Data Mining

Ajeet Pandey

Today is 2020-05-17.

Executive Summary

This dataset has been provided by Bank which is interested in increasing its asset base by giving out more loans to potential customers in order to earn Interest Income over a good period of financial years in future

Various variables or predictors have been provided like Income, Age, Mortgage ets. to gauge the Response on Personal Loan

Importing required libraries

```
library(readr)
library(dplyr)
```

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

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

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

```
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

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

```
library(lattice)
library(DataExplorer)
library(grDevices)
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## randomForest 4.6-14
```

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

```
##
## Attaching package: 'randomForest'
```

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

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

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

```
library(ranger)
```

```
##
## Attaching package: 'ranger'
```

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

```
library(Metrics)
```

```
##
## Attaching package: 'Metrics'
```

```
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```
library(ROCit)
library(kableExtra)
```

```
##
## Attaching package: 'kableExtra'
```

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

Data loading and data Cleaning

```
bank = read.csv("Bank_Personal_Loan_ds.csv", sep = "," , header = TRUE)
summary(bank)
```

```
##        ID            Age          Experience       Income          ZIP.Code
##  Min.   :   1   Min.   :23.00   Min.   :-3.0   Min.   :  8.00   Min.   : 9307
##  1st Qu.:1251   1st Qu.:35.00   1st Qu.:10.0   1st Qu.: 39.00   1st Qu.:91911
##  Median :2500   Median :45.00   Median :20.0   Median : 64.00   Median :93437
##  Mean   :2500   Mean   :45.34   Mean   :20.1   Mean   : 73.77   Mean   :93153
##  3rd Qu.:3750   3rd Qu.:55.00   3rd Qu.:30.0   3rd Qu.: 98.00   3rd Qu.:94608
##  Max.   :5000   Max.   :67.00   Max.   :43.0   Max.   :224.00   Max.   :96651
##      Family          CCAvg          Education        Mortgage
##  Min.   :1.000   Min.   : 0.000   Min.   :1.000   Min.   :  0.0
##  1st Qu.:1.000   1st Qu.: 0.700   1st Qu.:1.000   1st Qu.:  0.0
##  Median :2.000   Median : 1.500   Median :2.000   Median :  0.0
##  Mean   :2.396   Mean   : 1.938   Mean   :1.881   Mean   : 56.5
##  3rd Qu.:3.000   3rd Qu.: 2.500   3rd Qu.:3.000   3rd Qu.:101.0
##  Max.   :4.000   Max.   :10.000   Max.   :3.000   Max.   :635.0
##  Personal.Loan   Securities.Account   CD.Account         Online
##  Min.   :0.000   Min.   :0.0000     Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.000   1st Qu.:0.0000     1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.000   Median :0.0000     Median :0.0000   Median :1.0000
##  Mean   :0.096   Mean   :0.1044     Mean   :0.0604   Mean   :0.5968
##  3rd Qu.:0.000   3rd Qu.:0.0000     3rd Qu.:0.0000   3rd Qu.:1.0000
##  Max.   :1.000   Max.   :1.0000     Max.   :1.0000   Max.   :1.0000
##    CreditCard
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.294
##  3rd Qu.:1.000
##  Max.   :1.000
```

```
dim(bank)
```

```
## [1] 5000   14
```

Dataset has 5000 rows of observations and 14 variables

Family Members have 18 observations missing

```
any(is.na(bank)) ## check for missing values
```

```
## [1] FALSE
```

```
bank[is.na(bank)] = 0
any(is.na(bank))
```

```
## [1] FALSE
```

```
str(bank)
```

```
## 'data.frame':    5000 obs. of  14 variables:
##  $ ID                : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Age               : int  25 45 39 35 35 37 53 50 35 34 ...
##  $ Experience        : int  1 19 15 9 8 13 27 24 10 9 ...
##  $ Income            : int  49 34 11 100 45 29 72 22 81 180 ...
##  $ ZIP.Code          : int  91107 90089 94720 94112 91330 92121 91711 93943 90089 93023
...
##  $ Family            : int  4 3 1 1 4 4 2 1 3 1 ...
##  $ CCAvg             : num  1.6 1.5 1 2.7 1 0.4 1.5 0.3 0.6 8.9 ...
##  $ Education         : int  1 1 1 2 2 2 2 3 2 3 ...
##  $ Mortgage          : int  0 0 0 0 0 155 0 0 104 0 ...
##  $ Personal.Loan     : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ Securities.Account: int  1 1 0 0 0 0 0 0 0 0 ...
##  $ CD.Account        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Online            : int  0 0 0 0 0 1 1 0 1 0 ...
##  $ CreditCard        : int  0 0 0 0 1 0 0 1 0 0 ...
```

Looking at the dataset we realize the following aspects (raw check) ID and Zip code columns will not help much in analyis since they are basically addon - para information Experience has got negative values. We will fix them with corresponding positive values making more sense Columns like Personal Loan, CD Account, Online et.al are factor values with levels “0” and “1” . Save Education which is ordered factor with 3 levels 1 < 2 < 3

```
attach(bank)
bank = bank[, -c(1,5)]  ## removing ID and Zip code column from dataset

##  Converting multiple columns into factor columns
col = c("Education","Personal.Loan","Securities.Account", "CD.Account", "Online", "CreditCar
d")
bank[col] = lapply(bank[col], factor)
col
```

```
## [1] "Education"          "Personal.Loan"      "Securities.Account"
## [4] "CD.Account"         "Online"             "CreditCard"
```

```
summary(col)
```

```
##    Length     Class      Mode 
##         6 character character
```

```
## Converting Education into ordered factors . Ordinal variable
Education = factor(Education, levels = c("1", "2", "3"), ordered = TRUE )
names(bank)
```

```
##  [1] "Age"                "Experience"         "Income"
##  [4] "Family"             "CCAvg"              "Education"
##  [7] "Mortgage"           "Personal.Loan"      "Securities.Account"
## [10] "CD.Account"         "Online"             "CreditCard"
```

```
head(bank[Experience < 0,])
```

```
##     Age Experience Income Family CCAvg Education Mortgage Personal.Loan
## 90   25         -1    113      4  2.30         3        0             0
## 227  24         -1     39      2  1.70         2        0             0
## 316  24         -2     51      3  0.30         3        0             0
## 452  28         -2     48      2  1.75         3       89             0
## 525  24         -1     75      4  0.20         1        0             0
## 537  25         -1     43      3  2.40         2      176             0
##     Securities.Account CD.Account Online CreditCard
## 90                   0          0      0          1
## 227                  0          0      0          0
## 316                  0          0      1          0
## 452                  0          0      1          0
## 525                  0          0      1          0
## 537                  0          0      1          0
```

```
Experience = abs(Experience)  ## fixing them up
dim(bank)
```

```
## [1] 5000   12
```

```
summary(bank)
```

```
##       Age          Experience       Income           Family     
##  Min.   :23.00   Min.   :-3.0   Min.   :  8.00   Min.   :1.000  
##  1st Qu.:35.00   1st Qu.:10.0   1st Qu.: 39.00   1st Qu.:1.000  
##  Median :45.00   Median :20.0   Median : 64.00   Median :2.000  
##  Mean   :45.34   Mean   :20.1   Mean   : 73.77   Mean   :2.396  
##  3rd Qu.:55.00   3rd Qu.:30.0   3rd Qu.: 98.00   3rd Qu.:3.000  
##  Max.   :67.00   Max.   :43.0   Max.   :224.00   Max.   :4.000  
##      CCAvg        Education    Mortgage     Personal.Loan Securities.Account
##  Min.   : 0.000   1:2096    Min.   :  0.0   0:4520        0:4478            
##  1st Qu.: 0.700   2:1403    1st Qu.:  0.0   1: 480        1: 522            
##  Median : 1.500   3:1501    Median :  0.0                                   
##  Mean   : 1.938             Mean   : 56.5                                   
##  3rd Qu.: 2.500             3rd Qu.:101.0                                   
##  Max.   :10.000             Max.   :635.0                                   
##  CD.Account Online   CreditCard
##  0:4698     0:2016   0:3530    
##  1: 302     1:2984   1:1470    
## 
## 
## 
## 
```

#Exploratory Data Analysis

##Histogram Distributions of Dataset

Plotting density plot for all numerical variables

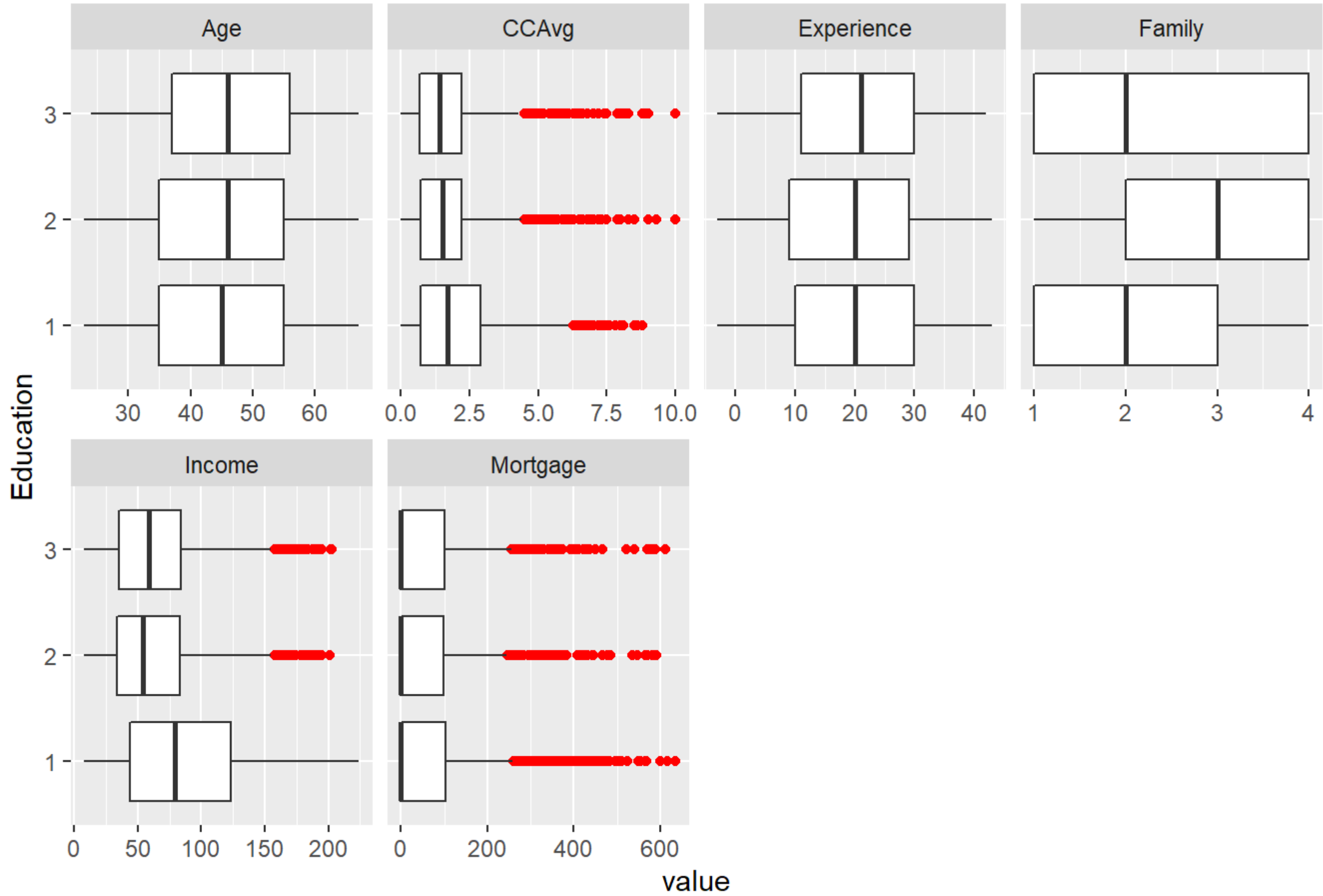
##BoxPlots by Education classes

##Insight

1. Credit Card and Mortagage predictors have lots of outliers accross all three levels of Education
2. Income has lots of outliers in Grad and Advanced professionals

Plotting boxplot by factor of Education for all the numerical variables

```
plot_boxplot(bank, by = "Education",
             geom_boxplot_args = list("outlier.color" = "red"))
```

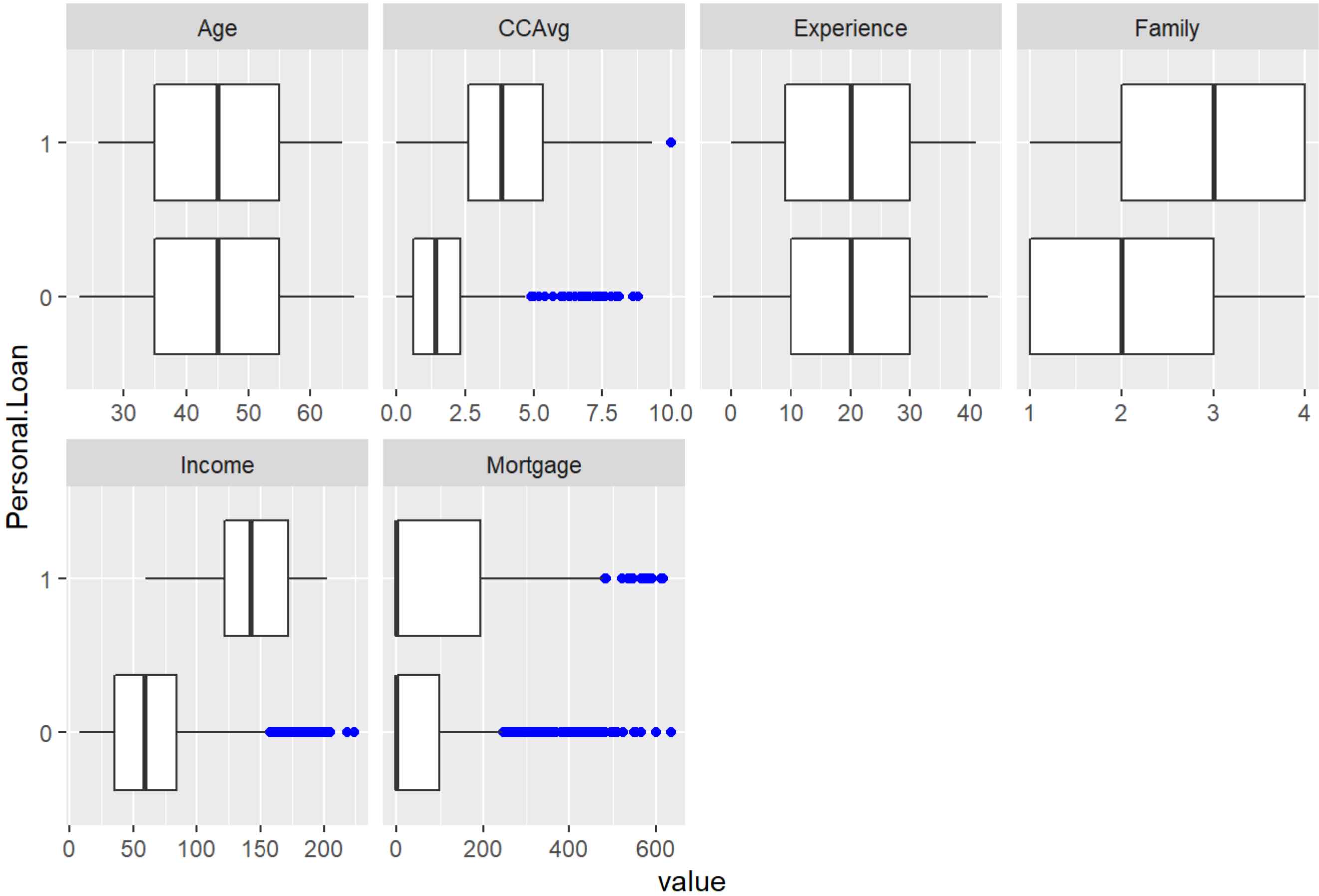


Boxplots by Personal Loan classes

Lots of “No” (Class 0) Personal loan takers are present as outliers in Credit Card, Mortgge and Income predictors

Plotting boxplot for Personal Loan (Response variable) for all numerical variables

```
plot_boxplot(bank, by = "Personal.Loan",
             geom_boxplot_args = list("outlier.color" = "blue"))
```



Following plots give us a good insight about how two categories of Personal Loan predictor are stacked across various other predictors like

1. Income vs Mortgage (scatter) 2.Income (density)

Mortgage (density)

Age (density)

Experience (density)

Income vs Education (histogram)

```
p1 = ggplot(bank, aes(Income, fill = Personal.Loan)) + geom_density(alpha = 0.4)
p2 = ggplot(bank, aes(Mortgage, fill = Personal.Loan)) + geom_density(alpha = 0.4)
p3 = ggplot(bank, aes(Age, fill = Personal.Loan)) + geom_density(alpha = 0.4)
p4 = ggplot(bank, aes(Experience, fill = Personal.Loan)) + geom_density(alpha = 0.4)
p5 = ggplot(bank, aes(Income, fill = Education)) + geom_histogram(alpha = 0.4, bins = 70)
p6 = ggplot(bank, aes(Income, Mortgage, color = Personal.Loan)) +
  geom_point(alpha = 0.7)
grid.arrange(p1, p2, p3, p4, p5, p6, ncol = 2, nrow = 3)
```

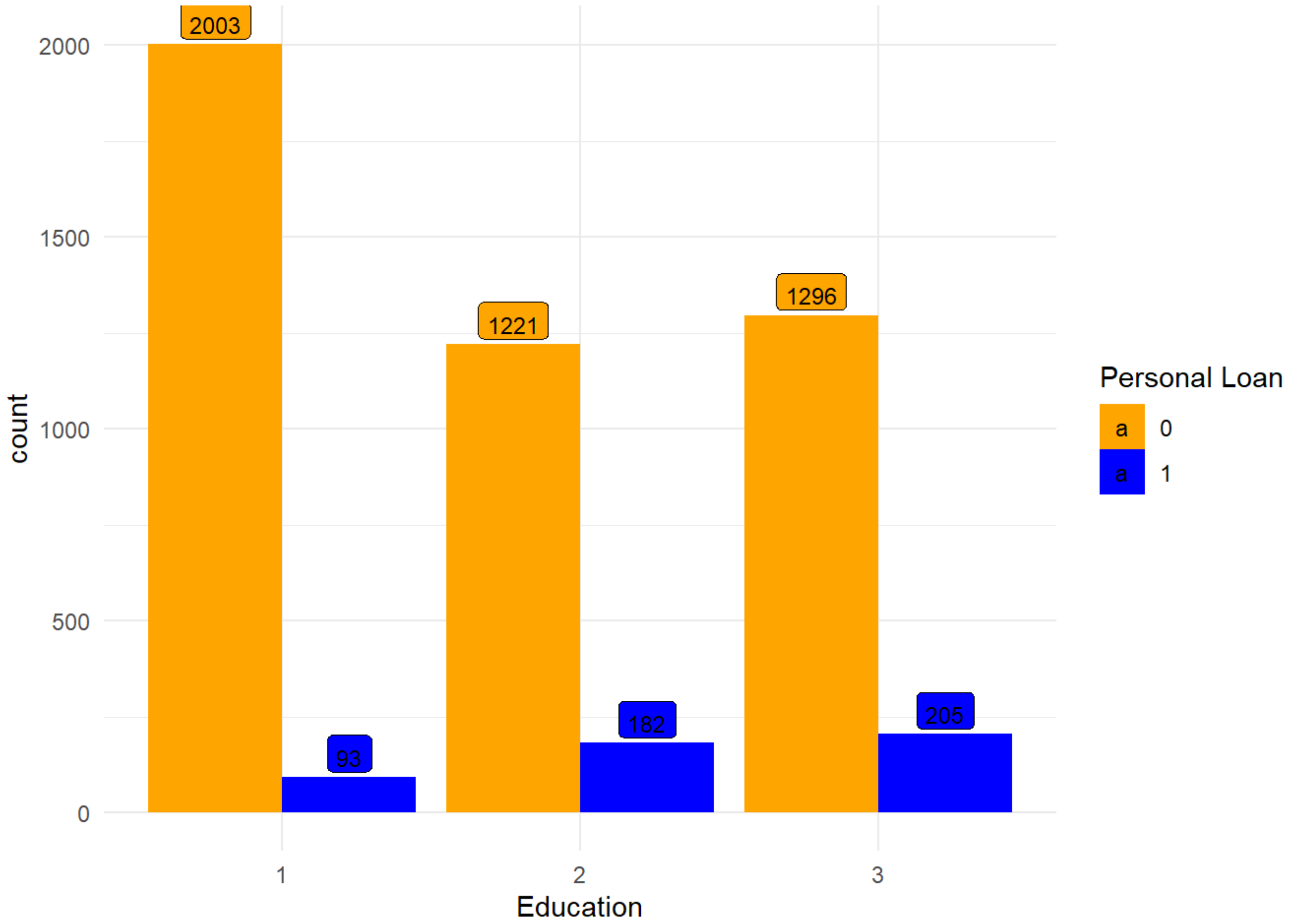
##Education

Proportion of no-loan takers is very high across all three categories of Education - Undergrad, Grad, and Advanced Profn

Data is almost skewed towards No-Personal Loans which makes good suspects and prospects depending on target category of bank

There is good jump from 93(undergrads) to 205 (Advanced Profs)

```
ggplot(bank, aes(Education,fill = Personal.Loan)) +
  geom_bar(stat = "count", position = "dodge") +
  geom_label(stat = "count", aes(label = ..count..),
             size = 3, position = position_dodge(width = 0.9), vjust = -0.15) +
  scale_fill_manual("Personal Loan", values = c("0" = "orange", "1" = "blue")) +
  theme_minimal()
```



Credit Card is very good indicator of who we can target bothways

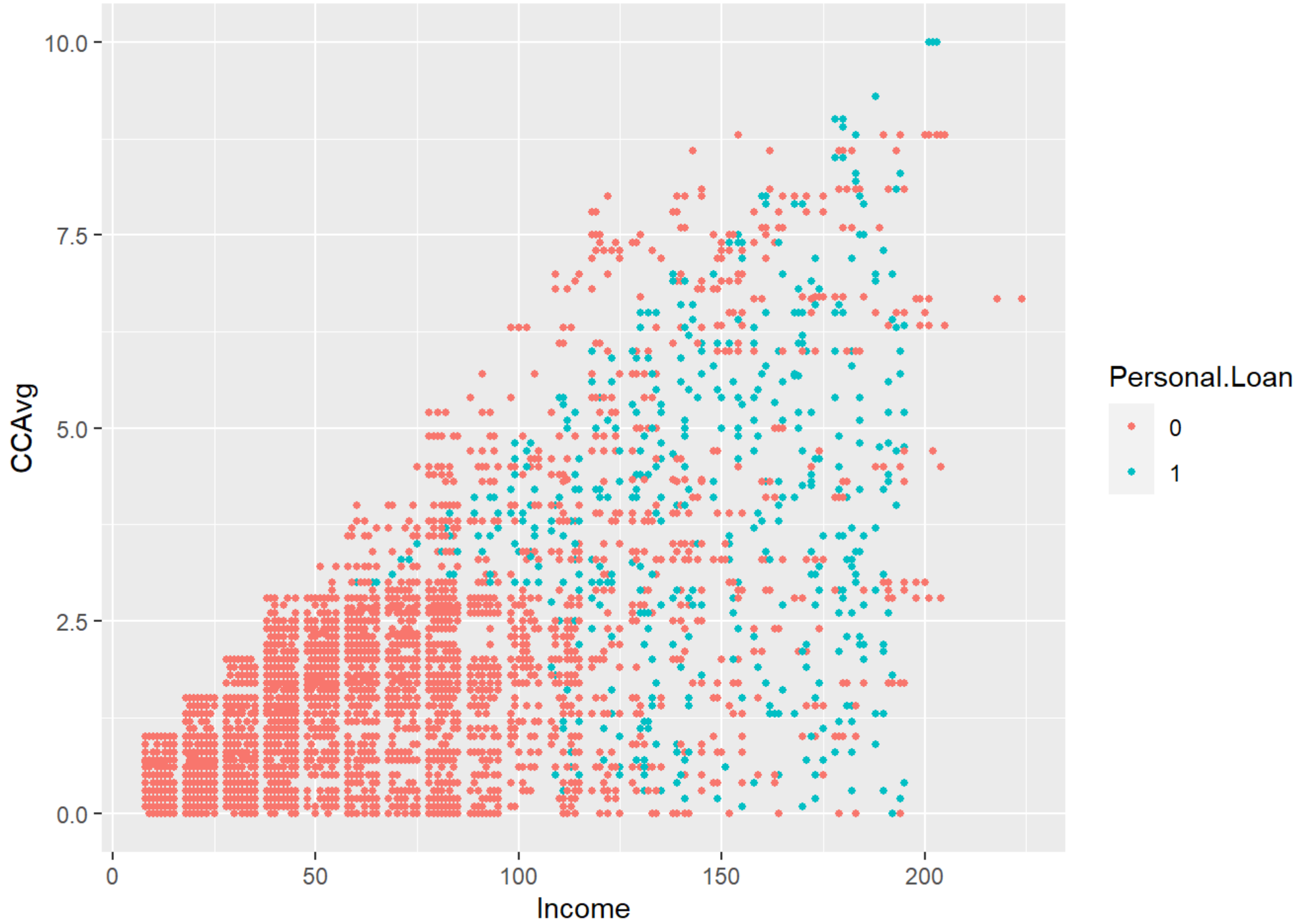
1.Prospects who spend more may need to pay off their debt by taking Personal Loan

Other category is who have good income but hesitate to spend can be offered loans on good conditions for their lifestyle and personal needs

Virtually People having income in 1st quartile i.e. between 38 K to 90K have no Personal loans and moderate Credit Card spending (under 3000)

People earning between 40K to 100K and having CC spend less than $ 2500 can become good prime targets keeping other predictors constant and we see a good chunk of them in graph

```
ggplot(bank, aes(Income,y = CCAvg, color = Personal.Loan)) +
    geom_point(size = 1)
```



Mortgage is another good indicator of who can be targeted.

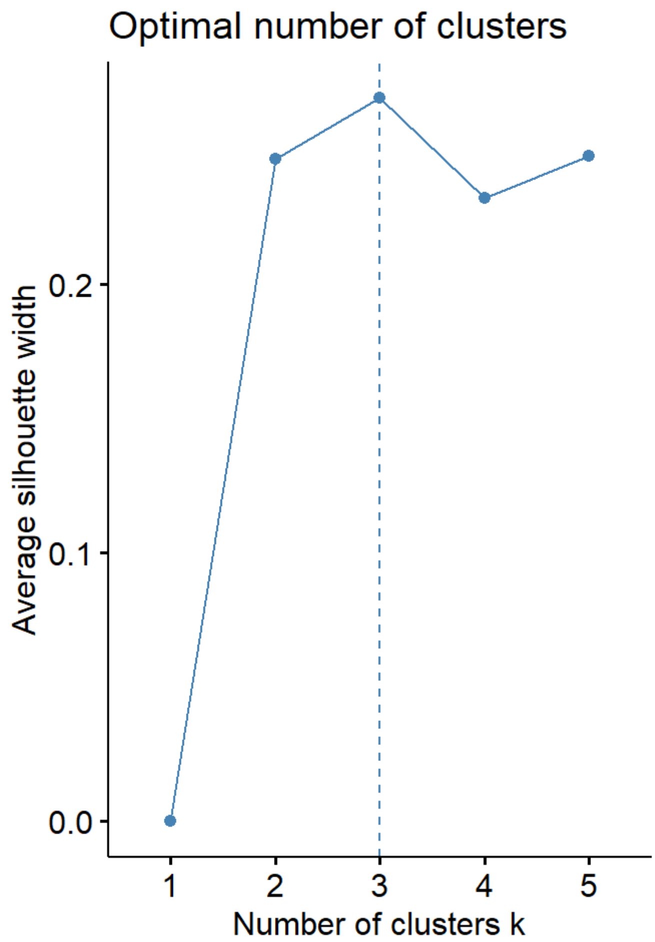
By offering good terms to people having zero Mortgage

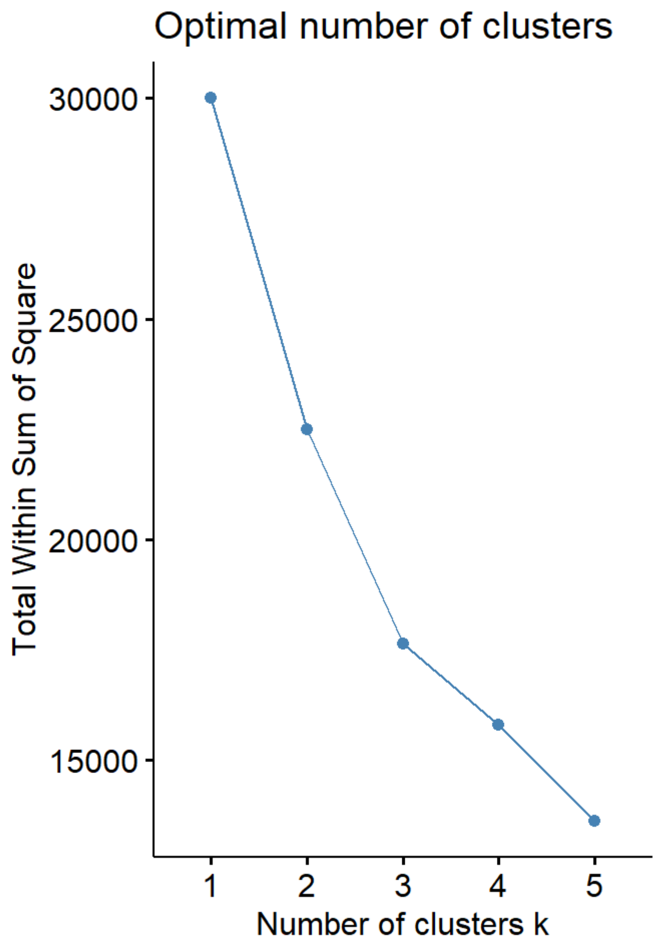
Others under considerate Mortgage like lets say 150K to settle their loans of high interest with low interest Personal Loans

```
ggplot(bank, aes(Income,y = Mortgage, color = Personal.Loan)) +
  geom_point(size = 1)
```

#Clustering Primarily hierarchial and kmeans clustering are two best suited methods for unsupervised learning Since we have a very large dataset (5000 obs) we cannot use hierarchial method.Kmeans suits this type of data categorization

```
bank.clus = bank %>% select_if(is.numeric)

bank.scaled = scale(bank.clus, center = TRUE)

bank.dist = dist(bank.scaled, method = "euclidean")

p12 = fviz_nbclust(bank.scaled, kmeans, method = "silhouette", k.max = 5)
p21 = fviz_nbclust(bank.scaled, kmeans, method = "wss", k.max = 5)

grid.arrange(p12, p21, ncol = 2)
```

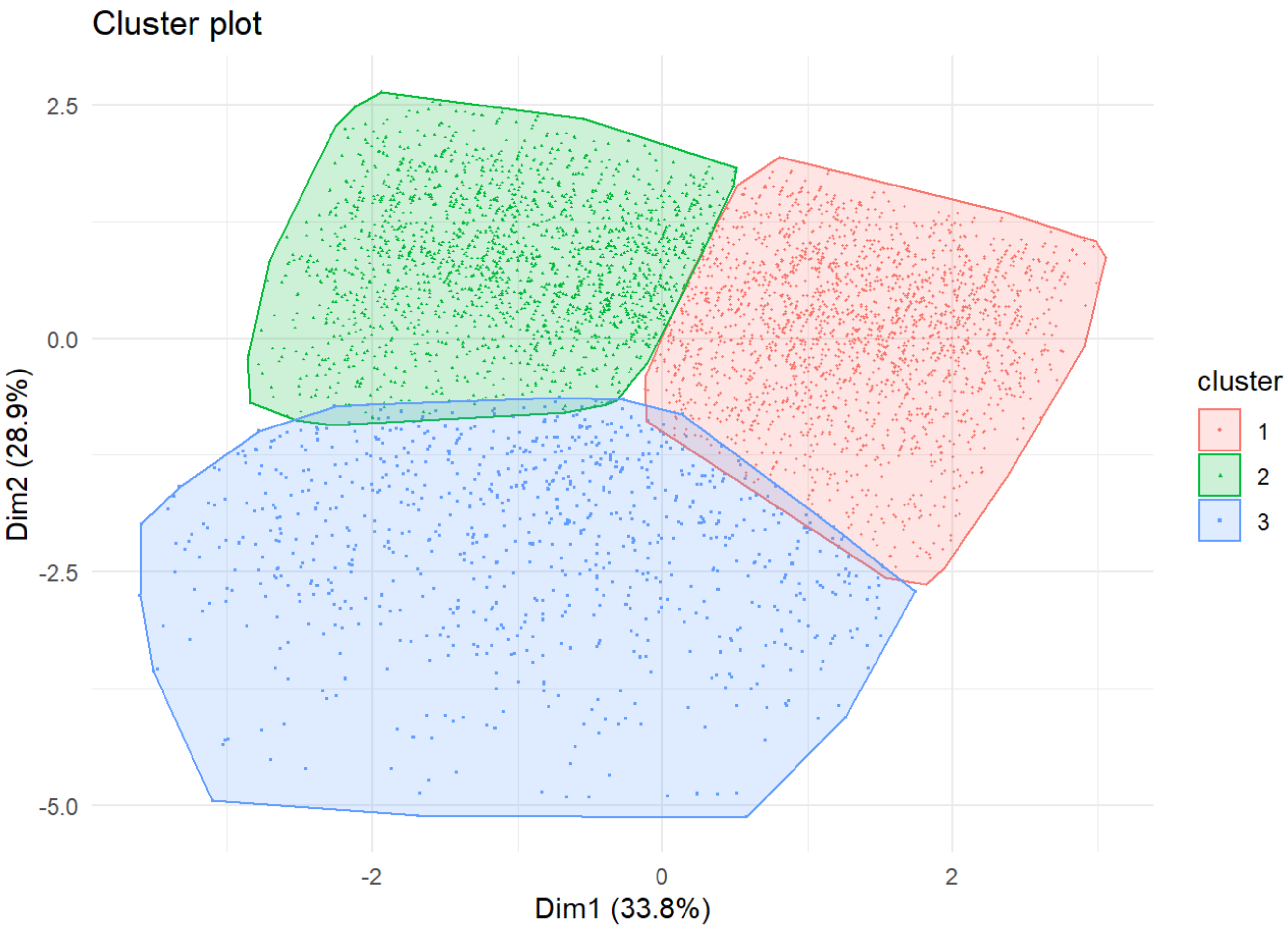




Running Kmeans with 3 centers and iterating it with nstart 10 times

```
set.seed(8787)
bank.clusters = kmeans(bank.scaled, 3, nstart = 10)
```

```
fviz_cluster(bank.clusters, bank.scaled, geom = "point",
             ellipse = TRUE, pointsize = 0.2, ) + theme_minimal()
```



checking optimal number of clusters to categorize dataset

#Insights

Silhouette and withing clusters sum of squares (wss) method , indicate we can divide our dataset into 3 clusters

This intuitively conincides with Education levels (3) as a wild guess. It makes sense that banks prefer Educated people who have good earning potential or may have in future increasing financial needs to support their lifestyle and needs

Kmeans divides the dataset into 3 clusters of size 2149, 2012, and 839

#Splitting of Dataset into Train - Test set

```
set.seed(1233)

## sampling 70% of data for training the algorithms using random sampling
bank.index = sample(1:nrow(bank), nrow(bank)*0.70)
bank.train = bank[bank.index,]
bank.test = bank[-bank.index,]

dim(bank.test)
```

```
## [1] 1500   12
```

```
dim(bank.train)
```

```
## [1] 3500   12
```

checking the ration of personal loans in each partition

```
table(bank.train$Personal.Loan)
```

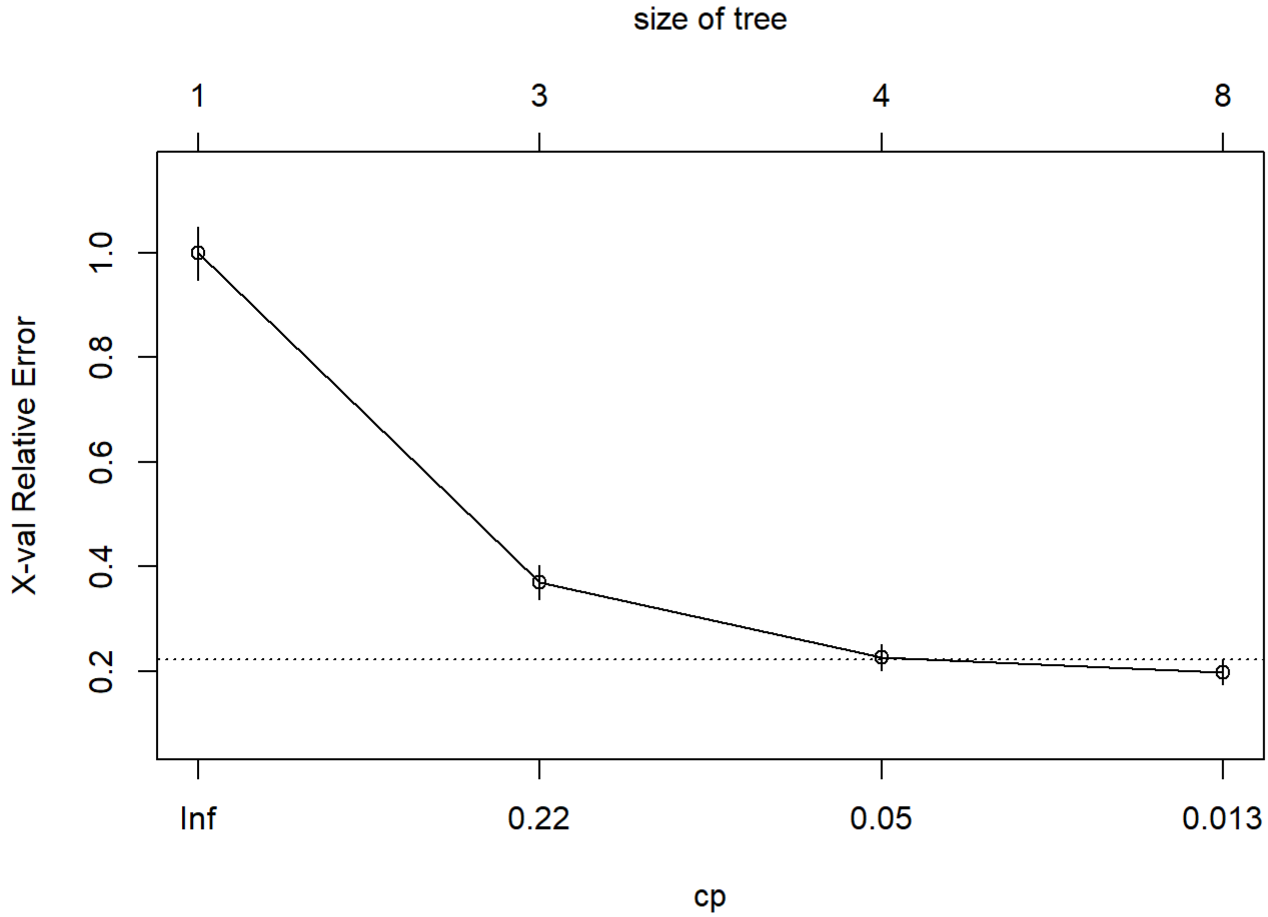
```
## 
##    0    1 
## 3151  349
```

```
table(bank.test$Personal.Loan)
```

```
## 
##    0    1 
## 1369  131
```

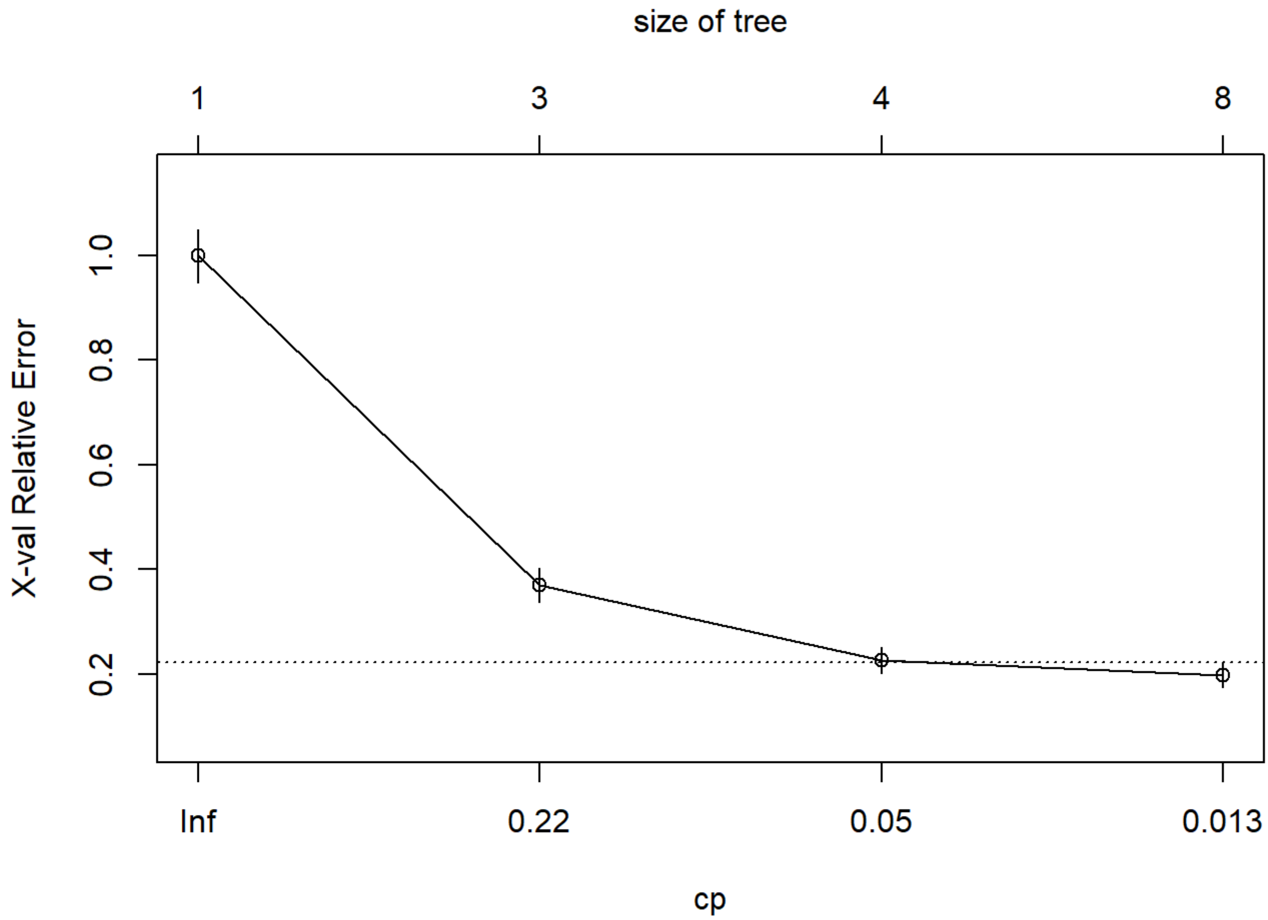
#CART Model Classification trees use recursive partitioning algorithms to learn and grow on data

```
set.seed(233)

cart.model.gini = rpart(Personal.Loan~., data = bank.train, method = "class",
                   parms = list(split = "gini"))

plotcp(cart.model.gini)
```



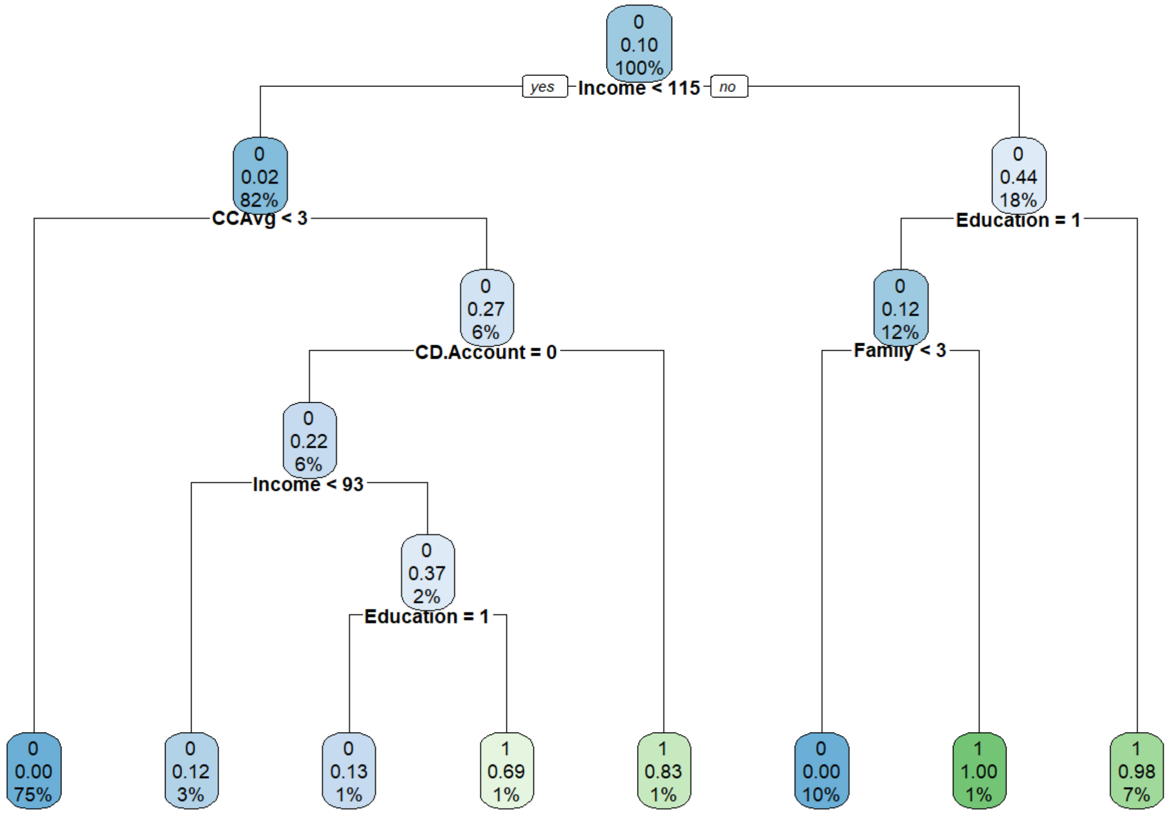
checking the complexity parameter

```
plotcp(cart.model.gini)
```



plotting the classification tree

```
rpart.plot(cart.model.gini, cex = 0.6)
```



checking the cptable to gauge the best crossvalidated error and correspoding

Complexity paramter

```
cart.model.gini$cptable
```

```
##           CP nsplit rel error    xerror       xstd
## 1 0.32521490      0 1.0000000 1.0000000 0.05078991
## 2 0.14326648      2 0.3495702 0.3696275 0.03193852
## 3 0.01719198      3 0.2063037 0.2263610 0.02517855
## 4 0.01000000      7 0.1346705 0.1977077 0.02356543
```

checking for the variable importance for splitting of tree

```
cart.model.gini$variable.importance
```

```
##    Education       Income       Family        CCAvg  CD.Account     Mortgage
## 232.1371068 191.3248866 143.4292520 106.6062571  56.9041764   27.3062762
##         Age       Online   Experience
##   3.4376722   1.7510401    0.6622231
```

##Insight

Education, Income , Family Member, CC Avg and CD Account are important predictors on which data is split by tree algo

Is clearly reflected in the built cart tree by the algorithm too

First split happens on whether Income is less than or greater than $ 115K

Complexity parameter almost lowers to 0.05 (graph) with relative 0.2 as cross validated error

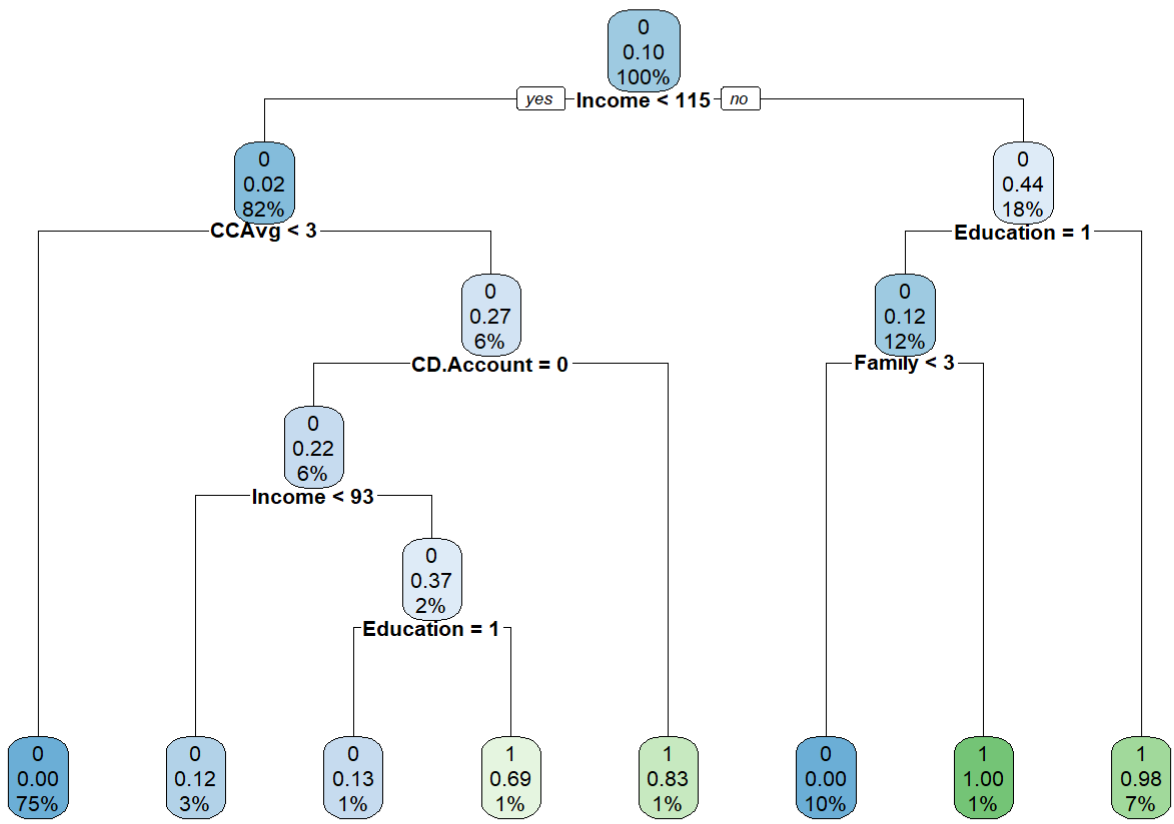
#Pruned Cart Tree Tree can be pruned using the complexity parameter for controlling the overfitting

prunning the tree using the best complexity parameter

```
pruned.model = prune(cart.model.gini, cp = 0.015)
```

plotting the prunned tree

```
pruned.model = prune(cart.model.gini, cp = 0.015)
rpart.plot(pruned.model, cex = 0.65)
```



#Cart Prediction

```
cart.pred = predict(pruned.model, bank.test, type = "prob")

cart.pred.prob.1 = cart.pred[,1]
head(cart.pred.prob.1, 10)
```

```
##          3          4          7          8         10         13         17
## 0.99734244 0.99734244 0.99734244 0.99734244 0.02109705 0.31428571 0.02109705
##         18         19         22
## 0.99734244 0.02109705 0.99734244
```

Since this is a loan prediction and we want to be more careful to weed out possible defaulters rather then deny the disbursal to dserving prospects We will set the threshold for probability as high as 0.70

###All the predicted probabilities >=0.7 will be considered as class “1” and rest class “0”

Using the Confusison Matrix to gauge the performance of Models ## setting the threshold for probabilities to be considered as 1

```
threshold = 0.70

bank.test$loanprediction = ifelse(cart.pred.prob.1 >= threshold, 1, 0)

bank.test$loanprediction = as.factor(bank.test$loanprediction)

Cart.Confusion.Matrix = confusionMatrix(bank.test$loanprediction,
                    reference = bank.test$Personal.Loan, positive = "1")
Cart.Confusion.Matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0    8  121
##          1 1361   10
##
##                Accuracy : 0.012
##                  95% CI : (0.0071, 0.0189)
##     No Information Rate : 0.9127
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.1738
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.076336
##             Specificity : 0.005844
##          Pos Pred Value : 0.007294
##          Neg Pred Value : 0.062016
##              Prevalence : 0.087333
##          Detection Rate : 0.006667
##    Detection Prevalence : 0.914000
##       Balanced Accuracy : 0.041090
##
##        'Positive' Class : 1
##
```

We can see that even Pruned CART tree has very low accuracy of just 1.67 % even after tuning its complexity parameter

#Random Forest Model

Random forest is an ensemble method used by combining weak and strong learners to give a better accuracy or output. Its a combination of multiple trees each choosen randomly to grow on dataset Uses averaging in the sense that weak and strong learners combined produce better results rather than a single CART tree

Two packages have been used to model the training dataset

Random Forest

Ranger (better than random forest)

#Modelling using Random Forest package

```
set.seed(1233)

RandomForest.model = randomForest(Personal.Loan~., data = bank.train)
print(RandomForest.model)
```

```
##
## Call:
##  randomForest(formula = Personal.Loan ~ ., data = bank.train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 1.26%
## Confusion matrix:
##      0   1 class.error
## 0 3146   5 0.001586798
## 1   39 310 0.111747851
```

Print the error rate

```
err = RandomForest.model$err.rate
head(err)
```

```
##             OOB          0         1
## [1,] 0.04977029 0.02303754 0.2835821
## [2,] 0.03820755 0.01731375 0.2242991
## [3,] 0.03591899 0.01444350 0.2281369
## [4,] 0.03509370 0.01360030 0.2326389
## [5,] 0.03399615 0.01317664 0.2225806
## [6,] 0.03121175 0.01055140 0.2151515
```

out of bag error

```
oob_err = err[nrow(err), "OOB"]
print(oob_err)  ## depicts the final out of bag error for all the samples
```

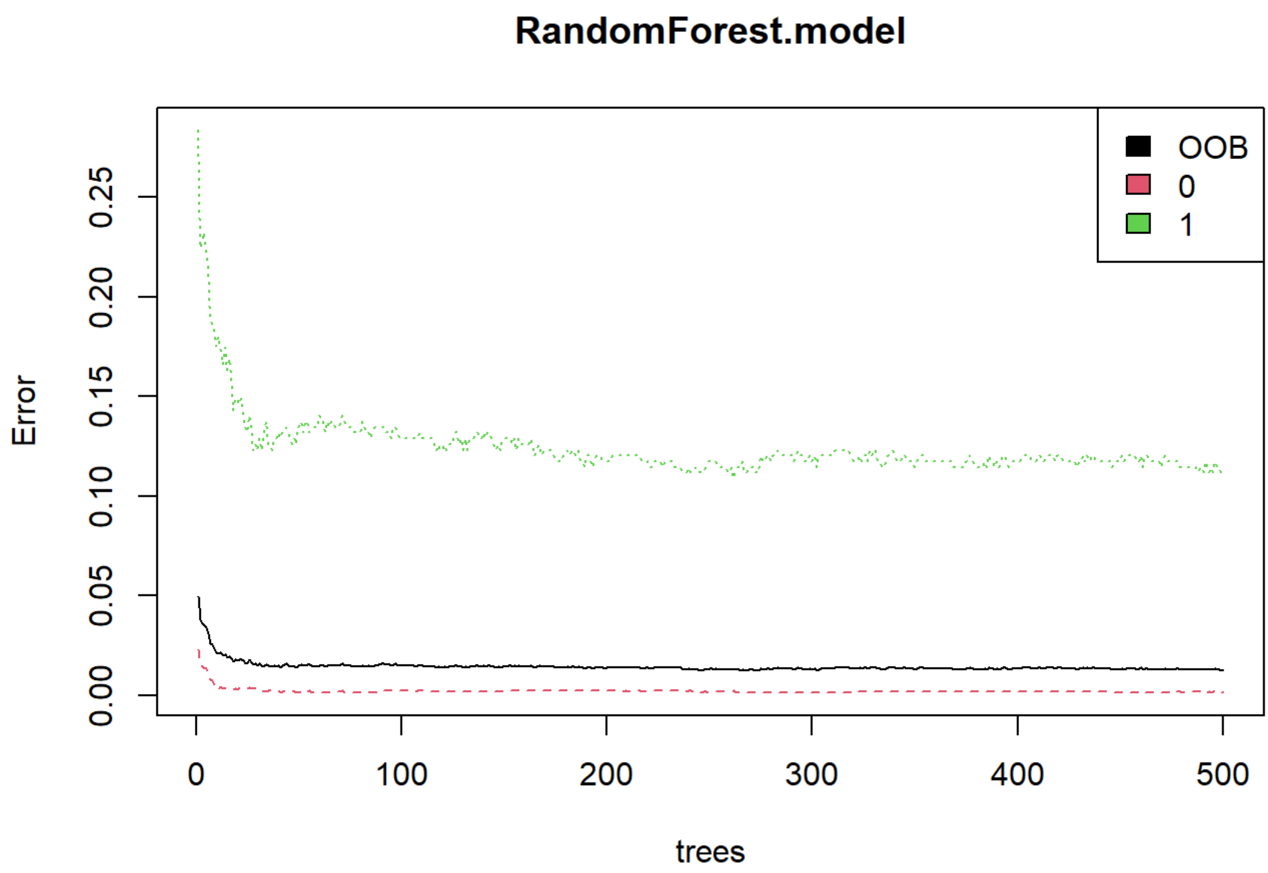
```
##        OOB
## 0.01257143
```

Following plot depecits the Out of Bag error for Class 0 and Class 1 and Overall OOB error.Also suggests the optimal trees we can use to tune Random forest model

Somewhere 250 - 350 trees should suffice as it saves time to train less trees and achieve same or even better results depending on cases

plot the OOB error

```
plot(RandomForest.model)
legend(x = "topright", legend = colnames(err), fill = 1:ncol(err))
```



#Prediction for Random Forest package

```
ranfost.pred = predict(RandomForest.model, bank.test, type = "prob")[,1]

bank.test$RFpred = ifelse(ranfost.pred >= 0.8,"1","0")

bank.test$RFpred = as.factor(bank.test$RFpred)

levels(bank.test$RFpred)
```

```
## [1] "0" "1"
```

```
RFConf.Matx = confusionMatrix(bank.test$RFpred, bank.test$Personal.Loan, positive = "1")
RFConf.Matx
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0   19  125
##          1 1350    6
##
##                Accuracy : 0.0167
##                  95% CI : (0.0108, 0.0245)
##     No Information Rate : 0.9127
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.1799
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.045802
##             Specificity : 0.013879
##          Pos Pred Value : 0.004425
##          Neg Pred Value : 0.131944
##              Prevalence : 0.087333
##          Detection Rate : 0.004000
##    Detection Prevalence : 0.904000
##       Balanced Accuracy : 0.029840
##
##        'Positive' Class : 1
##
```

```
table(bank.test$Personal.Loan)
```

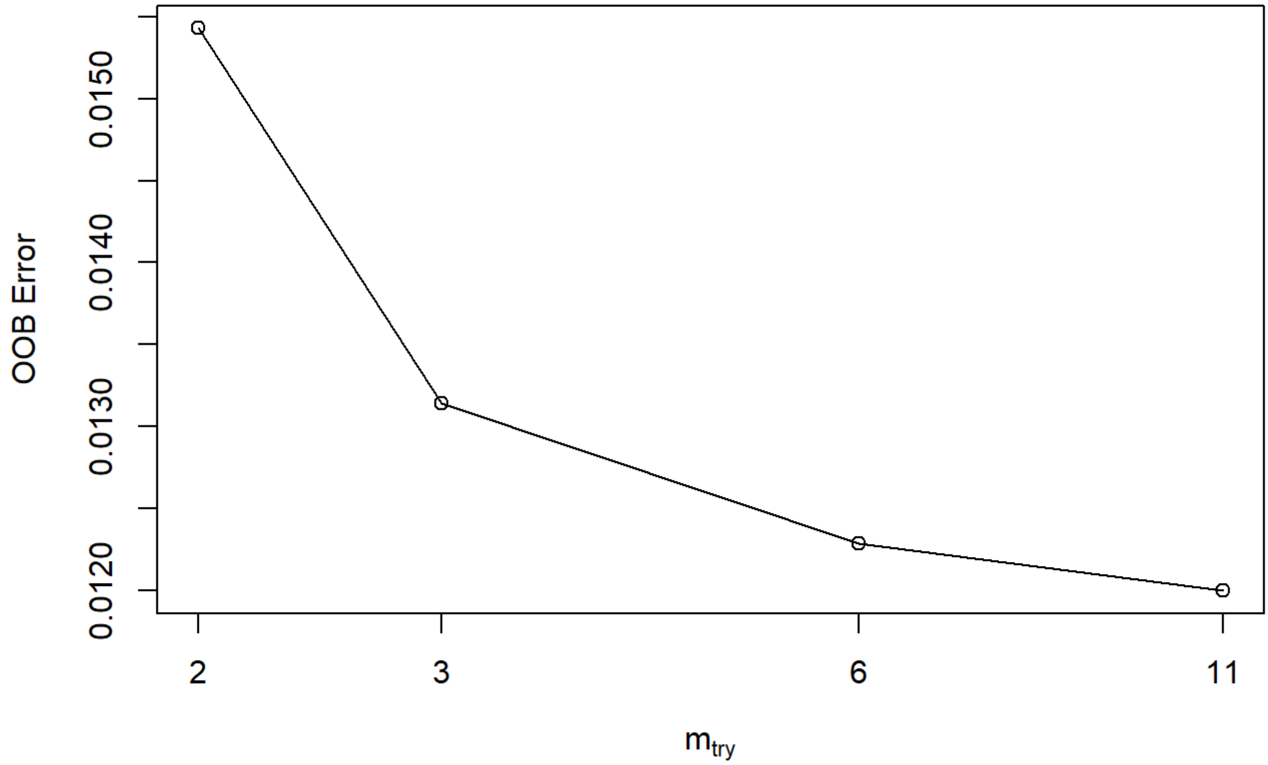
```
##
##    0    1
## 1369  131
```

#Tuning the Random Forest algo

Using the tuneRF function to random forest algorithm to get some idea about improving the performance

```
set.seed(333)

tuned.RandFors = tuneRF(x = subset(bank.train, select = -Personal.Loan),
             y = bank.train$Personal.Loan,
             ntreeTry = 501, doBest = T)
```

```
## mtry = 3  OOB error = 1.31%
## Searching left ...
## mtry = 2     OOB error = 1.54%
## -0.173913 0.05
## Searching right ...
## mtry = 6     OOB error = 1.23%
## 0.06521739 0.05
## mtry = 11    OOB error = 1.2%
## 0.02325581 0.05
```




```
print(tuned.RandFors)
```

```
##
## Call:
##  randomForest(x = x, y = y, mtry = res[which.min(res[, 2]), 1])
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##         OOB estimate of  error rate: 1.37%
## Confusion matrix:
##      0   1 class.error
## 0 3139  12 0.003808315
## 1   36 313 0.103151862
```

#Modelling using ranger package

Ranger package has been built atop randomforest package and has got better performance then rf models as it has less parameters to tune on.

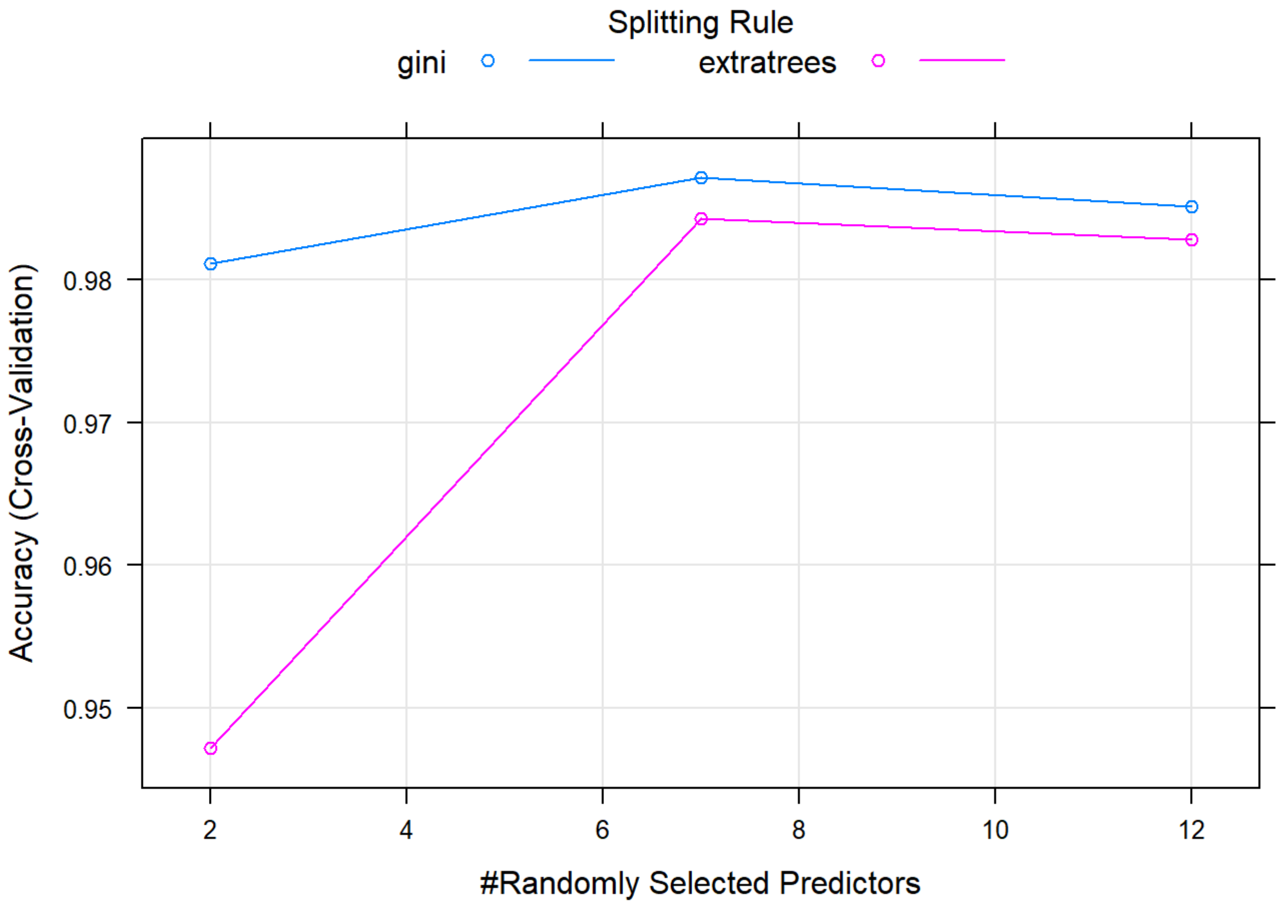
Mostly we need to bother about mtry only which is the number of variables we will use to build various trees. This takes care of other parameters like minimum number of splits , nodes etc.

Since this is classification problem it automatically chooses best method to for split rules and uses minimum node size as 1

```
set.seed(999)

RG.model = train(Personal.Loan~., data = bank.train, tuneLength = 3,
                 method = "ranger",
                 trControl = trainControl(method = 'cv',
                                          number = 5,
                                          verboseIter = FALSE))
RG.model
```

```
## Random Forest
##
## 3500 samples
##   11 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2800, 2799, 2801, 2800, 2800
## Resampling results across tuning parameters:
##
##   mtry  splitrule   Accuracy   Kappa
##    2    gini        0.9811396  0.8853611
##    2    extratrees  0.9471428  0.6135551
##    7    gini        0.9871424  0.9261296
##    7    extratrees  0.9842845  0.9080179
##   12    gini        0.9851404  0.9147375
##   12    extratrees  0.9828551  0.9006025
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 7, splitrule = gini
##  and min.node.size = 1.
```

```
plot(RG.model)
```

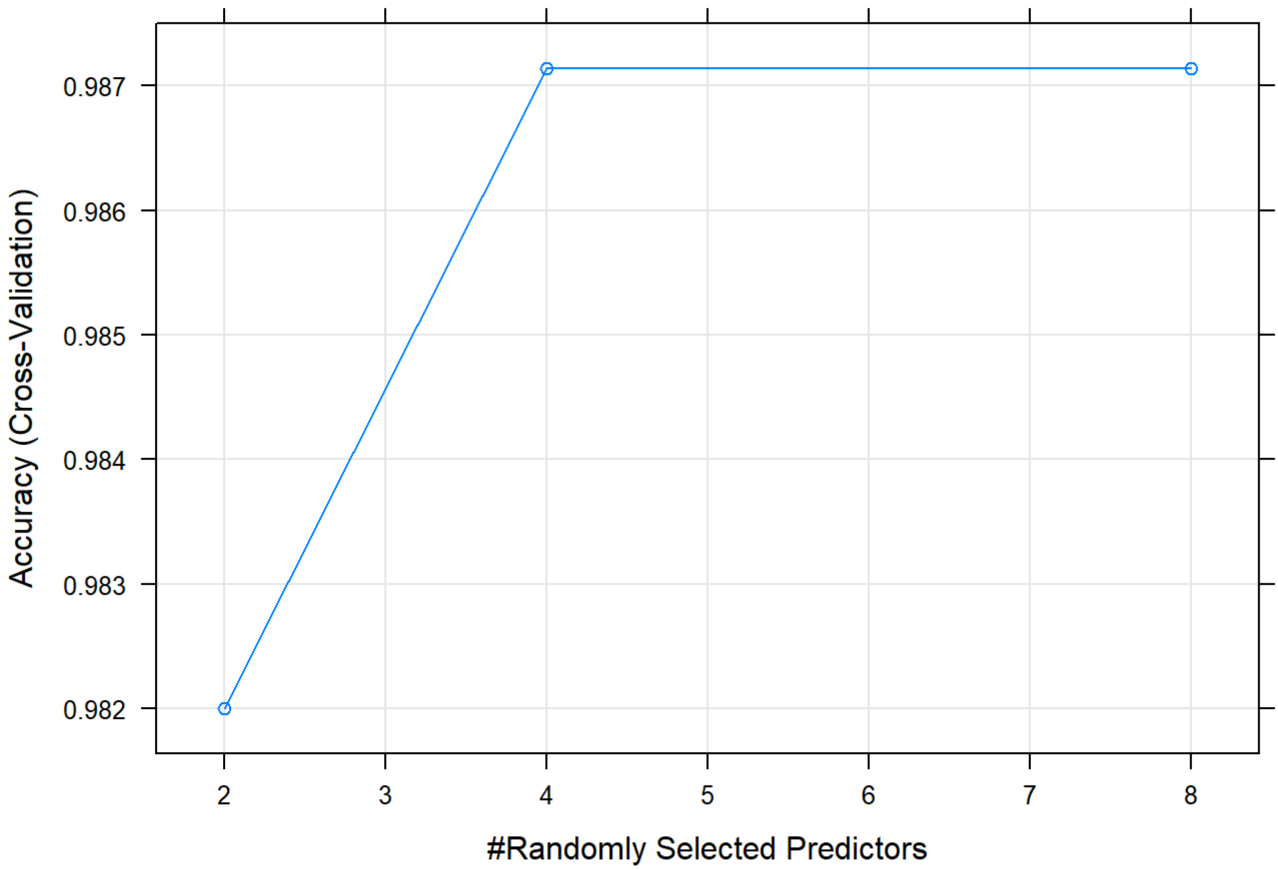


#Tuning ranger grid

```
tuneGrid = data.frame(.mtry = c(2,4,8), .splitrule = "gini", .min.node.size = 1)

set.seed(22222)
RFgrid.model = train(Personal.Loan~., data = bank.train, tuneGrid = tuneGrid,
                  method = "ranger",
                  trControl = trainControl(method = 'cv',
                                           number = 5, verboseIter = FALSE))
RFgrid.model
```

```
## Random Forest
##
## 3500 samples
##   11 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2800, 2800, 2799, 2801, 2800
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9819988  0.8908757
##   4     0.9871420  0.9254554
##   8     0.9871428  0.9261283
##
## Tuning parameter 'splitrule' was held constant at a value of gini
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 8, splitrule = gini
##  and min.node.size = 1.
```

```
plot(RFgrid.model)
```



#Refined ranger model

After the grid tunining we settle the number of trees to 511 and mtry = 4

```
set.seed(101)

Range.model = ranger(Personal.Loan~., data = bank.train, num.trees = 511,
                     mtry = 4, min.node.size = 1, verbose = FALSE)

Range.model
```

```
## Ranger result
##
## Call:
##  ranger(Personal.Loan ~ ., data = bank.train, num.trees = 511,      mtry = 4, min.node.siz
e = 1, verbose = FALSE)
##
## Type:                             Classification
## Number of trees:                  511
## Sample size:                      3500
## Number of independent variables:  11
## Mtry:                             4
## Target node size:                 1
## Variable importance mode:         none
## Splitrule:                        gini
## OOB prediction error:             1.34 %
```

#Prediction of RangeR package

Using ranger package and its grid model approach of cross validation we observe that its able to predict 120 cases of class 1 (Loan) correctly out of available 131 cases

This quantum jump from all previous models

```
range.pred = predict(Range.model, bank.test)

table(bank.test$Personal.Loan, range.pred$predictions)
```

```
##
##         0    1
##   0 1363    6
##   1   11  120
```

#Confusion Matrix of RangeR package

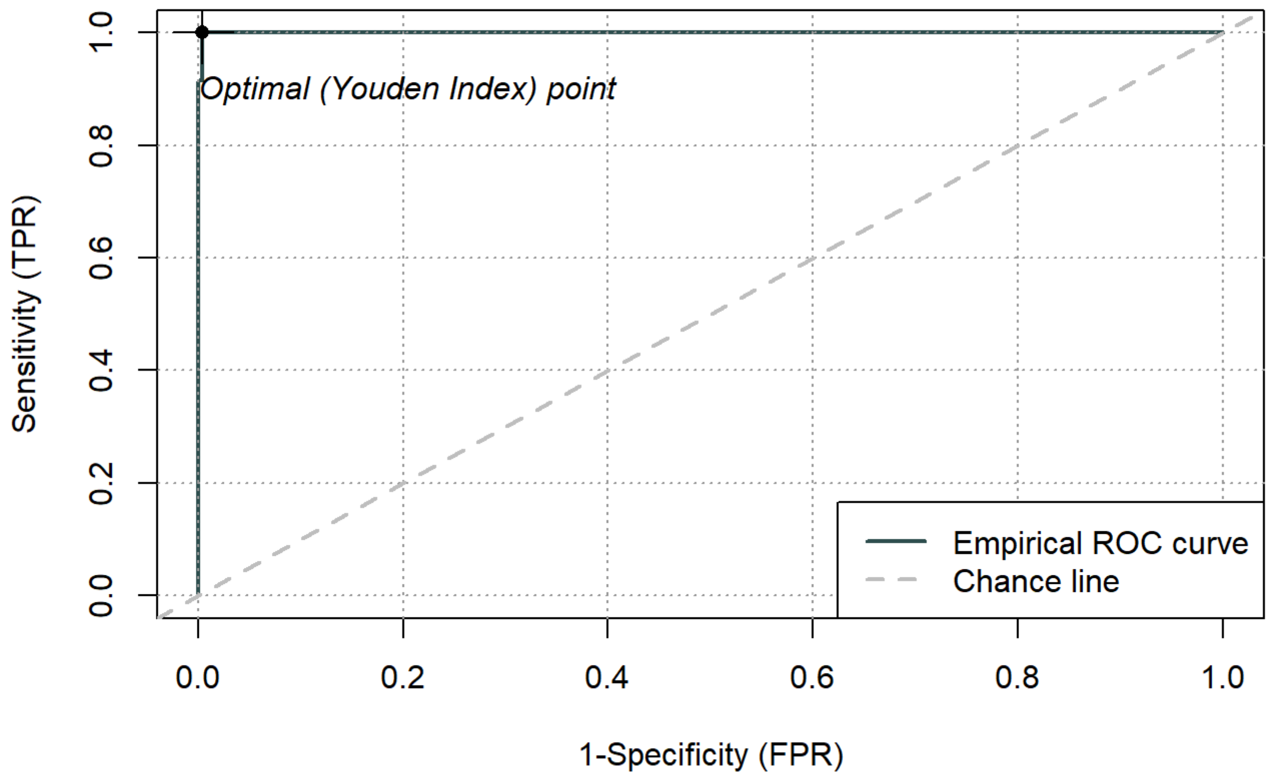
```
Range.ConMatx = confusionMatrix(range.pred$predictions,
                                bank.test$Personal.Loan, positive = "1")
Range.ConMatx
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1363   11
##          1    6  120
##
##                Accuracy : 0.9887
##                  95% CI : (0.9819, 0.9934)
##     No Information Rate : 0.9127
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9277
##
##  Mcnemar's Test P-Value : 0.332
##
##             Sensitivity : 0.91603
##             Specificity : 0.99562
##          Pos Pred Value : 0.95238
##          Neg Pred Value : 0.99199
##              Prevalence : 0.08733
##          Detection Rate : 0.08000
##    Detection Prevalence : 0.08400
##       Balanced Accuracy : 0.95582
##
##        'Positive' Class : 1
##
```

#Plotting of ROC curve

Plotting of ROC curve is another way of checking the Classification Model's performance. It is curve between Sensitivity (True Positive Rate) and 1 - Specifivity (False Positive Rate)

```
Prediction.Labels = as.numeric(range.pred$predictions)
Actual.Labels = as.numeric(bank.test$Personal.Loan)

roc_Rf = rocit(score = Prediction.Labels, class = Actual.Labels)

plot(roc_Rf)
```



#Conclusion

Various types of models were attempted Some raw , some refined and tuned to display the their dissimilarity in approaching the same dataset under mostly similar conditions.

If given a choice between low OOB (out of bag) error and Accuracy . I will go with accuracy as this case demands so.

As financial institution we want to be more than 100% sure that there should be no tolerance for defaults and we are able to earn from interest income

So under Circumstnces ranger (Random Forest) performs the best on dataset with accurancy of 98%

Model Name OOB errors % Accuracy % CART 0.21 1.67 Random Forest 1.2 1.8 tuned Random Forest 1.17 90.3 ranger Random Forest 1.31 98.8