Assignment 3

Divya Kamma UIN: 670505193

Preethi Reddy Nandanuru UIN: 654074552

Ashrita Cheetirala UIN: 659259358

```
#Loading the datasets
library(tidyverse)

## -- Attaching packages --------------------------------------- tidyverse
1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------
tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()

library(readxl)
library(dplyr)
library(rpart)
library(randomForest)

## Warning: package 'randomForest' was built under R version 4.1.3

## randomForest 4.7-1

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

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin

library(ROCR)

## Warning: package 'ROCR' was built under R version 4.1.3
```

```
library(ggplot2)
library(rpart.plot)

#Reading the dataset into R
stc <- read_excel("C:\\Users\\pnanda4\\Desktop\\excel\\data_mining.xlsx")
df <- stc
```

Preprocessing the data

```
#Removing the unnecessary columns by looking at the intial stage
df <- subset (df, select = -
c(ID,Special.Pay,Departure.Date,Return.Date,Deposit.Date,Special.Pay,Early.RP
L,Latest.RPL,Initial.System.Date,FirstMeeting,LastMeeting,SchoolGradeTypeLow,
SchoolGradeTypeHigh))

#Matching two columns as they have similar data and removing one of the
column
df$SPR.Group.Revenue = as.numeric(df$SPR.Group.Revenue)
df$Tuition = as.numeric(df$Tuition)
str(df$Tuition)

##  num [1:2389] 424 2350 1181 376 865 ...

d <- ifelse(df$SPR.Group.Revenue==df$Tuition,"Yes","No")
mutate(df,d)

## # A tibble: 2,389 x 45
##    Program.Code From.Grade To.Grade Group.State Is.Non.Annual.  Days
Travel.Type
##    <chr>        <chr>      <chr>    <chr>                <dbl> <dbl> <chr>
##  1 HS           4          4        CA                       0     1 A
##  2 HC           8          8        AZ                       0     7 A
##  3 HD           8          8        FL                       0     3 A
##  4 HN           9          12       VA                       1     3 B
##  5 HD           6          8        FL                       0     6 T
##  6 HC           10         12       LA                       0     4 A
##  7 SG           11         12       MA                       1     6 A
##  8 FN           9          9        MX                       0     8 A
##  9 CC           8          8        AZ                       0     8 A
## 10 HD           8          8        TX                       0     4 A
## # ... with 2,379 more rows, and 38 more variables: Tuition <dbl>,
## #   FRP.Active <dbl>, FRP.Cancelled <dbl>, FRP.Take.up.percent. <dbl>,
## #   Cancelled.Pax <dbl>, Total.Discount.Pax <dbl>, Poverty.Code <chr>,
## #   Region <chr>, CRM.Segment <chr>, School.Type <chr>,
## #   Parent.Meeting.Flag <dbl>, MDR.Low.Grade <chr>, MDR.High.Grade <chr>,
## #   Total.School.Enrollment <dbl>, Income.Level <chr>,
## #   EZ.Pay.Take.Up.Rate <dbl>, School.Sponsor <dbl>, ...

table(d)
```

```
## d
##  Yes
## 2389

df <- subset(df, select = -c(SPR.Group.Revenue))

#Replacing Cayman Islands with KY
df$Group.State <- gsub("Cayman Islands", "KY", df$Group.State)
summary(df$Group.State)

##    Length     Class      Mode
##      2389 character character

#Converting character into numeric variables
df$FPP.to.School.enrollment=as.numeric(df$FPP.to.School.enrollment)

## Warning: NAs introduced by coercion

df$DifferenceTraveltoLastMeeting =
as.numeric(df$DifferenceTraveltoLastMeeting)

## Warning: NAs introduced by coercion

df$DifferenceTraveltoFirstMeeting =
as.numeric(df$DifferenceTraveltoFirstMeeting)

## Warning: NAs introduced by coercion

#function for calculating the mode
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

#finding mode for categorical variables by using the above function
mode_from.grade <- getmode(df$From.Grade)
mode_Poverty.Code <- getmode(df$Poverty.Code)
mode_MDR.Low.Grade <- getmode(df$MDR.Low.Grade)
mode_To.Grade <- getmode(df$To.Grade)
mode_MDR.High.Grade <- getmode(df$MDR.High.Grade)
mode_Income.Level <- getmode(df$Income.Level)
mode_SchoolSizeIndicator <- getmode(df$SchoolSizeIndicator)

#imputing NA values with mode of the column for categorical variables
df <- df%>% mutate(From.Grade = ifelse(From.Grade ==
"NA",mode_from.grade,From.Grade))%>%
  mutate(To.Grade = ifelse(To.Grade == "NA",mode_To.Grade,To.Grade))%>%
  mutate(MDR.High.Grade = ifelse(MDR.High.Grade ==
"NA",mode_MDR.High.Grade,MDR.High.Grade))
```

```
df$Poverty.Code[is.na(df$Poverty.Code)] <- "U"
df$MDR.Low.Grade[is.na(df$MDR.Low.Grade)] <- mode_MDR.Low.Grade
df$Income.Level[is.na(df$Income.Level)] <- mode_Income.Level
df$SchoolSizeIndicator[is.na(df$SchoolSizeIndicator)] <-
mode_SchoolSizeIndicator

#Converting the character "NA" into 0
df$DifferenceTraveltoFirstMeeting[is.na(df$DifferenceTraveltoFirstMeeting)]
<- 0
df$DifferenceTraveltoLastMeeting[is.na(df$DifferenceTraveltoLastMeeting)] <-
0
df$FPP.to.School.enrollment[is.na(df$FPP.to.School.enrollment)] <- 0

#finding mean for numerical variables
mean_Total.School.Enrollment = mean(df$Total.School.Enrollment, na.rm=TRUE)
mean_DifferenceTraveltoFirstMeeting = mean(df$DifferenceTraveltoFirstMeeting)
mean_DifferenceTraveltoLastMeeting = mean(df$DifferenceTraveltoLastMeeting,
na.rm=TRUE)
mean_FPP.to.School.enrollment = mean(df$FPP.to.School.enrollment, na.rm=TRUE)

#imputing NA values with calculated means for numerical variables
df$Total.School.Enrollment[is.na(df$Total.School.Enrollment)] <-
mean_Total.School.Enrollment
df <- df %>% mutate(DifferenceTraveltoFirstMeeting =
ifelse(DifferenceTraveltoFirstMeeting ==
0,mean_DifferenceTraveltoFirstMeeting,DifferenceTraveltoFirstMeeting))%>%
  mutate(DifferenceTraveltoLastMeeting = ifelse(DifferenceTraveltoLastMeeting
== 0,mean_DifferenceTraveltoLastMeeting,DifferenceTraveltoLastMeeting))%>%
  mutate(FPP.to.School.enrollment = ifelse(FPP.to.School.enrollment ==
0,mean_FPP.to.School.enrollment,FPP.to.School.enrollment))
view(df)

#mutating various numeric columns to factor
df$Retained.in.2012. = as.factor(df$Retained.in.2012.)
df$SingleGradeTripFlag = as.factor(df$SingleGradeTripFlag)
df$ NumberOfMeetingswithParents = as.factor(df$ NumberOfMeetingswithParents)
df$School.Sponsor = as.factor(df$School.Sponsor)
df$Parent.Meeting.Flag = as.factor(df$Parent.Meeting.Flag)
df$Is.Non.Annual. = as.factor(df$Is.Non.Annual.)
df$Retained.in.2012. <- ifelse(df$Retained.in.2012.==1,"Yes","No")
view(df)
summary(df)

##  Program.Code        From.Grade          To.Grade          Group.State
##  Length:2389        Length:2389        Length:2389        Length:2389
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
```

```
##  Is.Non.Annual.      Days        Travel.Type          Tuition
##  0:2021         Min.   : 1.000   Length:2389        Min.   :  79
##  1: 368         1st Qu.: 4.000   Class :character   1st Qu.:1174
##                 Median : 5.000   Mode  :character   Median :1700
##                 Mean   : 4.575                      Mean   :1615
##                 3rd Qu.: 5.000                      3rd Qu.:2048
##                 Max.   :12.000                      Max.   :4200
##    FRP.Active     FRP.Cancelled    FRP.Take.up.percent. Cancelled.Pax
##  Min.   :  0.00   Min.   : 0.000   Min.   :0.0000       Min.   : 0.000
##  1st Qu.:  6.00   1st Qu.: 1.000   1st Qu.:0.4550       1st Qu.: 2.000
##  Median : 12.00   Median : 2.000   Median :0.6000       Median : 4.000
##  Mean   : 16.87   Mean   : 3.306   Mean   :0.5707       Mean   : 4.807
##  3rd Qu.: 23.00   3rd Qu.: 4.000   3rd Qu.:0.7270       3rd Qu.: 6.000
##  Max.   :257.00   Max.   :45.000   Max.   :1.0000       Max.   :39.000
##  Total.Discount.Pax Poverty.Code          Region          CRM.Segment
##  Min.   : 0.000     Length:2389        Length:2389        Length:2389
##  1st Qu.: 1.000     Class :character   Class :character   Class :character
##  Median : 2.000     Mode  :character   Mode  :character   Mode  :character
##  Mean   : 2.954
##  3rd Qu.: 4.000
##  Max.   :47.000
##  School.Type        Parent.Meeting.Flag MDR.Low.Grade      MDR.High.Grade
##  Length:2389        0: 337              Length:2389        Length:2389
##  Class :character   1:2052              Class :character   Class 
:character
##  Mode  :character                       Mode  :character   Mode  
:character
##
##
##
##  Total.School.Enrollment Income.Level       EZ.Pay.Take.Up.Rate
School.Sponsor
##  Min.   :  19.0          Length:2389        Min.   :0.0000      0:2136
##  1st Qu.: 367.0          Class :character   1st Qu.:0.1000      1: 253
##  Median : 609.0          Mode  :character   Median :0.2000
##  Mean   : 648.4                             Mean   :0.2079
##  3rd Qu.: 811.0                             3rd Qu.:0.2920
##  Max.   :3990.0                             Max.   :1.7500
##  SPR.Product.Type   SPR.New.Existing        FPP          Total.Pax
##  Length:2389        Length:2389        Min.   :  2.0   Min.   :  2.00
##  Class :character   Class :character   1st Qu.: 12.0   1st Qu.: 14.00
##  Mode  :character   Mode  :character   Median : 23.0   Median : 26.00
##                                        Mean   : 31.3   Mean   : 34.25
##                                        3rd Qu.: 41.0   3rd Qu.: 44.00
##                                        Max.   :286.0   Max.   :313.00
##  NumberOfMeetingswithParents DifferenceTraveltoFirstMeeting
##  0: 337                      Min.   :-204.0
##  1:1471                      1st Qu.: 216.0
##  2: 581                      Median : 238.0
##                              Mean   : 256.9
```

```
##                                3rd Qu.: 277.0
##                                Max.   : 749.0
##  DifferenceTraveltoLastMeeting SchoolGradeType    DepartureMonth
##  Min.   :-204.0                Length:2389        Length:2389
##  1st Qu.: 196.7                Class :character   Class :character
##  Median : 220.0                Mode  :character   Mode  :character
##  Mean   : 224.4
##  3rd Qu.: 258.0
##  Max.   : 749.0
##  GroupGradeTypeLow  GroupGradeTypeHigh GroupGradeType     MajorProgramCode
##  Length:2389        Length:2389        Length:2389        Length:2389
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  SingleGradeTripFlag FPP.to.School.enrollment   FPP.to.PAX
##  0:1059              Min.   :0.0009221        Min.   :0.6000
##  1:1330              1st Qu.:0.0216667        1st Qu.:0.8824
##                      Median :0.0480000        Median :0.9091
##                      Mean   :0.0660879        Mean   :0.9007
##                      3rd Qu.:0.0857664        3rd Qu.:0.9333
##                      Max.   :2.0526316        Max.   :1.0000
##  Num.of.Non_FPP.PAX SchoolSizeIndicator Retained.in.2012.
##  Min.   : 0.000     Length:2389          Length:2389
##  1st Qu.: 1.000     Class :character     Class :character
##  Median : 2.000     Mode  :character     Mode  :character
##  Mean   : 2.954
##  3rd Qu.: 4.000
##  Max.   :47.000
```

```
str(df$Retained.in.2012.)
```

```
##  chr [1:2389] "Yes" "Yes" "Yes" "No" "No" "Yes" "No" "No" "Yes" "Yes"
"Yes" ...
```

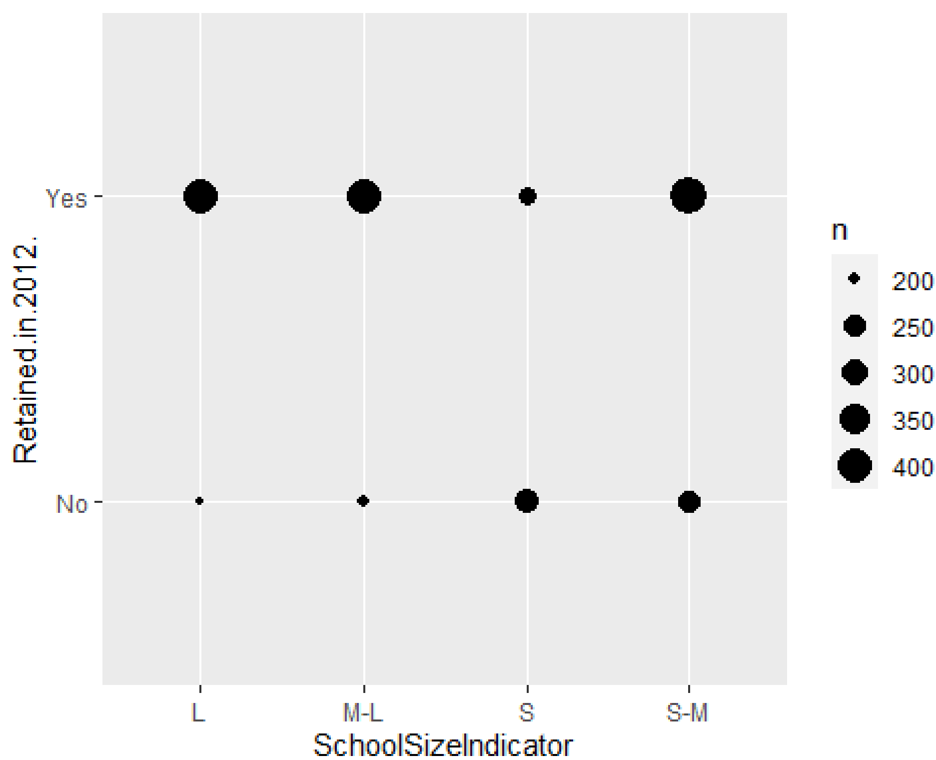
```
#mutating all the character variables to factors
colnames(df)
```

```
##  [1] "Program.Code"                   "From.Grade"
##  [3] "To.Grade"                       "Group.State"
##  [5] "Is.Non.Annual."                 "Days"
##  [7] "Travel.Type"                    "Tuition"
##  [9] "FRP.Active"                     "FRP.Cancelled"
## [11] "FRP.Take.up.percent."           "Cancelled.Pax"
## [13] "Total.Discount.Pax"             "Poverty.Code"
## [15] "Region"                         "CRM.Segment"
## [17] "School.Type"                    "Parent.Meeting.Flag"
## [19] "MDR.Low.Grade"                  "MDR.High.Grade"
## [21] "Total.School.Enrollment"        "Income.Level"
## [23] "EZ.Pay.Take.Up.Rate"            "School.Sponsor"
```

```
## [25] "SPR.Product.Type"                "SPR.New.Existing"
## [27] "FPP"                             "Total.Pax"
## [29] "NumberOfMeetingswithParents"     "DifferenceTraveltoFirstMeeting"
## [31] "DifferenceTraveltoLastMeeting"   "SchoolGradeType"
## [33] "DepartureMonth"                  "GroupGradeTypeLow"
## [35] "GroupGradeTypeHigh"              "GroupGradeType"
## [37] "MajorProgramCode"                "SingleGradeTripFlag"
## [39] "FPP.to.School.enrollment"        "FPP.to.PAX"
## [41] "Num.of.Non_FPP.PAX"              "SchoolSizeIndicator"
## [43] "Retained.in.2012."

df <- df %>% mutate_if(is.character,as.factor)
```

Exploratory Data Analysis

```
ggplot(df,aes(x=SchoolSizeIndicator,y=Retained.in.2012.))+
    geom_count()
```

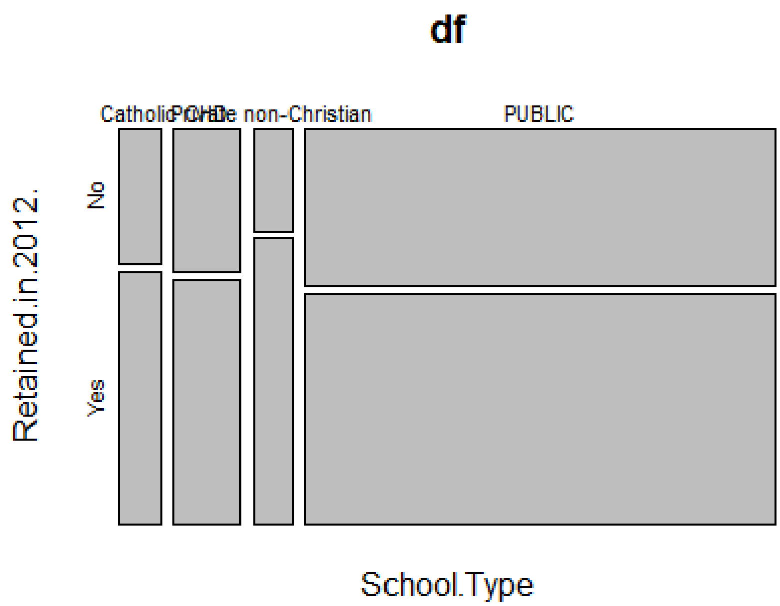


The following graph shows the various school sizes that have decided to retain in 2012 and we can infer that S-M size schools have the highest retention rate.

```
ggplot(df, aes(y=FRP.Active, x=FPP)) +
    geom_point(aes(color=Retained.in.2012.))
```

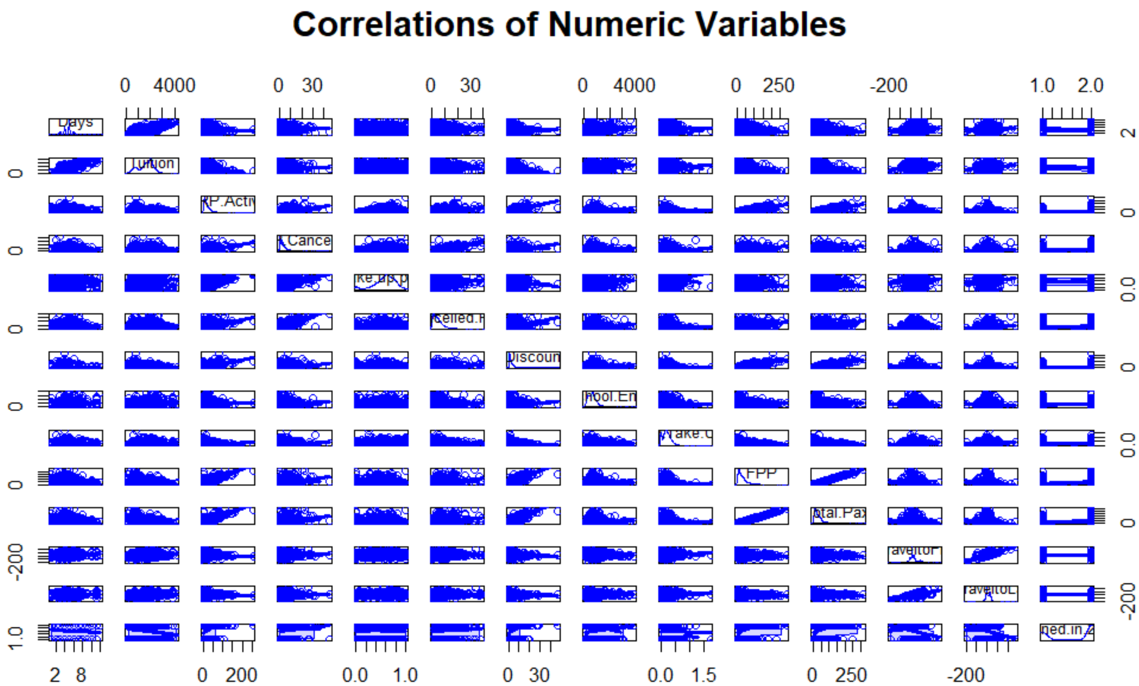
The scatterplot shows all the data points and differentiates between being retained or not retained. We can observe that as the number of full paying particpants increases the number of people who have taken the insurance and have been retained for the next year.

```
mosaicplot(School.Type~Retained.in.2012., data=df)
```



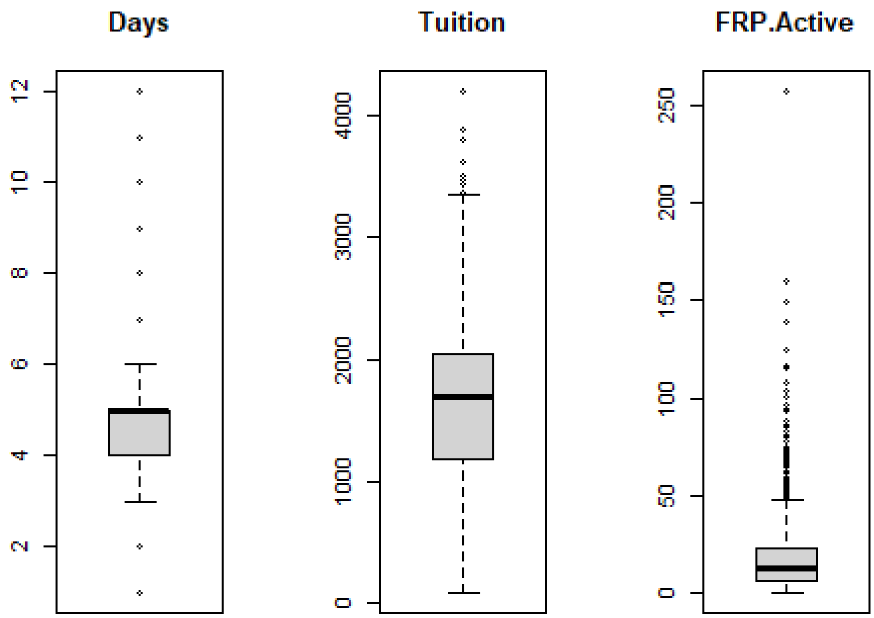
The public school type has retained STC the highest and Private non christian schools retained the STC least when compared to all the other school types.

```
# advanced scatter plots
library(car)

## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode

## The following object is masked from 'package:purrr':
##
##     some

scatterplotMatrix(~Days+Tuition+FRP.Active+FRP.Cancelled+FRP.Take.up.percent.
+Cancelled.Pax+Total.Discount.Pax+
Total.School.Enrollment+EZ.Pay.Take.Up.Rate+
FPP+Total.Pax+DifferenceTraveltoFirstMeeting+DifferenceTraveltoLastMeeting+Re
tained.in.2012. , data=df, main="Correlations of Numeric Variables")
```

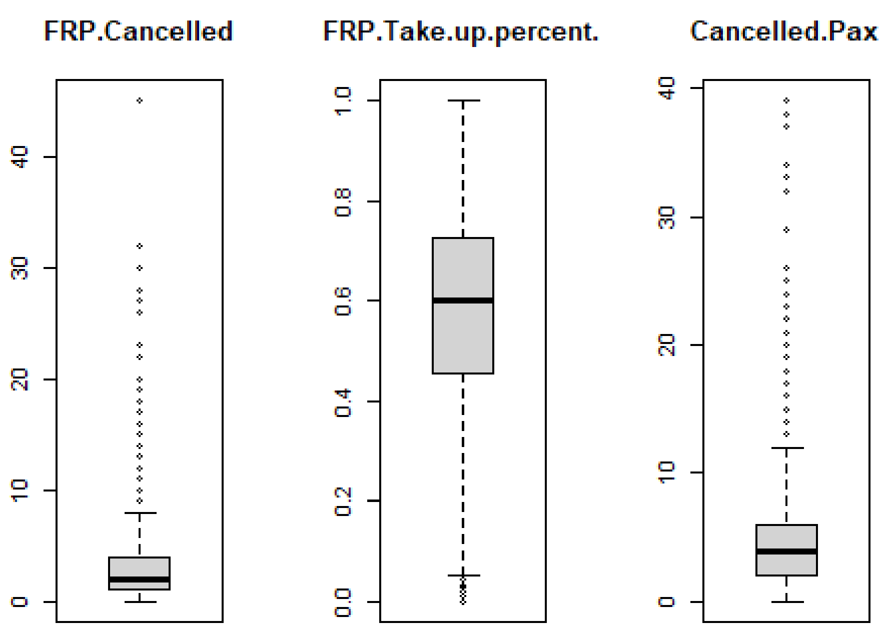


The above graph shows the correlation between all the numerical variables.

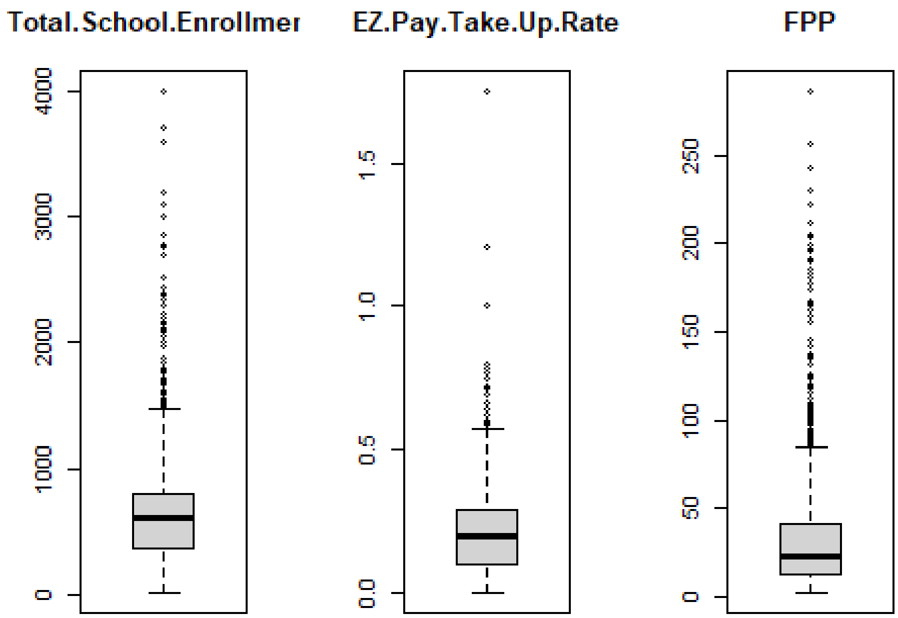
```
opar <- par(no.readonly = T)
par(mfrow = c(1,3))
boxplot(df$Days, main='Days')
boxplot(df$Tuition, main='Tuition')
boxplot(df$FRP.Active, main='FRP.Active')
```



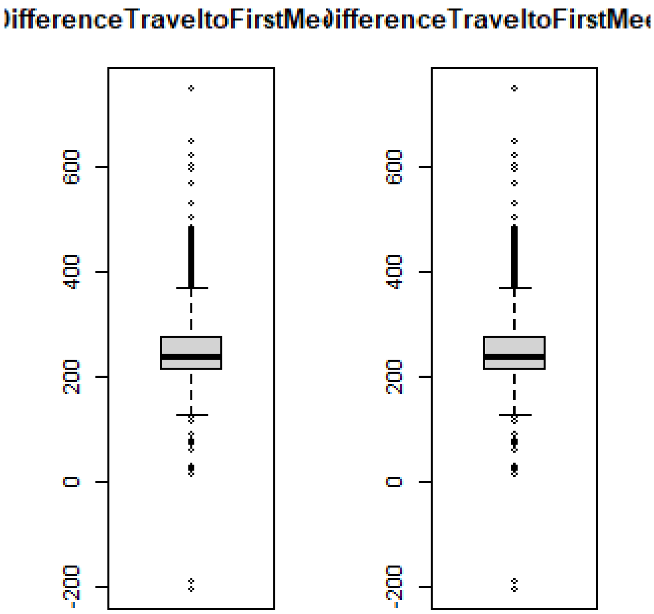

```
boxplot(df$FRP.Cancelled, main='FRP.Cancelled')
boxplot(df$FRP.Take.up.percent., main='FRP.Take.up.percent.')
boxplot(df$Cancelled.Pax, main='Cancelled.Pax')
```



```
boxplot(df$Total.School.Enrollment, main='Total.School.Enrollment')
boxplot(df$EZ.Pay.Take.Up.Rate, main='EZ.Pay.Take.Up.Rate')
boxplot(df$FPP, main='FPP')
```

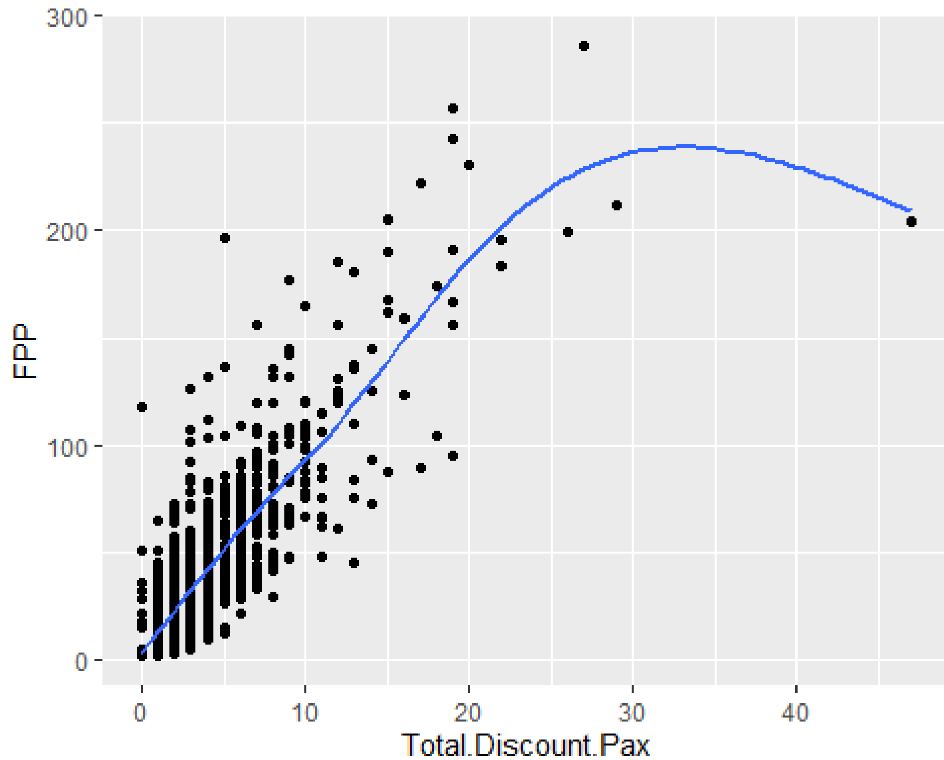



```
boxplot(df$DifferenceTraveltoFirstMeeting,
main='DifferenceTraveltoFirstMeeting')
boxplot(df$DifferenceTraveltoFirstMeeting,
main='DifferenceTraveltoFirstMeeting')
```



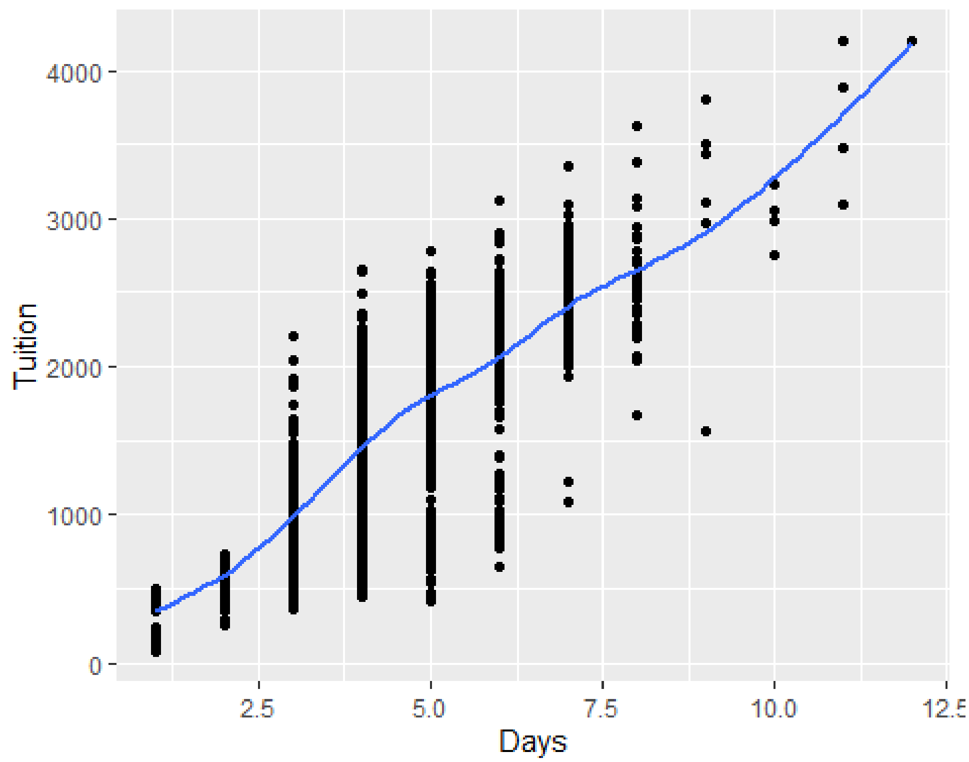
We have plotted the boxplots of all numerical variables to view where the median lies for each variable.

```
ggplot(df, aes(x=Total.Discount.Pax, y=FPP)) +
geom_point()+geom_smooth(se=FALSE)

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



The above graph shows curvilinear relationship between FPP and Total.Discount.Pax. We can infer that the discount paid by fully paying participants steadily increased and then started decreasing.

```
ggplot(df, aes(x=Days, y=Tuition)) + geom_point()+geom_smooth(se=FALSE)

## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



This plot shows the correlation between Days and Tuition. As the number of days the group on program increases, the price that costs for program also goes up. To verify the above, we have found out the correlation between days and tuition. As we can see there is a high correlation between these two variables.

```
cordata = df[,c("Days","Tuition")]
corr <- round(cor(cordata), 1)
corr

##          Days Tuition
## Days      1.0     0.8
## Tuition   0.8     1.0

#Finding the important variables using random forest and eliminating the
variables that have a mean gini decrease of less than 10.
rf <- randomForest(Retained.in.2012. ~ ., data = df,
                   mtry = sqrt(ncol(df)-1), ntree = 100,
                   proximity = T, importance = T)
print(rf)

##
## Call:
##  randomForest(formula = Retained.in.2012. ~ ., data = df, mtry =
sqrt(ncol(df) -      1), ntree = 100, proximity = T, importance = T)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 6
```

```
##
##         OOB estimate of  error rate: 20.64%
## Confusion matrix:
##      No  Yes class.error
## No  605  333   0.3550107
## Yes 160 1291   0.1102688
```

```
imp_variables<-importance(rf, type = 2)
imp_variables
```

```
##                                 MeanDecreaseGini
## Program.Code                           31.588914
## From.Grade                             57.359870
## To.Grade                               19.638742
## Group.State                            93.398981
## Is.Non.Annual.                         92.183460
## Days                                   10.278101
## Travel.Type                             2.473214
## Tuition                                29.062979
## FRP.Active                             33.610325
## FRP.Cancelled                          16.436619
## FRP.Take.up.percent.                   28.112418
## Cancelled.Pax                          19.291074
## Total.Discount.Pax                     16.645389
## Poverty.Code                           16.127554
## Region                                 19.388281
## CRM.Segment                            28.147945
## School.Type                             6.107431
## Parent.Meeting.Flag                     2.704012
## MDR.Low.Grade                          18.324497
## MDR.High.Grade                         12.107954
## Total.School.Enrollment                35.740266
## Income.Level                           78.014322
## EZ.Pay.Take.Up.Rate                    23.122769
## School.Sponsor                          2.087271
## SPR.Product.Type                        3.439195
## SPR.New.Existing                       60.159383
## FPP                                    44.900318
## Total.Pax                              39.261834
## NumberOfMeetingswithParents             6.138626
## DifferenceTraveltoFirstMeeting         27.878187
## DifferenceTraveltoLastMeeting          27.584738
## SchoolGradeType                        18.043801
## DepartureMonth                         15.652774
## GroupGradeTypeLow                      10.598107
## GroupGradeTypeHigh                      5.539089
## GroupGradeType                         26.957692
## MajorProgramCode                        2.095665
## SingleGradeTripFlag                    67.550958
## FPP.to.School.enrollment               32.267211
```

```
## FPP.to.PAX                                26.646651
## Num.of.Non_FPP.PAX                        15.779417
## SchoolSizeIndicator                       15.032886
```

```
#After removing the following variables we have a dataframe that finally
consists of 35 important variables acording to our analysis
df<- subset(df, select = -c(Travel.Type, School.Type, Parent.Meeting.Flag,
School.Sponsor, SPR.Product.Type,
NumberOfMeetingswithParents,GroupGradeTypeHigh, MajorProgramCode))
str(df)
```

```
## tibble [2,389 x 35] (S3: tbl_df/tbl/data.frame)
##  $ Program.Code                   : Factor w/ 28 levels "CC","CD","CN",..:
15 6 7 12 7 6 25 5 1 7 ...
##  $ From.Grade                     : Factor w/ 10 levels "10","11","12",..:
5 9 9 10 7 1 2 10 9 9 ...
##  $ To.Grade                       : Factor w/ 10 levels "10","11","12",..:
5 9 9 3 9 3 3 10 9 9 ...
##  $ Group.State                    : Factor w/ 53 levels "AB","AK","AL",..:
7 5 10 48 10 19 20 28 5 46 ...
##  $ Is.Non.Annual.                 : Factor w/ 2 levels "0","1": 1 1 1 2 1 1
2 1 1 1 ...
##  $ Days                           : num [1:2389] 1 7 3 3 6 4 6 8 8 4 ...
##  $ Tuition                        : num [1:2389] 424 2350 1181 376 865 ...
##  $ FRP.Active                     : num [1:2389] 25 9 17 0 40 9 16 10 30 51
...
##  $ FRP.Cancelled                  : num [1:2389] 3 9 6 0 8 4 4 0 0 1 ...
##  $ FRP.Take.up.percent.           : num [1:2389] 0.424 0.409 0.708 0 0.494
0.9 0.64 0.769 0.577 0.773 ...
##  $ Cancelled.Pax                  : num [1:2389] 3 11 6 1 9 3 5 1 0 1 ...
##  $ Total.Discount.Pax             : num [1:2389] 4 3 3 0 8 1 2 1 4 6 ...
##  $ Poverty.Code                   : Factor w/ 7 levels "0","A","B","C",..:
3 4 4 7 5 4 7 7 7 7 ...
##  $ Region                         : Factor w/ 6 levels
"Dallas","Houston",..: 6 4 4 4 4 4 4 4 4 2 ...
##  $ CRM.Segment                    : Factor w/ 12 levels "1","10","11",..: 6
2 2 9 2 10 10 9 7 7 ...
##  $ MDR.Low.Grade                  : Factor w/ 12 levels
"1","10","2","3",..: 11 8 7 7 7 2 10 7 7 12 ...
##  $ MDR.High.Grade                 : Factor w/ 12 levels "1","10","11",..: 8
11 11 11 11 4 4 11 4 11 ...
##  $ Total.School.Enrollment        : num [1:2389] 927 850 955 648 720 ...
##  $ Income.Level                   : Factor w/ 22 levels "A","B","C","D",..:
21 1 15 21 3 9 7 21 11 11 ...
##  $ EZ.Pay.Take.Up.Rate            : num [1:2389] 0.17 0.091 0.042 0 0.383
0.1 0.08 0 0.231 0.136 ...
##  $ SPR.New.Existing               : Factor w/ 2 levels "EXISTING","NEW": 1
1 1 1 1 2 1 1 1 1 ...
##  $ FPP                            : num [1:2389] 59 22 24 18 81 10 25 13 52
66 ...
```

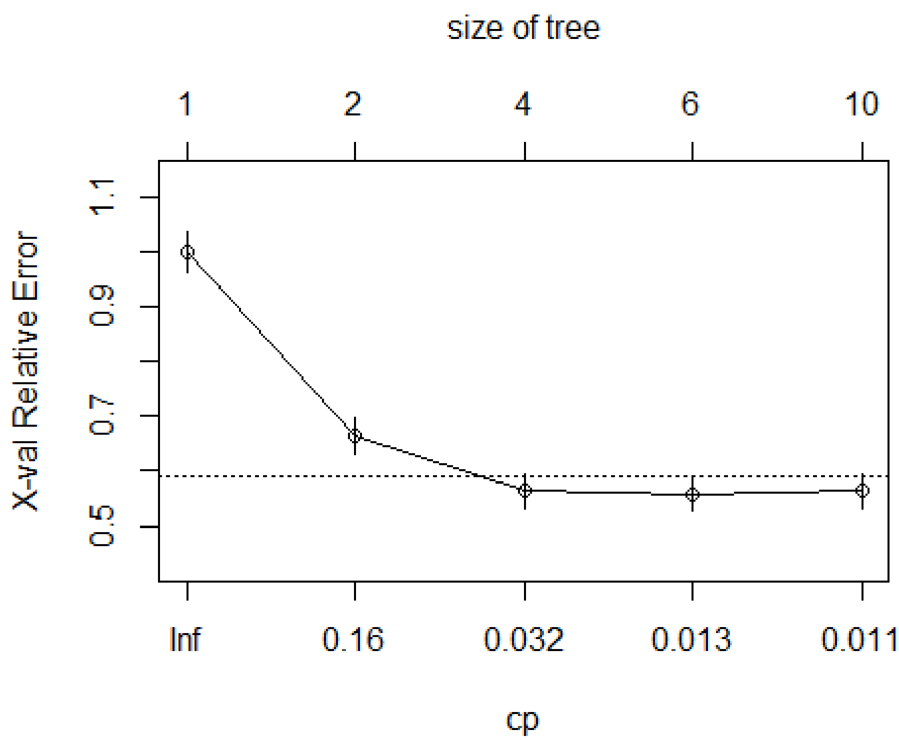
```
##  $ Total.Pax                        : num [1:2389] 63 25 27 18 89 11 27 14 56
72 ...
##  $ DifferenceTraveltoFirstMeeting: num [1:2389] 155 423 124 225 145 ...
##  $ DifferenceTraveltoLastMeeting : num [1:2389] 155 140 124 197 145 ...
##  $ SchoolGradeType               : Factor w/ 9 levels "Elementary-
>Elementary",..: 1 7 7 5 7 5 5 5 7 7 ...
##  $ DepartureMonth                : Factor w/ 6 levels
"April","February",..: 3 3 3 3 3 3 3 3 3 2 ...
##  $ GroupGradeTypeLow             : Factor w/ 6 levels
"Elementary","High",..: 3 4 4 6 4 2 2 6 4 5 ...
##  $ GroupGradeType                : Factor w/ 13 levels "Elementary-
>Elementary",..: 5 9 9 13 9 4 4 13 8 12 ...
##  $ SingleGradeTripFlag           : Factor w/ 2 levels "0","1": 2 2 2 1 1 1
1 2 2 2 ...
##  $ FPP.to.School.enrollment      : num [1:2389] 0.0636 0.0259 0.0251
0.0637 0.1125 ...
##  $ FPP.to.PAX                    : num [1:2389] 0.937 0.88 0.889 1 0.91
...
##  $ Num.of.Non_FPP.PAX            : num [1:2389] 4 3 3 0 8 1 2 1 4 6 ...
##  $ SchoolSizeIndicator           : Factor w/ 4 levels "L","M-L","S",..: 1
1 1 4 2 1 3 4 4 2 ...
##  $ Retained.in.2012.             : Factor w/ 2 levels "No","Yes": 2 2 2 1
1 2 1 1 2 2 ...
```

Construction of various decision trees by experimenting with different train and test splits.

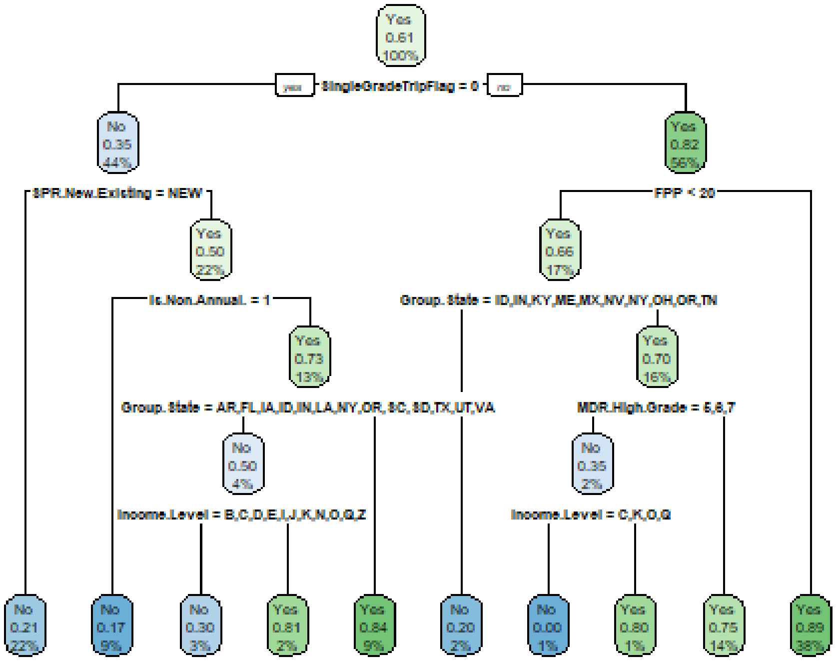
```
set.seed(60)
indx <- sample(2, nrow(df), replace=TRUE, prob=c(0.5,0.5))#dividing the
dataset into training and test with 50% in train and 50% in test
train <- df [indx==1, ] #assigning all the rows with index 1 to train
test <- df [indx==2, ] #assigning all the rows with index 2 to test
library("rpart")
tree_m1 <- rpart(Retained.in.2012. ~ ., train, parms = list(split = "gini" ))
#constructing the decision tree using rpart
print( tree_m1) #printing the decision tree
```

```
## n= 1221
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1221 474 Yes (0.3882064 0.6117936)
##    2) SingleGradeTripFlag=0 543 192 No (0.6464088 0.3535912)
##      4) SPR.New.Existing=NEW 273  56 No (0.7948718 0.2051282) *
##      5) SPR.New.Existing=EXISTING 270 134 Yes (0.4962963 0.5037037)
##       10) Is.Non.Annual.=1 109  19 No (0.8256881 0.1743119) *
##       11) Is.Non.Annual.=0 161  44 Yes (0.2732919 0.7267081)
##         22) Group.State=AR,FL,IA,ID,IN,LA,NY,OR,SC,SD,TX,UT,VA 54  27 No
(0.5000000 0.5000000)
##           44) Income.Level=B,C,D,E,I,J,K,N,O,Q,Z 33  10 No (0.6969697
```

```
0.3030303) *
##           45) Income.Level=A,F,G,H,L,M,P 21   4 Yes (0.1904762 0.8095238)
*
##         23)
Group.State=AL,AZ,CA,CO,HI,IL,KS,MA,MD,MI,MN,MO,MS,MT,NE,NM,NV,OH,OK,WA,WI
107  17 Yes (0.1588785 0.8411215) *
##    3) SingleGradeTripFlag=1 678 123 Yes (0.1814159 0.8185841)
##      6) FPP< 19.5 213  73 Yes (0.3427230 0.6572770)
##       12) Group.State=ID,IN,KY,ME,MX,NV,NY,OH,OR,TN 20   4 No (0.8000000
0.2000000) *
##       13)
Group.State=AK,AL,AZ,CA,CO,CT,FL,GA,IA,IL,KS,LA,MI,MN,MO,ND,NE,NM,OK,TX,UT,VA
,WA,WI 193  57 Yes (0.2953368 0.7046632)
##         26) MDR.High.Grade=5,6,7 23   8 No (0.6521739 0.3478261)
##           52) Income.Level=C,K,O,Q 13   0 No (1.0000000 0.0000000) *
##           53) Income.Level=D,F,I,J,M,N,P 10   2 Yes (0.2000000 0.8000000)
*
##         27) MDR.High.Grade=12,8,9 170  42 Yes (0.2470588 0.7529412) *
##      7) FPP>=19.5 465  50 Yes (0.1075269 0.8924731) *

plotcp(tree_m1)
```




```
rpart.plot(tree_m1)
```




```
printcp(tree_m1)

##
## Classification tree:
## rpart(formula = Retained.in.2012. ~ ., data = train, parms = list(split =
"gini"))
##
## Variables actually used in tree construction:
## [1] FPP                 Group.State         Income.Level
## [4] Is.Non.Annual.      MDR.High.Grade      SingleGradeTripFlag
## [7] SPR.New.Existing
##
## Root node error: 474/1221 = 0.38821
##
## n= 1221
##
##          CP nsplit rel error  xerror     xstd
## 1 0.335443      0   1.00000 1.00000 0.035926
## 2 0.077004      1   0.66456 0.66456 0.032254
## 3 0.013713      3   0.51055 0.56329 0.030471
## 4 0.012658      5   0.48312 0.55907 0.030389
## 5 0.010000      9   0.43038 0.56329 0.030471

minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
```

```
 for (j in minbckt){
tree_m1 <- rpart(Retained.in.2012. ~ ., train, parms = list(split = "gini" ),
control
= rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_1 <- predict(tree_m1, train, type = "class")#using predict
function to predict the classes of training data
trainerror_1 <- mean(tree_pred_class_1 != train$Retained.in.2012.)
#calculating the training error

tree_pred_test_1 <- predict(tree_m1, test, type = "class")#using predict
function to predict the classes of test data
testerror_1 <- mean(tree_pred_test_1 != test$Retained.in.2012.) #calculating
the test error
dif <- testerror_1-trainerror_1 #finding out the difference between test
error and training error

CM <- table(tree_pred_test_1, test$Retained.in.2012.)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
accuracy_model_test =(TP+TN)/(TP+TN+FP+FN) #calculating accuracy of test data
recall_test= (TP)/(TP+FN) #calculating recall of test data
F_score_test= (2*(recall_test*precision_test))/(recall_test+precision_test)
#calculating fscore of test data
#printing all the values
print(paste0("precision of test data: ", precision_test))
print(paste0("accuracy of test data: ", accuracy_model_test))
print(paste0("recall of test data: ", recall_test))
print(paste0("F-score of test data: ", F_score_test))
}
}

##
## tree_pred_test_1  No Yes
##               No  303 114
##               Yes 161 590
## [1] "precision of test data: 0.838068181818182"
## [1] "accuracy of test data: 0.764554794520548"
## [1] "recall of test data: 0.785619174434088"
## [1] "F-score of test data: 0.810996563573883"
##
## tree_pred_test_1  No Yes
##               No  304 121
##               Yes 160 583
## [1] "precision of test data: 0.828125"
## [1] "accuracy of test data: 0.759417808219178"
```

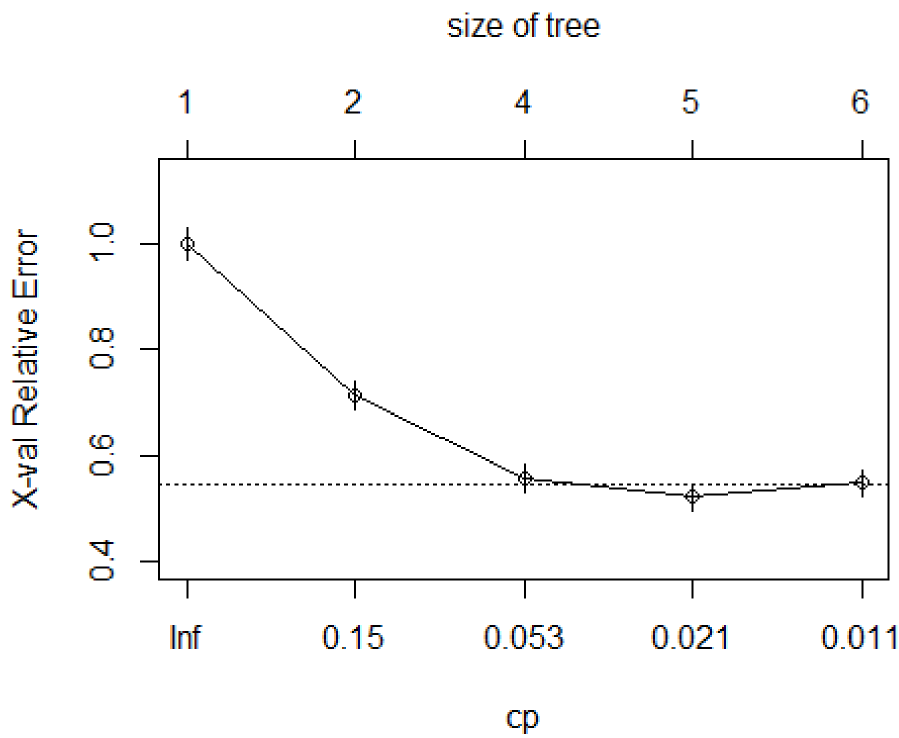
```
## [1] "recall of test data: 0.784656796769852"
## [1] "F-score of test data: 0.805805114029026"
##
## tree_pred_test_1  No Yes
##              No  282  76
##              Yes 182 628
## [1] "precision of test data: 0.892045454545455"
## [1] "accuracy of test data: 0.779109589041096"
## [1] "recall of test data: 0.775308641975309"
## [1] "F-score of test data: 0.829590488771466"
##
## tree_pred_test_1  No Yes
##              No  304 121
##              Yes 160 583
## [1] "precision of test data: 0.828125"
## [1] "accuracy of test data: 0.759417808219178"
## [1] "recall of test data: 0.784656796769852"
## [1] "F-score of test data: 0.805805114029026"
##
## tree_pred_test_1  No Yes
##              No  304 121
##              Yes 160 583
## [1] "precision of test data: 0.828125"
## [1] "accuracy of test data: 0.759417808219178"
## [1] "recall of test data: 0.784656796769852"
## [1] "F-score of test data: 0.805805114029026"
##
## tree_pred_test_1  No Yes
##              No  282  76
##              Yes 182 628
## [1] "precision of test data: 0.892045454545455"
## [1] "accuracy of test data: 0.779109589041096"
## [1] "recall of test data: 0.775308641975309"
## [1] "F-score of test data: 0.829590488771466"
##
## tree_pred_test_1  No Yes
##              No  291 104
##              Yes 173 600
## [1] "precision of test data: 0.852272727272727"
## [1] "accuracy of test data: 0.762842465753425"
## [1] "recall of test data: 0.776196636481242"
## [1] "F-score of test data: 0.812457684495599"
##
## tree_pred_test_1  No Yes
##              No  291 104
##              Yes 173 600
## [1] "precision of test data: 0.852272727272727"
## [1] "accuracy of test data: 0.762842465753425"
## [1] "recall of test data: 0.776196636481242"
## [1] "F-score of test data: 0.812457684495599"
```

```
##
## tree_pred_test_1  No Yes
##              No  282  76
##              Yes 182 628
## [1] "precision of test data: 0.892045454545455"
## [1] "accuracy of test data: 0.779109589041096"
## [1] "recall of test data: 0.775308641975309"
## [1] "F-score of test data: 0.829590488771466"
```

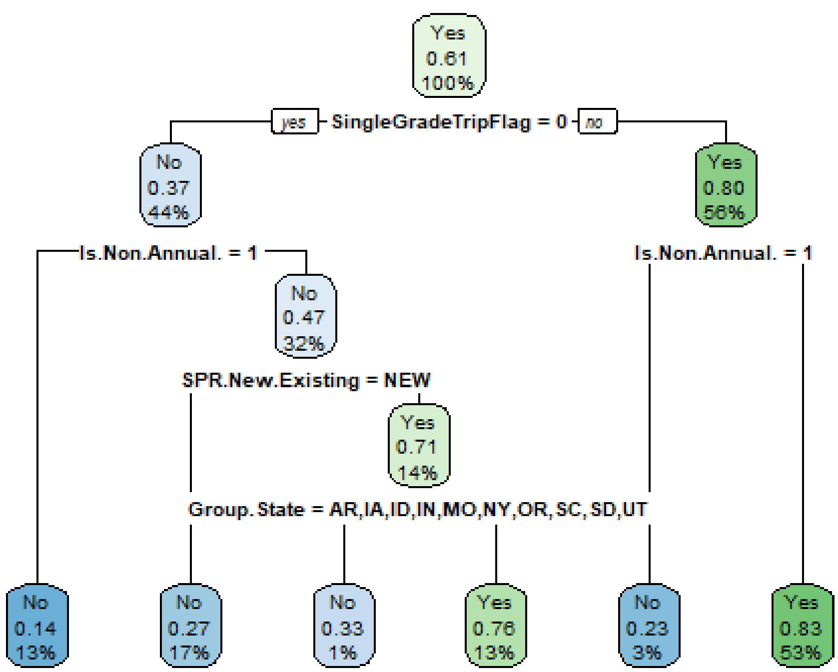
```
set.seed(60)
indx <- sample(2, nrow(df), replace=TRUE, prob=c(0.7,0.3))#dividing the
dataset into training and test with 70% in train and 30% in test
train <- df [indx==1, ] #assigning all the rows with index 1 to train
test <- df [indx==2, ] #assigning all the rows with index 2 to test
library("rpart")
tree_m2 <- rpart(Retained.in.2012. ~ ., train, parms = list(split = "gini" ))
#constructing the decision tree using rpart
print( tree_m2) #printing the decision tree
```

```
## n= 1677
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1677 652 Yes (0.3887895 0.6112105)
##    2) SingleGradeTripFlag=0 745 279 No (0.6255034 0.3744966)
##      4) Is.Non.Annual.=1 214  29 No (0.8644860 0.1355140) *
##      5) Is.Non.Annual.=0 531 250 No (0.5291902 0.4708098)
##       10) SPR.New.Existing=NEW 293  80 No (0.7269625 0.2730375) *
##       11) SPR.New.Existing=EXISTING 238  68 Yes (0.2857143 0.7142857)
##         22) Group.State=AR,IA,ID,IN,MO,NY,OR,SC,SD,UT 24   8 No (0.6666667
0.3333333) *
##         23)
Group.State=AL,AZ,CA,CO,CT,FL,HI,IL,KS,LA,MA,MD,ME,MI,MN,NC,ND,NH,NM,NV,OH,OK
,PA,TN,TX,VA,WA,WI 214  52 Yes (0.2429907 0.7570093) *
##    3) SingleGradeTripFlag=1 932 186 Yes (0.1995708 0.8004292)
##      6) Is.Non.Annual.=1 43  10 No (0.7674419 0.2325581) *
##      7) Is.Non.Annual.=0 889 153 Yes (0.1721035 0.8278965) *
```

```
plotcp(tree_m2)
```




```
rpart.plot(tree_m2)
```




```
printcp(tree_m2)
```

```
##
## Classification tree:
## rpart(formula = Retained.in.2012. ~ ., data = train, parms = list(split =
"gini"))
##
## Variables actually used in tree construction:
## [1] Group.State         Is.Non.Annual.      SingleGradeTripFlag
## [4] SPR.New.Existing
##
## Root node error: 652/1677 = 0.38879
##
## n= 1677
##
##          CP nsplit rel error  xerror     xstd
## 1 0.286810      0   1.00000 1.00000 0.030618
## 2 0.078221      1   0.71319 0.71319 0.028117
## 3 0.035276      3   0.55675 0.55675 0.025866
## 4 0.012270      4   0.52147 0.52147 0.025252
## 5 0.010000      5   0.50920 0.54755 0.025710
```

```
minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
 for (j in minbckt){
tree_m2 <- rpart(Retained.in.2012. ~ ., train, parms = list(split = "gini" ),
control
= rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_2 <- predict(tree_m2, train, type = "class")#using predict
function to predict the classes of training data
trainerror_2 <- mean(tree_pred_class_2 != train$Retained.in.2012.)
#calculating the training error
print(trainerror_1)
tree_pred_test_2 <- predict(tree_m2, test, type = "class")#using predict
function to predict the classes of test data
testerror_2 <- mean(tree_pred_test_2 != test$Retained.in.2012.) #calculating
the test error
dif <- testerror_2-trainerror_2 #finding out the difference between test
error and training error

CM <- table(tree_pred_test_2, test$Retained.in.2012.)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
accuracy_model_test =(TP+TN)/(TP+TN+FP+FN) #calculating accuracy of test data
recall_test= (TP)/(TP+FN) #calculating recall of test data
```

```
F_score_test= (2*(recall_test*precision_test))/(recall_test+precision_test)
#calculating fscore of test data
#printing all the values

print(paste0("precision of test data: ", precision_test))
print(paste0("accuracy of test data: ", accuracy_model_test))
print(paste0("recall of test data: ", recall_test))
print(paste0("F-score of test data: ", F_score_test))
}
}
```

```
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
##              No  204  51
##              Yes  82 375
## [1] "precision of test data: 0.880281690140845"
## [1] "accuracy of test data: 0.813202247191011"
## [1] "recall of test data: 0.820568927789934"
## [1] "F-score of test data: 0.849377123442809"
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
##              No  204  51
##              Yes  82 375
## [1] "precision of test data: 0.880281690140845"
## [1] "accuracy of test data: 0.813202247191011"
## [1] "recall of test data: 0.820568927789934"
## [1] "F-score of test data: 0.849377123442809"
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
##              No  198  46
##              Yes  88 380
## [1] "precision of test data: 0.892018779342723"
## [1] "accuracy of test data: 0.811797752808989"
## [1] "recall of test data: 0.811965811965812"
## [1] "F-score of test data: 0.850111856823266"
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
##              No  204  51
##              Yes  82 375
## [1] "precision of test data: 0.880281690140845"
## [1] "accuracy of test data: 0.813202247191011"
## [1] "recall of test data: 0.820568927789934"
## [1] "F-score of test data: 0.849377123442809"
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
```

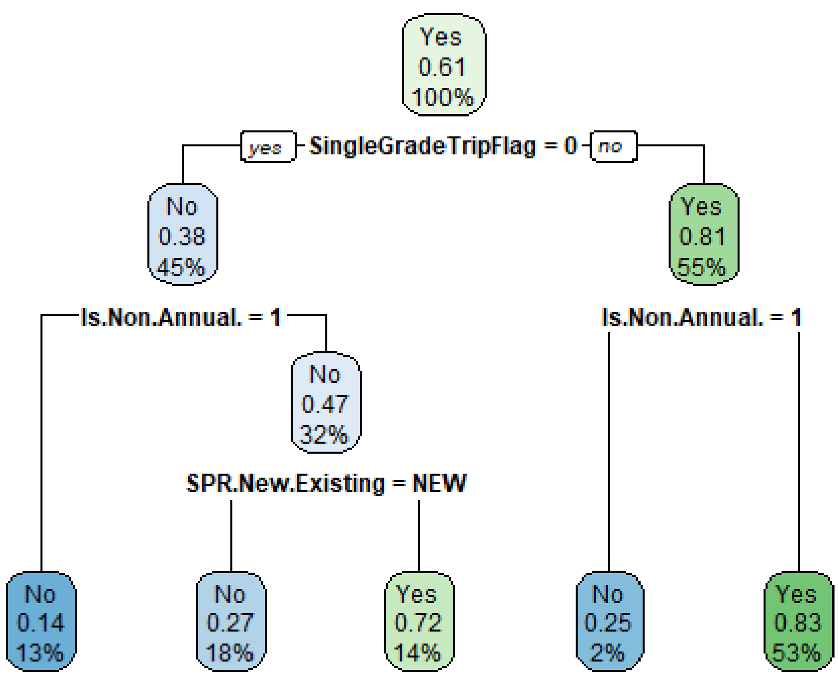
```
##               No  204  51
##               Yes  82 375
## [1] "precision of test data: 0.880281690140845"
## [1] "accuracy of test data: 0.813202247191011"
## [1] "recall of test data: 0.820568927789934"
## [1] "F-score of test data: 0.849377123442809"
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
##               No  198  46
##               Yes  88 380
## [1] "precision of test data: 0.892018779342723"
## [1] "accuracy of test data: 0.811797752808989"
## [1] "recall of test data: 0.811965811965812"
## [1] "F-score of test data: 0.850111856823266"
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
##               No  204  51
##               Yes  82 375
## [1] "precision of test data: 0.880281690140845"
## [1] "accuracy of test data: 0.813202247191011"
## [1] "recall of test data: 0.820568927789934"
## [1] "F-score of test data: 0.849377123442809"
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
##               No  204  51
##               Yes  82 375
## [1] "precision of test data: 0.880281690140845"
## [1] "accuracy of test data: 0.813202247191011"
## [1] "recall of test data: 0.820568927789934"
## [1] "F-score of test data: 0.849377123442809"
## [1] 0.1981982
##
## tree_pred_test_2  No Yes
##               No  198  46
##               Yes  88 380
## [1] "precision of test data: 0.892018779342723"
## [1] "accuracy of test data: 0.811797752808989"
## [1] "recall of test data: 0.811965811965812"
## [1] "F-score of test data: 0.850111856823266"
```

```r
set.seed(60)
indx <- sample(2, nrow(df), replace=TRUE, prob=c(0.8,0.2))#dividing the
dataset into training and test with 80% in train and 20% in test
train <- df [indx==1, ] #assigning all the rows with index 1 to train
test <- df [indx==2, ] #assigning all the rows with index 2 to test
library("rpart")
tree_m3 <- rpart(Retained.in.2012. ~ ., train, parms = list(split = "gini" ))
```

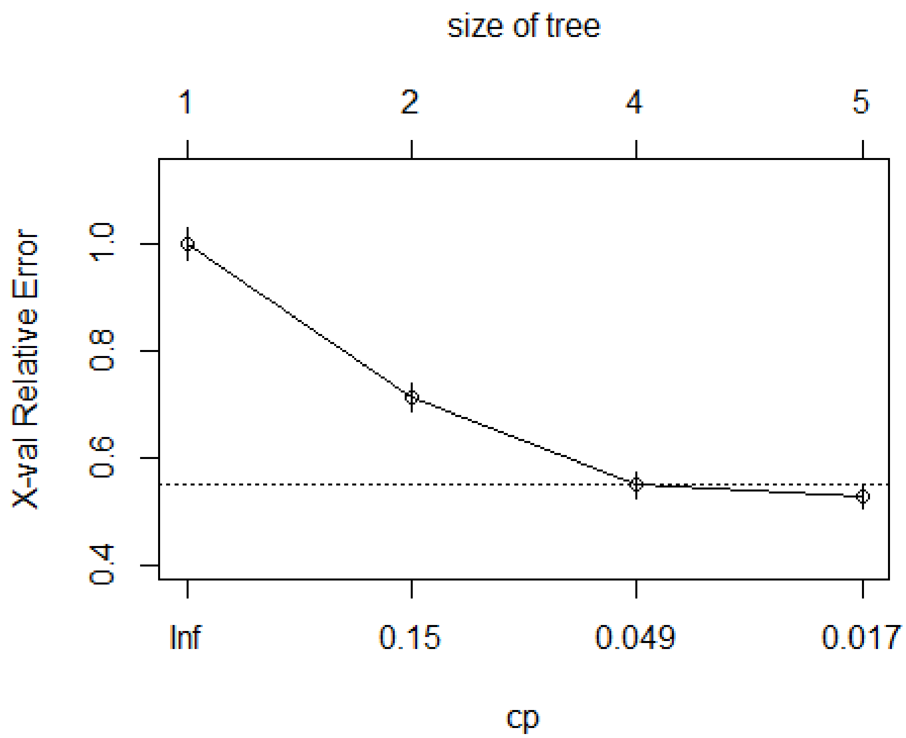
```r
#constructing the decision tree using rpart
print( tree_m3) #printing the decision tree
```

```
## n= 1902
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1902 735 Yes (0.3864353 0.6135647)
##    2) SingleGradeTripFlag=0 852 320 No (0.6244131 0.3755869)
##      4) Is.Non.Annual.=1 246  34 No (0.8617886 0.1382114) *
##      5) Is.Non.Annual.=0 606 286 No (0.5280528 0.4719472)
##       10) SPR.New.Existing=NEW 335  91 No (0.7283582 0.2716418) *
##       11) SPR.New.Existing=EXISTING 271  76 Yes (0.2804428 0.7195572) *
##    3) SingleGradeTripFlag=1 1050 203 Yes (0.1933333 0.8066667)
##      6) Is.Non.Annual.=1 44  11 No (0.7500000 0.2500000) *
##      7) Is.Non.Annual.=0 1006 170 Yes (0.1689861 0.8310139) *
```

```r
rpart.plot(tree_m3)
```




```r
plotcp(tree_m3)
```




```
printcp(tree_m3)

##
## Classification tree:
## rpart(formula = Retained.in.2012. ~ ., data = train, parms = list(split =
"gini"))
##
## Variables actually used in tree construction:
## [1] Is.Non.Annual.      SingleGradeTripFlag SPR.New.Existing
##
## Root node error: 735/1902 = 0.38644
##
## n= 1902
##
##         CP nsplit rel error  xerror     xstd
## 1 0.288435      0   1.00000 1.00000 0.028893
## 2 0.080952      1   0.71156 0.71156 0.026494
## 3 0.029932      3   0.54966 0.54966 0.024269
## 4 0.010000      4   0.51973 0.52789 0.023910

minsplt <- c(15, 51, 104) #assigning random vector values to minsplit
minbckt <- c(5, 17, 38) #assigning random vector values to minbckt
#looping through to try different combinations of minsplit and minbucket
for (i in minsplt){
 for (j in minbckt){
tree_m3 <- rpart(Retained.in.2012. ~ ., train, parms = list(split = "gini" ),
control
```

```
= rpart.control(minbucket = j, minsplit =i, cp=0.01))
tree_pred_class_3 <- predict(tree_m3, train, type = "class")#using predict
function to predict the classes of training data
trainerror_3 <- mean(tree_pred_class_3 != train$Retained.in.2012.)
#calculating the training error
print(trainerror_1)
tree_pred_test_3 <- predict(tree_m3, test, type = "class")#using predict
function to predict the classes of test data
testerror_3 <- mean(tree_pred_test_3 != test$Retained.in.2012.) #calculating
the test error
dif <- testerror_3-trainerror_3 #finding out the difference between test
error and training error

CM <- table(tree_pred_test_3, test$Retained.in.2012.)
print(CM)
#Assigning the values of matrix to the following variables
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
precision_test =(TP)/(TP+FP) #calculating precision of test data
accuracy_model_test =(TP+TN)/(TP+TN+FP+FN) #calculating accuracy of test data
recall_test= (TP)/(TP+FN) #calculating recall of test data
F_score_test= (2*(recall_test*precision_test))/(recall_test+precision_test)
#calculating fscore of test data
#printing all the values

print(paste0("precision of test data: ", precision_test))
print(paste0("accuracy of test data: ", accuracy_model_test))
print(paste0("recall of test data: ", recall_test))
print(paste0("F-score of test data: ", F_score_test))
}
}

## [1] 0.1981982
##
## tree_pred_test_3  No Yes
##              No  140  29
##              Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
## [1] 0.1981982
##
## tree_pred_test_3  No Yes
##              No  140  29
##              Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
```

```
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
## [1] 0.1981982
## 
## tree_pred_test_3  No Yes
##              No  140  29
##              Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
## [1] 0.1981982
## 
## tree_pred_test_3  No Yes
##              No  140  29
##              Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
## [1] 0.1981982
## 
## tree_pred_test_3  No Yes
##              No  140  29
##              Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
## [1] 0.1981982
## 
## tree_pred_test_3  No Yes
##              No  140  29
##              Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
## [1] 0.1981982
## 
## tree_pred_test_3  No Yes
##              No  140  29
##              Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
## [1] 0.1981982
## 
## tree_pred_test_3  No Yes
```

```
##               No  140  29
##               Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
## [1] 0.1981982
##
## tree_pred_test_3  No Yes
##               No  140  29
##               Yes  63 255
## [1] "precision of test data: 0.897887323943662"
## [1] "accuracy of test data: 0.811088295687885"
## [1] "recall of test data: 0.80188679245283"
## [1] "F-score of test data: 0.847176079734219"
```

If STC is actually being retained by customer, but they are predicted as not retained, it will be expensive. Hence, we considere recall to be a performance metric for this dataset as it will be crucial to reduce the number of False negatives (actually retained but predicted as not retained).

Gini 50:50 CP=0.01		
minsplit	minbucket	Recall
15	5	0.7856192
15	17	0.7846568
51	5	0.7846568
51	17	0.7846568
104	5	0.7761966
104	17	0.7761966
15	38	0.7753086
51	38	0.7753086
104	38	0.7753086

Gini 70:30 CP=0.01		
minsplit	minbucket	Recall
15	5	0.8205689
15	17	0.8205689
51	5	0.8205689
51	17	0.8205689
104	5	0.8205689
104	17	0.8205689
15	38	0.8119658
51	38	0.8119658
104	38	0.8119658

Gini 80:20 CP=0.01		
minsplit	minbucket	Recall
15	5	0.8018868
15	17	0.8018868
15	38	0.8018868
51	5	0.8018868
51	17	0.8018868
51	38	0.8018868
104	5	0.8018868
104	17	0.8018868
104	38	0.8018868

According to the decision trees that have been constructed above, we can conclude that 70:30 split with the highlighted pruning parameters gives us the best recall.

Constructing random forests with different ntree values and finding the best mtry for each model to derive a single model with the best performance

```
set.seed(60)
indx <- sample(2, nrow(df), replace=TRUE, prob=c(0.8,0.2))#dividing the
dataset into training and test with 80% in train and 20% in test
train <- df [indx==1, ] #assigning all the rows with index 1 to train
test <- df [indx==2, ] #assigning all the rows with index 2 to test
#Whichever gives the best mtry value for 100 ntree samples
pr.err <- c()
for(mt in seq(1,ncol(train)))
{
rf1 <- randomForest(Retained.in.2012.~., data = train, ntree = 100,
mtry = ifelse(mt == ncol(train),
mt-1, mt))
predicted <- predict(rf1, newdata = test, type = "class")
pr.err <- c(pr.err,mean(test$Retained.in.2012. != predicted))
}
```

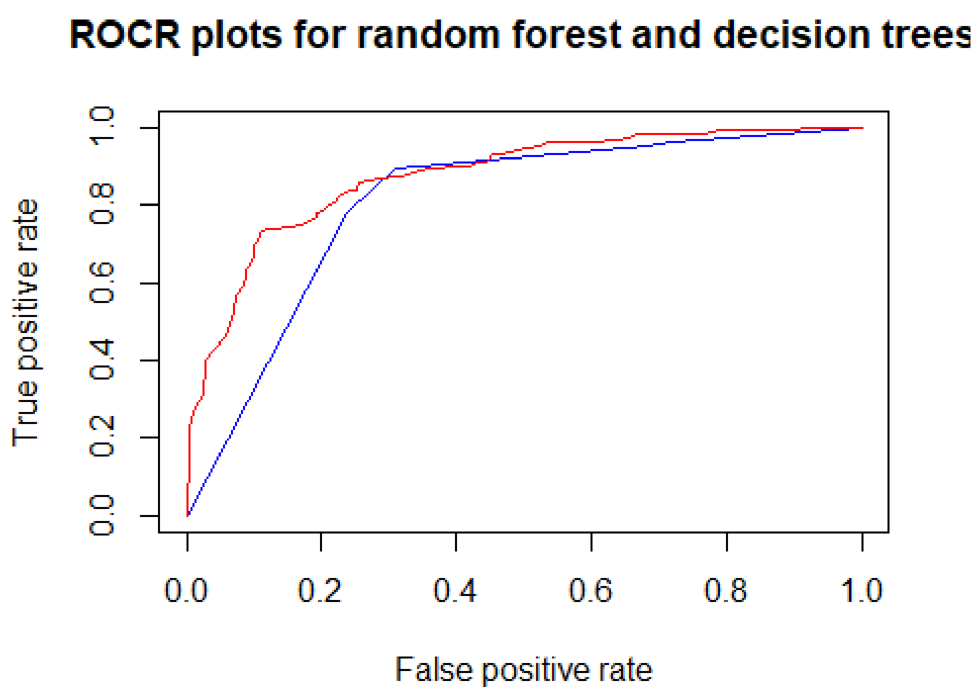
```
bestmtry <- which.min(pr.err)
print(bestmtry)

## [1] 6

rf1 <- randomForest(Retained.in.2012.~., data = train, ntree = 100,
mtry =bestmtry)
predicted <- predict(rf1, newdata = test, type = "class")
CM <- table(predicted, test$Retained.in.2012.)
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
recall_test1= (TP)/(TP+FN) #calculating recall of test data
F_score_test1= (2*(recall_test*precision_test))/(recall_test+precision_test)
pr.err <- c(pr.err,mean(test$Retained.in.2012. != predicted))
print(paste0("recall of test data: ", recall_test1))

## [1] "recall of test data: 0.757396449704142"

print(paste0("F-score of test data: ", F_score_test1))

## [1] "F-score of test data: 0.847176079734219"

#Whichever gives the best mtry value for 300 ntree samples
pr.err <- c()
for(mt in seq(1,ncol(train)))
{
rf2 <- randomForest(Retained.in.2012.~., data = train, ntree = 300,
mtry = ifelse(mt == ncol(train),
mt-1, mt))
predicted <- predict(rf2, newdata = test, type = "class")
pr.err <- c(pr.err,mean(test$Retained.in.2012. != predicted))
}
bestmtry1 <- which.min(pr.err)
print(bestmtry1)

## [1] 16

rf2 <- randomForest(Retained.in.2012.~., data = train, ntree = 300,
mtry = bestmtry1)
predicted <- predict(rf2, newdata = test, type = "class")
CM <- table(predicted, test$Retained.in.2012.)
TN =CM[1,1]
TP =CM[2,2]
FP =CM[1,2]
FN =CM[2,1]
recall_test2= (TP)/(TP+FN) #calculating recall of test data
F_score_test2= (2*(recall_test*precision_test))/(recall_test+precision_test)
pr.err <- c(pr.err,mean(test$Retained.in.2012. != predicted))
print(paste0("recall of test data: ", recall_test2))
```

```
## [1] "recall of test data: 0.761194029850746"

print(paste0("F-score of test data: ", F_score_test2))

## [1] "F-score of test data: 0.847176079734219"
```

According to the randomforest models constructed above, the randomforest model1 is considered as it gives us the better recall.

Plotting the ROC Curve for the two trees

```
#Plotting ROC curve for decision trees
score <- predict(tree_m2, test, type = "prob")[,"Yes"]
pred <- prediction(score, test$Retained.in.2012.)
perf <- performance(pred, "tpr", "fpr")
#Plotting ROC curve for random trees
score2 <- predict(rf1, test, type = "prob")[,"Yes"]
pred2 <- prediction(score2, test$Retained.in.2012.)
perf2 <- performance(pred2, "tpr", "fpr")
#Plotting multiple plots in one graph
plot( perf, colorize = FALSE, col="blue",main= "ROCR plots for random forest
and decision trees")
plot(perf2, add = TRUE, colorize = FALSE,col="red")
```




```
#Area under curve for Decision Tree
auc <- unlist(slot(performance(pred, "auc"), "y.values"))
print(auc)
```

```
## [1] 0.8088271

#Area under curve for RandomForest model
auc1 <- unlist(slot(performance(pred2, "auc"), "y.values"))
print(auc1)

## [1] 0.8752602
```

The red and the blue lines indicate the ROC curve for Random Forest and Decision Tree respectively. From this graph we can state that random forest has a better performance on this dataset when compared to decision tree.

From the auc that we calculated above for both the models, we have inferred that RandomForest model has better auc when compared to decision tree model. By this, we can say that RandomForest model performs better than decision tree model.

```
#Creating a duplicate of the preprocessed data to build further models on
data <- df
```

Performing 10 fold cross validation on the dataset for both random forest and decision tree to see the affect of the performance and calculate the weighted averages.

```
#Decision Tree
set.seed(60)
data <- data[sample(nrow(data)), ]
k <- 10
nmethod <- 2
folds <- cut(seq(1,nrow(data)), breaks=k, labels=FALSE)
model.recall <- matrix(-1,k,nmethod,
dimnames=list(paste0("Fold",1:k),c("Decision Tree","Random Forest")))
actual_p <- matrix(-1,k,nmethod, dimnames=list(paste0("Fold",1:k),c("Decision
Tree","Random Forest")))
weighted_recall <- matrix(-1,k,nmethod,
dimnames=list(paste0("Fold",1:k),c("Decision Tree","Random Forest")))

#CV
#total number of positive(i.e yes) instances in the entire dataset
total <- 1451
#Calculated the weighted recall
for(i in 1:k){
  testindexes <- which(folds == i, arr.ind = TRUE)
  test<- data[testindexes, ]
  train<- data[-testindexes, ]
  tree <- rpart(Retained.in.2012. ~ ., train, parms = list(split = "gini" ),
control= rpart.control(minbucket = 5, minsplit =15, cp=0.01))
  pred <- predict(tree, newdata=test, type="class")
  CM <- table(pred, test$Retained.in.2012.)
  TN =CM[1,1]
  TP =CM[2,2]
```

```
  FP =CM[1,2]
  FN =CM[2,1]

  #number of actual positives in each fold
  actual_p[i,"Decision Tree"]<- TP+FN
  model.recall[i,"Decision Tree"] <- (TP)/(TP+FN)
  weighted_recall[i,"Decision Tree"] <- ( actual_p[i,"Decision
Tree"]*model.recall[i,"Decision Tree"])/total
  tree2 <- randomForest(Retained.in.2012.~.,data= train,mtry = bestmtry,
ntree = 100, proximity=T,importance = T)
  pred2<- predict(tree2, newdata=test, type="class")
  CM <- table(pred2, test$Retained.in.2012.)
  TN =CM[1,1]
  TP =CM[2,2]
  FP =CM[1,2]
  FN =CM[2,1]

#number of actual positives in each fold
  actual_p[i,"Random Forest"]<- TP+FN

  model.recall[i,"Random Forest"] <- (TP)/(TP+FN)
  weighted_recall[i,"Random Forest"] <- (actual_p[i,"Random
Forest"]/total)*model.recall[i,"Random Forest"]
}
ap <- actual_p
ap

##        Decision Tree Random Forest
## Fold1            147           163
## Fold2            159           164
## Fold3            164           163
## Fold4            170           173
## Fold5            160           165
## Fold6            150           151
## Fold7            166           165
## Fold8            161           161
## Fold9            167           166
## Fold10           164           169

mr <-model.recall #calculating model recall
mr

##        Decision Tree Random Forest
## Fold1      0.8299320     0.8098160
## Fold2      0.7987421     0.8048780
## Fold3      0.7621951     0.7668712
## Fold4      0.8176471     0.8150289
## Fold5      0.6937500     0.6969697
## Fold6      0.7600000     0.7350993
## Fold7      0.8192771     0.8242424
## Fold8      0.8633540     0.8571429
```

```
## Fold9      0.8203593     0.8674699
## Fold10     0.7743902     0.7278107
```

```r
wr <-weighted_recall #Assigning weighted recall to wr
wr
```

```
##        Decision Tree Random Forest
## Fold1     0.08407994    0.09097174
## Fold2     0.08752584    0.09097174
## Fold3     0.08614748    0.08614748
## Fold4     0.09579600    0.09717436
## Fold5     0.07649897    0.07925569
## Fold6     0.07856651    0.07649897
## Fold7     0.09372846    0.09372846
## Fold8     0.09579600    0.09510682
## Fold9     0.09441764    0.09924190
## Fold10    0.08752584    0.08476912
```

```r
wr1 <- colSums(wr) #Calculating the sum of all the weighted recall for
decision tree and random forest models
wr1
```

```
## Decision Tree Random Forest
##     0.8800827     0.8938663
```

When considering 10 cross validation, we derived that RandomForest has a higher recall when compared to Decision tree model.