Project – Thera Bank Case Study

Apr 2020

Author : Snehal Gawand

Table of Contents

1. Project Objective

This case study is about Thera Bank which has a growing customer base. Majority of these customers are liability customers (depositors) with varying size of deposits. The number of customers who are also borrowers (asset customers) is quite small, and the bank is interested in expanding this base rapidly to bring in more loan business and in the process, earn more through the interest on loans. A campaign that the bank ran last year for liability customers showed a healthy conversion rate of over 9% success.

The objective is to build the best model which can classify the right customers who have a higher probability of purchasing the loan.

2. Assumptions

Below are few factors which might affect the likelihood of a liability customer buying personal loans.
Age: Customers within the age group of 30-50 are more likely to buy personal loans
Family members: Higher the number of family members earning, less is the probability of buying personal loan
Education: The education of customer can affect the personal loan buying probability. People who are graduated or Advanced Professionals are more viable to buy personal loans from a bank rather than people who are under-graduated.

3. Environmental setup, dataset import and variables identification

Let's import the data into R environment and perform preliminary analysis

```
#install packages/libraries
>library("readxl")

#Set-up working directory
> setwd("~/Documents/Projects/Project 3")

#upload dataset in R
> TheraBank <- read_excel("Thera Bank_Personal_Loan_Modelling-dataset.xlsx")

#check dimensions of dataset
> dim(TheraBank)
[1] 5000   14
```

Dataset has 5000 observations and 14 variables

```
#check for missing values
> any(is.na(TheraBank))
[1] TRUE
```

```
> colSums(is.na(TheraBank))
                ID        Age (in years) Experience (in years)   Income (in K/month)
                 0                     0                     0                     0
          ZIP Code        Family members                 CCAvg             Education
                 0                    18                     0                     0
          Mortgage          Personal Loan    Securities Account           CD Account
                 0                     0                     0                     0
            Online             CreditCard
                 0                     0
```

Column "Family members" has 18 observations with missing values

#Missing Value Treatment

As the family member has NA values and mean/median or any missing value treatment can result in adding some approximate value to those 18 observations, the data might get changed. Hence, replacing the NAs with zeroes.

```
> TheraBank[is.na(TheraBank)] = 0
> sum(is.na(TheraBank))
[1] 0
```

After replacing NAs with zeroes, dataset does not have any missing values

#Checking rows with negative values as Experience

```
 > TheraBank[TheraBank$`Experience (in years)`<0,]
# A tibble: 52 x 14
      ID `Age (in years)` `Experience (in~ `Income (in K/m~ `ZIP Code` `Family members`
   <dbl>          <dbl>            <dbl>          <dbl>      <dbl>            <dbl>
 1    90             25               -1            113      94303                4
 2   227             24               -1             39      94085                2
 3   316             24               -2             51      90630                3
 4   452             28               -2             48      94132                2
 5   525             24               -1             75      93014                4
 6   537             25               -1             43      92173                3
 7   541             25               -1            109       94010               4
 8   577             25               -1             48      92870                3
 9   584             24               -1             38      95045                2
10   598             24               -2            125       92835               2
# ... with 42 more rows, and 8 more variables: CCAvg <dbl>, Education <dbl>,
#   Mortgage <dbl>, `Personal Loan` <dbl>, `Securities Account` <dbl>, `CD
#   Account` <dbl>, Online <dbl>, CreditCard <dbl>
```

#fixing negative values

```
> TheraBank$`Experience (in years)` = abs(TheraBank$`Experience (in years)`)
```

#Checking negative values again

```
> TheraBank[TheraBank$`Experience (in years)`<0,]
# A tibble: 0 x 14
```

After replacing negatives with positive values, dataset does not have any Experience in negative values

```
#remove unwanted variables
>TheraBank = TheraBank[,-c(1,5)]
```
The 'CustomerID' and 'ZipCode' are not important variables to perform the analysis, hence removed them from dataset

```
#converting columns to factors
>TheraBank$Education = factor(TheraBank$Education,levels = c("1", "2", "3"), order =
TRUE)
>TheraBank$`Personal Loan` = as.factor(TheraBank$`Personal Loan`)
>TheraBank$`Securities Account`=as.factor(TheraBank$`Securities Account`)
>TheraBank$`CD Account`=as.factor(TheraBank$`CD Account`)
>TheraBank$Online = as.factor((TheraBank$Online))
>TheraBank$CreditCard = as.factor((TheraBank$CreditCard))

#summary
> summary(TheraBank)
 Age (in years)  Experience (in years) Income (in K/month) Family members
 Min.   :23.00   Min.   : 0.00        Min.   :  8.00      Min.   :0.000
 1st Qu.:35.00   1st Qu.:10.00        1st Qu.: 39.00      1st Qu.:1.000
 Median :45.00   Median :20.00        Median : 64.00      Median :2.000
 Mean   :45.34   Mean   :20.13        Mean   : 73.77      Mean   :2.389
 3rd Qu.:55.00   3rd Qu.:30.00        3rd Qu.: 98.00      3rd Qu.:3.000
 Max.   :67.00   Max.   :43.00        Max.   :224.00      Max.   :4.000
    CCAvg        Education    Mortgage     Personal Loan Securities Account CD Account
 Min.   : 0.000   1:2096    Min.   :  0.0   0:4520        0:4478             0:4698
 1st Qu.: 0.700   2:1403    1st Qu.:  0.0   1: 480        1: 522             1: 302
 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
 Online   CreditCard
 0:2016   0:3530
 1:2984   1:1470

#Ratio of personal loans
> prop.table(table(TheraBank$`Personal Loan`))*100

  0   1
90.4  9.6
```

Conclusion:

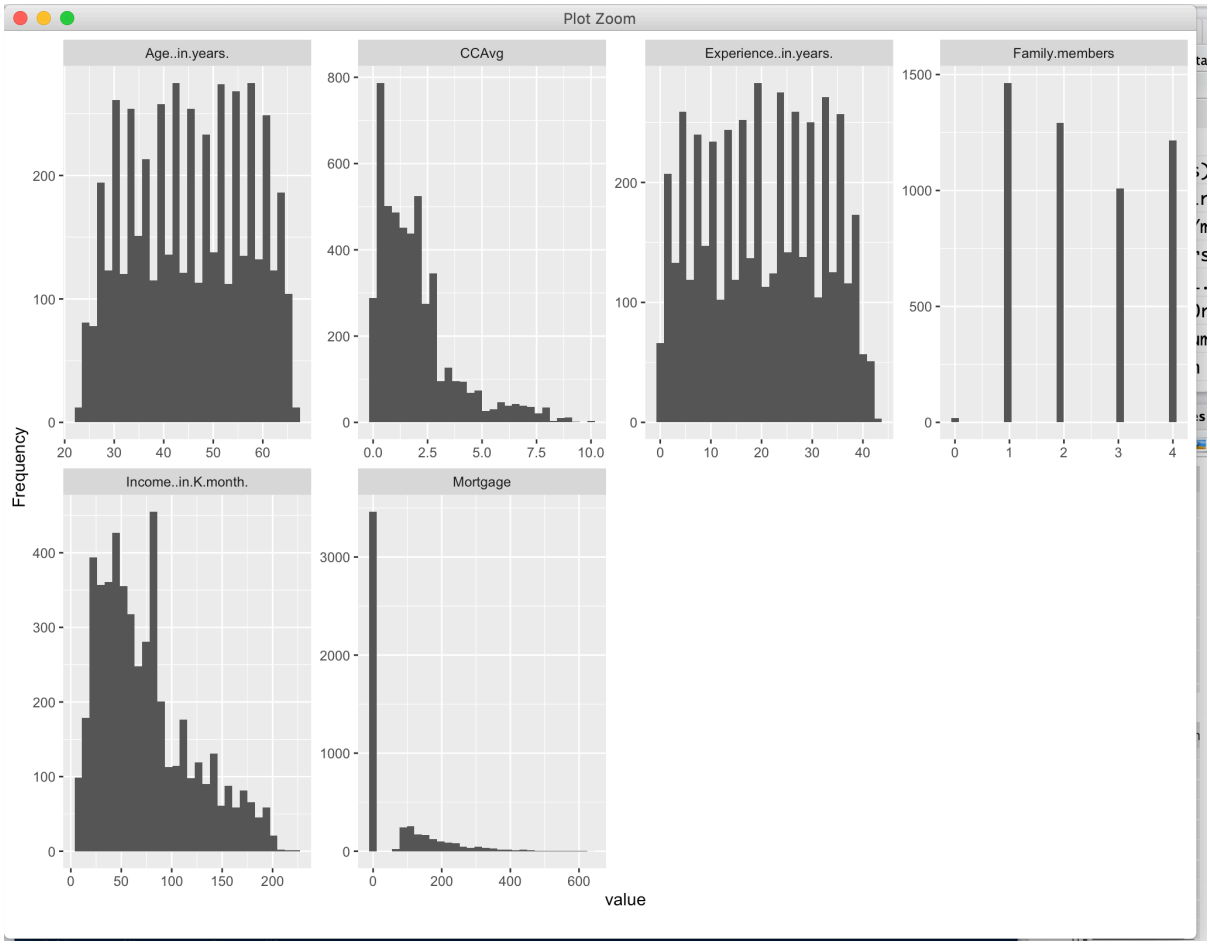
- The Thera Bank Dataset has data of 5000 customers and for 14 different variables
- Family Members have 18 missing observations (have values 'spaces"), hence replacing "spaces" with "Zeroes", as the information of those customers is useful and can't be discarded
- Customer experience in years cannot have negative value, however 52 customers have negative experience value. Hence, it is converted into positive value

- 9.6% customers accepted personal loan offer, whereas 90.4% customers didn't accept.

4. Exploratory Data Analysis

- Univariate Analysis - Histogram

```
>library("DataExplorer")
>plot_histogram(TheraBank)
```



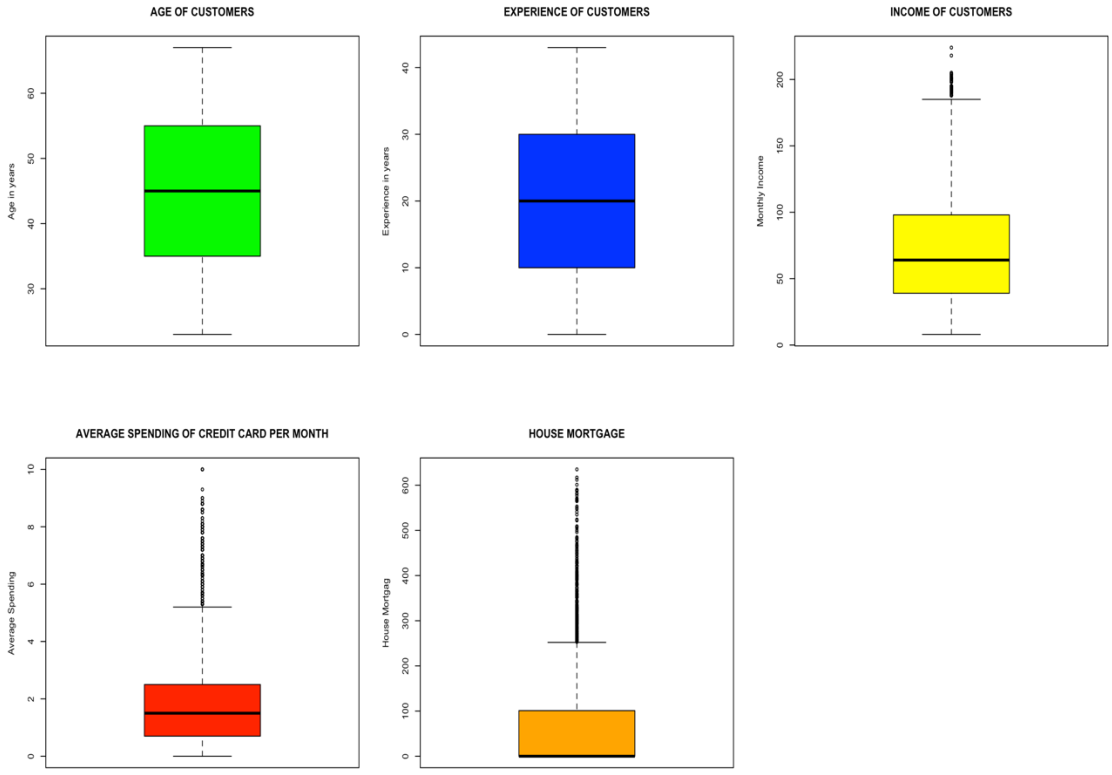
Conclusion:

- The histograms of Customer Age and Experience is very close to normal distribution

- Univariate Analysis – Box plot

```
>par(mfrow = c(2,3))
boxplot(TheraBank$`Age (in years)`,
  main = toupper("Age of customers"),
  ylab = "Age in years",
```

```
col = "green")

>boxplot(TheraBank$`Experience (in years)`,
main = toupper("Experience of customers"),
ylab = "Experience in years",
col = "blue")

>boxplot(TheraBank$`Income (in K/month)`,
main = toupper("Income of customers"),
ylab = "Monthly Income",
col = "yellow")

>boxplot(TheraBank$CCAvg,
main = toupper("Average Spending of credit card per month"),
ylab = "Average Spending",
col = "red")

>boxplot(TheraBank$Mortgage,
main = toupper("House Mortgage"),
ylab = "House Mortgag",
col = "orange")
```

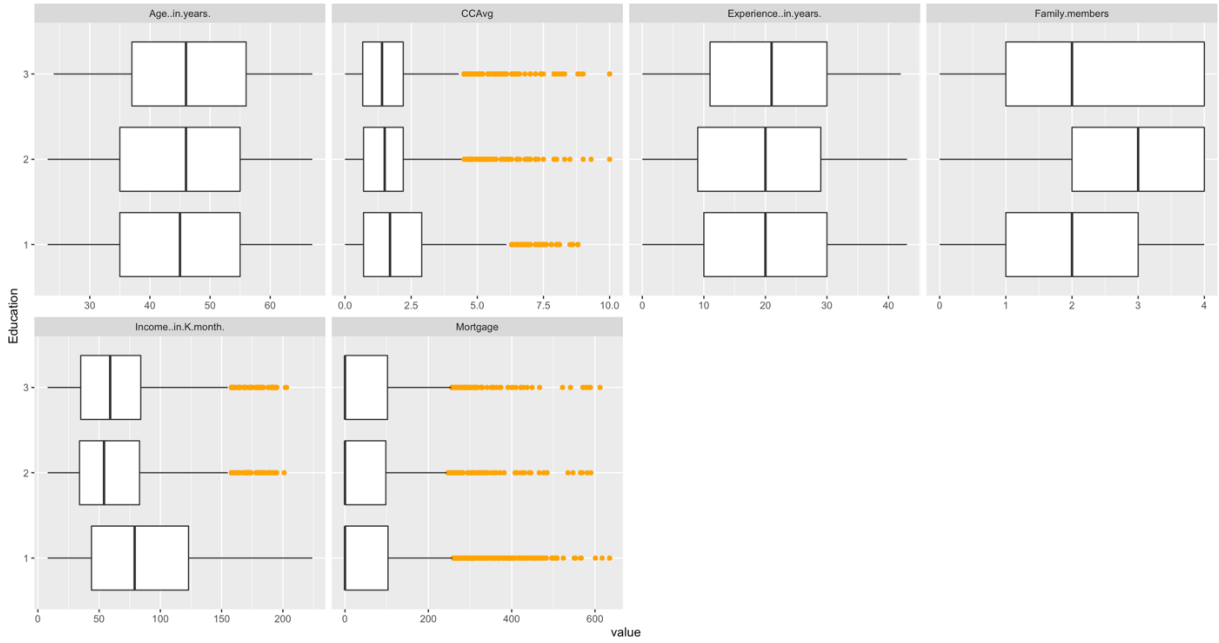


Conclusion:

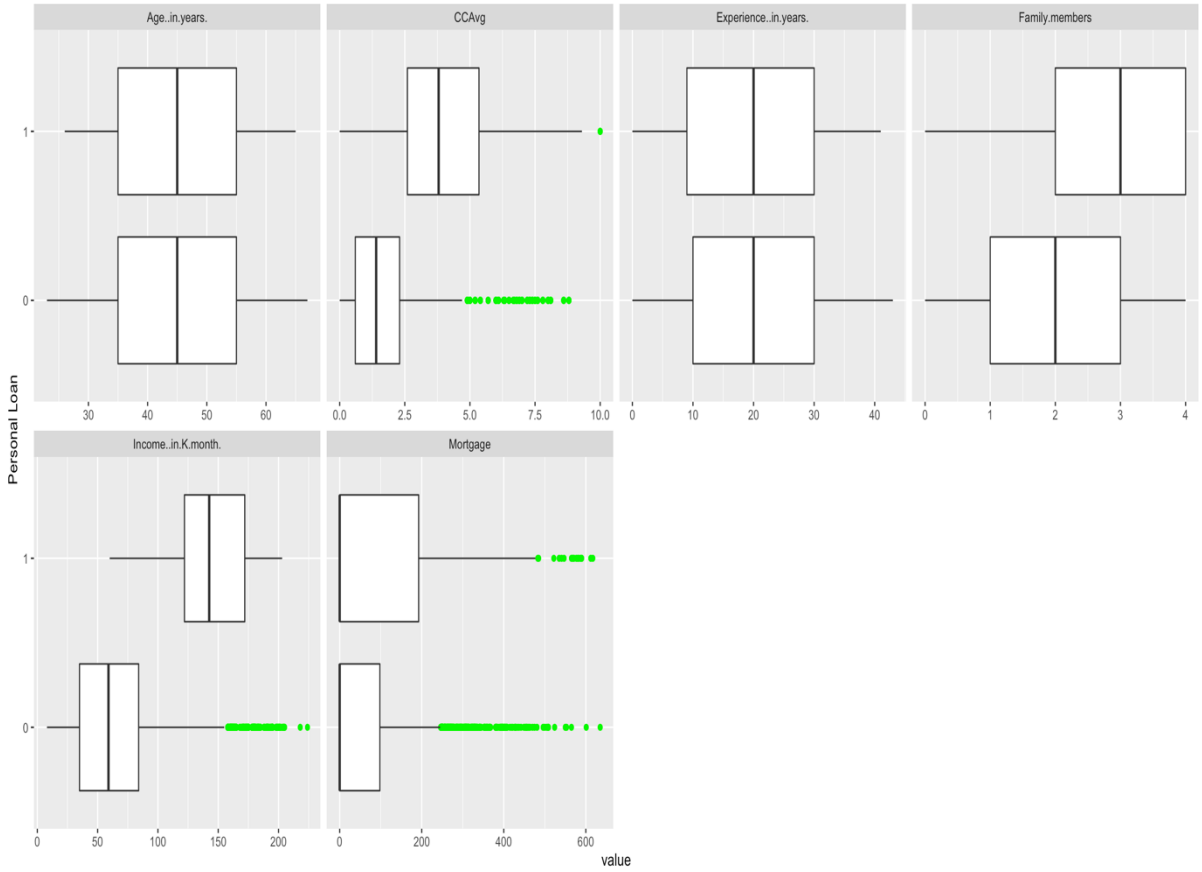
- There are very few outliers in Age and Experience column
- Monthly income, average spending of credit card and house mortgage has lots of outliers.

- Bivariate Analysis – Box plot

```
>plot_boxplot(TheraBank, by = "Education",
 geom_boxplot_args = list("outlier.color" = "orange"))
```



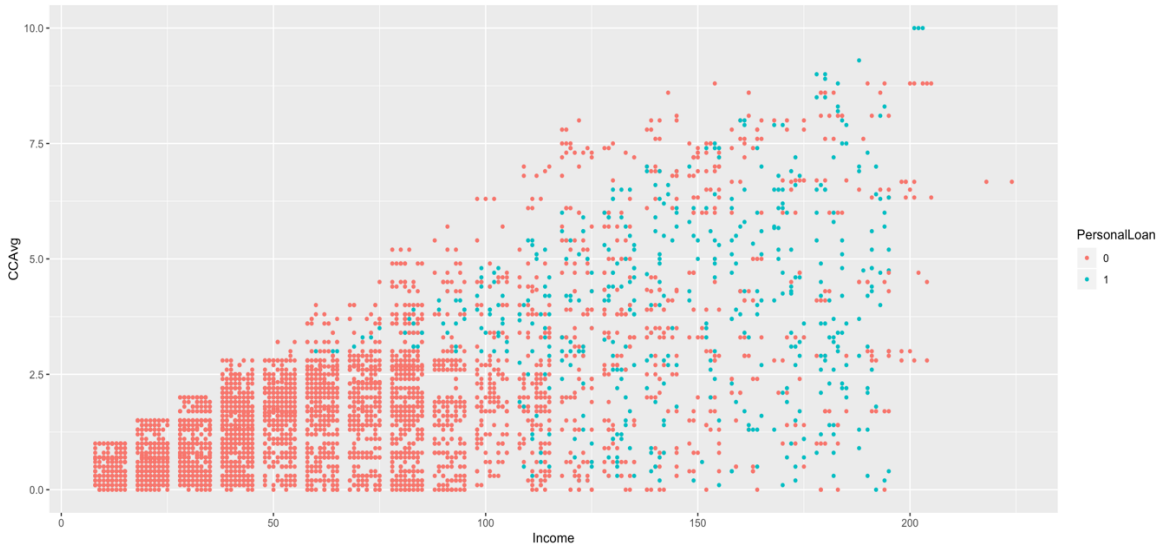

```
>plot_boxplot(TheraBank, by = "Personal Loan",
geom_boxplot_args = list("outlier.color" = "green"))
```



Conclusion:

- Customers using credit card and availing Mortgage have lots of outliers across all three levels of Education
- Customers with Education level of Graduates and Advanced professionals have outliers in Income.
- Customers who did not accepted Personal Loan offered during last campaign, have lots of outliers in Credit card, Income and Mortgage usage.

```
#convert column names for ease of use
>library("dplyr")
>TheraBank = TheraBank %>% rename(Age = `Age (in years)`,
                        Experience = `Experience (in years)`,
                         Income = `Income (in K/month)`,PersonalLoan = `Personal Loan`,
                         CDAccount = `CD Account`, SecuritiesAccount = `Securities Account`,
                         Familymembers = `Family members`)

>ggplot(TheraBank, aes(Income,y = CCAvg, color = PersonalLoan)) + geom_point(size = 1)
```



Conclusion:

- Customers having income upto 100K have no Personal loans and moderate Credit Card spending (under 3000)
- There are good number of customers with income in the range of 40K to 100K, having credit card average spend less than $ 2500. These customers can become good prime targets to buy loan.

5. Clustering

As clustering feature is only for numeric variables, removing the categorical variables from the dataset. Also scaling the dataset

```
>bank = TheraBank %>% select_if(is.numeric)
>bank.scaled = scale(bank,center=TRUE)
```
Here the dataset is scaled and has only the columns Age, Experience, Income, Family Members, CCAvg, Mortgage

For unsupervised learning, hierarchical and k-means clustering are two best suited methods.

- Hierarchical clustering

```
#print new matrix by computing the distance using hierarchical clustering
dist.bank.scaled = dist(bank.scaled,method = "euclidean")
print(dist.bank.scaled,digits=3)
```

As the dataset is large with 5000 observations, we cannot use hierarchical method.

- K-means clustering

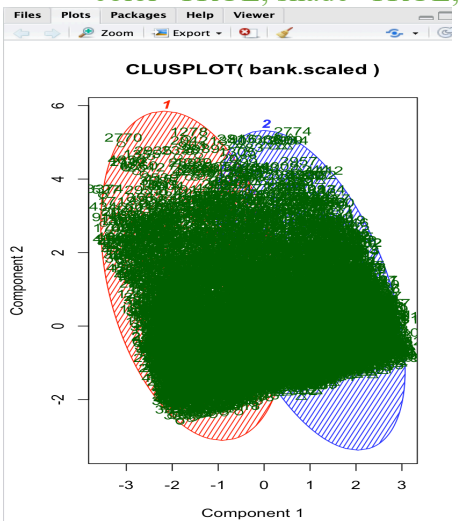
```
#k-means clustering
>seed=1000
>set.seed(seed) #since kmeans uses a randomized starting point for cluster centroids
```

We will first cluster the data into two clusters. Later below we will try to choose the optimal number of clusters.

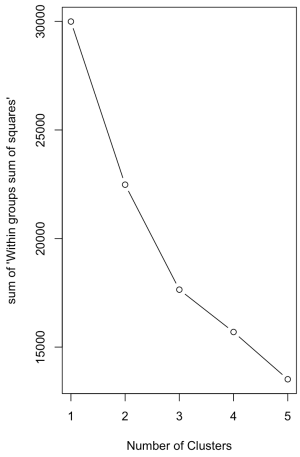
```
>cluster1 = kmeans(x=bank.scaled, centers = 2, nstart = 5)
```

Let's plot these two clusters

```
>library(cluster)
>clusplot(bank.scaled, cluster1$cluster,
>         color=TRUE, shade=TRUE, labels=2, lines=1)
```



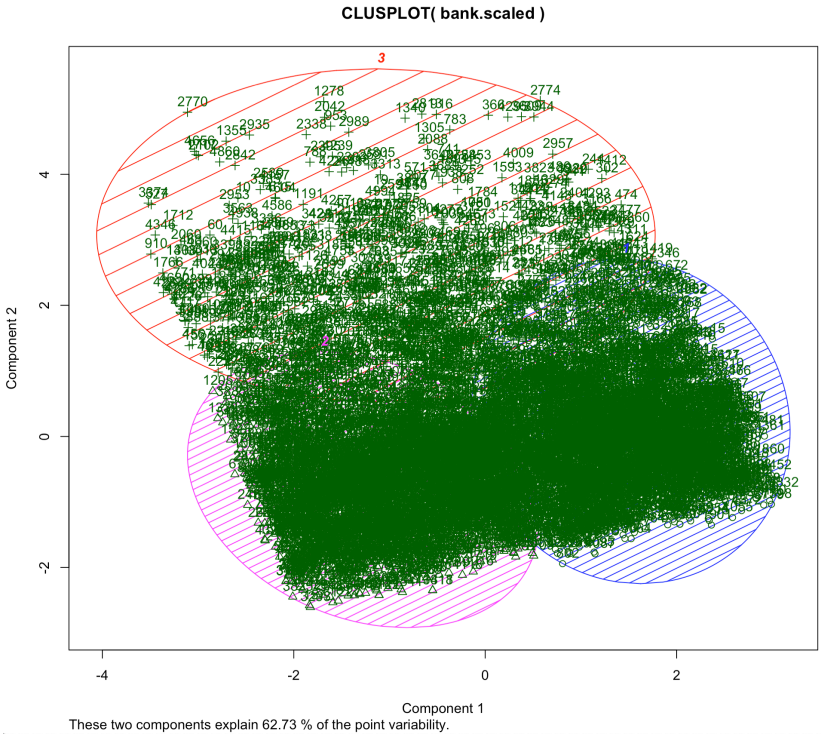
To look for optimal number of clusters, Let's try K=2 to 5 and for each plot the "sum of Within cluster sum of squares"



The graph indicates K=3 might be a good choice (elbow argument).

Creating cluster with k=3

```
>set.seed(seed) #since kmeans uses a randomized starting point for cluster centroids

>cluster2 = kmeans(x=bank.scaled, centers = 3, nstart = 10)

>clusplot(bank.scaled, cluster2$cluster, color=TRUE, shade=TRUE, labels=2, lines=1)
```



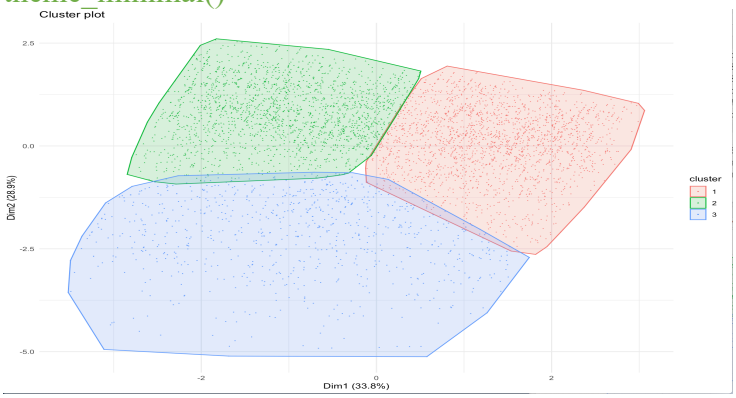
Adding cluster numbers into the input dataset, to view the customers belonging to each of the cluster

```
>bank$Clusters = cluster2$cluster
```

To view the clusters in better graphical representation,

```
>install.packages("factoextra")
>library("factoextra")
```

```
>fviz_cluster(cluster2, bank.scaled, geom = "point", ellipse = TRUE, pointsize = 0.2, ) +
theme_minimal()
```



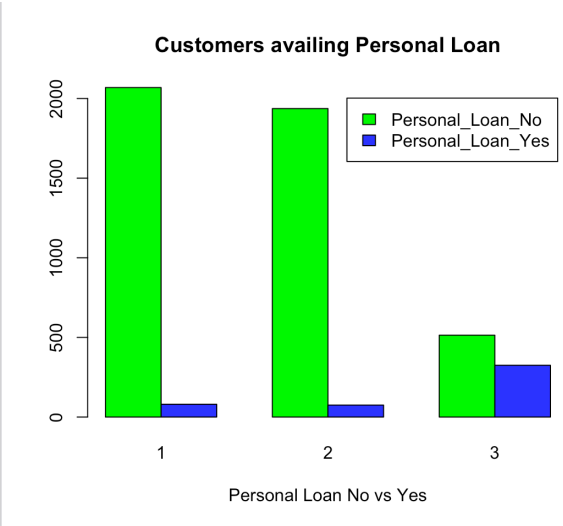

```
#Aggregating rows based on clusters they fall into
> custProfile = aggregate(bank,list(bank$Clusters),FUN="mean")
> print(custProfile)
```

	Group.1	Age	Experience	Income	Family members	CCAvg	Mortgage	Clusters
1	1	55.53792	30.243369	58.12704	2.376454	1.356524	44.77338	1
2	2	35.12724	9.939861	60.16849	2.606859	1.370641	45.59443	2
3	3	43.70083	18.690107	146.48033	1.896305	4.787592	112.68176	3

Adding these new Clusters back to original dataset

```
>bank1 = TheraBank
>bank1$Cluster = bank$Clusters
>View(bank1)
```

Plotting graph of Personal loan versus Clusters



Conclusion:

- K-means clustering mechanism was chosen over Hierarchical clustering as the dataset size is large
- Customers data in Thera Bank were divided into three clusters using k-means clustering with cluster size of 2149,2012 and 839

- Cluster 3 has customers which had majorly accepted personal loan in earlier campaign

6. Splitting dataset into Train and Test

Splitting the dataset into train and test in 70:30 ratio

```
> set.seed(1000)
> index = sample((1:nrow(TheraBank)),round(nrow(TheraBank)*0.7,0), replace = F)
> Train.TheraBank = TheraBank[index,]
> Test.TheraBank = TheraBank[-index,]
> dim(Train.TheraBank)
[1] 3500   12
> dim(Test.TheraBank)
[1] 1500   12

> table(Train.TheraBank$PersonalLoan)
   0    1
3158  342

> table(Test.TheraBank$PersonalLoan)
   0    1
1362  138
```

Conclusion:

- The dataset is divided into train dataset with 3500 observations, and test dataset with 1500 observations (70:30 ratio)
- Train dataset consist of 342 customers which opted for personal loan earlier, whereas 138 customers in Test dataset had opted for personal loan.

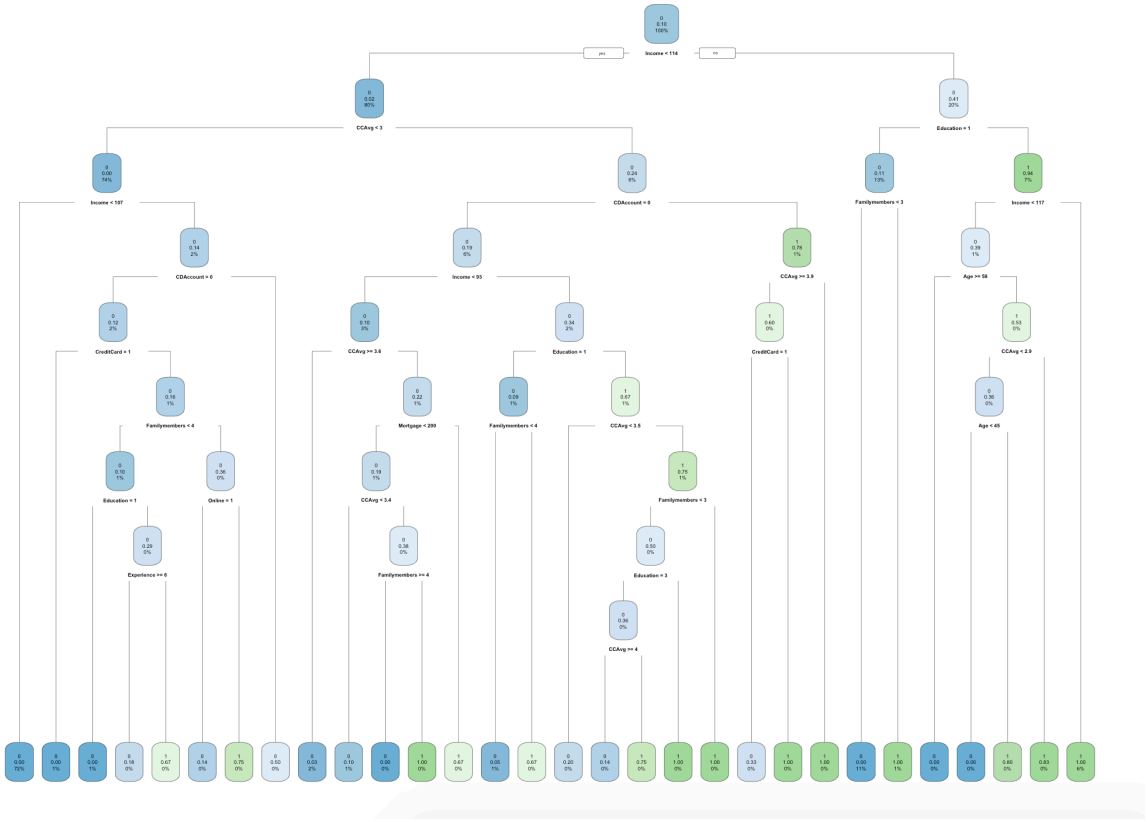
7. Model Building – CART

In the current dataset out of total customer count, 9.6% customers accepted personal loan offer. Hence, we need to analyze the attributes which can help to identify the right customers who have a higher probability of purchasing the loan in future.
Hence in our analysis Personal loan would be the target or the response variable i.e. the dependent variable and other variables would be independent or the predictor variables.

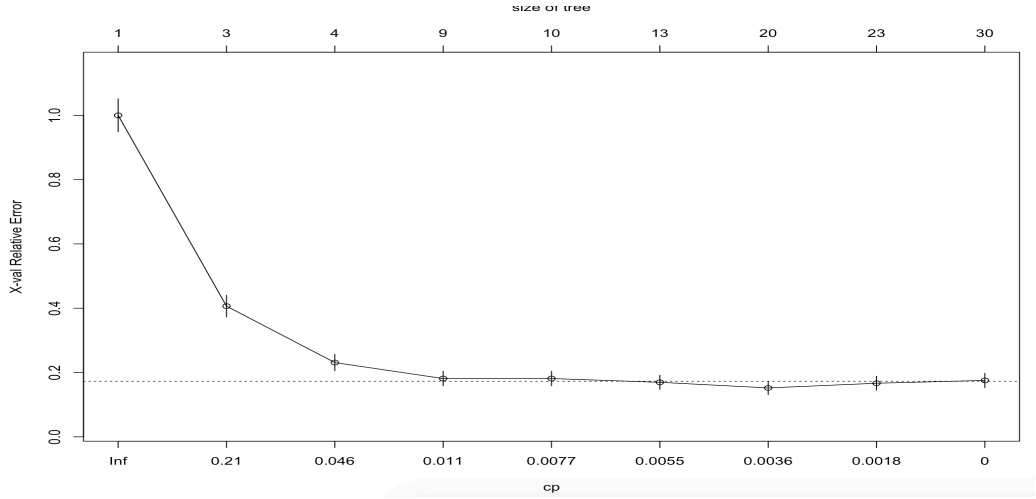
```
>library(rpart)
>library(rpart.plot)
>set.seed(1000)
```

We will begin by building a very complex classification tree, by setting the "cost complexity" threshold to "0" and the minimum bucket size to be 3. Then we will plot the tree using rpart.

```
> cart.model <- rpart(formula = PersonalLoan ~ .,
+                     data = Train.TheraBank, method = "class",cp=0, minbucket=3)
> rpart.plot(cart.model)
```



```
#Important variables
> cart.model$variable.importance
       Education           Income     Familymembers            CCAvg
      240.011332       199.666759       165.067488       103.298871
      CDAccount         Mortgage              Age       Experience
       51.717869        27.930525        17.010869        15.819857
         Online       CreditCard SecuritiesAccount
        6.242098         3.639625         1.352169

#Cost complexity table
>plotcp(cart.model)
```



Looking at the above graph of cost complexity, the unnecessarily complex tree above can be pruned using a cost complexity threshold of 0.045

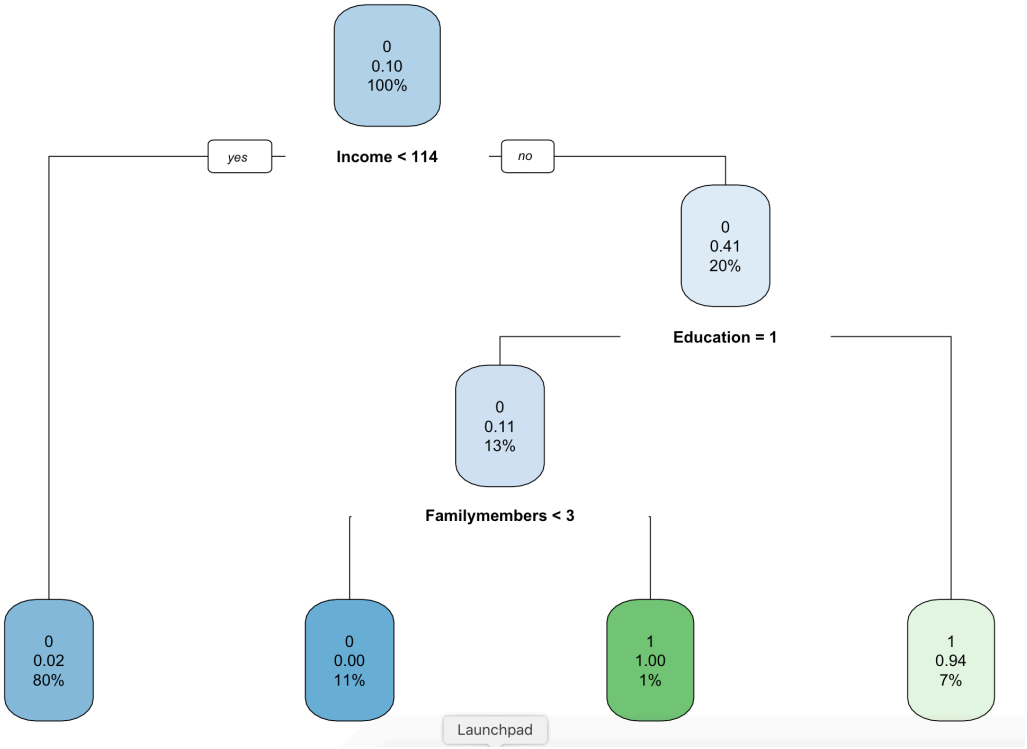
Conclusion:

- The first split is based on income greater than $114K
- Education, Income, Family member, CCAvg, CD account are the important variables for splitting according to above CART tree

- PRUNE CART TREE

Let us prune the CART tree using the complexity threshold value of 0.045

```
> pcart.model = prune(cart.model, cp= 0.045 ,"CP")
> rpart.plot(pcart.model)
```



Conclusion:

- First node shows there are 80% customers with income less than $114k
- Income is the first variable that is split in the decision tree hence is the most important variable as well while building any strategy.
- Targeting this low-income group customers would help the company to spend the marketing money on the correct customers rather than waste it on all customers

8. Model Building – Random Forest

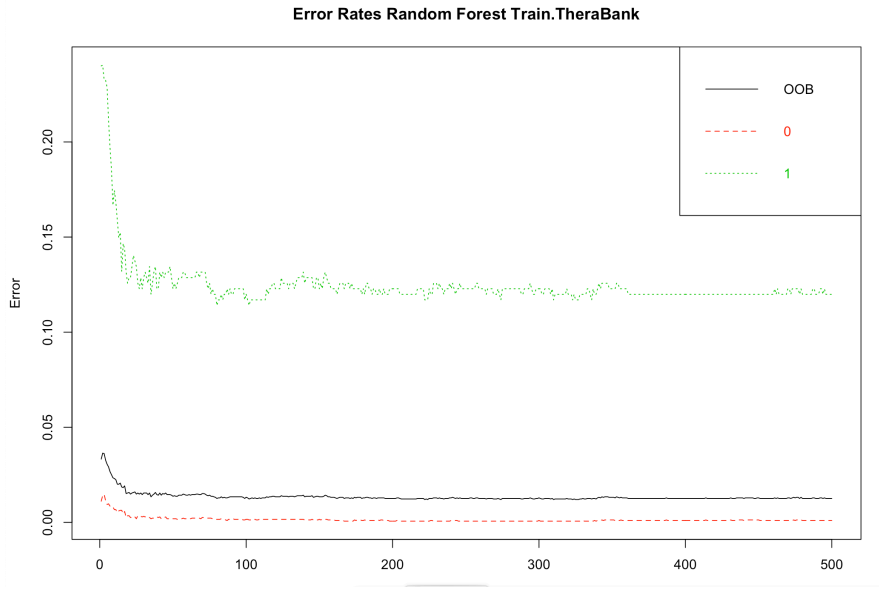
Random forest is an ensemble method used by combining weak and strong learners to give a better accuracy or output. It's a combination of multiple trees each chosen randomly to grow on dataset.

```
> library(randomForest)
> seed=1000
> set.seed(seed)
> rndForest.model = randomForest(PersonalLoan ~ ., data = Train.TheraBank)
> print(rndForest.model)

Call:
randomForest(formula = PersonalLoan ~ ., data = Train.TheraBank)
```

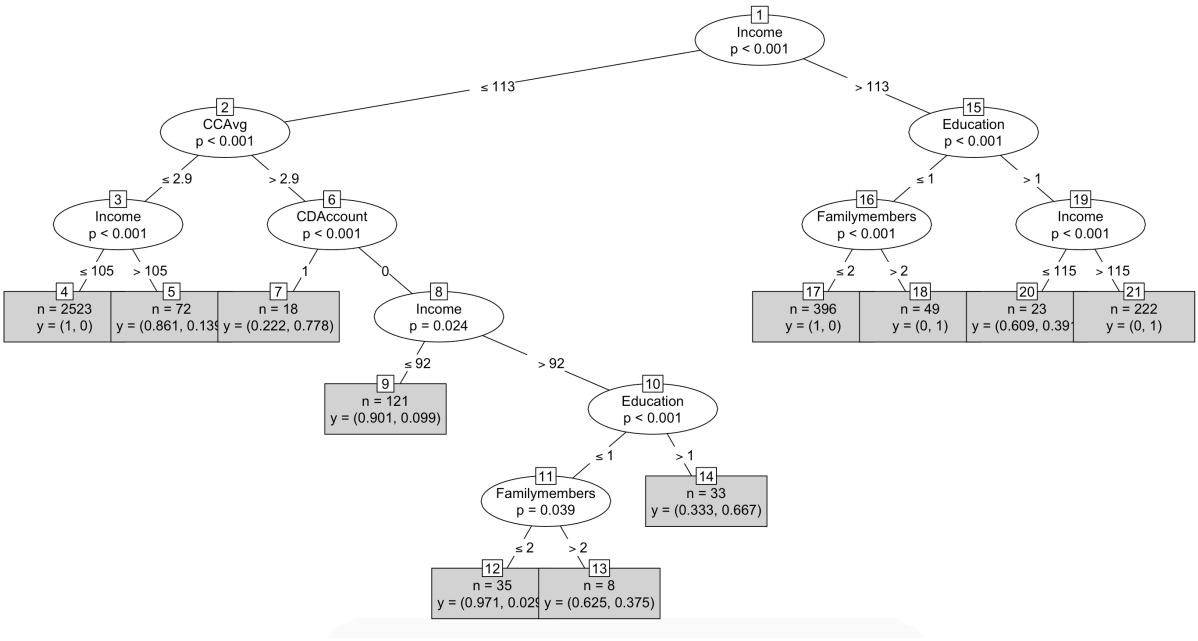
```
   Type of random forest: classification
         Number of trees: 500
No. of variables tried at each split: 3

OOB estimate of  error rate: 1.26%
Confusion matrix:
      0      1      class.error
0   3155   3      0.0009499683
1   41       301    0.1198830409
```

Plotting the OOB erro and the random forest model

```
> plot(rndForest.model, main="")
> legend("topright", c("OOB", "0", "1"), text.col=1:6, lty=1:3, col=1:3)
> title(main="Error Rates Random Forest Train.TheraBank")
```




```
> library("party")
> x <- ctree(PersonalLoan ~ ., data = Train.TheraBank)
> plot(x, type="simple")
```

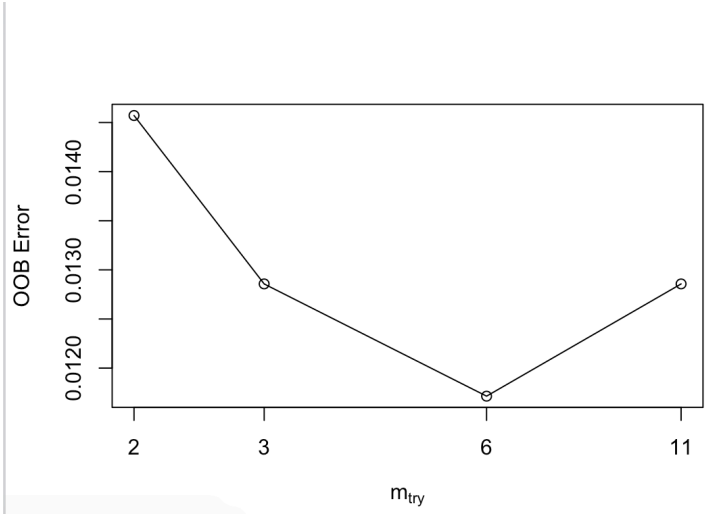


The above graph shows complex random forest model, hence tuning the model to increase the predictive power

- TUNING RANDOM FOREST MODEL

```
> seed=1000
> set.seed(seed)
> tune.rndForest = tuneRF(x=subset(Train.TheraBank,select = -PersonalLoan),
+                         y=Train.TheraBank$PersonalLoan,
+                         ntreeTry=501, doBest=T)
mtry = 3  OOB error = 1.29%
Searching left ...
mtry = 2        OOB error = 1.46%
-0.1333333 0.05
Searching right ...
mtry = 6        OOB error = 1.17%
0.08888889 0.05
mtry = 11       OOB error = 1.29%
-0.09756098 0.05
```

Tuning shows the when mtry (the number of variables we will use to build various trees) is 6, the Out Of Bag error is the lowest.



Let us print the tuned random forest model

```
> print(tune.rndForest)

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: 6

              OOB estimate of  error rate: 1.2%
Confusion matrix:
     0       1    class.error
0   3151   7      0.002216593
 1   35     307  0.102339181
```

Conclusion

- The Out Of Bag error is 1.2% which is very good, i.e. 98.8% accuracy.
- If we look at the Confusion Matrix, we can see that classification error is quite low. This shows that our RF model is performing well in classifying the train set.

9. Model Performance check and remarks

- CART PREDICTION FOR TEST AND TRAIN DATASET

For prediction, as we are going to add additional columns to the dataset, mirroring the test and train datasets

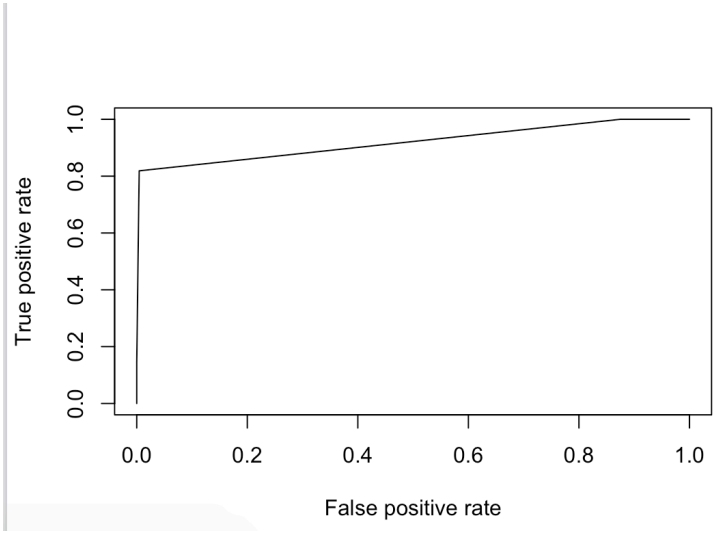
```
> Train1 = Train.TheraBank
> Test1 = Test.TheraBank
```

Scoring the train dataset and deciling the code to generate predictions

```
> Train1$predict.class <- predict(pcart.model, Train1, type="class")
> Train1$predict.score <- predict(pcart.model, Train1, type="prob")
```

Let's plot the graph between True Positive Rate and False Positive Rate of train dataset

```
> library(ROCR)
> library(ineq)
> pred <- prediction(Train1$predict.score[,2], Train1$PersonalLoan)
> perf <- performance(pred, "tpr", "fpr")
> plot(perf)
```




```
> KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
> auc <- performance(pred,"auc");
> auc <- as.numeric(auc@y.values)
> gini = ineq(Train1$predict.score[,2], type="Gini")
> with(Train1, table(PersonalLoan, predict.class))
            predict.class
PersonalLoan    0    1
           0 3144   14
           1   62  280

> auc
[1] 0.918824
> KS
[1] 0.8142803
> gini
[1] 0.7557978
```

Let us score the Test dataset using the same model

```
> Test1$predict.class <- predict(pcart.model, Test1, type="class")
> Test1$predict.score <- predict(pcart.model, Test1, type="prob")
> pred <- prediction(Test1$predict.score[,2], Test1$PersonalLoan)
> perf <- performance(pred, "tpr", "fpr")
> KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
> auc <- performance(pred,"auc");
```

```
> auc <- as.numeric(auc@y.values)
> gini = ineq(Test1$predict.score[,2], type="Gini")
> with(Test1, table(PersonalLoan, predict.class))
           predict.class
PersonalLoan    0    1
           0 1359    3
           1   22  116

> require(pROC)
> library(caret)
> confusionmatrix.cart <- confusionMatrix(Test1$PersonalLoan, Test1$predict.class)
> confusionmatrix.cart
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1359    3
         1   22  116

               Accuracy : 0.9833
                 95% CI : (0.9755, 0.9892)
    No Information Rate : 0.9207
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8937

 Mcnemar's Test P-Value : 0.0003182

            Sensitivity : 0.9841
            Specificity : 0.9748
         Pos Pred Value : 0.9978
         Neg Pred Value : 0.8406
             Prevalence : 0.9207
         Detection Rate : 0.9060
   Detection Prevalence : 0.9080
      Balanced Accuracy : 0.9794

       'Positive' Class : 0

> auc
[1] 0.9292973
> KS
[1] 0.8383771
> gini
[1] 0.7520567
```

- RANDOM FOREST PREDICTION FOR TRAIN AND TEST DATASET

In order to compare the performance of both CART and random forest model, lets perform predictions on the random forest model for both train and test datasets.

For prediction, as we are going to add additional columns to the dataset, mirroring the test and train datasets

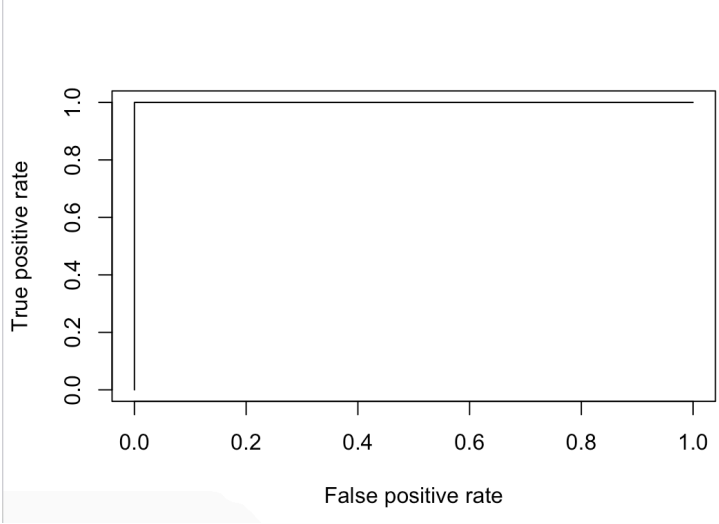
```
> Train2 = Train.TheraBank
> Test2 = Test.TheraBank
```

Scoring the train dataset and deciling the code to generate predictions

```
> Train2$predict.class <- predict(tune.rndForest, Train2, type="class")
> Train2$predict.score <- predict(tune.rndForest, Train2, type="prob")
```

Let's plot the graph between True Positive Rate and False Positive Rate of train dataset

```
> pred <- prediction(Train2$predict.score[,2], Train2$PersonalLoan)
> perf <- performance(pred, "tpr", "fpr")
> plot(perf)
```



The graph shows a high level of accuracy

```
> KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
> auc <- performance(pred,"auc");
> auc <- as.numeric(auc@y.values)
> gini = ineq(Train2$predict.score[,2], type="Gini")
> with(Train2, table(PersonalLoan, predict.class))
            predict.class
PersonalLoan    0       1
           0    3158    0
           1    0       342

> auc
[1] 1
> KS
[1] 1
> gini
[1] 0.9000495
```

The accuracy of random forest model on train dataset is 100%

Let us score the Test dataset using the same random forest model

```
> Test2$predict.class <- predict(tune.rndForest, Test2, type="class")
> Test2$predict.score <- predict(tune.rndForest, Test2, type="prob")

> KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
> auc <- performance(pred,"auc");
> auc <- as.numeric(auc@y.values)
> gini = ineq(Test1$predict.score[,2], type="Gini")
> with(Test2, table(PersonalLoan, predict.class))
             predict.class
PersonalLoan    0        1
           0          1359    3
           1           14      124

> confusionmatrix.rndforest <- confusionMatrix(Test2$PersonalLoan,
Test2$predict.class)
> confusionmatrix.rndforest
Confusion Matrix and Statistics

          Reference
Prediction    0    1
         0 1359    3
         1   14  124

               Accuracy : 0.9887
                 95% CI : (0.9819, 0.9934)
    No Information Rate : 0.9153
    P-Value [Acc > NIR] : < 2e-16

                  Kappa : 0.9296

 Mcnemar's Test P-Value : 0.01529

            Sensitivity : 0.9898
            Specificity : 0.9764
         Pos Pred Value : 0.9978
         Neg Pred Value : 0.8986
             Prevalence : 0.9153
         Detection Rate : 0.9060
   Detection Prevalence : 0.9080
      Balanced Accuracy : 0.9831

       'Positive' Class : 0

> auc
[1] 0.9985874
```

```
> KS
[1] 0.9693545
> gini
[1] 0.7520567
```

	CART		RANDOM FOREST	
	TRAIN	TEST	TRAIN	TEST
ACCURACY	97.83%	98.33%	100%	98.87%
AUC	91.88%	92.92%	100%	99.95%
KS	81.4%	83.83%	100%	96.93%
GINI	75.57%	75.20%	90%	75.20%

Conclusion:

By the Model performance parameters, we can understand that by using the random forest model, we can better predict the probability of a customer buying a Loan from There Bank since the sensitivity of the Random Forest model is 98.98% and accuracy is 100% for train dataset and 98.87% for test dataset.

So, we can safely say, Random Forest Model performs better to classify the customers who have higher probability to buy loan.

10. Appendix – R Code

```
#install package/library
library("readxl")
library("ggplot2")

#Set-up working directory
setwd("~/Documents/Projects/Project 3")

#upload dataset in R
TheraBank <- read_excel("Thera Bank_Personal_Loan_Modelling-dataset.xlsx")

#check dimensions of dataset
dim(TheraBank)

#check for missing values
any(is.na(TheraBank))
colSums(is.na(TheraBank))

#replacing Missing values with zeroes
TheraBank[is.na(TheraBank)] = 0
sum(is.na(TheraBank))

#checking rows with negative value as Experience
```

```
TheraBank[TheraBank$`Experience (in years)`<0,]

#fixing negative values
TheraBank$`Experience (in years)` = abs(TheraBank$`Experience (in years)`)

TheraBank[TheraBank$`Experience (in years)`<0,]

#remove unwanted variables
TheraBank = TheraBank[,-c(1,5)]

#converting columns to factors
TheraBank$Education = factor(TheraBank$Education,levels = c("1", "2", "3"), order =
TRUE)
TheraBank$`Personal Loan` = as.factor(TheraBank$`Personal Loan`)
TheraBank$`Securities Account`=as.factor(TheraBank$`Securities Account`)
TheraBank$`CD Account`=as.factor(TheraBank$`CD Account`)
TheraBank$Online = as.factor((TheraBank$Online))
TheraBank$CreditCard = as.factor((TheraBank$CreditCard))

#summary
summary(TheraBank)

#Ratio of personal loans
prop.table(table(TheraBank$`Personal Loan`))*100

## Univariate analysis
library("DataExplorer")
plot_histogram(TheraBank)

par(mfrow = c(2,3))
boxplot(TheraBank$`Age (in years)`,
        main = toupper("Age of customers"),
        ylab = "Age in years",
        col = "green")

boxplot(TheraBank$`Experience (in years)`,
        main = toupper("Experience of customers"),
        ylab = "Experience in years",
        col = "blue")

boxplot(TheraBank$`Income (in K/month)`,
        main = toupper("Income of customers"),
        ylab = "Monthly Income",
        col = "yellow")

boxplot(TheraBank$CCAvg,
        main = toupper("Average Spending of credit card per month"),
        ylab = "Average Spending",
        col = "red")
```

```
boxplot(TheraBank$Mortgage,
        main = toupper("House Mortgage"),
        ylab = "House Mortgag",
        col = "orange")

#Bivariate analyis
plot_boxplot(TheraBank, by = "Education",
             geom_boxplot_args = list("outlier.color" = "orange"))

plot_boxplot(TheraBank, by = "Personal Loan",
             geom_boxplot_args = list("outlier.color" = "green"))

#convert column names for ease of use
library(dplyr)
TheraBank = TheraBank %>% rename(Age = `Age (in years)`, Experience = `Experience (in
years)`,
                                 Income = `Income (in K/month)`,PersonalLoan = `Personal Loan`,
                                 CDAccount = `CD Account`, SecuritiesAccount = `Securities Account`,
                                 Familymembers = `Family members`)

ggplot(TheraBank, aes(Income,y = CCAvg, color = PersonalLoan)) + geom_point(size = 1)


################################
#CLUSTERING

bank = TheraBank %>% select_if(is.numeric)
bank.scaled = scale(bank,center=TRUE)

#print new matrix by computing the distance using hierarchial clustering
dist.bank.scaled = dist(bank.scaled,method = "euclidean")
print(dist.bank.scaled,digits=3)

#k-means clustering
seed=1000
set.seed(seed) #since kmeans uses a randomized starting point for cluster centroids

cluster1 = kmeans(x=bank.scaled, centers = 2, nstart = 5)
print(cluster1)

library(cluster)
clusplot(bank.scaled, cluster1$cluster,
         color=TRUE, shade=TRUE, labels=2, lines=1)

totWss=rep(0,5)
for(k in 1:5){
  set.seed(seed)
  clust=kmeans(x=bank.scaled, centers=k, nstart=5)
  totWss[k]=clust$tot.withinss
}
```

```
plot(c(1:5), totWss, type="b", xlab="Number of Clusters",
     ylab="sum of 'Within groups sum of squares'")


set.seed(seed) #since kmeans uses a randomized starting point for cluster centroids

cluster2 = kmeans(x=bank.scaled, centers = 3, nstart = 10)
print(cluster2)
clusplot(bank.scaled, cluster2$cluster, color=TRUE, shade=TRUE, labels=2, lines=1)

bank$Clusters = cluster2$cluster
View(bank)


install.packages("factoextra")
library("factoextra")
fviz_cluster(cluster2, bank.scaled, geom = "point", ellipse = TRUE, pointsize = 0.2, ) +
theme_minimal()

custProfile = aggregate(bank,list(bank$Clusters),FUN="mean")
print(custProfile)


bank1 = TheraBank

bank1$Cluster = bank$Clusters
View(bank1)

counts <- table( bank1$PersonalLoan,bank1$Cluster)
barplot(counts, main="Customers availing Personal Loan",
        xlab="Personal Loan No vs Yes", col=c("green","blue"),
        legend = c("Personal_Loan_No","Personal_Loan_Yes"), beside=TRUE)


############################################################
#Splitting dataset into Train and Test
############################################################

set.seed(1000)
index = sample((1:nrow(TheraBank)),round(nrow(TheraBank)*0.7,0), replace = F)

Train.TheraBank = TheraBank[index,]
Test.TheraBank = TheraBank[-index,]

dim(Train.TheraBank)
dim(Test.TheraBank)

table(Train.TheraBank$PersonalLoan)
table(Test.TheraBank$PersonalLoan)
```

```
########################################## CART
#####################################
library(rpart)
library(rpart.plot)
set.seed(1000)

#We beging by building a very complex classification tree, by setting the "cost complexity"
threshold to "0"
#and the minimum bucket size to be 3. Then we plot the tree using rpart.
cart.model <- rpart(formula = PersonalLoan ~ .,
                    data = Train.TheraBank, method = "class",cp=0, minbucket=3)
cart.model
rpart.plot(cart.model)

#The cost complexity table can be obtained using the printcp or plotcp functions
printcp(cart.model)
plotcp(cart.model)
cart.model$variable.importance

########################PRUNE CART TREE ########################
#The unncessarily complex tree above can be pruned using a cost complexity threshold.
#Using a complexity threshold of 0.045 gives us a much simpler tree.
pcart.model = prune(cart.model, cp= 0.045 ,"CP")
printcp(pcart.model)
rpart.plot(pcart.model)

##############################CART prediction###################################
## Let's use rattle to see various model evaluation measures
##rattle()

View(Train.TheraBank)
Train1 = Train.TheraBank
Test1 = Test.TheraBank
## Scoring Train dataset

Train1$predict.class <- predict(pcart.model, Train1, type="class")
Train1$predict.score <- predict(pcart.model, Train1, type="prob")



## deciling code
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
           ifelse(x<deciles[2], 2,
```

```
                ifelse(x<deciles[3], 3,
                    ifelse(x<deciles[4], 4,
                        ifelse(x<deciles[5], 5,
                            ifelse(x<deciles[6], 6,
                                ifelse(x<deciles[7], 7,
                                    ifelse(x<deciles[8], 8,
                                        ifelse(x<deciles[9], 9, 10
                                        ))))))))))
}

class(Train1$predict.class)
## deciling
Train1$deciles <- decile(Train1$predict.score[,2])
View(Train1)

##install.packages("ROCR")
##install.packages("ineq")
library(ROCR)
library(ineq)
pred <- prediction(Train1$predict.score[,2], Train1$PersonalLoan)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(Train1$predict.score[,2], type="Gini")

with(Train1, table(PersonalLoan, predict.class))
auc
KS
gini

confusionmatrix.cart.train <- confusionMatrix(Train1$PersonalLoan, Train1$predict.class)
confusionmatrix.cart.train

## Syntax to get the node path
tree.path <- path.rpart(pcart.model, node = c(2, 12))
nrow(Test1)

## Scoring Test sample
Test1$predict.class <- predict(pcart.model, Test1, type="class")
Test1$predict.score <- predict(pcart.model, Test1, type="prob")


Test1$deciles <- decile(Test1$predict.score[,2])
View(Test1)
```

```
pred <- prediction(Test1$predict.score[,2], Test1$PersonalLoan)
perf <- performance(pred, "tpr", "fpr")
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(Test1$predict.score[,2], type="Gini")

with(Test1, table(PersonalLoan, predict.class))
require(pROC)
library(caret)
confusionmatrix.cart <- confusionMatrix(Test1$PersonalLoan, Test1$predict.class)
confusionmatrix.cart
auc
KS
gini



############################## RANDOM
FOREST#####################################



library(randomForest)
seed=1000
set.seed(seed)
rndForest.model = randomForest(PersonalLoan ~ ., data = Train.TheraBank)
print(rndForest.model)


##OOB error rate estimation 1.26%

#print error rate
rndForest.model$err.rate
plot(rndForest.model, main="")
legend("topright", c("OOB", "0", "1"), text.col=1:6, lty=1:3, col=1:3)
title(main="Error Rates Random Forest Train.TheraBank")

library("party")
x <- ctree(PersonalLoan ~ ., data = Train.TheraBank)
plot(x, type="simple")

####
#########################Tuning Random forest ############################
seed=1000
set.seed(seed)
tune.rndForest = tuneRF(x=subset(Train.TheraBank,select = -PersonalLoan),
                        y=Train.TheraBank$PersonalLoan,
                        ntreeTry=501, doBest=T)
```

```
y <- ctree(PersonalLoan ~ ., data = Train.TheraBank)
plot(y, type="simple")
importance(tune.rndForest)
print(tune.rndForest)


Train2 = Train.TheraBank
Test2 = Test.TheraBank
## Scoring Train dataset
?predict
Train2$predict.class <- predict(tune.rndForest, Train2, type="class")
Train2$predict.score <- predict(tune.rndForest, Train2, type="prob")



## deciling code
decile <- function(x){
  deciles <- vector(length=10)
  for (i in seq(0.1,1,.1)){
    deciles[i*10] <- quantile(x, i, na.rm=T)
  }
  return (
    ifelse(x<deciles[1], 1,
           ifelse(x<deciles[2], 2,
                  ifelse(x<deciles[3], 3,
                         ifelse(x<deciles[4], 4,
                                ifelse(x<deciles[5], 5,
                                       ifelse(x<deciles[6], 6,
                                              ifelse(x<deciles[7], 7,
                                                     ifelse(x<deciles[8], 8,
                                                            ifelse(x<deciles[9], 9, 10
                                                            ))))))))))
}

class(Train2$predict.class)
## deciling
Train2$deciles <- decile(Train2$predict.score[,2])
View(Train1)


pred <- prediction(Train2$predict.score[,2], Train2$PersonalLoan)
perf <- performance(pred, "tpr", "fpr")
plot(perf)
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(Train2$predict.score[,2], type="Gini")
```

```
with(Train2, table(PersonalLoan, predict.class))
auc
KS
gini
require(pROC)
library(caret)
confusionmatrix.rfor.train <- confusionMatrix(Train2$PersonalLoan, Train2$predict.class)
confusionmatrix.rfor.train


## Scoring Test sample
Test2$predict.class <- predict(tune.rndForest, Test2, type="class")
Test2$predict.score <- predict(tune.rndForest, Test2, type="prob")


Test2$deciles <- decile(Test2$predict.score[,2])
View(Test2)



pred <- prediction(Test2$predict.score[,2], Test2$PersonalLoan)
perf <- performance(pred, "tpr", "fpr")
KS <- max(attr(perf, 'y.values')[[1]]-attr(perf, 'x.values')[[1]])
auc <- performance(pred,"auc");
auc <- as.numeric(auc@y.values)

gini = ineq(Test1$predict.score[,2], type="Gini")

with(Test2, table(PersonalLoan, predict.class))

confusionmatrix.rndforest <- confusionMatrix(Test2$PersonalLoan, Test2$predict.class)
confusionmatrix.rndforest
auc
KS
gini
```