Qu.: 902   3rd Qu.: 78.00                            
##  Max.   :6392   Max.   :118.00                            
```

```r
cat("\n \n")
```

```r
print("Attributes of dataset")
```

```
## [1] "Attributes of dataset"
```

```r
cat("\n")
```

```r
attributes(cdata)
```

```
## $class
## [1] "tbl_df"     "tbl"        "data.frame"
## 
## $row.names
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102
## [103] 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
## [120] 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
## [137] 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153
## [154] 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170
## [171] 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187
## [188] 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204
## [205] 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221
## [222] 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236 237 238
## [239] 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255
## [256] 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272
## [273] 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289
## [290] 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306
## [307] 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323
## [324] 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340
```

```
## [341] 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357
## [358] 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374
## [375] 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391
## [392] 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408
## [409] 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425
## [426] 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442
## [443] 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459
## [460] 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476
## [477] 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493
## [494] 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510
## [511] 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527
## [528] 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544
## [545] 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561
## [562] 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578
## [579] 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595
## [596] 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612
## [613] 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629
## [630] 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646
## [647] 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663
## [664] 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680
## [681] 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697
## [698] 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714
## [715] 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731
## [732] 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748
## [749] 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765
## [766] 766 767 768 769 770 771 772 773 774 775 776 777
##
## $names
## [1] "X1"          "name"        "accept_rate" "Outstate"    "Enroll"
## [6] "Grad.Rate"   "Private"     "isElite"
##
## $spec
## cols(
##   X1 = col_integer(),
##   name = col_character(),
##   accept_rate = col_double(),
##   Outstate = col_integer(),
##   Enroll = col_integer(),
##   Grad.Rate = col_integer(),
##   Private = col_factor(levels = c("Yes", "No"), ordered = FALSE),
##   isElite = col_factor(levels = c("Elite", "Not Elite"), ordered = FALSE)
## )
```

```r
print("Summary of isElite column")
```

```
## [1] "Summary of isElite column"
```

```r
cat("\n")
```

```r
summary(cdata$isElite)
```

```
##     Elite Not Elite
##        78       699
```

From summary table of descriptive analytics we can know the basic statistics such as mean,median, range, etc. We can generate the summary table by two ways as discussed below :

```r
df <- unique(cdata[c(3:6)])
Des_stats <- do.call(data.frame,
                    list(Average = apply(df, 2, mean),
                    Standard_Deviation = apply(df, 2, sd),
                    Median = apply(df, 2, median),
                    Minimum = apply(df, 2, min),
                    Maximum = apply(df, 2, max)))

print(Des_stats)
```

```
##                  Average Standard_Deviation     Median      Minimum
## accept_rate 7.469277e-01          0.1471039    0.77875    0.1544863
## Outstate    1.044067e+04       4023.0164841 9990.00000 2340.0000000
## Enroll      7.799730e+02        929.1761901  434.00000   35.0000000
## Grad.Rate   6.546332e+01         17.1777099   65.00000   10.0000000
##             Maximum
## accept_rate       1
## Outstate      21700
## Enroll         6392
## Grad.Rate       118
```

To see more details we can generate correlation methods so that we can establish the mutual relationship amongst variables and we can forsee the trends among different variables. Correlation plot can be generated by creating scatterplot matrix. There are two methods through which we can generate a scatterplot matrix described as follow:

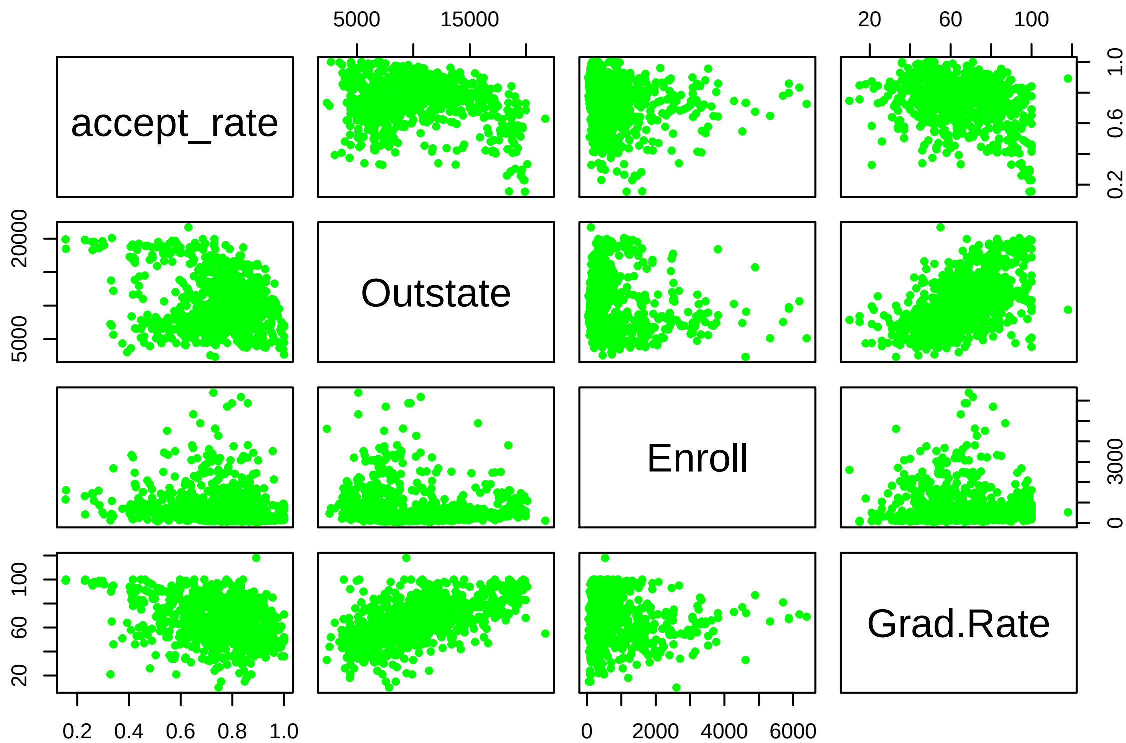
```r
corr = cor(df,use = "complete.obs",method="kendall")
print(round(corr,digits = 3))
```

```
##             accept_rate Outstate Enroll Grad.Rate
## accept_rate       1.000   -0.086 -0.169    -0.158
## Outstate         -0.086    1.000 -0.056     0.420
## Enroll           -0.169   -0.056  1.000     0.065
## Grad.Rate        -0.158    0.420  0.065     1.000
```

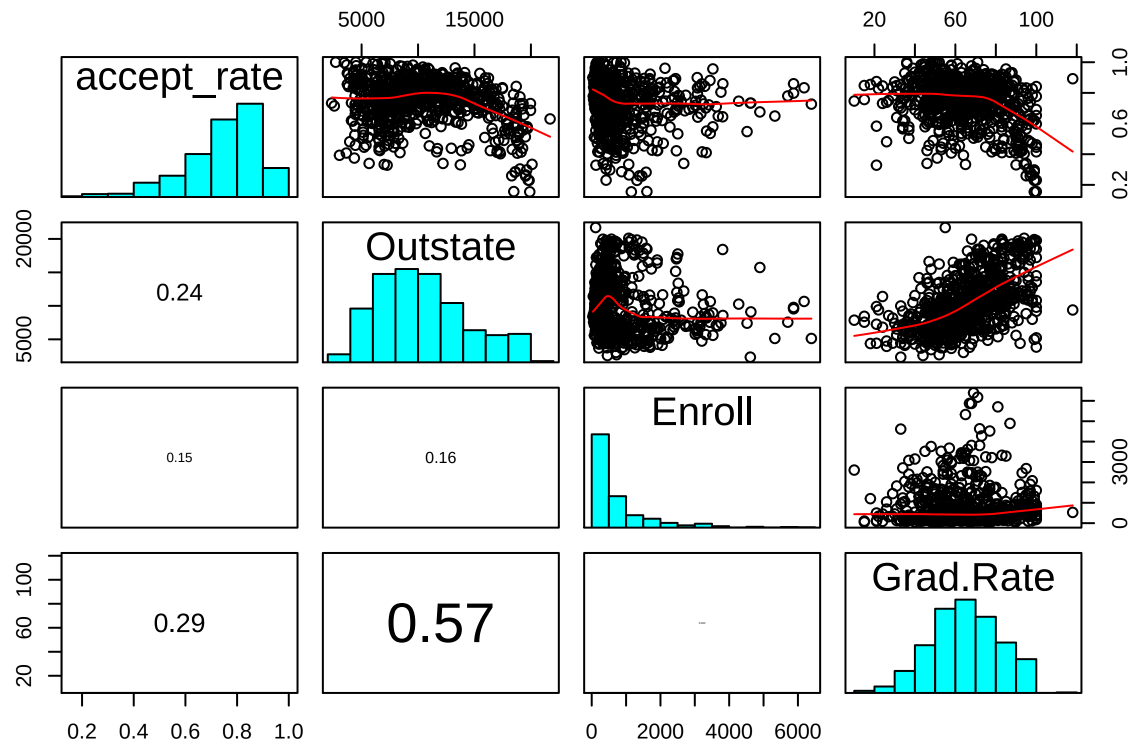
```r
library(s20x)
```

```
## Warning: package 's20x' was built under R version 3.3.2
```

```r
pairs(df,col="green",pch=20)
```




```r
pairs20x(df)
```



Before applying classification algorithm to any dataset, first we need to create reusable partitions.

```
set.seed(8)

set <- cdata[-2]
partitiondata <- sample(2, nrow(set),replace=TRUE,prob=c(0.7,0.3))
trainData <- set[partitiondata ==1,]
testData <- set[partitiondata ==2,]

print("Train Data")
```

```
## [1] "Train Data"
```

```
dim(trainData)
```

```
## [1] 543   7
```

```
cat("\n")
```

```
print("Test Data")
```

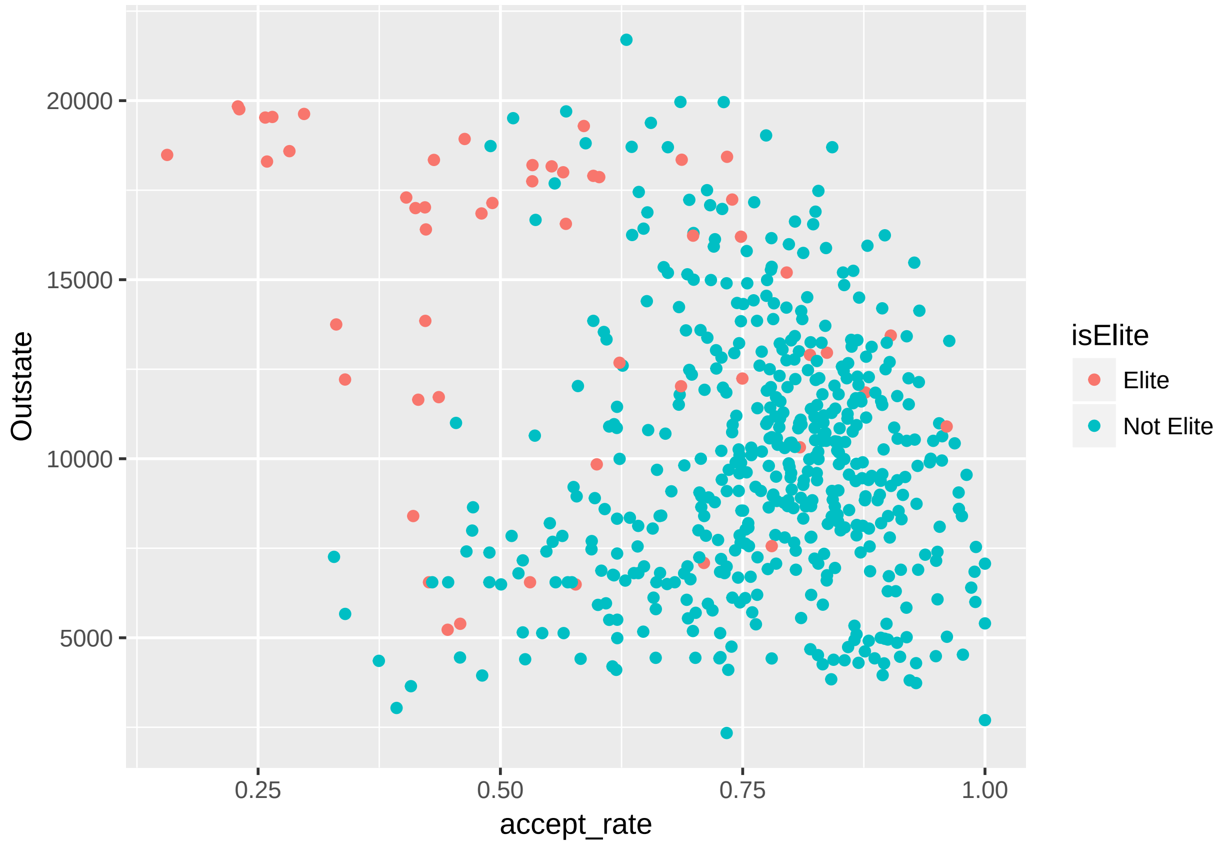
```
## [1] "Test Data"
```

```
dim(testData)
```

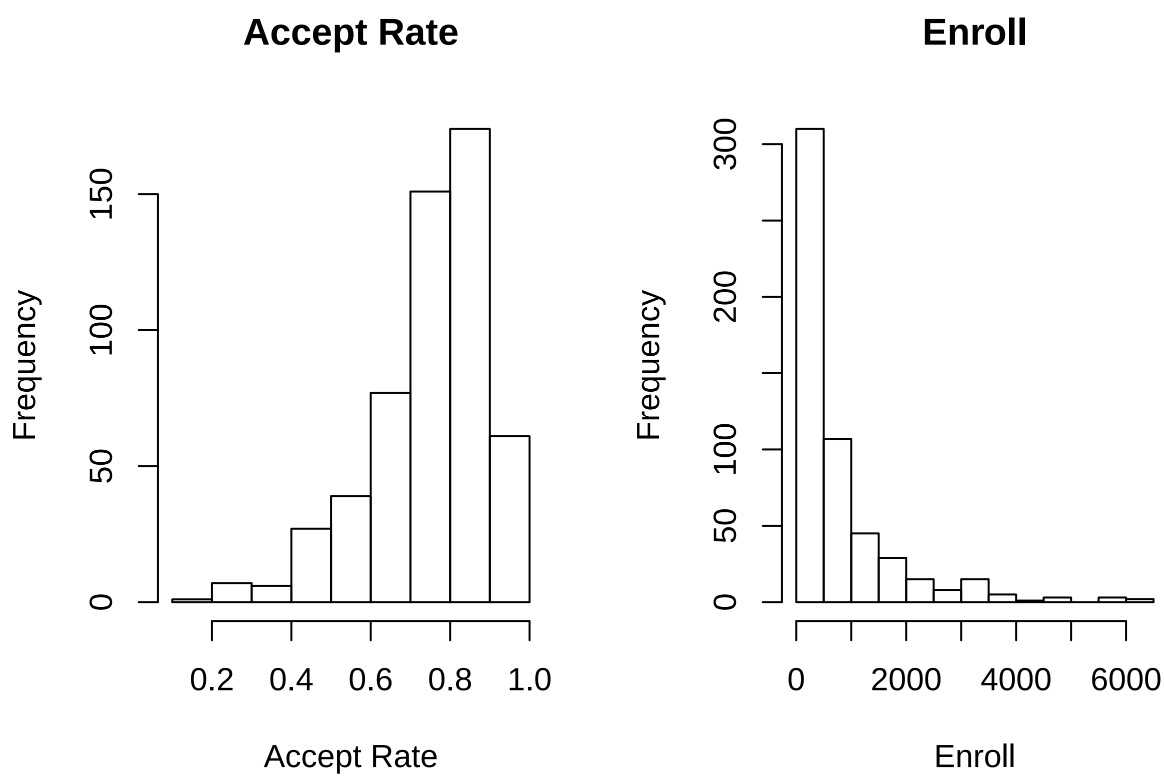
```
## [1] 234   7
```

By using histograms and density curve, we can interpret the distribution of the demographic and competitive variables whether the distribution is normal distribution or not.

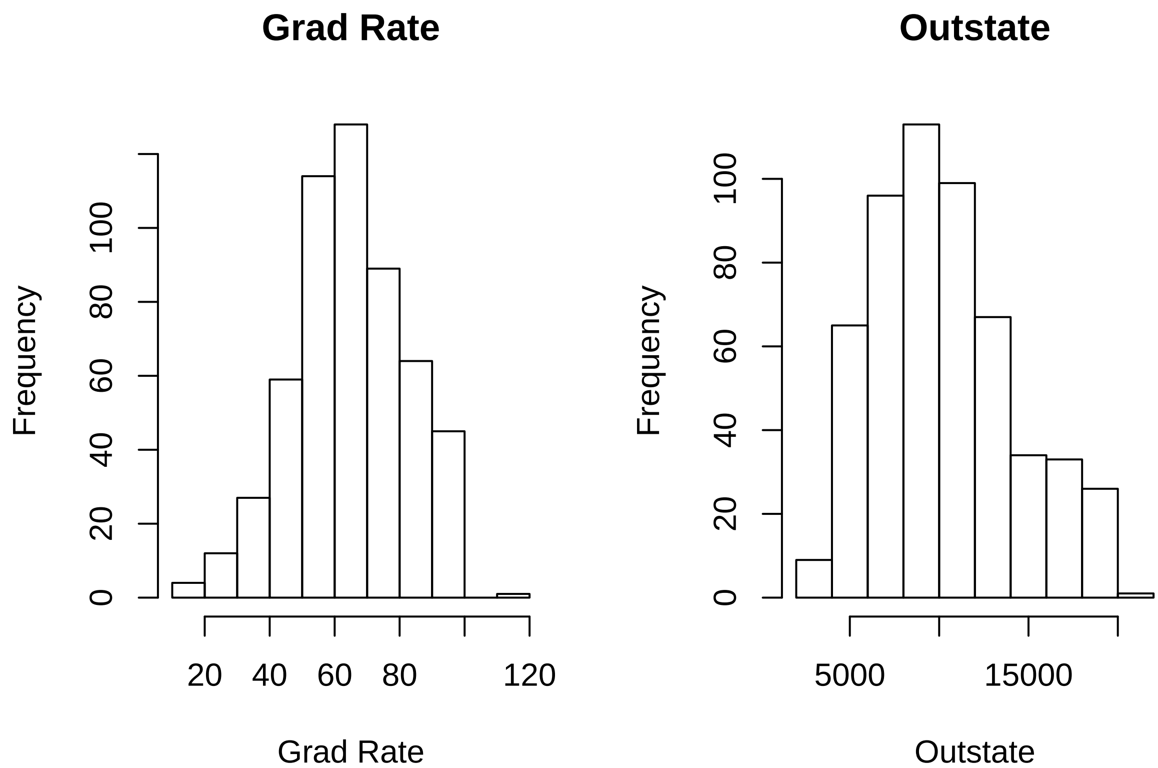
```r
qplot(accept_rate, Outstate, colour=isElite, data=trainData)
```




```r
par(mfrow=c(1,2))

hist(trainData$accept_rate, main = "Accept Rate",xlab = "Accept Rate")
hist(trainData$Enroll, main = "Enroll",xlab = "Enroll")
```




```
hist(trainData$Grad.Rate, main = "Grad Rate",xlab = "Grad Rate")
hist(trainData$Outstate, main="Outstate",xlab = "Outstate")
```



Classification using Tree

```r
library(tree)

tree.trainData = tree(trainData$isElite~.,trainData)
print("Summary of tree using TREE")
```

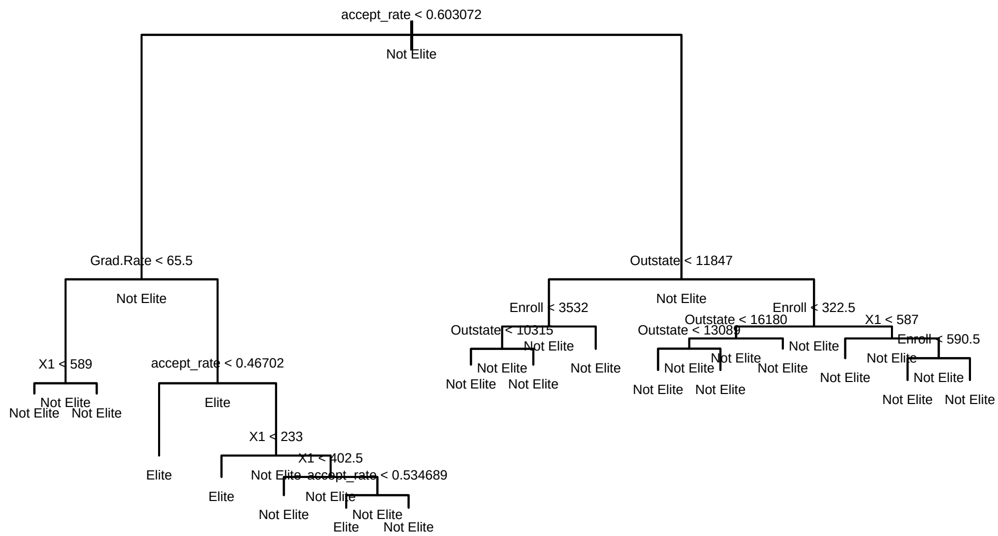
```
## [1] "Summary of tree using TREE"
```

```r
cat("\n")
```

```r
summary(tree.trainData)
```

```
##
## Classification tree:
## tree(formula = trainData$isElite ~ ., data = trainData)
## Variables actually used in tree construction:
## [1] "accept_rate" "Grad.Rate"   "X1"          "Outstate"    "Enroll"
## Number of terminal nodes:  16
## Residual mean deviance:  0.2061 = 108.6 / 527
## Misclassification error rate: 0.04052 = 22 / 543
```

```r
plot(tree.trainData)
text(tree.trainData, all = TRUE, cex = 0.45)
```




```r
#Confusion Matrix of trained data
confusionMatrix(table(predict(tree.trainData,trainData,type = "class"),trainData$isElite))
```

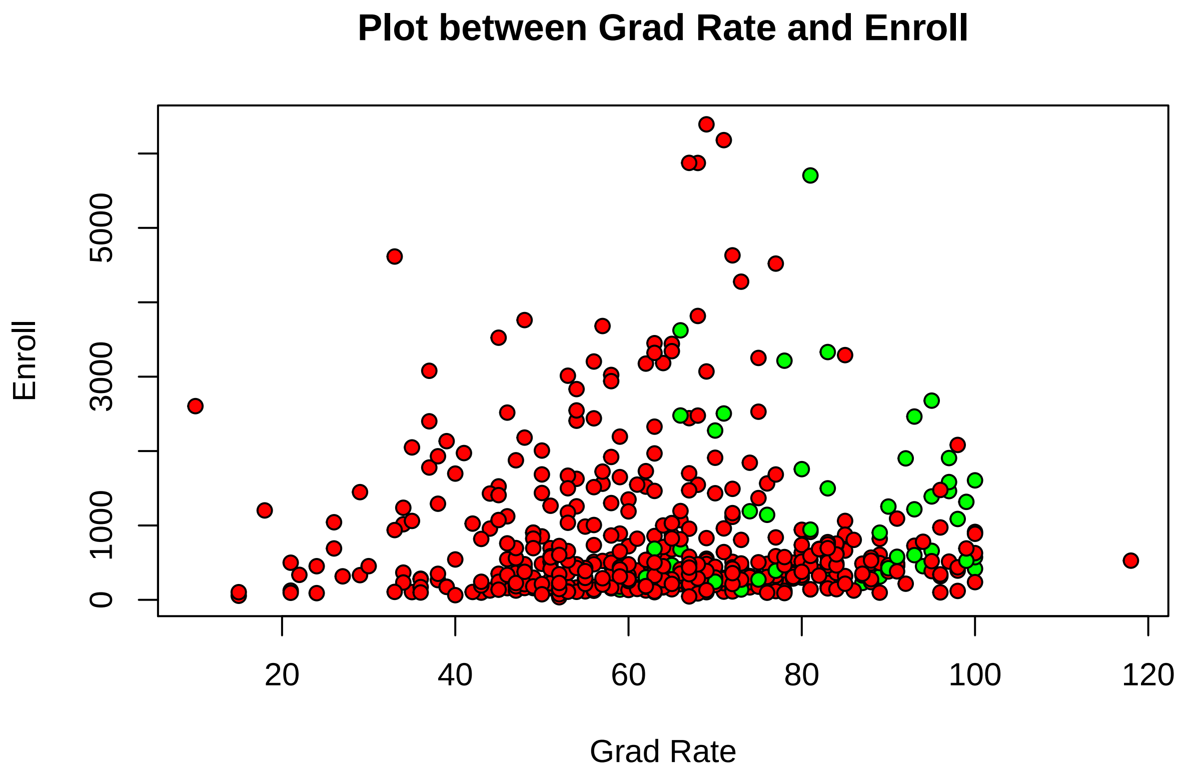
```
## Confusion Matrix and Statistics
##
##
##             Elite Not Elite
##   Elite        35         6
##   Not Elite    18       484
##
##                Accuracy : 0.9558
##                  95% CI : (0.9349, 0.9715)
##     No Information Rate : 0.9024
##     P-Value [Acc > NIR] : 2.811e-06
##
##                   Kappa : 0.7209
##  Mcnemar's Test P-Value : 0.02474
##
##             Sensitivity : 0.66038
##             Specificity : 0.98776
##          Pos Pred Value : 0.85366
##          Neg Pred Value : 0.96414
##              Prevalence : 0.09761
##          Detection Rate : 0.06446
##    Detection Prevalence : 0.07551
##       Balanced Accuracy : 0.82407
```

```
##
##        'Positive' Class : Elite
##
```

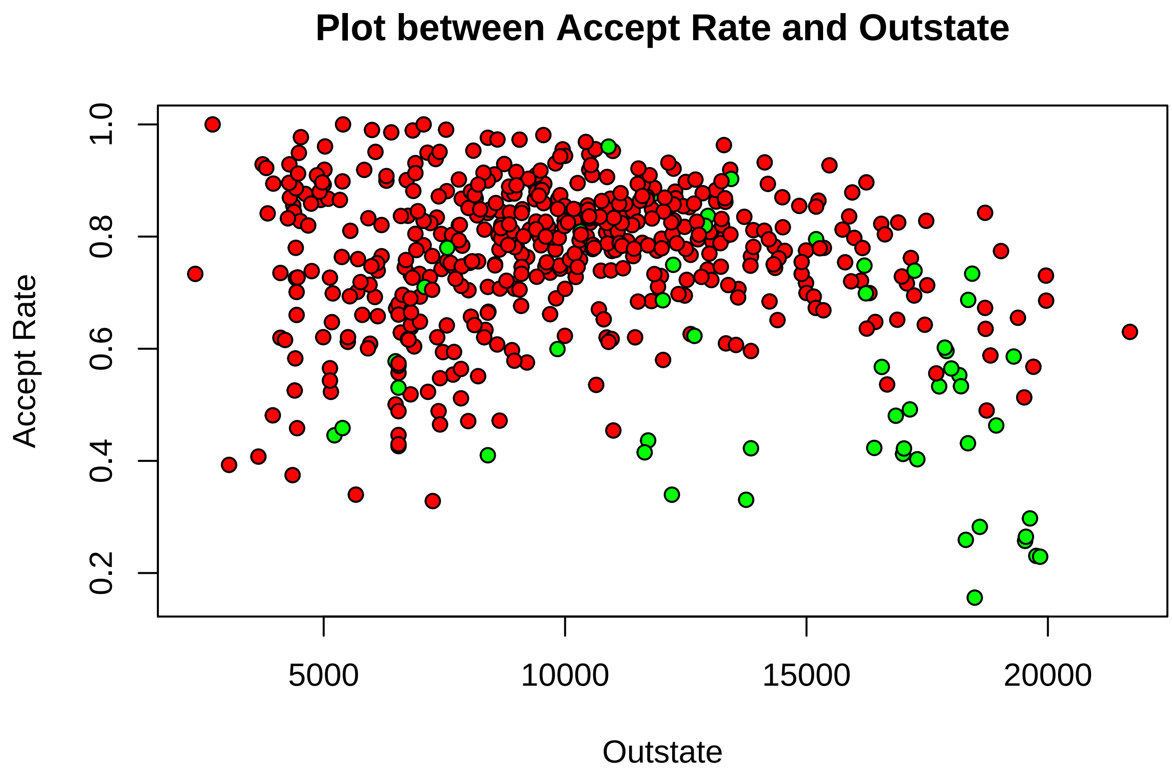
```r
#Confusion Matrix of test data
confusionMatrix(table(predict(tree.trainData,testData,type = "class"),testData$isElite))
```

```
## Confusion Matrix and Statistics
##
##
##             Elite Not Elite
##   Elite        13        10
##   Not Elite    12       199
##
##                Accuracy : 0.906
##                  95% CI : (0.8611, 0.9401)
##     No Information Rate : 0.8932
##     P-Value [Acc > NIR] : 0.3055
##
##                   Kappa : 0.4894
##  Mcnemar's Test P-Value : 0.8312
##
##             Sensitivity : 0.52000
##             Specificity : 0.95215
##          Pos Pred Value : 0.56522
##          Neg Pred Value : 0.94313
##              Prevalence : 0.10684
##          Detection Rate : 0.05556
##    Detection Prevalence : 0.09829
##       Balanced Accuracy : 0.73608
##
##        'Positive' Class : Elite
##
```

```r
plot(trainData$Enroll~trainData$Grad.Rate,pch=21,main="Plot between Grad Rate and Enroll",xlab = "Grad R
```




```
plot(trainData$accept_rate~trainData$Outstate,pch=21,main="Plot between Accept Rate and Outstate",xlab =
```



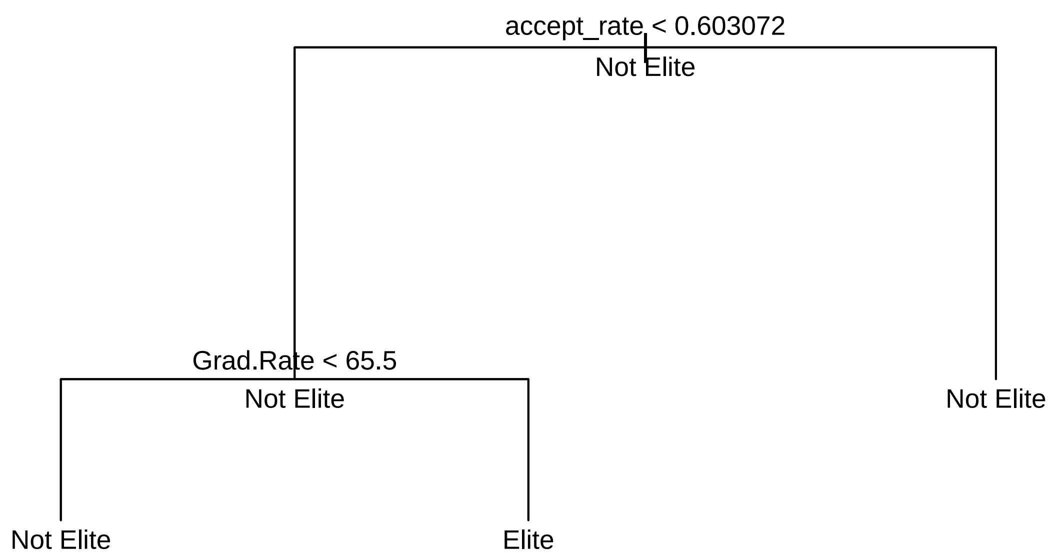

```
#Pruned Data

p.tree = prune.misclass(tree.trainData,best=2)
print("Summary of pruned decision tree with best 2")
```

```
## [1] "Summary of pruned decision tree with best 2"
```

```
summary(p.tree)
```

```
##
## Classification tree:
## snip.tree(tree = tree.trainData, nodes = 3:5)
## Variables actually used in tree construction:
## [1] "accept_rate" "Grad.Rate"
## Number of terminal nodes:  3
## Residual mean deviance:  0.4075 = 220.1 / 540
## Misclassification error rate: 0.06446 = 35 / 543
```

```
plot(p.tree)
text(p.tree, all=TRUE, cex = 0.8)
```



From unpruned and pruned decision tree, we can see that Misclassification error rate is 37% bigger than unpruned tree and residual mean deviance is almost 50% lesser than unpruned tree. So unpruned tree is a better fitment.

```
#Confusion Matrix of train data of pruned tree
confusionMatrix(table(predict(p.tree,trainData,type = "class"),trainData$isElite))
```

```
## Confusion Matrix and Statistics
##
##
##             Elite Not Elite
##   Elite        35        17
##   Not Elite    18       473
##
##                Accuracy : 0.9355
##                  95% CI : (0.9115, 0.9547)
##     No Information Rate : 0.9024
##     P-Value [Acc > NIR] : 0.00396
##
##                   Kappa : 0.631
##  Mcnemar's Test P-Value : 1.00000
##
##             Sensitivity : 0.66038
##             Specificity : 0.96531
##          Pos Pred Value : 0.67308
##          Neg Pred Value : 0.96334
##              Prevalence : 0.09761
```

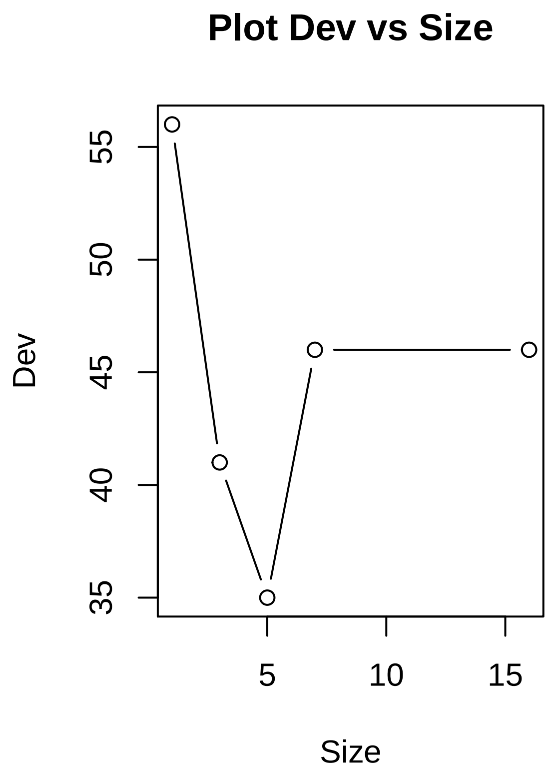
```
##          Detection Rate : 0.06446
##    Detection Prevalence : 0.09576
##       Balanced Accuracy : 0.81284
##
##        'Positive' Class : Elite
##
```

```r
#Confusion Matrix of test data of pruned tree
confusionMatrix(table(predict(p.tree,testData,type = "class"),testData$isElite))
```

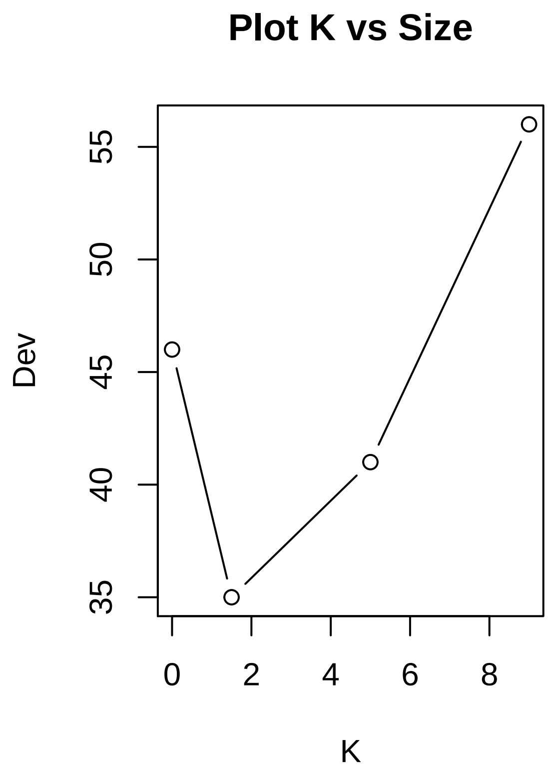
```
## Confusion Matrix and Statistics
##
##
##              Elite Not Elite
##   Elite         15        11
##   Not Elite     10       198
##
##                Accuracy : 0.9103
##                  95% CI : (0.8661, 0.9436)
##     No Information Rate : 0.8932
##     P-Value [Acc > NIR] : 0.2333
##
##                   Kappa : 0.5379
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.6000
##             Specificity : 0.9474
##          Pos Pred Value : 0.5769
##          Neg Pred Value : 0.9519
##              Prevalence : 0.1068
##          Detection Rate : 0.0641
##    Detection Prevalence : 0.1111
##       Balanced Accuracy : 0.7737
##
##        'Positive' Class : Elite
##
```

From statistics of pruned decision tree, we can see that there are 11 misidentified elements in test data for Elite and 10 for Not Elite.

```r
cv1=cv.tree(tree.trainData,FUN=prune.misclass)
par(mfrow=c(1,2))
plot(cv1$size,cv1$dev,type="b",xlab = "Size",ylab = "Dev",main = "Plot Dev vs Size")

par(mfrow=c(1,2))
```




```r
plot(cv1$k,cv1$dev,type="b",xlab = "K",ylab = "Dev",main = "Plot K vs Size")
```



Random Forest Classification

```
library(randomForest)
```

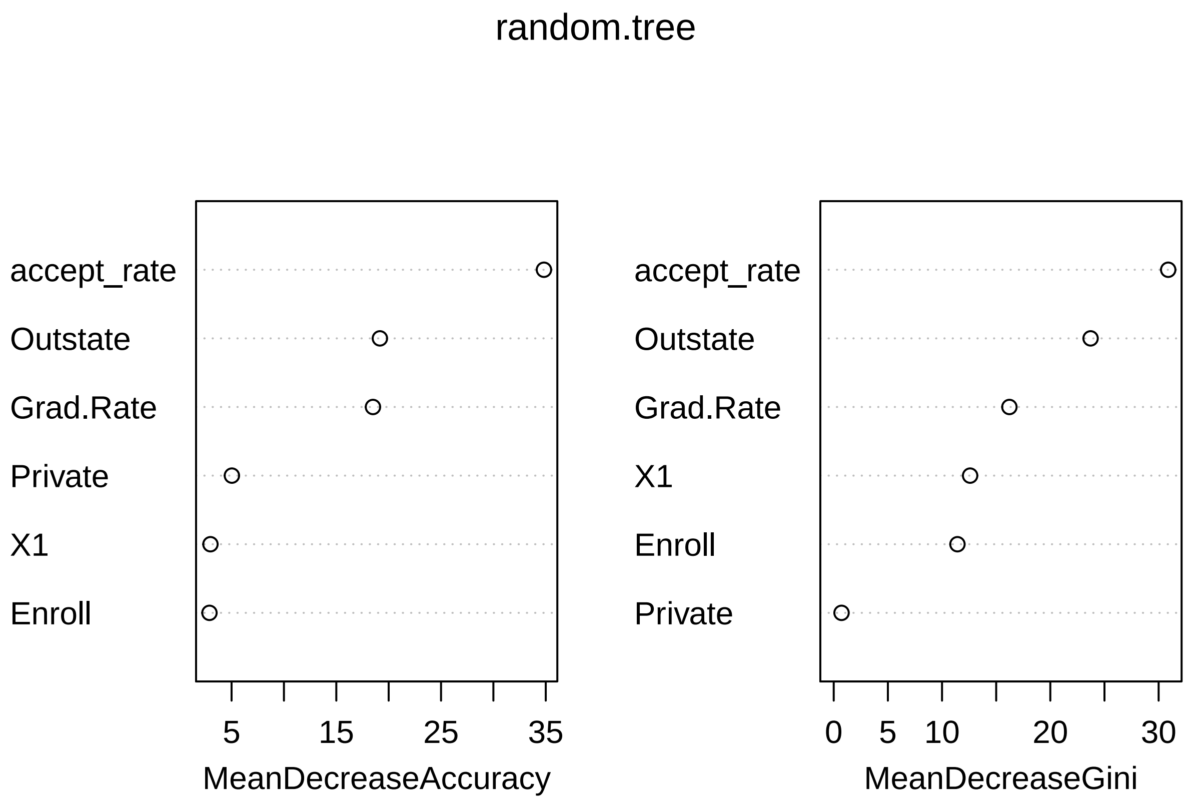
```
## Warning: package 'randomForest' was built under R version 3.3.3
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
random.tree <- randomForest(isElite~.,data=trainData, importance = TRUE,tree=2500)
print(random.tree)
```

```
##
## Call:
##  randomForest(formula = isElite ~ ., data = trainData, importance = TRUE,      tree = 2500)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 6.26%
## Confusion matrix:
##           Elite Not Elite class.error
```

```
## Elite         25         28    0.5283019
## Not Elite      6        484    0.0122449
```

```r
varImpPlot(random.tree)
```




```r
pdata <- predict(random.tree, testData)
print("Summary of prediction data")
```

```
## [1] "Summary of prediction data"
```

```r
summary(pdata)
```

```
##     Elite Not Elite
##        16       218
```

```r
cat("\n")
```

```r
print("Summary of isElite")
```

```
## [1] "Summary of isElite"
```

```r
summary(testData$isElite)
```

```
##     Elite Not Elite
##        25       209
```

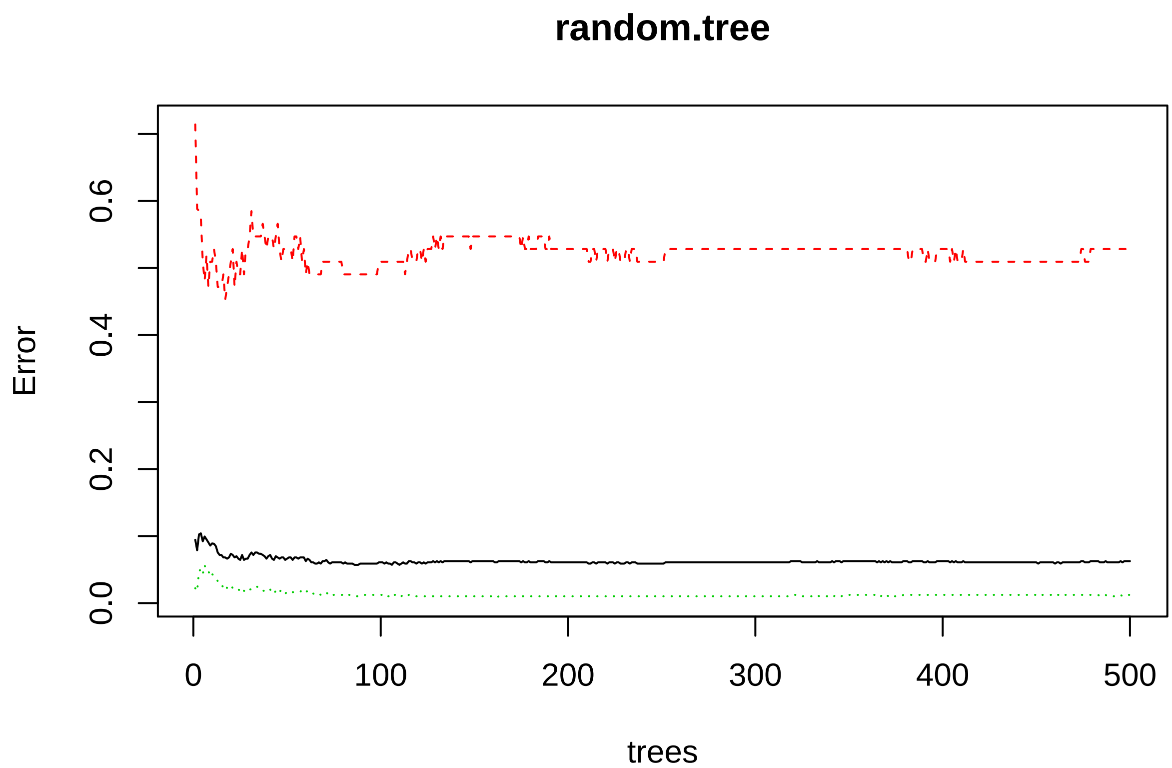
```r
cat("\n\n")
```

```r
confusionMatrix(table(pdata,testData$isElite))
```

```
## Confusion Matrix and Statistics
```

```
##
##
## pdata       Elite Not Elite
##   Elite        12         4
##   Not Elite    13       205
##
##                Accuracy : 0.9274
##                  95% CI : (0.8862, 0.9571)
##     No Information Rate : 0.8932
##     P-Value [Acc > NIR] : 0.05071
##
##                   Kappa : 0.5476
##  Mcnemar's Test P-Value : 0.05235
##
##             Sensitivity : 0.48000
##             Specificity : 0.98086
##          Pos Pred Value : 0.75000
##          Neg Pred Value : 0.94037
##              Prevalence : 0.10684
##          Detection Rate : 0.05128
##    Detection Prevalence : 0.06838
##       Balanced Accuracy : 0.73043
##
##        'Positive' Class : Elite
##
```

```r
plot(random.tree)
```




```r
importance(random.tree)
```

```
##                  Elite Not Elite MeanDecreaseAccuracy MeanDecreaseGini
## X1           3.2537662  1.793129             2.982418       12.5945145
## accept_rate 37.2989900 18.717809            34.827368       30.8842228
## Outstate    16.2683890 14.386806            19.161603       23.7163831
## Enroll       3.8802610  1.106227             2.885866       11.4239079
## Grad.Rate   18.7756613  9.998622            18.499139       16.2209855
## Private      0.3182931  4.760088             5.028227        0.7237341
```

```r
library(party)
```

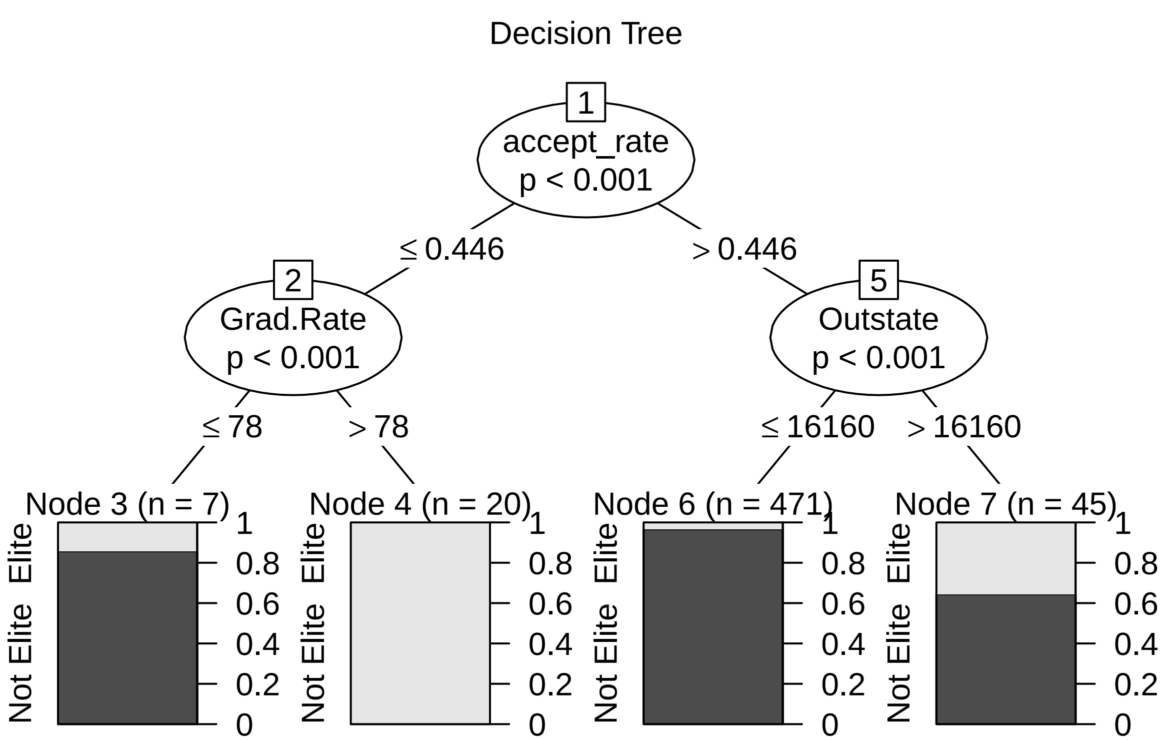
```
## Warning: package 'party' was built under R version 3.3.3

## Loading required package: grid

## Loading required package: mvtnorm

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

## Loading required package: modeltools

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

## Loading required package: stats4

## Loading required package: strucchange

## Warning: package 'strucchange' was built under R version 3.3.3

## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 3.3.2

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

## Loading required package: sandwich

## Warning: package 'sandwich' was built under R version 3.3.2
```

```
set.seed(1234)
r.tree <- cforest(as.factor(isElite) ~ .,data=trainData, controls=cforest_unbiased(ntree=200,mtry=3))

prediction <- predict(r.tree,testData,OOB = TRUE,type="response")
summary(prediction)
```

```
##     Elite Not Elite
##        15       219
```

```
temp <- ctree(trainData$isElite~.,data=trainData)
plot(temp,main = "Decision Tree",cex = 0.5)
```



Classification using Regression

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.3.3
```

```r
set.seed(145)
rp.tree <- rpart(isElite~.,data=trainData)

print("Summary of rpart decision tree")
```

```
## [1] "Summary of rpart decision tree"
```

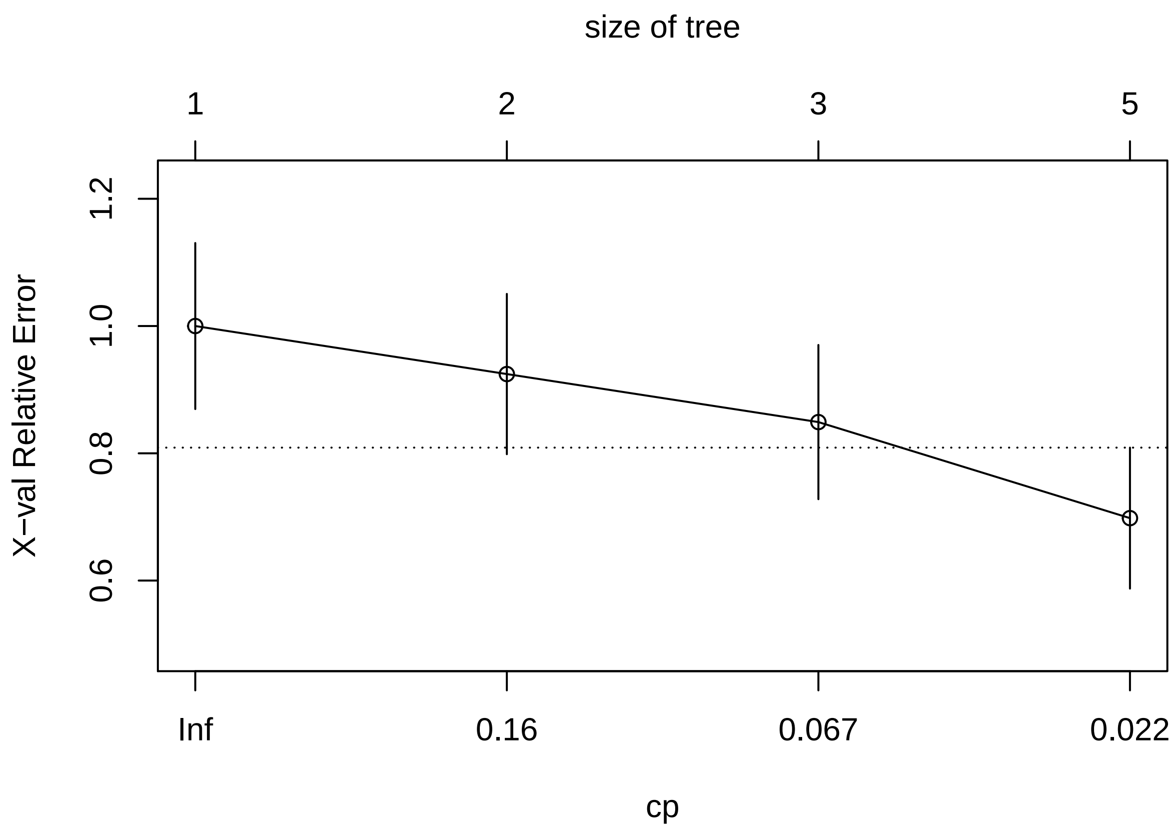
```r
cat("\n\n")
```

```r
summary(rp.tree)
```

```
## Call:
## rpart(formula = isElite ~ ., data = trainData)
##   n= 543
## 
##           CP nsplit rel error    xerror      xstd
## 1 0.28301887      0 1.0000000 1.0000000 0.1304849
## 2 0.09433962      1 0.7169811 0.9245283 0.1259754
## 3 0.04716981      2 0.6226415 0.8490566 0.1212119
## 4 0.01000000      4 0.5283019 0.6981132 0.1107899
## 
## Variable importance
## accept_rate    Outstate   Grad.Rate      Enroll     Private 
##          53          25          16           3           2 
## 
## Node number 1: 543 observations,    complexity param=0.2830189
##   predicted class=Not Elite  expected loss=0.09760589  P(node) =1
##     class counts:    53   490
##    probabilities: 0.098 0.902 
##   left son=2 (27 obs) right son=3 (516 obs)
##   Primary splits:
##       accept_rate < 0.4458624 to the left,  improve=26.289430, (0 missing)
##       Outstate    < 16180     to the right, improve=21.029030, (0 missing)
##       Grad.Rate   < 88.5      to the right, improve=13.679000, (0 missing)
##       Enroll      < 1080      to the right, improve= 3.059383, (0 missing)
##       X1          < 590       to the right, improve= 1.357474, (0 missing)
##   Surrogate splits:
##       Outstate < 19519     to the right, agree=0.952, adj=0.037, (0 split)
## 
## Node number 2: 27 observations,    complexity param=0.09433962
##   predicted class=Elite      expected loss=0.2222222  P(node) =0.04972376
##     class counts:    21     6
##    probabilities: 0.778 0.222 
##   left son=4 (20 obs) right son=5 (7 obs)
##   Primary splits:
##       Grad.Rate < 79        to the right, improve=7.619048, (0 missing)
##       Outstate  < 7830      to the right, improve=6.333333, (0 missing)
##       Enroll    < 884       to the right, improve=2.871795, (0 missing)
##       X1        < 258       to the left,  improve=1.568627, (0 missing)
##       Private   splits as  LR, improve=1.333333, (0 missing)
##   Surrogate splits:
##       Outstate < 7830      to the right, agree=0.889, adj=0.571, (0 split)
##       Enroll   < 376.5     to the right, agree=0.778, adj=0.143, (0 split)
##       Private  splits as  LR, agree=0.778, adj=0.143, (0 split)
## 
```

```
## Node number 3: 516 observations,    complexity param=0.04716981
##   predicted class=Not Elite  expected loss=0.0620155  P(node) =0.9502762
##     class counts:    32   484
##    probabilities: 0.062 0.938
##   left son=6 (45 obs) right son=7 (471 obs)
##   Primary splits:
##       Outstate    < 16180     to the right, improve=8.4958340, (0 missing)
##       accept_rate < 0.6030717 to the left,  improve=5.4666220, (0 missing)
##       Grad.Rate   < 64.5      to the right, improve=2.2325580, (0 missing)
##       Enroll      < 587.5     to the right, improve=0.5408290, (0 missing)
##       X1          < 407.5     to the right, improve=0.5126699, (0 missing)
##
## Node number 4: 20 observations
##   predicted class=Elite      expected loss=0  P(node) =0.03683241
##     class counts:    20     0
##    probabilities: 1.000 0.000
##
## Node number 5: 7 observations
##   predicted class=Not Elite  expected loss=0.1428571  P(node) =0.01289134
##     class counts:     1     6
##    probabilities: 0.143 0.857
##
## Node number 6: 45 observations,    complexity param=0.04716981
##   predicted class=Not Elite  expected loss=0.3555556  P(node) =0.08287293
##     class counts:    16    29
##    probabilities: 0.356 0.644
##   left son=12 (17 obs) right son=13 (28 obs)
##   Primary splits:
##       accept_rate < 0.6160172 to the left,  improve=4.643231, (0 missing)
##       Enroll      < 625       to the right, improve=2.468376, (0 missing)
##       Outstate    < 18566     to the left,  improve=1.838812, (0 missing)
##       X1          < 404.5     to the right, improve=1.251852, (0 missing)
##       Grad.Rate   < 88.5      to the right, improve=1.050030, (0 missing)
##   Surrogate splits:
##       Grad.Rate < 87.5      to the right, agree=0.756, adj=0.353, (0 split)
##       Outstate  < 17594     to the right, agree=0.689, adj=0.176, (0 split)
##       Enroll    < 1495.5    to the right, agree=0.667, adj=0.118, (0 split)
##       X1        < 561       to the right, agree=0.644, adj=0.059, (0 split)
##
## Node number 7: 471 observations
##   predicted class=Not Elite  expected loss=0.03397028  P(node) =0.8674033
##     class counts:    16   455
##    probabilities: 0.034 0.966
##
## Node number 12: 17 observations
##   predicted class=Elite      expected loss=0.3529412  P(node) =0.03130755
##     class counts:    11     6
##    probabilities: 0.647 0.353
##
## Node number 13: 28 observations
##   predicted class=Not Elite  expected loss=0.1785714  P(node) =0.05156538
##     class counts:     5    23
##    probabilities: 0.179 0.821
```

```r
plotcp(rp.tree)
```

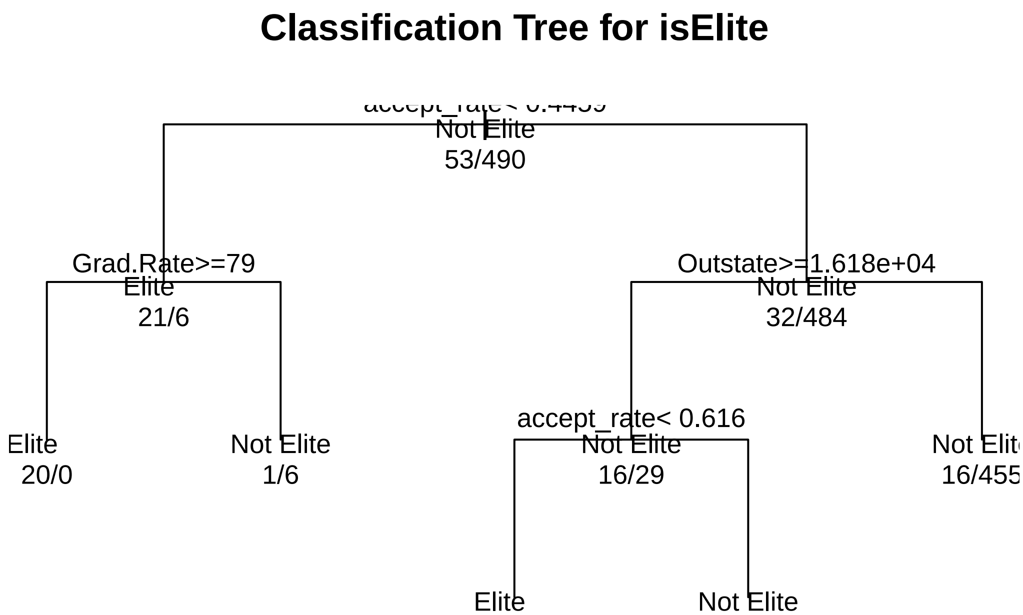



```r
cat("\n")
```

```r
printcp(rp.tree)
```

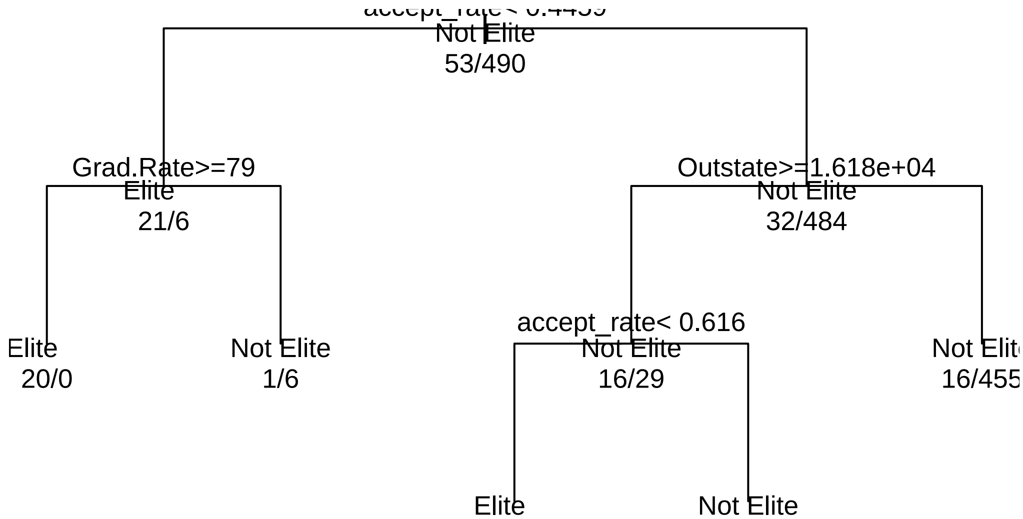
```
##
## Classification tree:
## rpart(formula = isElite ~ ., data = trainData)
##
## Variables actually used in tree construction:
## [1] accept_rate Grad.Rate   Outstate
##
## Root node error: 53/543 = 0.097606
##
## n= 543
##
##        CP nsplit rel error  xerror    xstd
## 1 0.28302      0   1.00000 1.00000 0.13048
## 2 0.09434      1   0.71698 0.92453 0.12598
## 3 0.04717      2   0.62264 0.84906 0.12121
## 4 0.01000      4   0.52830 0.69811 0.11079
```

```r
plot(rp.tree, uniform=TRUE, main="Classification Tree for isElite")
text(rp.tree, use.n=TRUE, all=TRUE, cex=.8)
```



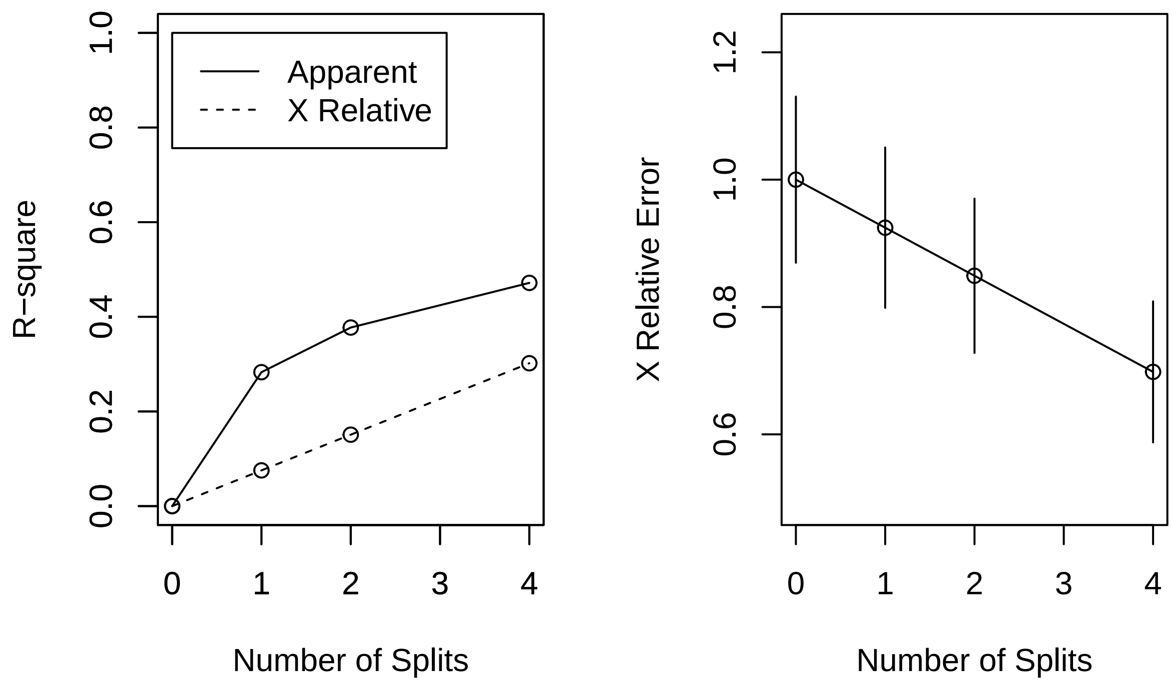

```r
#Pruning the tree
prp.tree <- prune(rp.tree, cp=rp.tree$cptable[which.min(rp.tree$cptable[,"xerror"]),"CP"])

plot(prp.tree, uniform=TRUE, main="Pruned Classification Tree for isElite")
text(prp.tree, use.n=TRUE, all=TRUE, cex=.8)
```

Pruned Classification Tree for isElite




```r
par(mfrow=c(1,2))
rsq.rpart(rp.tree)
```

```
##
## Classification tree:
## rpart(formula = isElite ~ ., data = trainData)
##
## Variables actually used in tree construction:
## [1] accept_rate Grad.Rate   Outstate
##
## Root node error: 53/543 = 0.097606
##
## n= 543
##
##         CP nsplit rel error  xerror    xstd
## 1 0.28302      0   1.00000 1.00000 0.13048
## 2 0.09434      1   0.71698 0.92453 0.12598
## 3 0.04717      2   0.62264 0.84906 0.12121
## 4 0.01000      4   0.52830 0.69811 0.11079

## Warning in rsq.rpart(rp.tree): may not be applicable for this method
```



From all three algorithms used for classfication on college dataset, we can determine that random forest classification has highest accuracy compared to other two. Because random forest improves predictive accuracy which generates large number of bootstrapped trees, classifying each option using every tree in newly created forest whihc decides a final predicted result by aggregating all the results.

```
library(factoextra)
```

```
## Welcome! Related Books: `Practical Guide To Cluster Analysis in R` at https://goo.gl/13EFCZ
```

```
library(ggplot2)

df <- na.omit(df)
df <- scale(df)
head(round(df,2))
```

```
##      accept_rate Outstate Enroll Grad.Rate
## [1,]       -0.03    -0.75  -0.06     -0.32
## [2,]        0.91     0.46  -0.29     -0.55
## [3,]        0.14     0.20  -0.48     -0.67
## [4,]        0.61     0.63  -0.69     -0.38
## [5,]        0.06    -0.72  -0.78     -2.94
## [6,]        0.47     0.76  -0.67     -0.61
```

```
set.seed(234)

km <- kmeans(df,2,nstart=25)

cat("_________________________________________________________________")
```

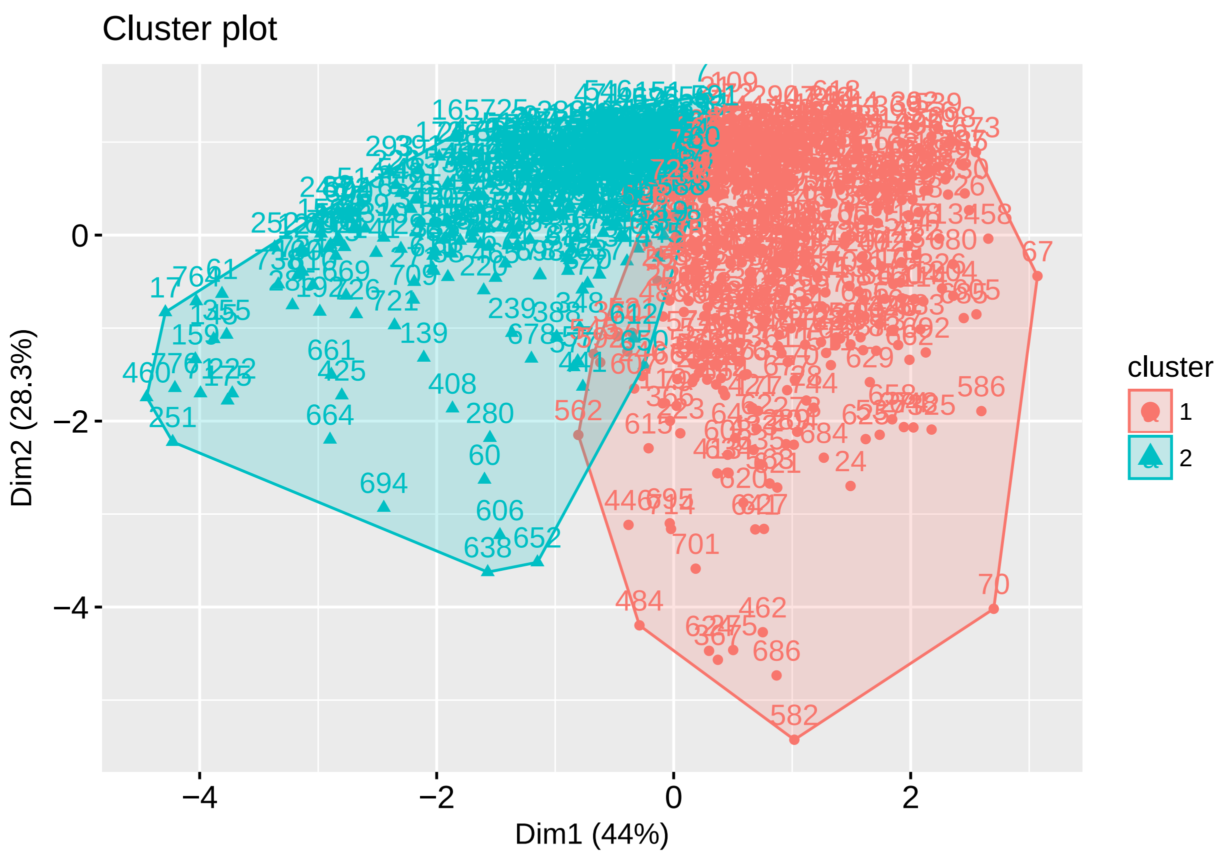
```
## --------------------------------------------------------------------
```

```r
cat("\n\n")
```

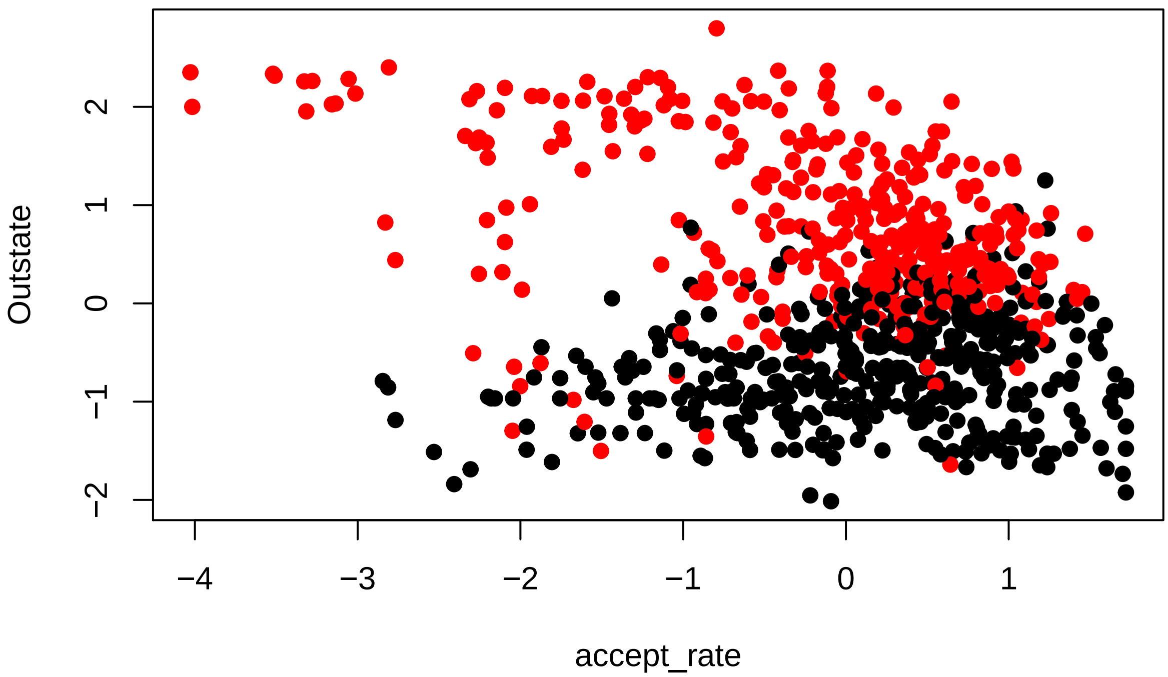
```r
print(km)
```

```
## K-means clustering with 2 clusters of sizes 419, 358
##
## Cluster means:
##   accept_rate   Outstate     Enroll  Grad.Rate
## 1   0.1455959 -0.6638777  0.1855451 -0.6781741
## 2  -0.1704042  0.7769965 -0.2171604  0.7937289
##
## Clustering vector:
##   [1] 1 1 1 2 1 2 2 2 2 1 2 2 2 2 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1 2 2 2 1 1 2
##  [36] 2 2 2 2 1 2 2 1 1 2 1 1 2 2 1 1 1 1 1 2 1 2 1 1 2 2 1 1 2 2 1 1 1 2 1
##  [71] 2 2 2 2 2 2 1 2 1 1 2 1 1 1 2 1 2 2 2 1 2 2 2 1 2 2 2 1 1 1 1 2 1 1 1
## [106] 1 2 2 1 2 1 1 1 1 2 2 2 2 1 1 2 2 2 2 2 1 2 2 1 2 2 2 2 1 2 2 2 2 2 2
## [141] 2 1 1 2 2 1 1 1 2 2 2 2 2 1 1 1 1 2 2 2 1 1 2 2 2 1 1 1 1 2 1 2 2 1 2
## [176] 2 1 1 1 1 1 1 2 1 2 2 2 2 1 1 2 2 1 2 1 1 2 1 1 1 2 1 1 1 1 2 1 1 2 1
## [211] 1 1 1 2 1 1 2 1 1 2 1 2 1 1 2 2 1 2 2 1 2 1 1 1 2 1 1 2 2 2 2 2 2 2 2
## [246] 1 1 1 1 2 2 2 1 1 2 2 2 1 2 2 2 1 1 1 1 2 2 1 2 1 2 1 2 1 1 1 1 1 2 2
## [281] 1 1 1 2 2 1 1 2 1 1 1 1 2 1 2 1 2 1 2 2 1 2 1 1 1 1 2 2 2 2 1 1 1 2 1
## [316] 1 1 2 2 2 1 1 2 1 1 1 2 2 1 1 2 2 2 1 2 1 1 1 2 2 1 1 2 2 2 1 2 2 2 2
## [351] 1 1 1 2 2 1 1 1 1 2 2 1 2 1 2 1 1 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 1 1 1
## [386] 1 2 2 1 1 2 1 2 1 1 2 2 2 2 1 2 1 2 2 1 2 1 2 2 1 1 1 1 1 2 1 2 1 1 1
## [421] 1 1 1 1 2 2 1 1 2 2 2 2 1 2 1 1 1 2 1 1 2 2 2 1 1 1 2 1 1 2 1 1 1 2 1
## [456] 1 2 1 2 2 2 1 1 2 2 1 1 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 1 1 1 2 2 2 1
## [491] 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 1 1 1 2 2 1 1 2 2 2 2 2 1 1 2 1 2
## [526] 2 1 2 2 1 1 1 1 1 2 1 1 1 1 1 2 2 1 1 2 2 2 2 1 2 1 1 2 1 1 2 2 2 1 2
## [561] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 1 1 2 1 1 1 1 1 1 1 2 1 1 2 1 1 2 2
## [596] 2 2 2 1 1 1 2 1 1 1 2 1 1 1 2 1 2 1 2 1 2 1 1 2 1 1 2 1 1 1 2 1 1 1 1
## [631] 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 2 1 1 2 1
## [666] 2 2 1 2 2 2 2 1 2 1 1 1 2 1 1 1 2 1 1 1 1 1 2 2 2 1 1 2 2 1 1 1 1 1 1
## [701] 1 1 1 2 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 2 2 1 1 2 2 2 2 1 2 1 1 2 1 2 2
## [736] 1 2 2 1 1 2 1 2 1 1 1 1 1 1 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 1 1 1 2
## [771] 2 2 1 2 1 2 2
##
## Within cluster sum of squares by cluster:
## [1] 1293.7238  940.6421
##  (between_SS / total_SS =  28.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

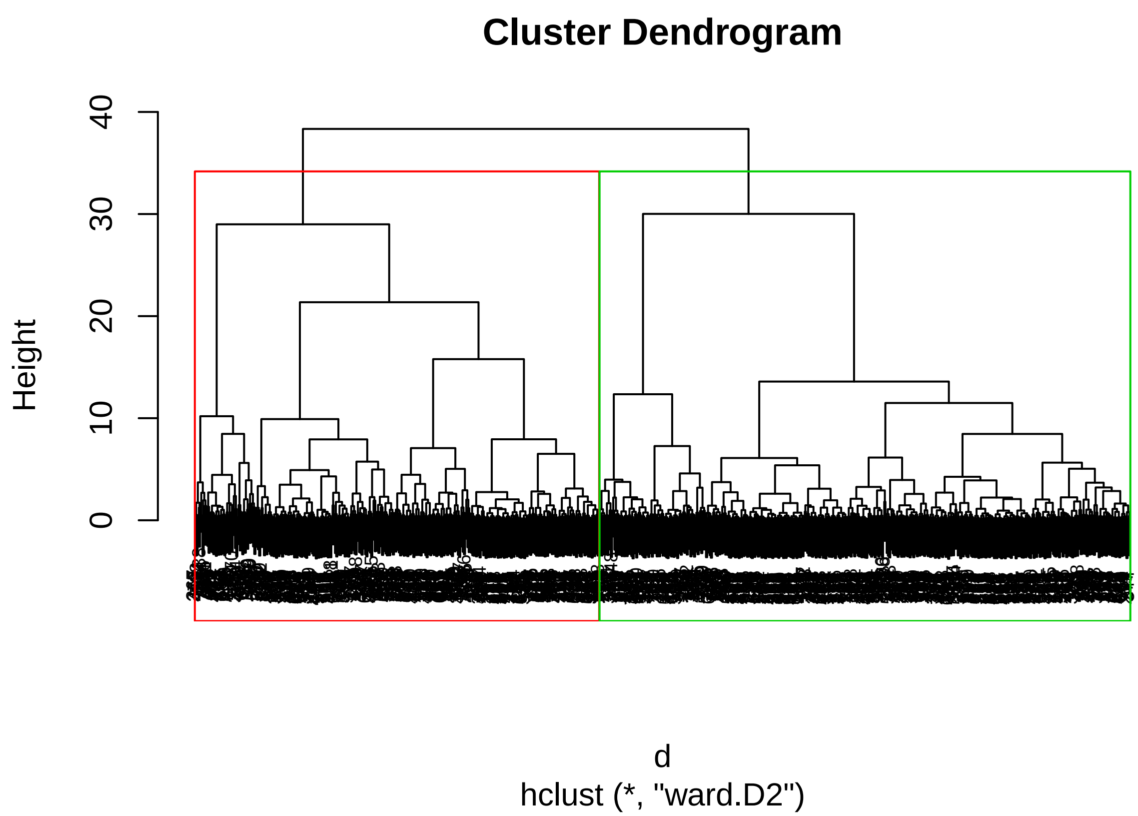
```r
fviz_cluster(km,data=df)
```




```
plot(df, col = km$cluster, pch = 19)
points(km$cluter, col = 1:2, pch = 8, cex = 3)
```

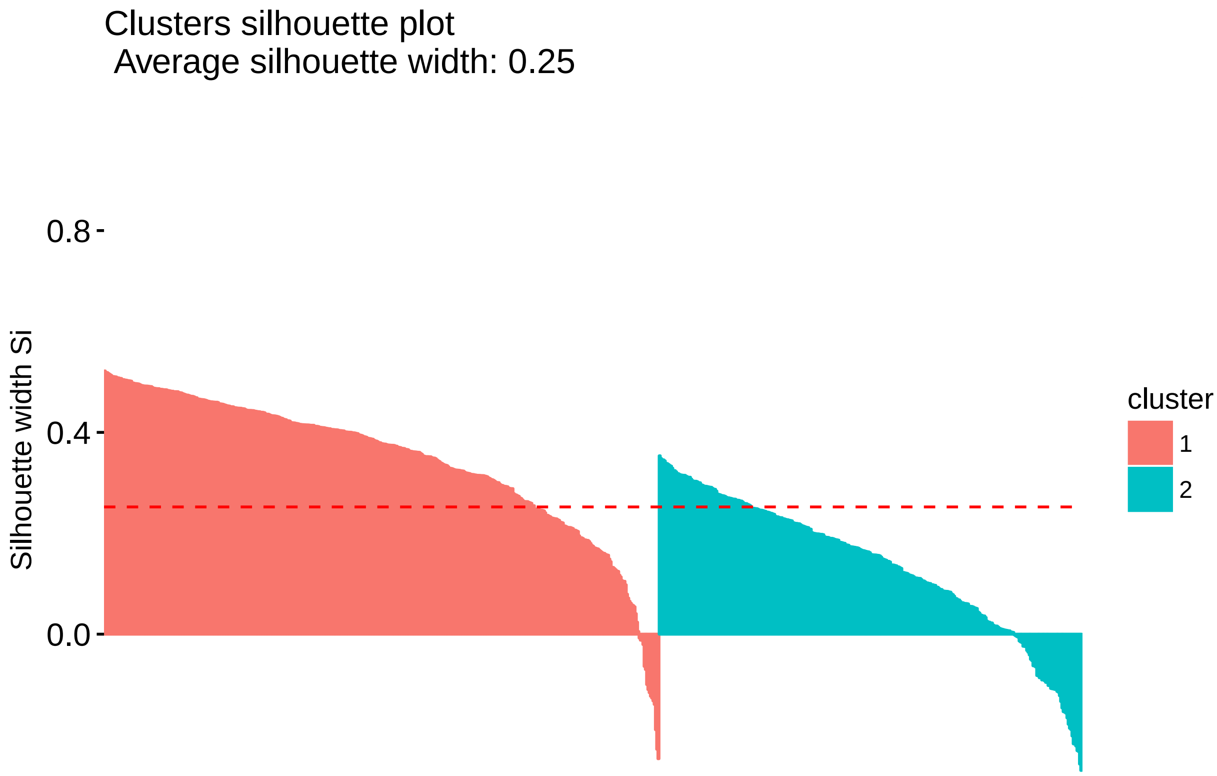



```
#Hierarchial Clustering

#Creation of dissimilarity matrix
d <- dist(df,method = "euclidean")

#applying hclust() function
res_hc1 <- hclust(d, method = "ward.D2")

#Cutting groups into 4 clusters
grp1 <- cutree(res_hc1,k=2)

#Plotting the cluster
plot(res_hc1, cex = 0.6)
rect.hclust(res_hc1, k = 2, border = 2:5)
```




```r
res.hc1 <- eclust(df, "hclust", k = 2, graph = FALSE)
fviz_silhouette(res.hc1)
```

```
##   cluster size ave.sil.width
## 1       1  441          0.34
## 2       2  336          0.13
```



After applying clustering on the college dataset, obervation is that clustering is not appropriate to determine ifnormation from dataset because clusters are not clear and crisp that can provide resourceful instances.