Meghashree Maddihally Nagoji

3/18/2022

Exploratory Data Analysis

Reading data from the excel

```
data<-read_excel("Retention modeling.xlsx", sheet =2)

## Warning in read_fun(path = enc2native(normalizePath(path)), sheet_i = shee
t, :
## Expecting numeric in A2393 / R2393C1: got 'Data source: Company data adjus
ted by
## author using unspecified constants.'
```

Data Cleaning

Removing the last 3 Redundant Rows

```
n<-nrow(data)
df<-data[1:(n-3),]
```

Group.State has 54 Levels and Random Forest can only take 53 Levels So, Removing the First least occurred entry.

```
df <- df %>% filter(!Group.State == "Bermuda")
```

Keeping the Special.pay Column to add it after we convert all the other Columns NA

```
x <- df$Special.Pay
df <- subset(df, select = -c(ID,Departure.Date,Return.Date,Deposit.Date,Early
.RPL,
                             Latest.RPL,Initial.System.Date,FirstMeeting,Last
Meeting,
                             Special.Pay))
```

Replacing to True NA values. * Read_Excel doesn’t read the NA values as na *

```
df[df == "NA"] <- NA
```

Adding back the Special.Pay Column. NA in the Special.Pay is a Example Value, we are not removing them

```
df <- cbind(df,Special.Pay=x)

cols <- c('Program.Code','From.Grade', 'To.Grade', 'Group.State','Is.Non.Annu
al.',
          'Travel.Type','Special.Pay','Poverty.Code','Region','CRM.Segment',
          'School.Type','Parent.Meeting.Flag','MDR.Low.Grade','MDR.High.Grade
',
          'Income.Level','School.Sponsor','SPR.Product.Type','SPR.New.Existin
g',
```

```
          'NumberOfMeetingswithParents','SchoolGradeTypeLow','SchoolGradeType
High',
          'SchoolGradeType','DepartureMonth','GroupGradeTypeLow','GroupGradeT
ypeHigh',
          'GroupGradeType','MajorProgramCode','SingleGradeTripFlag','SchoolSi
zeIndicator'
)
```

Factoring the Columns

```
df[cols] <- lapply(df[cols], factor)
```

Converting to numeric values

```
df$DifferenceTraveltoFirstMeeting <- as.numeric(df$DifferenceTraveltoFirstMee
ting)
df$DifferenceTraveltoLastMeeting <- as.numeric(df$DifferenceTraveltoLastMeeti
ng)
df$FPP.to.School.enrollment <- as.numeric(df$FPP.to.School.enrollment)
```

Number of NA values before replacing

```
sum(is.na(df))

## [1] 2018
```

Function to replace NA's in both Categorical and Numerical Values

```
for (i in colnames(df)){
  if(class(df[[i]]) == 'factor'){
    tt <- table(df[,i])
    df[is.na(df[,i]), i] <- names(tt[tt==max(tt)])
  } else {
    df[is.na(df[,i]), i] <- mean(df[,i], na.rm = TRUE)
  }
}
```

Checking the type of all variables

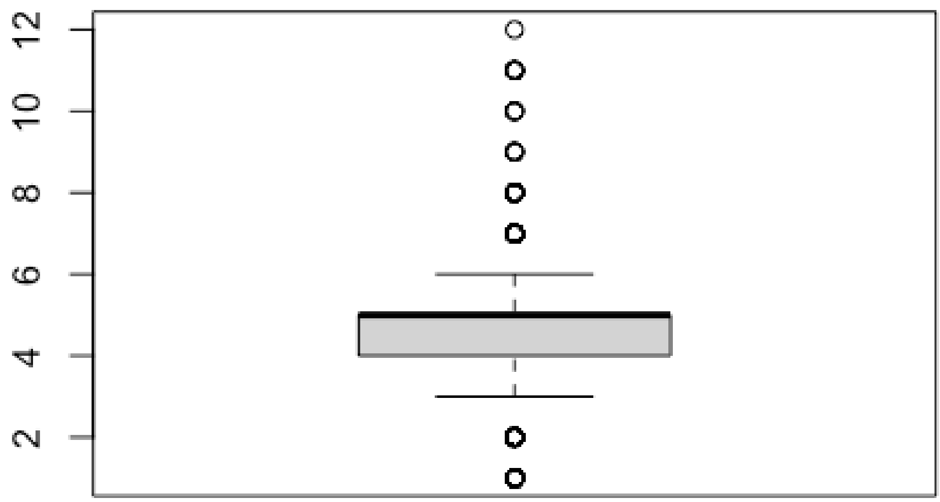
```
str(df)

## 'data.frame':    2388 obs. of  47 variables:
##  $ 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",..:
6 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 7 3 3 6 4 6 8 8 4 ...
##  $ Travel.Type                   : Factor w/ 4 levels "A","B","N","T": 1 1
1 2 4 1 1 1 1 1 ...
##  $ Tuition                       : num  424 2350 1181 376 865 ...
##  $ FRP.Active                    : num  25 9 17 0 40 9 16 10 30 51 ...
##  $ FRP.Cancelled                 : num  3 9 6 0 8 4 4 0 0 1 ...
##  $ FRP.Take.up.percent.          : num  0.424 0.409 0.708 0 0.494 0.9 0.64
0.769 0.577 0.773 ...
##  $ Cancelled.Pax                 : num  3 11 6 1 9 3 5 1 0 1 ...
##  $ Total.Discount.Pax            : num  4 3 3 0 8 1 2 1 4 6 ...
##  $ Poverty.Code                  : Factor w/ 6 levels "0","A","B","C",..:
3 4 4 3 5 4 3 3 3 3 ...
##  $ Region                        : Factor w/ 6 levels "Dallas","Houston",.
.: 6 4 4 4 4 4 4 4 4 2 ...
##  $ CRM.Segment                   : Factor w/ 11 levels "1","10","11",..: 6
2 2 9 2 10 10 9 7 7 ...
##  $ School.Type                   : Factor w/ 4 levels "Catholic","CHD",..:
4 4 4 2 4 4 1 2 2 3 ...
##  $ Parent.Meeting.Flag           : Factor w/ 2 levels "0","1": 2 2 2 1 2 2
2 2 2 2 ...
##  $ 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  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  0.17 0.091 0.042 0 0.383 0.1 0.08
0 0.231 0.136 ...
##  $ School.Sponsor                : Factor w/ 2 levels "0","1": 2 1 1 1 1 1
1 1 1 1 ...
##  $ SPR.Product.Type              : Factor w/ 6 levels "CA History","Costa
Rica",..: 1 3 3 3 3 3 6 3 3 3 ...
##  $ SPR.New.Existing              : Factor w/ 2 levels "EXISTING","NEW": 1
1 1 1 1 2 1 1 1 1 ...
##  $ FPP                           : num  59 22 24 18 81 10 25 13 52 66 ...
##  $ Total.Pax                     : num  63 25 27 18 89 11 27 14 56 72 ...
##  $ SPR.Group.Revenue             : num  424 2350 1181 376 865 ...
##  $ NumberOfMeetingswithParents   : Factor w/ 3 levels "0","1","2": 2 3 2 1
2 2 2 2 2 2 ...
##  $ DifferenceTraveltoFirstMeeting: num  155 423 124 262 145 ...
##  $ DifferenceTraveltoLastMeeting : num  155 140 124 229 145 ...
##  $ SchoolGradeTypeLow            : Factor w/ 4 levels "Elementary","High",
..: 1 3 3 2 3 2 2 2 3 3 ...
##  $ SchoolGradeTypeHigh           : Factor w/ 4 levels "Elementary","High",
..: 1 3 3 2 3 2 2 2 3 3 ...
##  $ SchoolGradeType               : Factor w/ 9 levels "Elementary->Element
ary",..: 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 ...
##  $ GroupGradeTypeHigh           : Factor w/ 4 levels "Elementary","High",
..: 1 3 3 4 3 2 2 4 2 3 ...
##  $ GroupGradeType               : Factor w/ 13 levels "Elementary->Elemen
tary",..: 5 9 9 13 9 4 4 13 8 12 ...
##  $ MajorProgramCode             : Factor w/ 4 levels "C","H","I","S": 2 2
2 2 2 2 4 3 1 2 ...
##  $ SingleGradeTripFlag          : Factor w/ 2 levels "0","1": 2 2 2 1 1 1
1 2 2 2 ...
##  $ FPP.to.School.enrollment     : num  0.0636 0.0259 0.0251 0.0662 0.1125
...
##  $ FPP.to.PAX                   : num  0.937 0.88 0.889 1 0.91 ...
##  $ Num.of.Non_FPP.PAX           : num  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.            : num  1 1 1 0 0 1 0 0 1 1 ...
##  $ Special.Pay                  : Factor w/ 4 levels "CP","FR","NA",..: 3
1 3 3 3 3 3 3 1 3 ...
```

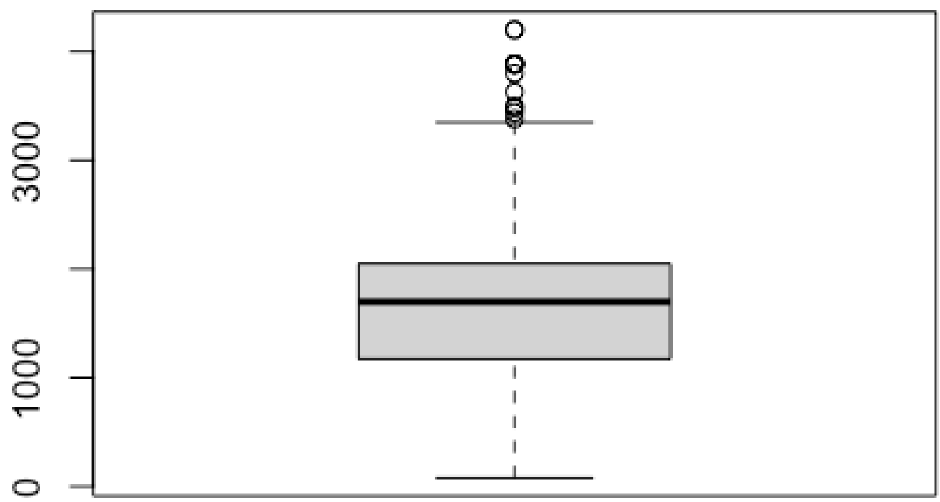
Finding the outliers in numerical variables

```
boxplot(df$Days)
boxplot(df$Days)$out
```

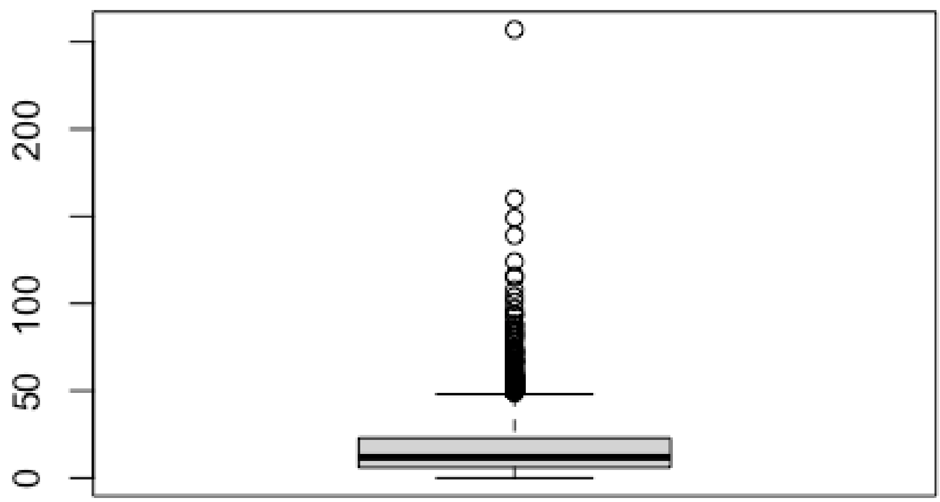



```
##   [1]  1  7  8  8  7  7  1  8  7  7  7  8  7  7  8  8  7  7 11  1  1  7  2
7  7
##  [26]  7  7  7  7  7  7  7  7  1  2  2  2  2  1  1  1  1  1  7  7  7  7  7
8 10
##  [51]  1  8  7  2  2  2  8  8  2  2  1  1  1  1  1  2  7  8  8  1  1  2  7
7  7
##  [76]  7  8  9  7  1  2  2  2  7  7  2  2  2  2  2  2  1  1  2  7  1  7  8
7  8
## [101]  7  7  7  7  2  2  7  2  2  2  1  1  1  7  7  1  7  7  7  7  7  7  2
2  2
## [126]  2  2  2  2  1  1  1  7  7  7  7  7  2  1  1  1  1  1  2  1  1  1  7
1  1
## [151]  7  2  1  1  2  2  1  1  2  2  2  2  2  2  2  2  2  2  2  8  1  1  1
1  2
## [176]  2 10  2  2 10  1  2  2  2  2  8  1  1  1  1  2  2  2  7  7  7  7  1
10  2
## [201]  2  2  2  1  1  2  2  8  1  2  7  7  7  7  1  7  2  2  2  2  7  8 10
1  1
## [226]  1  2  2  2  1  1  1  1  1  8  8  1  7  7  7  8 12  1  2  7  7  7  7
1  2
## [251]  2  7  1  2  2  7  1  1  9  1  1  1  2  2  2  7  7  7  7  7  7  8  7
7  7
## [276]  8  8  2  7  7  8 11  7  1  7  7  7  7  7  8  8  8  9  7  7  7  8  8
```

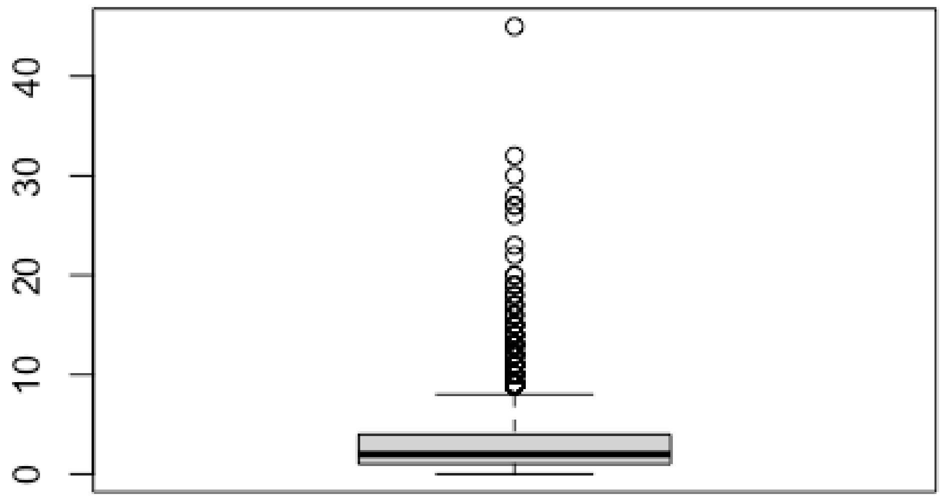
```
7  7
## [301]  7  8  7  7  7 11 11 11 11 11 11 11 11  1  8  7  7  7  7  9  9  9  8
7  7
## [326]  7  7  1

boxplot(df$Tuition)
boxplot(df$Tuition)$out
```



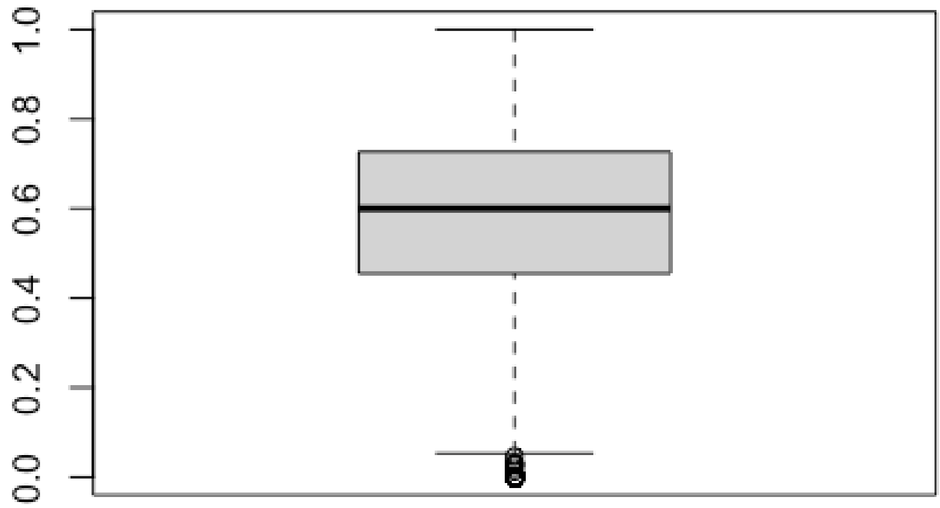

```
##  [1] 3379 3479 3628 4200 4199 3884 3884 3884 3884 3884 3884 3884 3799 3439
3499

boxplot(df$FRP.Active)
boxplot(df$FRP.Active)$out
```




```
## [1]  51  56  53  50  78  72  97  70  50  94  65  88  50  58 149  49  54
64
## [19]  51  72  54  81  50  57  73 257  83 104  85  55  58  54  55 116  61
71
## [37]  58  67  66  58  88  54  52  50  55  61  51  56  50  57 124  54  64
68
## [55]  53  74  49 139 101 160  68  53  51  56 108  83  59  81  86  83  59
67
## [73]  67  55  50  50  94  55  67  67  55  72  52  54  62  57  59  51  50
88
## [91] 115  62  58  71  57  52  51  59  52  74  93  51  80  50  52  75  55

boxplot(df$FRP.Cancelled)
boxplot(df$FRP.Cancelled)$out
```

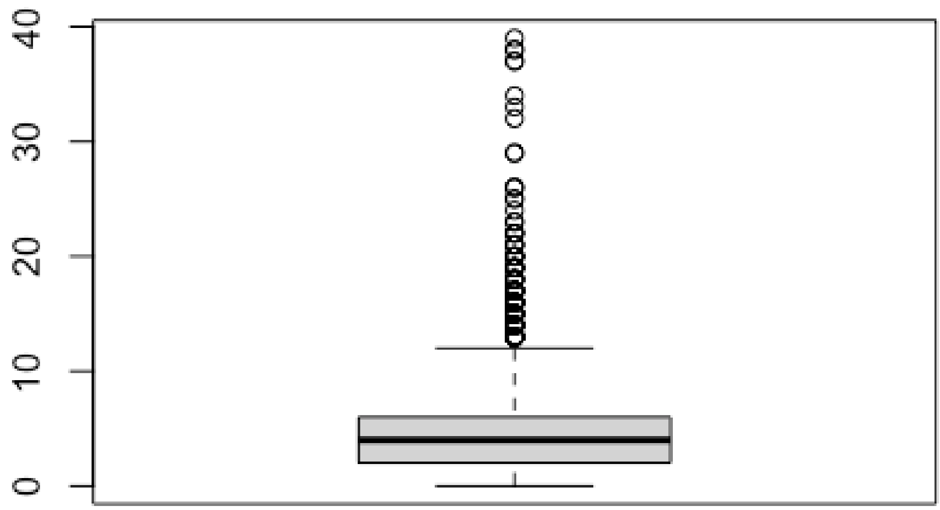



```
##   [1]  9  9  9 13 30 10 11 18 11  9 12 22 15  9 10  9 11 13 10 12 14 14 11
9 13
##  [26] 11 12 10 15  9  9 10 11 11 17  9 10 10 12 32  9 15 18 11  9 20 23 11
9 15
##  [51]  9 13 10  9 14 11  9  9 18 11 10 11 11 13 10 10 12 13 10 10 10  9 12
15  9
##  [76] 13 13 11 13 16 13  9  9 11 19 12  9 11  9  9  9 11 11 17 11 13 13 10
28 13
## [101] 15 11  9 11  9 20 11  9 13 17 10 10  9 13 12 10 12 10 12  9 11 14  9
14 13
## [126] 12  9 11 12 13 14  9  9 12  9 16 10  9 10 17 12 45 11 10  9 15 13 27
12 19
## [151] 16 11 11  9  9 11 12 14  9 11 15 12 11 17 15 11  9 10 10  9 10 11 10
26  9
## [176] 15 10  9 19 10 13 13 13  9  9 16 10 13 11 27 10 14 10  9 10 14 14  9
9

boxplot(df$FRP.Take.up.percent.)
boxplot(df$FRP.Take.up.percent.)$out
```



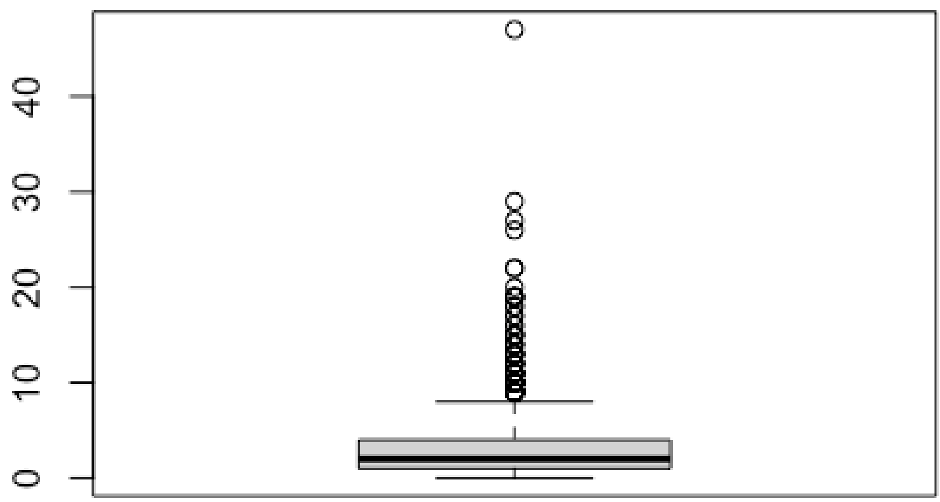

```
##   [1] 0.000 0.000 0.000 0.031 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000
##  [13] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000
##  [25] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000
##  [37] 0.028 0.000 0.000 0.000 0.000 0.000 0.029 0.000 0.046 0.000 0.000 0.
000
##  [49] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.028 0.000 0.000 0.
000
##  [61] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000
##  [73] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000
##  [85] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000
##  [97] 0.013 0.000 0.000 0.000 0.020 0.000 0.000 0.000 0.000 0.000 0.000 0.
000
## [109] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.
000
## [121] 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000 0.000
```

```
boxplot(df$Cancelled.Pax)
boxplot(df$Cancelled.Pax)$out
```

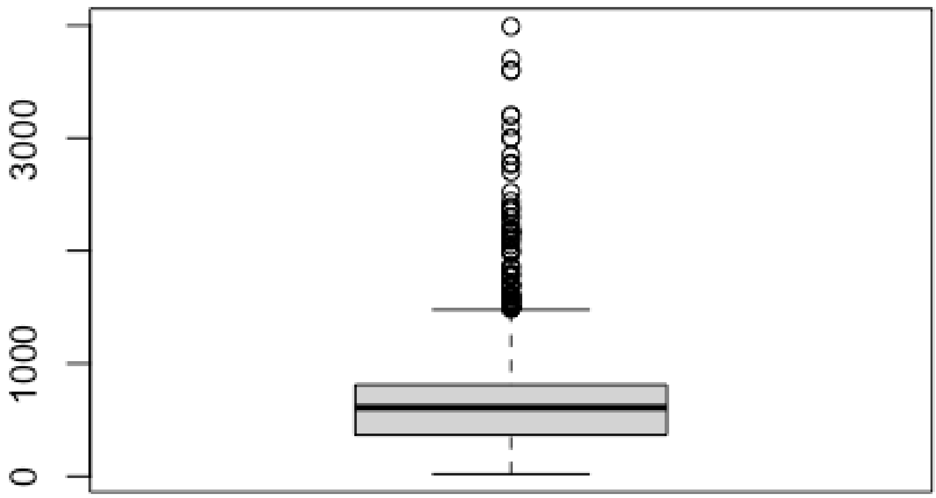



```
##   [1] 15 37 14 21 14 20 21 14 14 13 14 17 19 15 20 13 13 15 17 17 19 15 26
25 17
##  [26] 16 14 21 20 25 14 26 19 21 13 14 18 15 16 23 13 15 14 14 16 15 15 16
19 14
##  [51] 18 17 13 15 26 16 13 14 18 20 17 16 13 37 16 13 19 14 19 17 33 18 17
16 20
##  [76] 29 15 16 13 18 14 22 22 17 14 19 13 13 15 14 13 14 16 15 20 16 19 20
13 16
## [101] 24 13 17 38 29 17 22 39 13 20 16 14 16 15 15 13 20 13 17 17 17 17 14
13 13
## [126] 18 19 34 22 23 18 13 13 19 16 15 22 16 21 32 16 18 38 14 17
```

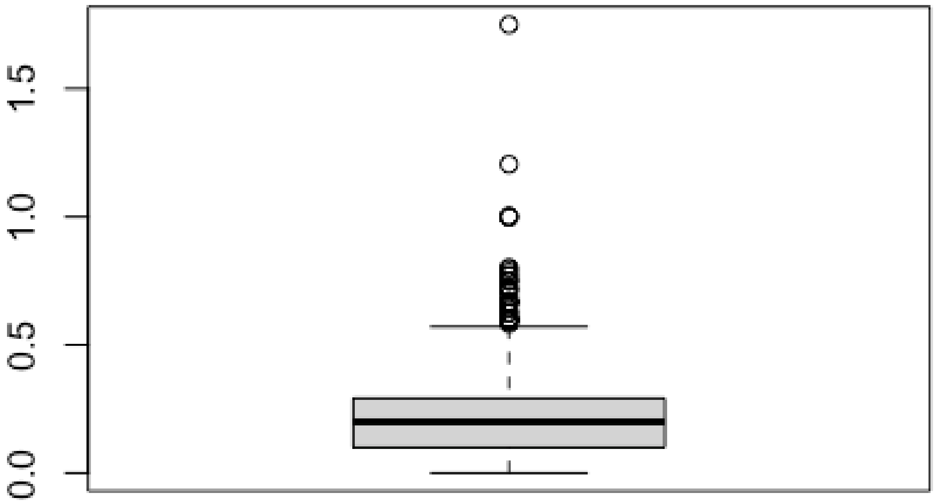
```
boxplot(df$Total.Discount.Pax)
boxplot(df$Total.Discount.Pax)$out
```



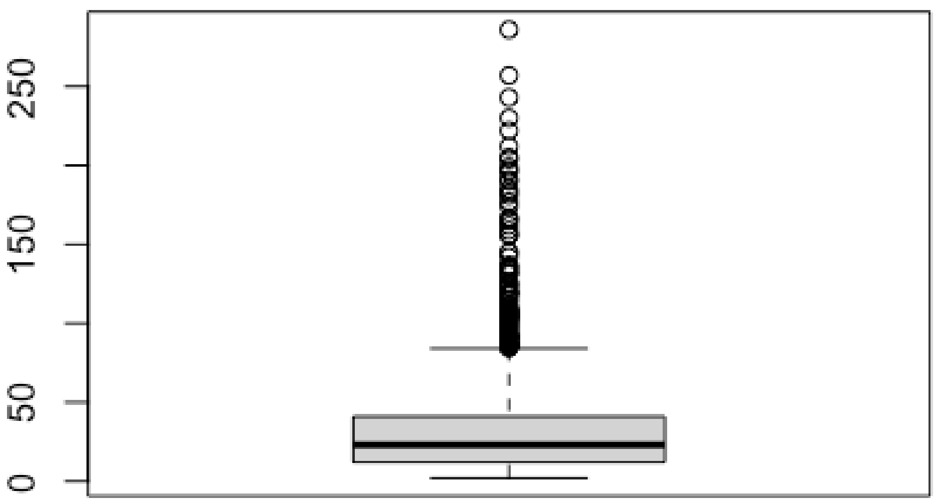

```
##  [1] 12 10 14  9 13 13 10 11 10 13 10 12 10 10 11 11 17 22  9 19 26 11  9
12  9
## [26] 10  9 10 18 10 11  9 29 12 13 19 11 11 18 22 11 10 19 15 11  9  9  9
9 15
## [51] 14 15  9 17 10 13 27 47 10  9 10 16 15  9 12 20 12  9 19  9 10  9 15
16 10
## [76]  9  9 10  9 12 19 10 13  9 11 10 14 12 14  9 13 19 11 10 11 10

boxplot(df$Total.School.Enrollment)
boxplot(df$Total.School.Enrollment)$out
```



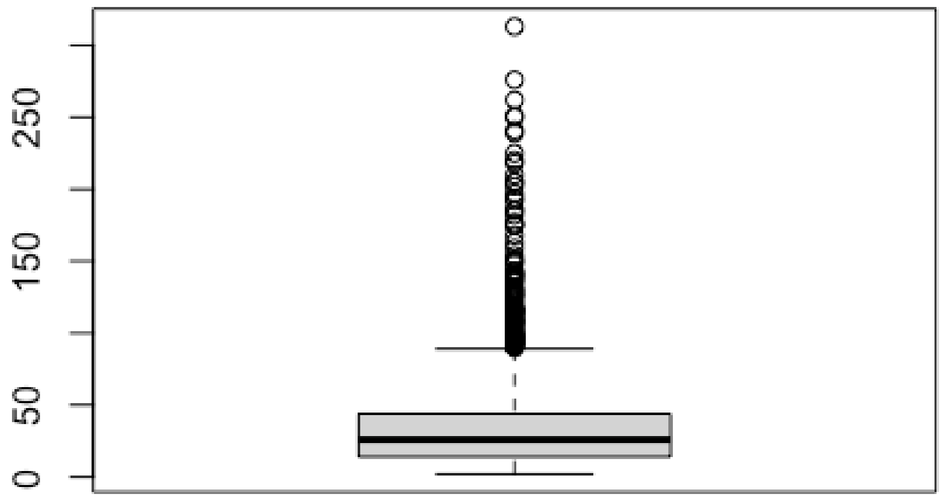

```
##  [1] 1693 2393 1688 1555 2159 1590 1514 1792 2127 1500 2778 3600 1853 3200
3200
## [16] 1693 1602 1600 2098 1538 1606 1611 1554 1625 2200 2850 1700 3990 2050
3100
## [31] 2000 1500 1500 2300 1974 1559 3600 1494 1700 1558 2175 2000 1528 1486
1587
## [46] 2441 1500 3700 1785 3000 2169 2700 2765 2169 1800 1769 1497 3000 1500
1558
## [61] 1563 1712 2120 2165 2520 2235 2087 2375 2351 1672 1500 2000 1871 2300
2168
## [76] 1606 1844

boxplot(df$EZ.Pay.Take.Up.Rate)
boxplot(df$EZ.Pay.Take.Up.Rate)$out
```



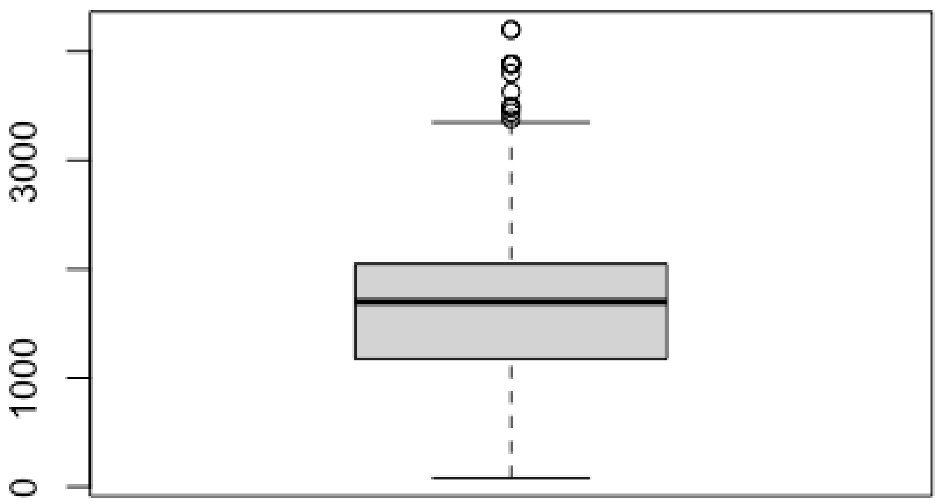

```
##  [1] 1.000 1.205 0.750 0.773 0.714 0.667 0.600 0.667 0.667 0.600 0.750 0.6
25
## [13] 0.722 0.750 1.000 0.667 0.600 0.667 0.625 0.667 0.667 0.600 0.643 0.5
91
## [25] 0.625 0.583 0.600 0.667 0.600 1.750 0.786 0.692 0.667 0.769 0.600 0.7
14
## [37] 0.750 0.667 0.594 1.000 0.600 1.000 0.722 0.600 0.800 0.600 0.800

boxplot(df$FPP)
boxplot(df$FPP)$out
```



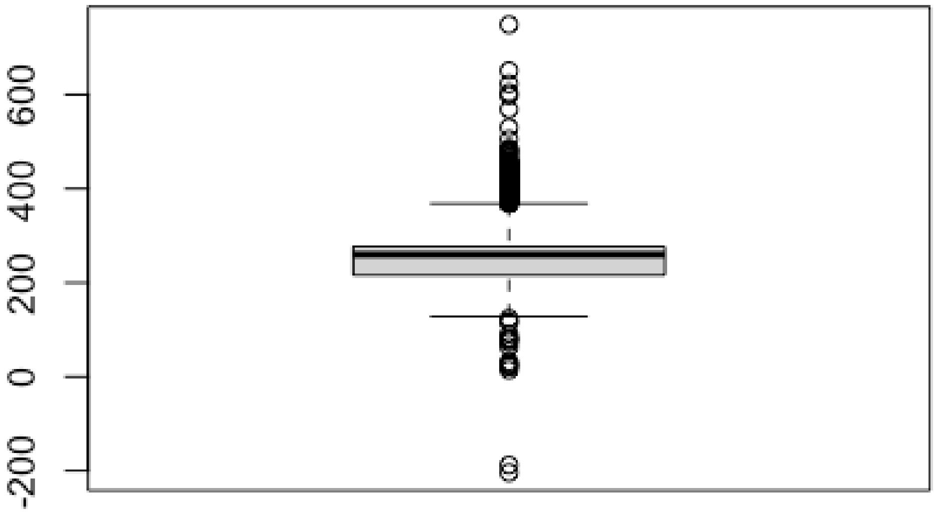

```
##   [1] 156 185  98 105 145 110 137  92 115 101 181 119 108  91 103 106  89
87
##  [19] 183 257 199 177 156 104 104 107 174  85  87  98  86  90  94  89 119
100
##  [37] 212 191  91  93  89  85 132 104  91 196 118  95 103 162 115  86  87
119
##  [55] 190 145 222 109  86 107  85 286 108 204 112 126 100 132 165 104 123
167
##  [73]  92 108 125 230  97 123 101  86 102 243 142 197 120  92 205 159  95
97
##  [91] 105 135 131 156 119  91  85 110 125 120  93  87  88  85 135 166  92
132
## [109] 106 108 136

boxplot(df$Total.Pax)
boxplot(df$Total.Pax)$out
```




```
##   [1] 163 197 108 112 159 123 150 102 126 109 194  94 131 118  99 113 117
106
##  [19]  95 205 276 225 186 168 109 113 117 192  97 106  94  96 101 100 126
108
##  [37] 241 210  99 100  96  96 136 122  97 218 118 114 107 177 126  91 102
127
##  [55] 205 154 239 115  92 110  97  92 313 115 251 116 129 110 141 175  90
112
##  [73] 139 182  98 117 137 250 104 135 110  93 105 262 151 202 130  98 220
175
##  [91] 102  92  90 104  93 114 143 143 175 129  98  94 120 139 132 107  95
96
## [109]  94 148 185  95 140 116 115 141  92

boxplot(df$SPR.Group.Revenue)
boxplot(df$SPR.Group.Revenue)$out
```



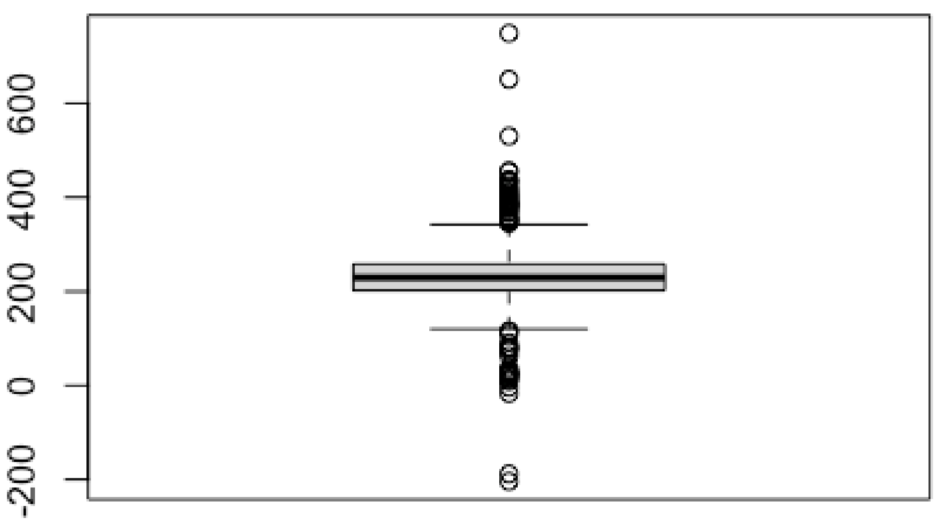

```
## [1] 3379 3479 3628 4200 4199 3884 3884 3884 3884 3884 3884 3884 3799 3439
3499

boxplot(df$DifferenceTraveltoFirstMeeting)
boxplot(df$DifferenceTraveltoFirstMeeting)$out
```




```
##   [1]  423  124   91   63 -204  392   32  380  380  399  369  404   28  37
7  369
##  [16]  388  395  382  116   80 -188  394  412  374  438  389  384  417  37
2  417
##  [31]  402  375  397  404  436  377  433  412  378  397  391  377  400  39
7  377
##  [46]  383  424  421  406  369  389  440  374  569  370  390  431  369  43
4  451
##  [61]  377  393  386  409  419  374  379  374  401  375  370  369  391  37
5  116
##  [76]  393  372  604  455  436   14  406  456  376  382  394  381  411  43
8   82
##  [91]  389  376  410  383  399  385  382  370  377  385  414  407  433  37
7  379
## [106]  749  407  378  425  397  386  408  379  406  377  407  400  384  42
8  468
## [121]  461  374  401   78  372  380  408  389  431  375  396  383  438  45
2  374
## [136]  381  412  395  419  444  383  431  418  380  382  389  466  391  42
4  410
## [151]  385  448  412  391  411  391  384  418  432  411  391  439  383  39
1  413
## [166]   22  378  399  385  428  391  378  407  379  371  390  385  389  38
```

```
5  399
## [181]  386  455  391  378  399  390  432  425  453  384  414  376  412  45
6  426
## [196]  440  392  399  399  422  414  505  401  403   22  460  390  394  39
7  403
## [211]  402  388  445  390  405  384  409  451  382  383  424  369  376  40
3  384
## [226]  384  459  424  438  403  369  384   25  452  440  439  399  412  38
5  399
## [241]  384  448  399  390  404   82  475  453  483  651  384  623  403  37
7  394
## [256]  408  426  424  418  388  395  394  477  452   73  403  396  598  48
0  437
## [271]  396  419  398  424  391  412  530  417  393  425  412  474  404  39
8  419
## [286]  390  369  456  427  426  379  414  458  420  485  428  399  485  40
6  394
## [301]  410  396  390  426  416  405  419
```

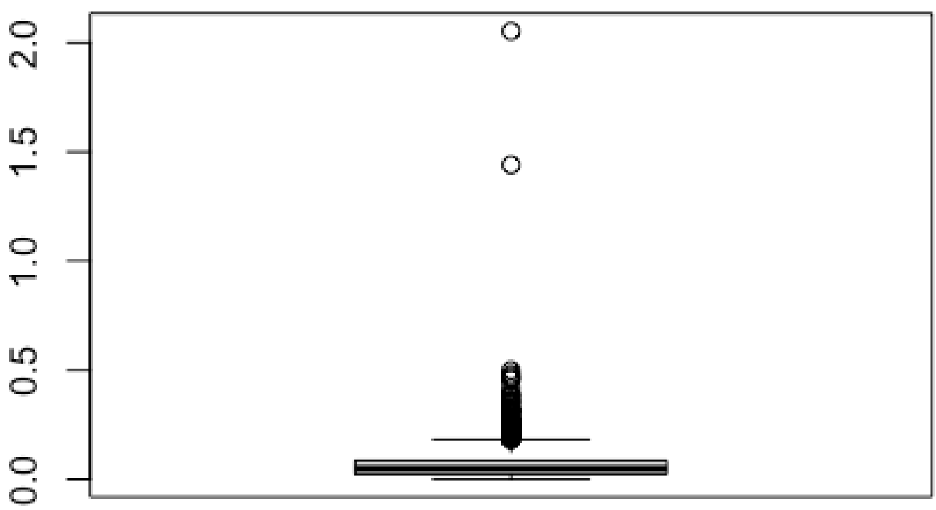
```
boxplot(df$DifferenceTraveltoLastMeeting)
```




```
boxplot(df$DifferenceTraveltoLastMeeting)$out
```

```
##  [1]   91   63   38 -204  392   32   28  109  345  116   80  352 -188  389
384
## [16]  354  402  367   -4  356  354  347  377  348  368   24  -17  434  116
455
## [31]   14   82   15  377  749  358   78  389  381  419  383  410  432  383
22
## [46]  391   14  376   23   22  402  438  369   25  385   82  651  403   30
395
## [61]   73  437  419  530  425  412  398   27  456    9  410
```

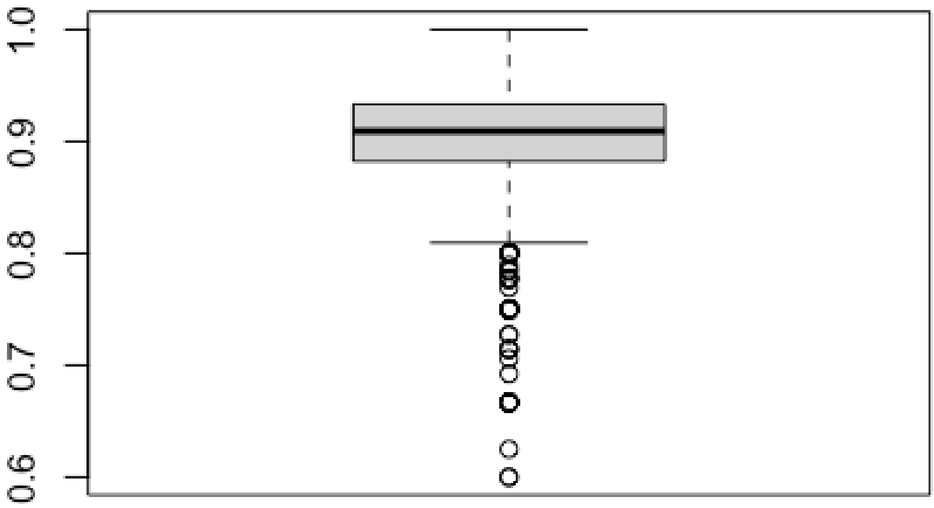
```
boxplot(df$FPP.to.School.enrollment)
boxplot(df$FPP.to.School.enrollment)$out
```




```
##   [1] 0.2700730 0.4583333 0.1956522 0.1924686 0.1962457 0.2032193 0.386752
1
##   [8] 0.3375000 0.2213930 0.2033333 0.2725345 0.3161290 0.2694064 0.259090
9
##  [15] 0.2189474 0.2440393 0.2057143 0.2709677 0.2071429 0.2560000 0.211428
6
##  [22] 0.2537313 0.2000000 0.3183183 0.4807692 0.1903485 0.2623626 0.235714
3
##  [29] 0.1851852 0.2188235 0.2368421 0.2523364 0.4583333 0.3888889 0.185628
7
```

```
##  [36] 0.1866667 0.2025316 0.1954887 0.2204082 0.3136364 0.2520000 0.366666
7
##  [43] 0.3241758 0.2202381 0.2280702 2.0526316 0.3161290 0.1863118 0.228169
0
##  [50] 0.3800000 0.2875000 0.2000000 0.1934307 0.2470238 0.1836735 0.216560
5
##  [57] 0.3057851 0.2235294 0.2059701 1.4400000 0.2380952 0.1868687 0.200000
0
##  [64] 0.1977819 0.4705882 0.1925926 0.2709677 0.2863636 0.2860000 0.311450
4
##  [71] 0.2000000 0.1821429 0.2111111 0.2307692 0.2089552 0.3586957 0.311594
2
##  [78] 0.2241087 0.3586957 0.2000000 0.2300000 0.2460000 0.1857143 0.418604
7
##  [85] 0.3538462 0.1885246 0.3884615 0.1900000 0.2217153 0.2028571 0.230409
4
##  [92] 0.1923077 0.2243902 0.2252252 0.2159091 0.2781547 0.2571429 0.195454
5
##  [99] 0.3063584 0.1885246 0.2545455 0.3013699 0.2266667 0.2073171 0.247169
8
## [106] 0.2016949 0.3478261 0.2323944 0.2344214 0.2815315 0.2000000 0.223880
6
## [113] 0.2224880 0.1885714 0.1882353 0.1909548 0.2300000 0.2100000 0.304587
2
## [120] 0.3245614 0.1866667 0.2401747 0.1977401 0.2480000 0.1927083 0.205714
3
## [127] 0.2750000 0.2051282 0.1971154 0.2094595 0.2158730 0.1857143 0.186666
7
## [134] 0.2368421 0.5000000 0.2700000 0.2447257 0.1902439
```

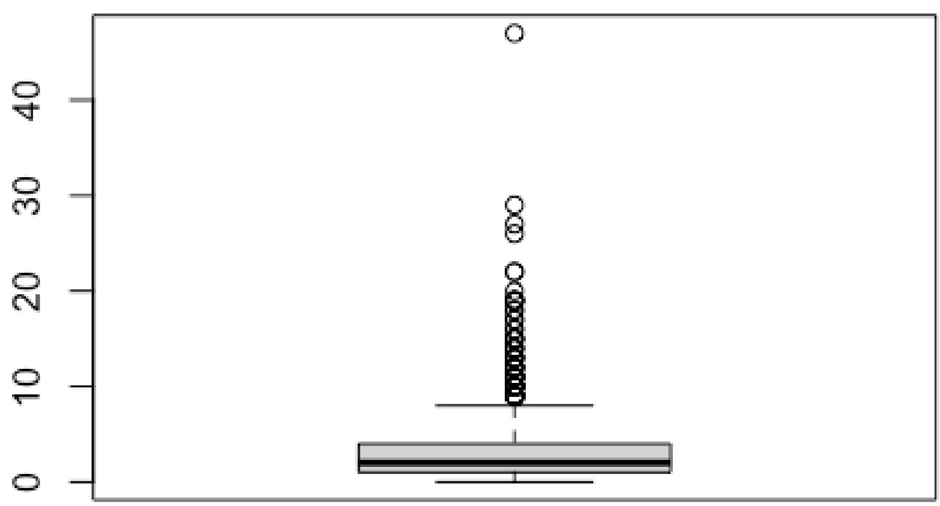
```
boxplot(df$FPP.to.PAX)
boxplot(df$FPP.to.PAX)$out
```




```
##   [1] 0.8000000 0.8000000 0.7500000 0.8000000 0.7857143 0.7142857 0.800000
0
##   [8] 0.7777778 0.7500000 0.7142857 0.6666667 0.8000000 0.8000000 0.783783
8
##  [15] 0.7894737 0.8000000 0.8000000 0.8000000 0.7142857 0.8000000 0.727272
7
##  [22] 0.8000000 0.8000000 0.7500000 0.7758621 0.8000000 0.7500000 0.666666
7
##  [29] 0.8000000 0.7500000 0.7500000 0.6923077 0.8000000 0.7777778 0.800000
0
##  [36] 0.7500000 0.7500000 0.7777778 0.7500000 0.8000000 0.7692308 0.750000
0
##  [43] 0.8000000 0.8000000 0.7500000 0.8000000 0.7500000 0.7500000 0.666666
7
##  [50] 0.8000000 0.7500000 0.7500000 0.8000000 0.7777778 0.8000000 0.600000
0
##  [57] 0.8000000 0.7500000 0.8000000 0.6666667 0.8000000 0.8000000 0.750000
0
##  [64] 0.8000000 0.8000000 0.6666667 0.7500000 0.7500000 0.6666667 0.750000
0
##  [71] 0.7500000 0.7500000 0.6000000 0.6250000 0.8000000 0.7500000 0.800000
0
##  [78] 0.7500000 0.8000000 0.8000000 0.7500000 0.7500000 0.7500000 0.800000
```

```
0
##  [85] 0.8000000 0.8000000 0.7500000 0.6666667 0.7500000 0.7500000 0.800000
0
##  [92] 0.8000000 0.7500000 0.8000000 0.7500000 0.8000000 0.7500000 0.714285
7
##  [99] 0.7500000 0.7500000 0.8000000 0.7272727 0.6666667 0.8000000 0.666666
7
## [106] 0.8000000 0.8000000 0.6666667 0.7500000 0.7500000 0.7500000 0.800000
0
## [113] 0.8000000 0.6666667 0.7777778 0.7500000 0.8000000 0.8000000 0.750000
0
## [120] 0.7500000 0.7500000 0.7058824 0.6666667 0.7500000 0.7500000 0.800000
0
## [127] 0.8000000 0.8000000 0.6666667
```

```
boxplot(df$Num.of.Non_FPP.PAX)
boxplot(df$Num.of.Non_FPP.PAX)$out
```




```
##  [1] 12 10 14  9 13 13 10 11 10 13 10 12 10 10 11 11 17 22  9 19 26 11  9
12  9
## [26] 10  9 10 18 10 11  9 29 12 13 19 11 11 18 22 11 10 19 15 11  9  9  9
9 15
## [51] 14 15  9 17 10 13 27 47 10  9 10 16 15  9 12 20 12  9 19  9 10  9 15
```

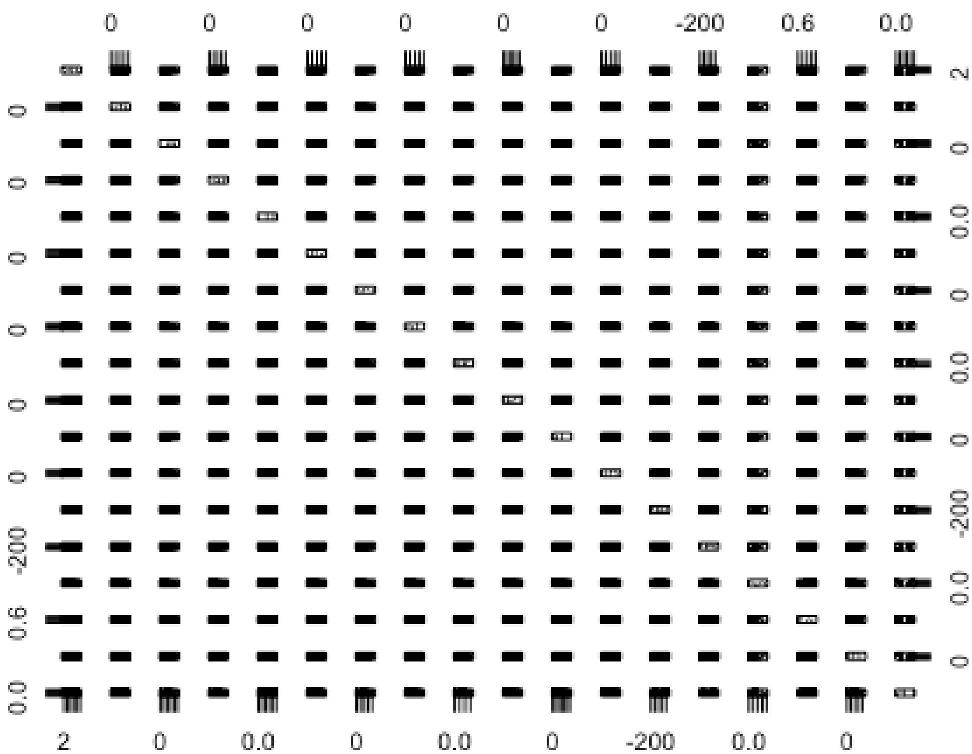
```
16 10
## [76]  9  9 10  9 12 19 10 13  9 11 10 14 12 14  9 13 19 11 10 11 10
```

Finding correlation between numerical variables and target variable

```
cors <- cor(df[,(colnames(df) %in% c('Days', 'Tuition','FRP.Active','FRP.Canc
elled',
                                     'FRP.Take.up.percent.','Cancelled.Pax','
Total.Discount.Pax','Total.School.Enrollment',
                                     'EZ.Pay.Take.Up.Rate','FPP','Total.Pax',
'SPR.Group.Revenue','DifferenceTraveltoFirstMeeting',
                                     'DifferenceTraveltoLastMeeting','FPP.to.
School.enrollment','FPP.to.PAX','Num.of.Non_FPP.PAX'))],
            df[,colnames(df) %in% c('Retained.in.2012.')])
kable(cors,col.names=c('Retained.in.2012.'))
```

	Retained.in.2012.
Days	-0.0492645
Tuition	-0.1188375
FRP.Active	0.2503010
FRP.Cancelled	0.0730723
FRP.Take.up.percent.	-0.0163190
Cancelled.Pax	0.0496069
Total.Discount.Pax	0.2161201
Total.School.Enrollment	0.0842149
EZ.Pay.Take.Up.Rate	-0.0187687
FPP	0.2608753
Total.Pax	0.2602503
SPR.Group.Revenue	-0.1188375
DifferenceTraveltoFirstMeeting	-0.1216149
DifferenceTraveltoLastMeeting	-0.0968051
FPP.to.School.enrollment	0.0703062
FPP.to.PAX	0.1446134
Num.of.Non_FPP.PAX	0.2161201

```
plot(df[,(colnames(df) %in% c('Days', 'Tuition','FRP.Active','FRP.Cancelled',
                              'FRP.Take.up.percent.','Cancelled.Pax','Total.D
iscount.Pax','Total.School.Enrollment',
                              'EZ.Pay.Take.Up.Rate','FPP','Total.Pax','SPR.Gr
oup.Revenue','DifferenceTraveltoFirstMeeting',
                              'DifferenceTraveltoLastMeeting','FPP.to.School.
enrollment','FPP.to.PAX','Num.of.Non_FPP.PAX','Retained.in.2012.'))])
```



Based on the correlation values, below are the important numerical variables:

FPP
Total.Pax
FRP.Active
Total.Discount.Pax
Num.of.Non_FPP.PAX FPP.to.PAX Total.School.Enrollment FRP.Cancelled
FPP.to.School.enrollment Cancelled.Pax

Converting target variable from numeric to factor

```
df$Retained.in.2012. <- as.factor(ifelse(df$Retained.in.2012. == 1,"HIGH","LO
W"))
```

Calculate chi-square values for categorical variables (Descending order of X-Squared, Higher = More important)

```
chisq.test(df$Program.Code, df$Retained.in.2012., correct=FALSE)
```

```
## Warning in chisq.test(df$Program.Code, df$Retained.in.2012., correct = FAL
SE):
## Chi-squared approximation may be incorrect
```

```
## 
##  Pearson's Chi-squared test
## 
## data:  df$Program.Code and df$Retained.in.2012.
## X-squared = 117.14, df = 27, p-value = 3.374e-13

chisq.test(df$From.Grade, df$Retained.in.2012., correct=FALSE)

## Warning in chisq.test(df$From.Grade, df$Retained.in.2012., correct = FALSE
## ):
## Chi-squared approximation may be incorrect

## 
##  Pearson's Chi-squared test
## 
## data:  df$From.Grade and df$Retained.in.2012.
## X-squared = 392.73, df = 9, p-value < 2.2e-16

chisq.test(df$To.Grade, df$Retained.in.2012., correct=FALSE)

## Warning in chisq.test(df$To.Grade, df$Retained.in.2012., correct = FALSE):
Chi-
## squared approximation may be incorrect

## 
##  Pearson's Chi-squared test
## 
## data:  df$To.Grade and df$Retained.in.2012.
## X-squared = 160.95, df = 9, p-value < 2.2e-16

chisq.test(df$Group.State, df$Retained.in.2012., correct=FALSE)

## Warning in chisq.test(df$Group.State, df$Retained.in.2012., correct = FALS
## E):
## Chi-squared approximation may be incorrect

## 
##  Pearson's Chi-squared test
## 
## data:  df$Group.State and df$Retained.in.2012.
## X-squared = 122.95, df = 52, p-value = 1.117e-07

chisq.test(df$Is.Non.Annual., df$Retained.in.2012., correct=FALSE)

## 
##  Pearson's Chi-squared test
## 
## data:  df$Is.Non.Annual. and df$Retained.in.2012.
## X-squared = 365.07, df = 1, p-value < 2.2e-16

chisq.test(df$Travel.Type, df$Retained.in.2012., correct=FALSE)
```

```
## Warning in chisq.test(df$Travel.Type, df$Retained.in.2012., correct = FALS
E):
## Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  df$Travel.Type and df$Retained.in.2012.
## X-squared = 16.063, df = 3, p-value = 0.001101

chisq.test(df$Special.Pay, df$Retained.in.2012., correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  df$Special.Pay and df$Retained.in.2012.
## X-squared = 26.297, df = 3, p-value = 8.266e-06

chisq.test(df$Poverty.Code, df$Retained.in.2012., correct=FALSE)

## Warning in chisq.test(df$Poverty.Code, df$Retained.in.2012., correct = FAL
SE):
## Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  df$Poverty.Code and df$Retained.in.2012.
## X-squared = 38.474, df = 5, p-value = 3.029e-07

chisq.test(df$Region, df$Retained.in.2012., correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  df$Region and df$Retained.in.2012.
## X-squared = 36.463, df = 5, p-value = 7.674e-07

chisq.test(df$CRM.Segment, df$Retained.in.2012., correct=FALSE)

## Warning in chisq.test(df$CRM.Segment, df$Retained.in.2012., correct = FALS
E):
## Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  df$CRM.Segment and df$Retained.in.2012.
## X-squared = 150.58, df = 10, p-value < 2.2e-16

chisq.test(df$School.Type, df$Retained.in.2012., correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$School.Type and df$Retained.in.2012.
## X-squared = 14.336, df = 3, p-value = 0.002481
```

```r
chisq.test(df$Parent.Meeting.Flag, df$Retained.in.2012., correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$Parent.Meeting.Flag and df$Retained.in.2012.
## X-squared = 0.9819, df = 1, p-value = 0.3217
```

```r
chisq.test(df$MDR.Low.Grade, df$Retained.in.2012., correct=FALSE)
```

```
## Warning in chisq.test(df$MDR.Low.Grade, df$Retained.in.2012., correct = FA
LSE):
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$MDR.Low.Grade and df$Retained.in.2012.
## X-squared = 93.134, df = 11, p-value = 4.044e-15
```

```r
chisq.test(df$MDR.High.Grade, df$Retained.in.2012., correct=FALSE)
```

```
## Warning in chisq.test(df$MDR.High.Grade, df$Retained.in.2012., correct = F
ALSE):
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$MDR.High.Grade and df$Retained.in.2012.
## X-squared = 82.672, df = 11, p-value = 4.48e-13
```

```r
chisq.test(df$Income.Level, df$Retained.in.2012., correct=FALSE)
```

```
## Warning in chisq.test(df$Income.Level, df$Retained.in.2012., correct = FAL
SE):
## Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$Income.Level and df$Retained.in.2012.
## X-squared = 85.559, df = 21, p-value = 9.312e-10
```

```r
chisq.test(df$School.Sponsor, df$Retained.in.2012., correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$School.Sponsor and df$Retained.in.2012.
## X-squared = 34.721, df = 1, p-value = 3.806e-09
```

```
chisq.test(df$SPR.Product.Type, df$Retained.in.2012., correct=FALSE)
```

```
## Warning in chisq.test(df$SPR.Product.Type, df$Retained.in.2012., correct =
## FALSE): Chi-squared approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$SPR.Product.Type and df$Retained.in.2012.
## X-squared = 64.122, df = 5, p-value = 1.704e-12
```

```
chisq.test(df$SPR.New.Existing, df$Retained.in.2012., correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$SPR.New.Existing and df$Retained.in.2012.
## X-squared = 324.19, df = 1, p-value < 2.2e-16
```

```
chisq.test(df$NumberOfMeetingswithParents, df$Retained.in.2012., correct=FALS
E)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$NumberOfMeetingswithParents and df$Retained.in.2012.
## X-squared = 8.1604, df = 2, p-value = 0.0169
```

```
chisq.test(df$SchoolGradeTypeLow, df$Retained.in.2012., correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$SchoolGradeTypeLow and df$Retained.in.2012.
## X-squared = 78.122, df = 3, p-value < 2.2e-16
```

```
chisq.test(df$SchoolGradeTypeHigh, df$Retained.in.2012., correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$SchoolGradeTypeHigh and df$Retained.in.2012.
## X-squared = 144.13, df = 3, p-value < 2.2e-16
```

```
chisq.test(df$SchoolGradeType, df$Retained.in.2012., correct=FALSE)
```

```
## Warning in chisq.test(df$SchoolGradeType, df$Retained.in.2012., correct =
## FALSE): Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  df$SchoolGradeType and df$Retained.in.2012.
## X-squared = 168.21, df = 8, p-value < 2.2e-16

chisq.test(df$DepartureMonth, df$Retained.in.2012., correct=FALSE)

## Warning in chisq.test(df$DepartureMonth, df$Retained.in.2012., correct = F
ALSE):
## Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  df$DepartureMonth and df$Retained.in.2012.
## X-squared = 86.099, df = 5, p-value < 2.2e-16

chisq.test(df$GroupGradeTypeLow, df$Retained.in.2012., correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  df$GroupGradeTypeLow and df$Retained.in.2012.
## X-squared = 87.573, df = 5, p-value < 2.2e-16

chisq.test(df$GroupGradeTypeHigh, df$Retained.in.2012., correct=FALSE)

##
##  Pearson's Chi-squared test
##
## data:  df$GroupGradeTypeHigh and df$Retained.in.2012.
## X-squared = 63.001, df = 3, p-value = 1.342e-13

chisq.test(df$GroupGradeType, df$Retained.in.2012., correct=FALSE)

## Warning in chisq.test(df$GroupGradeType, df$Retained.in.2012., correct = F
ALSE):
## Chi-squared approximation may be incorrect

##
##  Pearson's Chi-squared test
##
## data:  df$GroupGradeType and df$Retained.in.2012.
## X-squared = 121.86, df = 12, p-value < 2.2e-16

chisq.test(df$MajorProgramCode, df$Retained.in.2012., correct=FALSE)

##
##  Pearson's Chi-squared test
```

```
##
## data:  df$MajorProgramCode and df$Retained.in.2012.
## X-squared = 56.628, df = 3, p-value = 3.086e-12
```

```
chisq.test(df$SingleGradeTripFlag, df$Retained.in.2012., correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$SingleGradeTripFlag and df$Retained.in.2012.
## X-squared = 495.58, df = 1, p-value < 2.2e-16
```

```
chisq.test(df$SchoolSizeIndicator, df$Retained.in.2012., correct=FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df$SchoolSizeIndicator and df$Retained.in.2012.
## X-squared = 69.82, df = 3, p-value = 4.664e-15
```

Based on the chi-square values, below are the important categorical variables:

SingleGradeTripFlag From.Grade Is.Non.Annual. SPR.New.Existing SchoolGradeType To.Grade CRM.Segment SchoolGradeTypeHigh Group.State GroupGradeType Program.Code

Random Forest Model construction

```
# Identifying the best value of mtry using validation set
set.seed(222)
ind <- sample(2, nrow(df), replace = TRUE, prob = c(0.7, 0.3))
train <- df[ind==1,]
test <- df[ind==2,]

pr.err <- c()
for(mt in seq(1, sqrt(ncol(train) - 1)))
{
  rf <- randomForest(Retained.in.2012. ~., data = train, ntree = 100, mtry =
mt,
                     proximity = T, importance = T)
  pred <- predict(rf, newdata = test, type = "class")
  pr.err<- c(pr.err, mean(pred != test$Retained.in.2012.))
}

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

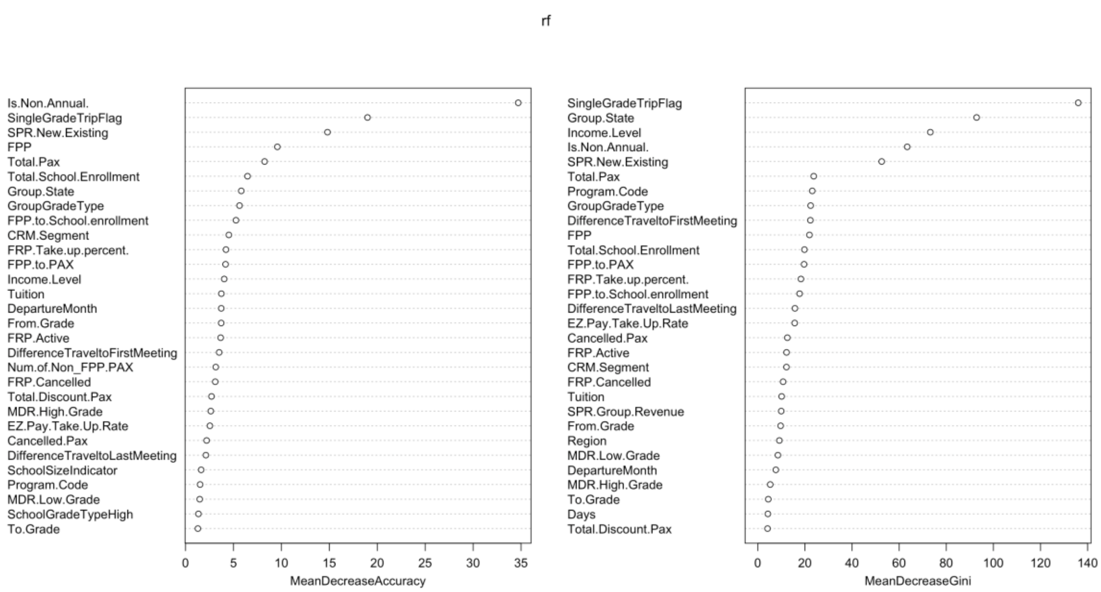
```
## [1] 5
```

```
rf <- randomForest(Retained.in.2012. ~., data = train, ntree = 100, mtry = be
stmtry,
                   proximity = T, importance = T)
p1 <- predict(rf, data = train)
```

```
p2<- predict(rf, newdata = test)

#we can find the accuracy of train and test data from the confusion matrix
confusionMatrix(data=p1, reference = train$Retained.in.2012.)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction HIGH LOW
##       HIGH  823 149
##       LOW   203 487
##
##                Accuracy : 0.7882
##                  95% CI : (0.7678, 0.8076)
##     No Information Rate : 0.6173
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5588
##
##  Mcnemar's Test P-Value : 0.004729
##
##             Sensitivity : 0.8021
##             Specificity : 0.7657
##          Pos Pred Value : 0.8467
##          Neg Pred Value : 0.7058
##              Prevalence : 0.6173
##          Detection Rate : 0.4952
##    Detection Prevalence : 0.5848
##       Balanced Accuracy : 0.7839
##
##        'Positive' Class : HIGH
##

confusionMatrix(data=p2, reference = test$Retained.in.2012.)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction HIGH LOW
##       HIGH  325  66
##       LOW   100 235
##
##                Accuracy : 0.7713
##                  95% CI : (0.739, 0.8014)
##     No Information Rate : 0.5854
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.5366
##
##  Mcnemar's Test P-Value : 0.01043
```

```
##
##             Sensitivity : 0.7647
##             Specificity : 0.7807
##          Pos Pred Value : 0.8312
##          Neg Pred Value : 0.7015
##              Prevalence : 0.5854
##          Detection Rate : 0.4477
##    Detection Prevalence : 0.5386
##       Balanced Accuracy : 0.7727
##
##        'Positive' Class : HIGH
##
```

```
# Function to calculate Evaluation Measures
evaluation.measure <- function(actual, prediction)
{
  y <- as.vector(table(actual,prediction))
  names(y) <- c("TP","FN","FP","TN")
  Accuracy <- (y["TP"]+y["TN"])/sum(y)
  Error <- 1- Accuracy
  Recall <- ((y["TP"])/(y["TP"]+y["FN"])) * (y["TP"]+y["FN"])
  em <- c(Accuracy, Error, Recall)
  return(em)
}

#varImpPlot(rf)

knitr::include_graphics("plot_zoom_png")
```



Decision tree construction based on pruning parameters minsplit, minbucket and CP

```
set.seed(96)
```

MODEL 1: 70:30 SPLIT

```
#Using the index function to assign 1&2 to the observations in the dataset na
med data.
index <- sample(2, nrow(df), replace = T, prob = c(0.7,0.3))

#selecting index 1 for training data
train <- df[index == 1,]

#selecting index 2 for testing data
test <- df[index == 2,]

#Creating formula with all the Retained.in.2012. variables using ., to serve
as an input parameter to rpart.
MyFormula = Retained.in.2012. ~.

#Develop a decision tree.
mytree_70_30_basic <- rpart(MyFormula, data=train)

#Predict function to predict the classes for the decision tree mytree_70_30_b
asic for training data.
mytree_train_predict_70_30 <- predict(mytree_70_30_basic, data = train , type
= "class")

#Calculating the training error by comparing predicted classes with Retained.
in.2012. variable of original dataset.
mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 != train$Retained
.in.2012.)

#Predict function to predict the classes for the decision tree mytree_70_30 f
or testing data.
mytree_test_predict_70_30 <- predict(mytree_70_30_basic, newdata = test, type
= "class")

#Calculating the testing error by comparing predicted classes with Retained.i
n.2012. variable of original dataset.
mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 != test$Retained.in
.2012.)

#Calculating the performance of the model by finding the difference between t
he test error & train data.
diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30
print(diff_70_30)

## [1] 0.05485196
```

Based on summary, using with CP = 0.01000000 for the least xerror

APPLYING PARAMETER VALUES TO ARRIVE AT BETTER PERFORMANCE FOR MODEL 2: 70-30 SPLIT

Creating vectors for minsplit and minbucket values to be used for different combinations to test performance.

```
msplt <- c(12,48,102)
mbckt <- c(4,16,34)

for (i in msplt)
{
  for (j in mbckt)
  {
    mytree_70_30 <- rpart(MyFormula, data = train,control = rpart.control (mi
nsplit = i,minbucket = j, cp = 0.01000000))

    mytree_train_predict_70_30 <- predict(mytree_70_30, data = train , type =
"class")

    mytree_train_error_70_30 <- mean(mytree_train_predict_70_30 != train$Reta
ined.in.2012.)

    mytree_test_predict_70_30 <- predict(mytree_70_30, newdata = test, type =
"class")

    mytree_test_error_70_30 <- mean(mytree_test_predict_70_30 != test$Retaine
d.in.2012.)

    diff_70_30 = mytree_test_error_70_30 - mytree_train_error_70_30
    diff_70_30
    print(diff_70_30)

    ############################## Confusion Matrix for 70:30 Split #########
########################################

    cfmt <- table(train$Retained.in.2012.,mytree_train_predict_70_30)
    print(cfmt)

    fp = cfmt[2,1]
    fn = cfmt[1,2]
    tn = cfmt[2,2]
    tp = cfmt[1,1]

    #Calculating precision by dividing true positive with the sum of true pos
itive and false positive.
    precision_train =  (tp)/(tp+fp)
    accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
```

```
    recall_train = (tp)/(tp+fn)
    fscore_train = (2*(recall_train*precision_train))/(recall_train+precision
_train)

    cfmt <- table(test$Retained.in.2012.,mytree_test_predict_70_30)
    print(cfmt)

    fp = cfmt[2,1]
    fn = cfmt[1,2]
    tn = cfmt[2,2]
    tp = cfmt[1,1]

    #Calculating precision by dividing true positive with the sum of true pos
itive and false positive.
    precision_test =  (tp)/(tp+fp)
    accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
    recall_test = (tp)/(tp+fn)
    fscore_test = (2*(recall_test*precision_test))/(recall_test+precision_tes
t)

    # Printing the values for train data error, test data error, performance
and other parameters.
    print(paste("Train data error: ", mytree_train_error_70_30))
    print(paste("Test data error: ", mytree_test_error_70_30))
    print(paste("Difference/performance", diff_70_30))
    print(paste("precision of training data: ", precision_train))
    print(paste("accuracy of training data: ", accuracymodel_train))
    print(paste("recall of training data: ", recall_train))
    print(paste("F-score of training data: ", fscore_train))
    print(paste("precision of test data: ", precision_test))
    print(paste("accuracy of test data: ", accuracymodel_test))
    print(paste("recall of test data: ", recall_test))
    print(paste("F-score of test data: ", fscore_test))
  }
}

## [1] 0.05485196
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  866 129
##   LOW   145 500
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  384  72
##   LOW    94 198
## [1] "Train data error:  0.167073170731707"
## [1] "Test data error:  0.22192513368984"
## [1] "Difference/performance 0.0548519629581322"
## [1] "precision of training data:  0.856577645895153"
## [1] "accuracy of training data:  0.832926829268293"
```

```
## [1] "recall of training data:  0.87035175879397"
## [1] "F-score of training data:  0.863409770687936"
## [1] "precision of test data:  0.803347280334728"
## [1] "accuracy of test data:  0.77807486631016"
## [1] "recall of test data:  0.842105263157895"
## [1] "F-score of test data:  0.822269807280514"
## [1] 0.04790987
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  857 138
##   LOW   143 502
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  384  72
##   LOW    92 200
## [1] "Train data error:  0.171341463414634"
## [1] "Test data error:  0.219251336898396"
## [1] "Difference/performance 0.0479098734837616"
## [1] "precision of training data:  0.857"
## [1] "accuracy of training data:  0.828658536585366"
## [1] "recall of training data:  0.861306532663317"
## [1] "F-score of training data:  0.859147869674185"
## [1] "precision of test data:  0.80672268907563"
## [1] "accuracy of test data:  0.780748663101604"
## [1] "recall of test data:  0.842105263157895"
## [1] "F-score of test data:  0.824034334763948"
## [1] 0.04108517
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  866 129
##   LOW   161 484
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  391  65
##   LOW    98 194
## [1] "Train data error:  0.176829268292683"
## [1] "Test data error:  0.217914438502674"
## [1] "Difference/performance 0.0410851702099909"
## [1] "precision of training data:  0.843232716650438"
## [1] "accuracy of training data:  0.823170731707317"
## [1] "recall of training data:  0.87035175879397"
## [1] "F-score of training data:  0.856577645895153"
## [1] "precision of test data:  0.79959100204499"
## [1] "accuracy of test data:  0.782085561497326"
## [1] "recall of test data:  0.857456140350877"
## [1] "F-score of test data:  0.827513227513227"
## [1] 0.05485196
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  866 129
```

```
##   LOW   145 500
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  384  72
##   LOW    94 198
## [1] "Train data error:  0.167073170731707"
## [1] "Test data error:  0.22192513368984"
## [1] "Difference/performance 0.0548519629581322"
## [1] "precision of training data:  0.856577645895153"
## [1] "accuracy of training data:  0.832926829268293"
## [1] "recall of training data:  0.87035175879397"
## [1] "F-score of training data:  0.863409770687936"
## [1] "precision of test data:  0.803347280334728"
## [1] "accuracy of test data:  0.77807486631016"
## [1] "recall of test data:  0.842105263157895"
## [1] "F-score of test data:  0.822269807280514"
## [1] 0.04790987
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  857 138
##   LOW   143 502
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  384  72
##   LOW    92 200
## [1] "Train data error:  0.171341463414634"
## [1] "Test data error:  0.219251336898396"
## [1] "Difference/performance 0.0479098734837616"
## [1] "precision of training data:  0.857"
## [1] "accuracy of training data:  0.828658536585366"
## [1] "recall of training data:  0.861306532663317"
## [1] "F-score of training data:  0.859147869674185"
## [1] "precision of test data:  0.80672268907563"
## [1] "accuracy of test data:  0.780748663101604"
## [1] "recall of test data:  0.842105263157895"
## [1] "F-score of test data:  0.824034334763948"
## [1] 0.04108517
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  866 129
##   LOW   161 484
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  391  65
##   LOW    98 194
## [1] "Train data error:  0.176829268292683"
## [1] "Test data error:  0.217914438502674"
## [1] "Difference/performance 0.0410851702099909"
## [1] "precision of training data:  0.843232716650438"
## [1] "accuracy of training data:  0.823170731707317"
```

```
## [1] "recall of training data:  0.87035175879397"
## [1] "F-score of training data:  0.856577645895153"
## [1] "precision of test data:  0.79959100204499"
## [1] "accuracy of test data:  0.782085561497326"
## [1] "recall of test data:  0.857456140350877"
## [1] "F-score of test data:  0.827513227513227"
## [1] 0.05485196
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  866 129
##   LOW   145 500
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  384  72
##   LOW    94 198
## [1] "Train data error:  0.167073170731707"
## [1] "Test data error:  0.22192513368984"
## [1] "Difference/performance 0.0548519629581322"
## [1] "precision of training data:  0.856577645895153"
## [1] "accuracy of training data:  0.832926829268293"
## [1] "recall of training data:  0.87035175879397"
## [1] "F-score of training data:  0.863409770687936"
## [1] "precision of test data:  0.803347280334728"
## [1] "accuracy of test data:  0.77807486631016"
## [1] "recall of test data:  0.842105263157895"
## [1] "F-score of test data:  0.822269807280514"
## [1] 0.04790987
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  857 138
##   LOW   143 502
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  384  72
##   LOW    92 200
## [1] "Train data error:  0.171341463414634"
## [1] "Test data error:  0.219251336898396"
## [1] "Difference/performance 0.0479098734837616"
## [1] "precision of training data:  0.857"
## [1] "accuracy of training data:  0.828658536585366"
## [1] "recall of training data:  0.861306532663317"
## [1] "F-score of training data:  0.859147869674185"
## [1] "precision of test data:  0.80672268907563"
## [1] "accuracy of test data:  0.780748663101604"
## [1] "recall of test data:  0.842105263157895"
## [1] "F-score of test data:  0.824034334763948"
## [1] 0.04108517
##       mytree_train_predict_70_30
##        HIGH LOW
##   HIGH  866 129
```

```
##   LOW   161 484
##       mytree_test_predict_70_30
##        HIGH LOW
##   HIGH  391  65
##   LOW    98 194
## [1] "Train data error:  0.176829268292683"
## [1] "Test data error:  0.217914438502674"
## [1] "Difference/performance 0.0410851702099909"
## [1] "precision of training data:  0.843232716650438"
## [1] "accuracy of training data:  0.823170731707317"
## [1] "recall of training data:  0.87035175879397"
## [1] "F-score of training data:  0.856577645895153"
## [1] "precision of test data:  0.79959100204499"
## [1] "accuracy of test data:  0.782085561497326"
## [1] "recall of test data:  0.857456140350877"
## [1] "F-score of test data:  0.827513227513227"
```

MODEL 3: 80:20 SPLIT

```
#Assigning 1 & 2 as index to split test and train data
set.seed(96)
index <- sample(2, nrow(df), replace = T, prob = c(0.8,0.2))

#selecting index 1 for training
train <- df[index == 1,]

#selecting index 1 for testing
test <- df[index == 2,]

#Creating formula with all the Retained.in.2012. variables using ., to serve
as an input parameter to rpart.
MyFormula = Retained.in.2012.~.

mytree_80_20_basic = rpart(MyFormula, data=train)

#Predict function to predict the classes for the decision tree mytree_80_20_b
asic for training data.
mytree_train_predict_80_20 <- predict(mytree_80_20_basic, data = train , type
= "class")

#Calculating the training error by comparing predicted classes with Retained.
in.2012. variable of original dataset.
mytree_train_error_80_20 <- mean(mytree_train_predict_80_20 != train$Retained
.in.2012.)

#Predict function to predict the classes for the decision tree mytree_80_20 f
or testing data.
mytree_test_predict_80_20 <- predict(mytree_80_20_basic, newdata = test, type
= "class")
```

```
#Calculating the testing error by comparing predicted classes with Retained.i
n.2012. variable of original dataset.
mytree_test_error_80_20 <- mean(mytree_test_predict_80_20 != test$Retained.in
.2012.)

#Calculating the performance of the model by finding the difference between t
he test error & train data.
diff_80_20 = mytree_test_error_80_20 - mytree_train_error_80_20

print(diff_80_20)

## [1] 0.02291841
```

Based on summary, using with CP = 0.02732240 for the least xerror

APPLYING PARAMETER VALUES TO ARRIVE AT BETTER PERFORMANCE FOR MODEL 4: 80-20 SPLIT

Creating vectors for minsplit and minbucket values to be used for different combinations to test performance.

```
msplt <- c(12,48,102)
mbckt <- c(4,16,34)
for (i in msplt)
{
  for (j in mbckt)
  {

    mytree_80_20 <- rpart(MyFormula, data = train, parms = list(split="gini")
,control = rpart.control (minsplit = i,minbucket = j,cp=0.02732240))

    mytree_train_predict_80_20 <- predict(mytree_80_20, data = train , type =
"class")

    mytree_train_error_80_20 <- mean(mytree_train_predict_80_20 != train$Reta
ined.in.2012.)

    mytree_test_predict_80_20 <- predict(mytree_80_20, newdata = test, type =
"class")

    mytree_test_error_80_20 <- mean(mytree_test_predict_80_20 != test$Retaine
d.in.2012.)

    diff_80_20 = mytree_test_error_80_20 - mytree_train_error_80_20
    diff_80_20
    print(diff_80_20)

    ############################## Confusion Matrix for 80:20 Split #########
########################################
```

```r
    cfmt <- table(train$Retained.in.2012.,mytree_train_predict_80_20)
    print(cfmt)

    fp = cfmt[2,1]
    fn = cfmt[1,2]
    tn = cfmt[2,2]
    tp = cfmt[1,1]

    #Calculating precision by dividing true positive with the sum of true pos
itive and false positive.
    precision_train =  (tp)/(tp+fp)
    accuracymodel_train = (tp+tn)/(tp+tn+fp+fn)
    recall_train = (tp)/(tp+fn)
    fscore_train = (2*(recall_train*precision_train))/(recall_train+precision
_train)

    cfmt <- table(test$Retained.in.2012.,mytree_test_predict_80_20)
    print(cfmt)

    fp = cfmt[2,1]
    fn = cfmt[1,2]
    tn = cfmt[2,2]
    tp = cfmt[1,1]

    #Calculating precision by dividing true positive with the sum of true pos
itive and false positive.
    precision_test =  (tp)/(tp+fp)
    accuracymodel_test = (tp+tn)/(tp+tn+fp+fn)
    recall_test = (tp)/(tp+fn)
    fscore_test = (2*(recall_test*precision_test))/(recall_test+precision_tes
t)

    #Printing the values for train data error, test data error, performance a
nd other parameters.
    print(paste("Train data error: ", mytree_train_error_80_20))
    print(paste("Test data error: ", mytree_test_error_80_20))
    print(paste("Difference/performance", diff_80_20))
    print(paste("precision of training data: ", precision_train))
    print(paste("accuracy of training data: ", accuracymodel_train))
    print(paste("recall of training data: ", recall_train))
    print(paste("F-score of training data: ", fscore_train))
    print(paste("precision of test data: ", precision_test))
    print(paste("accuracy of test data: ", accuracymodel_test))
    print(paste("recall of test data: ", recall_test))
    print(paste("F-score of test data: ", fscore_test))
```

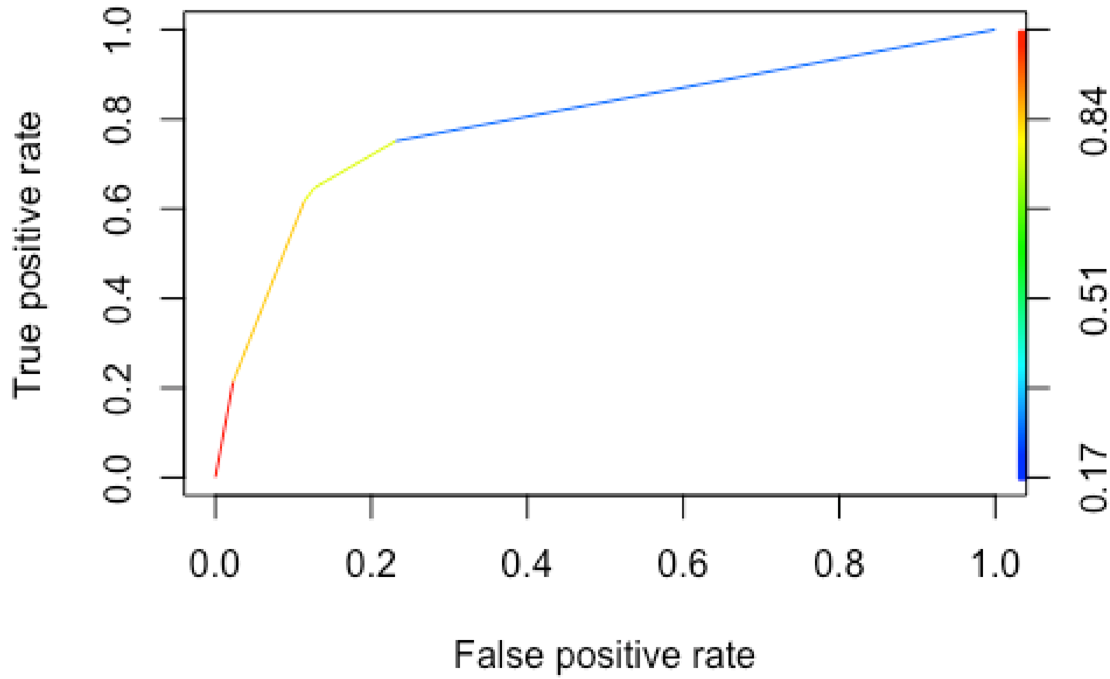
```
  }
}

## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
```

```
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
```

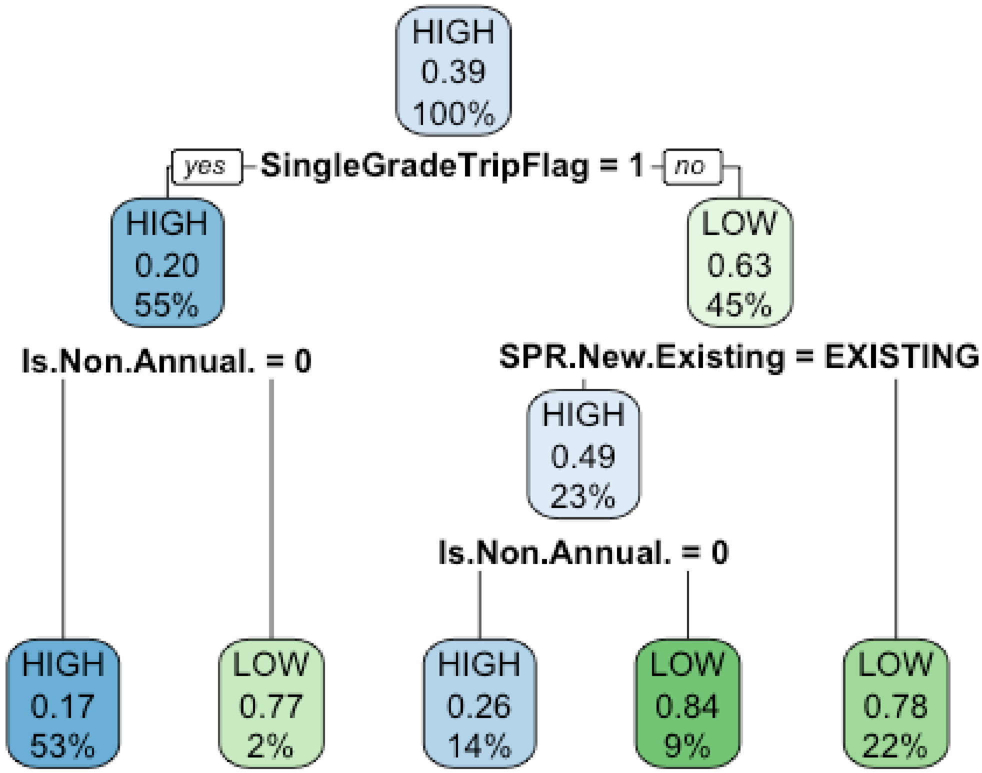
```
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
```

```
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
## [1] 0.02291841
##       mytree_train_predict_80_20
##        HIGH  LOW
##   HIGH 1018  126
##   LOW   238  498
##       mytree_test_predict_80_20
##        HIGH LOW
##   HIGH  268  39
##   LOW    71 130
## [1] "Train data error:  0.193617021276596"
## [1] "Test data error:  0.216535433070866"
## [1] "Difference/performance 0.0229184117942704"
## [1] "precision of training data:  0.810509554140127"
## [1] "accuracy of training data:  0.806382978723404"
## [1] "recall of training data:  0.88986013986014"
## [1] "F-score of training data:  0.848333333333333"
## [1] "precision of test data:  0.790560471976401"
## [1] "accuracy of test data:  0.783464566929134"
## [1] "recall of test data:  0.872964169381108"
## [1] "F-score of test data:  0.829721362229102"
```

```
predicteddtProb <- predict(mytree_80_20, newdata = test, type = "prob")[,2]
pred1 <- prediction(predicteddtProb, test$Retained.in.2012.)
perf1 <- performance(pred1, "tpr", "fpr")
plot(perf1,colorize=TRUE)
```




```
rpart.plot(mytree_80_20)
```



The below shows the error, recall values on train and test data for both 70-30 split and 80-20 split.

```r
knitr::include_graphics("Values.jpeg")
```

70:30 Split - Basic

Train data error	Test data error	error
0.1780488	0.2149533	0.0369045

70:30 Split - Basic - Pre Pruning

minsplit	minbucket	Train data error	Test data error	error	Recall(train)	F-score(train)	Recall(test)	F-score(test)
12	4	0.17804878	0.214953271	0.03690449	0.87814703	0.85658153	0.87336245	0.83246618
	16	0.18597561	0.213618158	0.02764255	0.88922457	0.85272815	0.89519651	0.83673469
	34	0.186585366	0.217623498	0.02970302	0.88620342	0.85188771	0.88864629	0.83401639
48	4	0.18597561	0.213618158	0.02764255	0.88922457	0.85272815	0.89519651	0.83673469
	16	0.18597561	0.213618158	0.02764255	0.88922457	0.85272815	0.89519651	0.83673469
	34	0.186585366	0.217623498	0.02764255	0.88922457	0.85272815	0.89519651	0.83673469
102	4	0.18597561	0.213618158	0.02764255	0.88922457	0.85272815	0.89519651	0.83673469
	16	0.18597561	0.213618158	0.02764255	0.88922457	0.85272815	0.89519651	0.83673469
	34	0.186585366	0.217623498	0.02970302	0.88620342	0.85188771	0.88864629	0.83401639

80:20 Split - Basic

Train data error	Test data error	error
0.1929825	0.2244094	0.031427

80:20 Split - Basic - Pre Pruning

Gini 80:20

minsplit	minbucket	Train data error	Test data error	error	Recall(trai	F-score(train)	Recall(test	F-score(test)
12	4	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709
	16	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709
	34	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709
48	4	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709
	16	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709
	34	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709
102	4	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709
	16	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709
	34	0.197235513	0.202755906	0.00552	0.882507	0.84535223	0.900662	0.840803709

Cross Validation :: K Fold Approach

```r
k <- 10
folds <- cut(seq(1,nrow(df)),breaks = k, labels = FALSE)

models.acc <- matrix(-1,k,2,dimnames=list(paste0("Fold ", 1:k, " Accuracy"),
c("DecisionTree","RandomForest")))
models.err <- matrix(-1,k,2,dimnames=list(paste0("Fold ", 1:k , " Error"), c(
"DecisionTree","RandomForest")))

emeasure.model.dt <- matrix(-1,k,3,dimnames=list(paste0("Fold", 1:k), c("Accu
racy","Error","Recall")))
```

```
emeasure.model.rf <- matrix(-1,k,3,dimnames=list(paste0("Fold", 1:k), c("Accu
racy","Error","Recall")))

for(i in 1:k)
{
  testIndexes <- which(folds==i, arr.ind=TRUE)
  testData <- df[testIndexes, ]
  trainData <- df[-testIndexes, ]

  # Decision Tree
  dt <- rpart(Retained.in.2012. ~ ., data = trainData, parms = list(split = "
information")
              ,control=rpart.control(minsplit = 12, minbucket = 4, cp=0.02))
  predicteddt <- predict(dt, newdata = testData,type="class")
  emeasure.model.dt[i,"Accuracy"] <- evaluation.measure(testData$Retained.in.
2012.,predicteddt)[1]
  emeasure.model.dt[i,"Error"] <- evaluation.measure(testData$Retained.in.201
2.,predicteddt)[2]
  emeasure.model.dt[i,"Recall"] <- evaluation.measure(testData$Retained.in.20
12.,predicteddt)[3]

  # Random Forest
  rf <- randomForest(Retained.in.2012. ~ ., data= trainData, ntree = 100, mtr
y=
                       bestmtry, proximity = T, importance = T)

  predictedrf <- predict(rf, newdata = testData, type = "class")
  emeasure.model.rf[i,"Accuracy"] <- evaluation.measure(testData$Retained.in.
2012.,predictedrf)[1]
  emeasure.model.rf[i,"Error"] <- evaluation.measure(testData$Retained.in.201
2.,predictedrf)[2]
  emeasure.model.rf[i,"Recall"] <- evaluation.measure(testData$Retained.in.20
12.,predictedrf)[3]
}

totalPositive <- table(df$Retained.in.2012.)[[1]]

Final <- matrix(c(mean(emeasure.model.dt[,"Accuracy"]),
                  mean(emeasure.model.dt[,"Error"]),
                  sum(emeasure.model.dt[,"Recall"])/totalPositive,
                  mean(emeasure.model.rf[,"Accuracy"]),
                  mean(emeasure.model.rf[,"Error"]),
                  sum(emeasure.model.rf[,"Recall"])/totalPositive),ncol = 2)
colnames(Final) <- c("DecisionTree","RandomForest")
rownames(Final) <- c("Accuracy","Error","Weighted Recall")

Final
```

```
##                  DecisionTree RandomForest
## Accuracy            0.7906227     0.780987
## Error               0.2093773     0.219013
## Weighted Recall     0.8959338     0.789111
```

Summary

The first step for our model building is doing the EDA. We identified the NA values in the dataset. For the numerical variables, we replaced the NA values with the mean and for categorical variables, we replaced the NA values with the frequently repeated value. We also constructed boxplots to identify the outliers. To find the important variables, we calculated the correlation values between numerical variables and the target variable. We calculated chisquare values between categorical variables and the target variable.

The second step is we constructed the random forest with different mtry values. The best mtry value for our model is 45. We are also constructing ROC curve.

The third step is we constructed decision trees with 70-30 split and 80-20 split. We used different minsplit, minbucket and cp values to get the best decision tree. In our case the best decision tree is for 80-20 split as it has the maximum recall value of 90% on the test data. We chose recall as the measure as we are focused on false-negatives. i.e telling a customer who can be retained as not retained. this would affect the business. We are also constructing ROC curve for the best decision tree.

The fourth step is, we are performing cross-validation on both random forest and decision tree with 10 folds. We are calculating the average of accuracy and error, weighted average of recall to identify which model performs better. We have identified decision tree as the best model based on recall value which is around 89.5%. We also know that decision tree performs better on small dataset and since our training dataset is small, which is close to 1600 rows, the base classifier is better than ensemble classifier. This strengthens our conclusion of choosing decision tree over random forest.