Thera Bank Case Analysis

– *Vompolu Sai Tanuj*

PGP-BABI(JUNE 2019) G1

Table of contents:

1) Project Objective:

The project is based on **Thera Bank** case study which has a growing database. Bank has **huge base of depositors (Liabilities)** but very **low number of burrowers (Assets**). So the bank has decided to expand it's base of burrowers from the existing base of depositors. So to help the campaign of the Marketing team to achieve the above objective, As an Analyst, must create a Classification model based on the **data of the customers** provided to us to build a **classification model** to help the Marketing team **target the right customers** and in due process **reducing the costs incurred** for campaigning.

2) Exploration of the dataset:

The dataset provided to us is the result of the campaigning of Market Team to convert the Depositors to Burrowers which was done previously. The data is stored in the file **"bank_data.xlsx".** The data contains the **information of customers** which will help us in creating a **classification model.**

a) Invoking of necessary packages to be used in the R Code:

The required packages must be installed and their respective libraries must be called out in the R code before calling out any function from that library. The libraries if not found in the R directory can be installed by using **install.packages ("package name").** The internet must be connected so that required package gets downloaded and installed. After installation, the library must be called into the R code using the function, **library(packagename).** For this project, the following libraries must be invoked to be used in the R code.

- **readr** – This library is used to **import the dataset** from the **CSV file to the R Studio.**

```
install.packages("readxl")
library(readxl)
```

- **ggplot2** – This library is used to **create many types of graphs** not found in the in-built library and it has lots of customizations.

```
install.packages("ggplot2")
library(ggplot2)
```

- **hmisc** – This library is useful to **create histograms** for **Uni-Variate Analysis**

```
install.packages("Hmisc")
library(Hmisc)
```

- **rpart –** This library is used to create CART decision trees.

```
install.packages("rpart")
library(rpart)
```

- **rpart.plot –** This library is used to plot CART decision trees.

```
install.packages("rpart.plot")
library(rpart.plot)
```

- **caTools** – This library is used to split training and testing data

```
install.packages("caTools")
library(caTools)
```

- **ROCR** – This library is used to get performance measure object

```
install.packages("ROCR")
library(ROCR)
```

- **InformationValue** – This library is useful for getting Concordance values

```
install.packages("InformationValue")
library(InformationValue)
```

- **cluster –** This library is useful for K means clustering

```
install.packages("cluster")
library(cluster)
```

- **NbClust –** This library is used for finding out the right number of clusters for Kmeans.

```
install.packages("NbClust")
library(NbClust)
```

- **ineq –** This library is used for finding the KS values

```
install.packages("ineq")
library(ineq)
```

- **randomForest –** This library is used to build Random Forest

```
install.packages("randomForest")
library(randomForest)
```

b) Setting the working directory and importing the dataset:

The working directory in the **R Console** must be set to the directory where the XLSX file exists. The working directory can be changed using the function **setwd().** To get the current directory of the R Console, the function **getwd()** can be used.

```
setwd("C:/R programs great lakes/DATA MINING/prjct")
getwd()
```

After setting the working directory, we must now import the dataset from the **xlsx file** to the R console using the function **read_xlsx().** The dataset assigned to this project is assigned to a data frame with the name **bank_data** and the same will be used to call the dataset further when required in the **R Console.**

```
bank_data = read_xlsx("Bank_data.xlsx")
bank_data
```

c) Identification of different variables:

To get an idea on all the variables in the dataset, we must first be able to identify all the variables in the dataset and get an idea about all of them before performing Uni-Variate and Bi-Variate Analysis.

The following functions can be used for identification of the variables:

- **summary() –** This function is used to get a **Simple Statistical summary** on each of the variables.

```
> summary(bank_data)
       ID        Age (in years)  Experience (in years) Income (in K/month)    ZIP Code     Family members
 Min.   :   1   Min.   :23.00   Min.   :-3.0          Min.   :  8.00      Min.   : 9307   Min.   :1.000
 1st Qu.:1251   1st Qu.:35.00   1st Qu.:10.0          1st Qu.: 39.00      1st Qu.:91911   1st Qu.:1.000
 Median :2500   Median :45.00   Median :20.0          Median : 64.00      Median :93437   Median :2.000
 Mean   :2500   Mean   :45.34   Mean   :20.1          Mean   : 73.77      Mean   :93153   Mean   :2.397
 3rd Qu.:3750   3rd Qu.:55.00   3rd Qu.:30.0          3rd Qu.: 98.00      3rd Qu.:94608   3rd Qu.:3.000
 Max.   :5000   Max.   :67.00   Max.   :43.0          Max.   :224.00      Max.   :96651   Max.   :4.000
                                                                                           NA's   :18
     CCAvg          Education        Mortgage     Personal Loan   Securities Account   CD Account         Online
 Min.   : 0.000   Min.   :1.000   Min.   :  0.0   Min.   :0.000   Min.   :0.0000     Min.   :0.0000   Min.   :0.0000
 1st Qu.: 0.700   1st Qu.:1.000   1st Qu.:  0.0   1st Qu.:0.000   1st Qu.:0.0000     1st Qu.:0.0000   1st Qu.:0.0000
 Median : 1.500   Median :2.000   Median :  0.0   Median :0.000   Median :0.0000     Median :0.0000   Median :1.0000
 Mean   : 1.938   Mean   :1.881   Mean   : 56.5   Mean   :0.096   Mean   :0.1044     Mean   :0.0604   Mean   :0.5968
 3rd Qu.: 2.500   3rd Qu.:3.000   3rd Qu.:101.0   3rd Qu.:0.000   3rd Qu.:0.0000     3rd Qu.:0.0000   3rd Qu.:1.0000
 Max.   :10.000   Max.   :3.000   Max.   :635.0   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() –** This function is used to **give out the dimensions** of the Dataset

```
> dim(bank_data)
[1] 5000   14
```

- **colnames() –** This function is used to **give out all the column names** in the Dataset.

```
> colnames(bank_data)
 [1] "ID"                    "Age (in years)"        "Experience (in years)" "Income (in K/month)"
 [5] "ZIP Code"              "Family members"        "CCAvg"                 "Education"
 [9] "Mortgage"              "Personal Loan"         "Securities Account"    "CD Account"
[13] "Online"                "CreditCard"
```

- **str() –** This function is used to give the **class of all the variables** present in the Dataset.

```
> str(bank_data)
Classes 'tbl_df', 'tbl' and 'data.frame':       5000 obs. of  14 variables:
 $ ID                    : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Age (in years)        : num  25 45 39 35 35 37 53 50 35 34 ...
 $ Experience (in years): num  1 19 15 9 8 13 27 24 10 9 ...
 $ Income (in K/month)   : num  49 34 11 100 45 29 72 22 81 180 ...
 $ ZIP Code              : num  91107 90089 94720 94112 91330 ...
 $ Family members        : num  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             : num  1 1 1 2 2 2 2 3 2 3 ...
 $ Mortgage              : num  0 0 0 0 0 155 0 0 104 0 ...
 $ Personal Loan         : num  0 0 0 0 0 0 0 0 0 1 ...
 $ Securities Account    : num  1 1 0 0 0 0 0 0 0 0 ...
 $ CD Account            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Online                : num  0 0 0 0 0 1 1 0 1 0 ...
 $ CreditCard            : num  0 0 0 0 1 0 0 1 0 0 ...
```

- **head() –** This function is used to give out **the top values** in the dataset. The default is **6.**

```
> head(bank_data)
# A tibble: 6 x 14
     ID `Age (in years)` `Experience (in~ `Income (in K/m~ `ZIP Code` `Family members` CCAvg Education Mortgage
  <dbl>            <dbl>            <dbl>            <dbl>      <dbl>            <dbl> <dbl>     <dbl>    <dbl>
1     1               25                1               49      91107                4   1.6         1        0
2     2               45               19               34      90089                3   1.5         1        0
3     3               39               15               11      94720                1   1           1        0
4     4               35                9              100      94112                1   2.7         2        0
5     5               35                8               45      91330                4   1           2        0
6     6               37               13               29      92121                4   0.4         2      155
# ... with 5 more variables: `Personal Loan` <dbl>, `Securities Account` <dbl>, `CD Account` <dbl>, Online <dbl
>,
#   CreditCard <dbl>
```

- **tail() –** This function is used to give out **the bottom values** in the dataset. The default is **6.**

```
> tail(bank_data)
# A tibble: 6 x 14
     ID `Age (in years)` `Experience (in~ `Income (in K/m~ `ZIP Code` `Family members` CCAvg Education Mortgage
  <dbl>            <dbl>            <dbl>            <dbl>      <dbl>            <dbl> <dbl>     <dbl>    <dbl>
1  4995               64               40               75      94588                3   2           3        0
2  4996               29                3               40      92697                1   1.9         3        0
3  4997               30                4               15      92037                4   0.4         1       85
4  4998               63               39               24      93023                2   0.3         3        0
5  4999               65               40               49      90034                3   0.5         2        0
6  5000               28                4               83      92612                3   0.8         1        0
# ... with 5 more variables: `Personal Loan` <dbl>, `Securities Account` <dbl>, `CD Account` <dbl>, Online <dbl
```

Inferences:

1) The dataset **bank_data** has **14 Columns** and **5000 Rows.** This means that the data consists of 14 Variables describing a total of 5000 customers.

2) The order of the data is set in the order of the number of the **ID variable** without taking into consideration the order of the other variables.

3) Out of the 14 **variables,** there are 6 variables describing the Personals of a customer which consists of details like **Age, Income, Family members, etc.**

4) There are **8 variables** like CCAvg, Usage of Credit Card, Presence of Securities Account, etc. which are bank-related information of the customers.

5) We can see that all the variables are in **Numeric** format even though only few can be considered as **Numeric variable.**

6) We can see that there are discrepancies in the data in the **Zip code variable and the Experience variable.**

7) The column names of the variables are **long and complex** which would make it difficult for us to call the variable names while **coding.**

8) We can see that most of the customers are of **45 years** with income of **64k per month** and working experience of around **20 years.**

9) From the data description given below, we can understand that there are **7 Categorical Variables of importance and 5 Numerical variables that are of importance.**

ID	Customer ID
Age	Customer's age in years
Experience	Years of professional experience
Income	Annual income of the customer ($000)
ZIPCode	Home Address ZIP code.
Family	Family size of the customer
CCAvg	Avg. spending on credit cards per month ($000)
Education	Education Level. 1: Undergrad; 2: Graduate; 3: Advanced/Professional
Mortgage	Value of house mortgage if any. ($000)
Personal Loan	Did this customer accept the personal loan offered in the last campaign?

Securities Account	Does the customer have a securities account with the bank?
CD Account	Does the customer have a certificate of deposit (CD) account with the bank?
Online	Does the customer use internet banking facilities?
CreditCard	Does the customer use a credit card issued by the bank?

10) We can conclude that one of the **Categorical variable**, **Personal Loan,** is the **Response Variable** while the rest of them are **Independent** variables.

d) Dataset transformation:

As seen in the initial analysis of the variables, the column names were **long** and **contained quotes** and also there were discrepancies in two of the variables, namely Family Member and Experience.

I. *Changing the Column names in the dataset:*

The column names in the dataset can be selected using the function **colnames()**. After selecting the column names, we must convert them into factor names which will make it **simple and easy** to call in the R Code further down. This can be done using the function **make.names()**.

```
> colnames(bank_data) = make.names(colnames(bank_data))
> colnames(bank_data)
 [1] "ID"                "Age..in.years."        "Experience..in.years." "Income..in.K.month."   "ZIP.Code"
 [6] "Family.members"    "CCAvg"                 "Education"             "Mortgage"              "Personal.Loan"
[11] "Securities.Account" "CD.Account"            "Online"                "CreditCard"
```

II. *Removing the negative values from the Experience variable:*

The variable **Experience** has many negative values which is not possible in real life. We can find the number of **negative values** using the **sum()** function on **conditional slicing.**

```
> sum(bank_data$Experience..in.years. < 0)
[1] 52
```

We can see that there are **52 values** which are negative. Rather than converting them to positive, it would be better to change them to **Zero** as it would least affect the analysis of the dataset.

```
> sum(bank_data$Experience..in.years. < 0)
[1] 52
> bank_data[bank_data < 0] = 0
> sum(bank_data$Experience..in.years. < 0)
[1] 0
```

III. *Fixing the values in the Zip code variable:*

The variable Zip Code has values which contain only **four digits** instead of **five.** The values which have four digits can be corrected by adding **5** on the leading side since it is the most probable that from which ever number we choose from **0 to 9,** we can be sure that it will be the shortest distance **probable** to that number and hence the shortest distance to the location also.

```
> ## Fixing the Pincodes ####
> pincode = bank_data
> sum(pincode$ZIP.Code < 10000)
[1] 1
> which(pincode$ZIP.Code < 10000)
[1] 385
> which(colnames(pincode) == "ZIP.Code")
[1] 5
> pincode[385,5] = pincode[385,5] * 10
> pincode[385,5]
# A tibble: 1 x 1
  ZIP.Code
     <dbl>
1    93070
> pincode[385,5] = pincode[385,5] + 5
> pincode[385,5]
# A tibble: 1 x 1
  ZIP.Code
     <dbl>
1    93075
> sum(pincode$ZIP.Code < 10000)
[1] 0
> |
```

IV.*Converting the Categorical variables:*

As seen in the analysis, we saw that the **Categorical variables** were still shown as **Numeric** variables. Hence they must be converted to **Factor variable.** This can be done by using the function **as.factor()**.

```
### Converting the requiered variables into Categorical variable ###
bank_data$Personal.Loan = factor(bank_data$Personal.Loan,levels = c(1,0),labels = c("Yes","No"))
bank_data$Education = factor(bank_data$Education,levels = c(1,2,3),labels = c("Undergrad","Postgrad","Proffesional"))
bank_data$Securities.Account = factor(bank_data$Securities.Account,levels = c(1,0),labels = c("Yes","No"))
bank_data$CD.Account = factor(bank_data$CD.Account,levels = c(1,0),labels = c("Yes","No"))
bank_data$Online = factor(bank_data$Online,levels = c(1,0),labels = c("Yes","No"))
bank_data$CreditCard = factor(bank_data$CreditCard,levels = c(1,0),labels = c("Yes","No"))
bank_data$Family.members = as.factor(bank_data$Family.members)
```

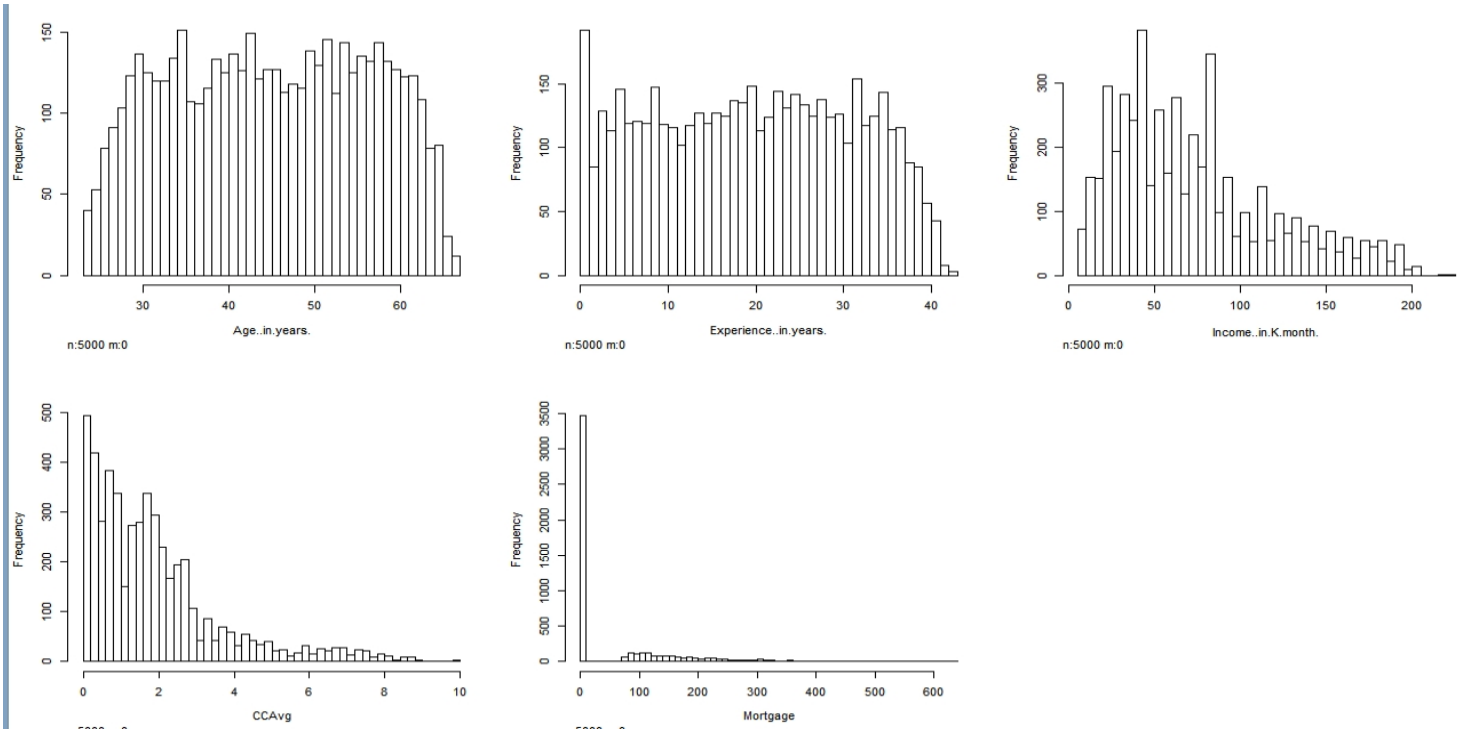
e.) Uni – Variate Analysis:

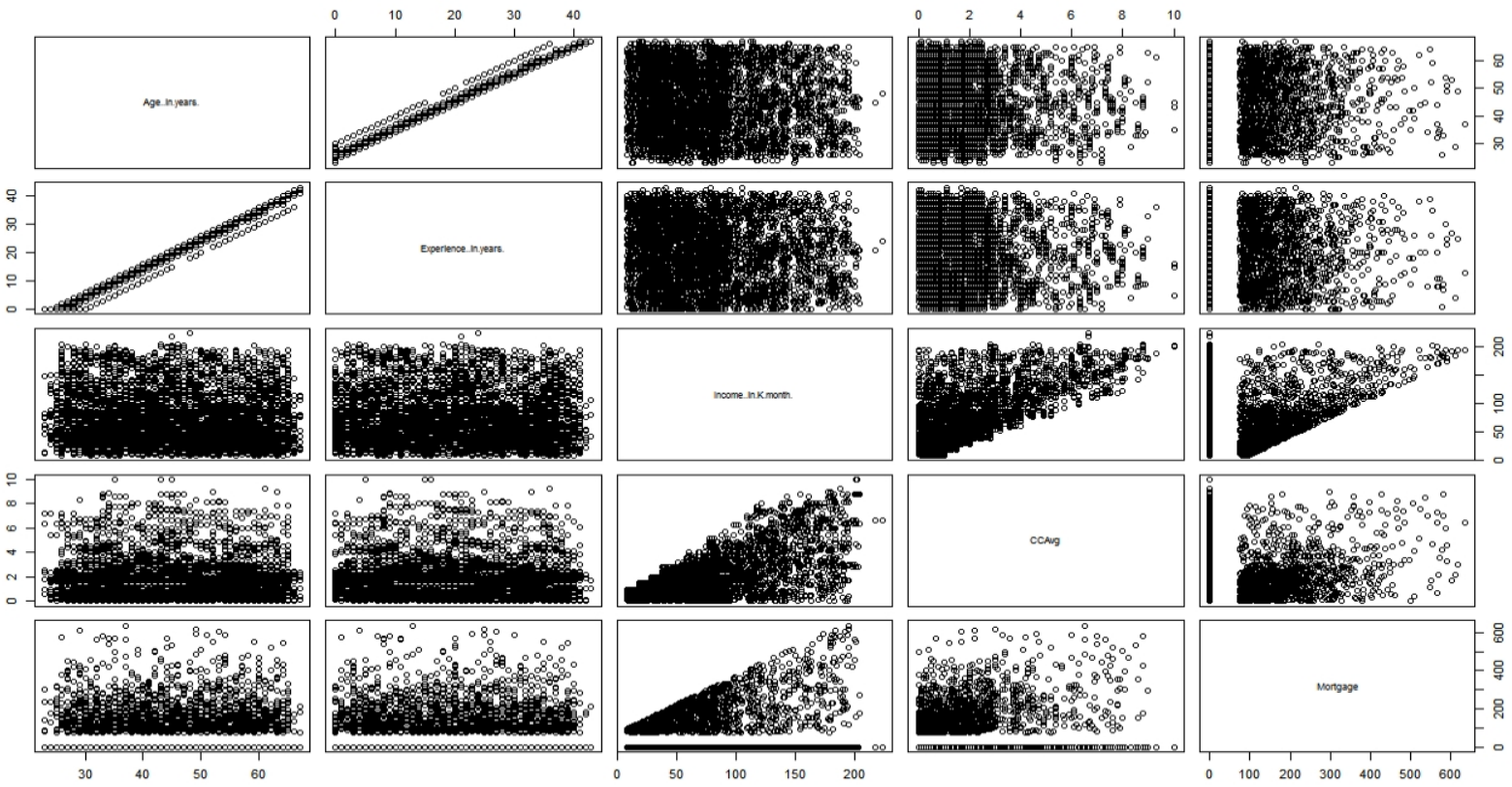
We can exclude the **ID** and **Zip Code** variables from the analysis as they do not contribute to the analysis. That leaves us with **7 Categorical Variables of importance and 5 Numerical variables that are of importance** with **Personal Loan** as **Response variable.**

I. Analysis of Numerical variables:

The **numerical variables** be analysed by plotting **Histograms.** To plot histograms for all the variables, we can use the function **hist.data.frame()** from the **hmisc()** package on the **spliced data frame** containing only the numerical variables. A **Scatter plot** between all the numerical variables will also help us analyse how the **Values** are scattered in the data frame. That can be done using the **plot()** function on sliced data frame containing only **Numerical Variables.**

```
# Plotting all the numerical variables #
hist.data.frame(bank_data[,c(2,3,4,7,9)])
plot(bank_data[,c(2,3,4,7,9)])
```





Inferences:

1) **Age** is the only variable to be **fairly** following a **normal distribution**.
2) **Experience** and **Mortgage** are the variables which don't seem to follow a **normal distribution.**
3) **CCAvg** and **Income** showcase a **left-skewed bell curve.**
4) The customers with **income on the lower side** contribute the highest to the strength of the **Customer base.**
5) The customers with **low CCAvg** contribute the highest to the strength of the **Customer base.**
6) The **Age and Professional experience** are **highly correlated** which makes sense because more the **age of the customer,** more likely he has gained more **professional experience.**
7) Rest of the variables have large **spread** and do not show **correlation.**

8) Looking at that proximity of the values in some of the **scatter plots,** we can roughly say that **3 to 4 clusters** are possible in the dataset.

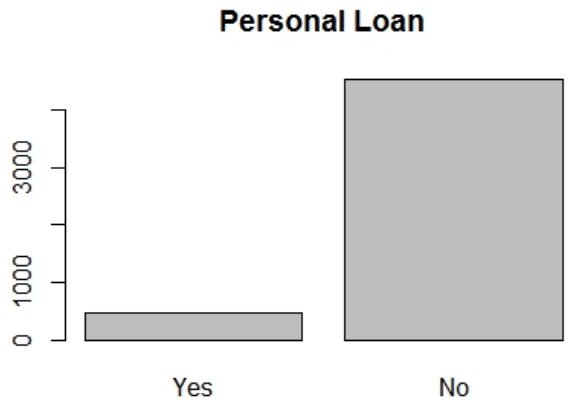
II. Analysis of Categorical variables:

The categorical variables can be analysed by creating a **frequency and Proportion tables** and also by plotting a **Barplot** using the function **Barplot().**

```
## Working with the response variable #####
print(table(bank_data$Personal.Loan))
print(prop.table(table(bank_data$Personal.Loan))*100)
plot(bank_data$Personal.Loan,main = "Personal Loan")
## Working with other categorical variables ##
print(table(bank_data$Family.members))
print(prop.table(table(bank_data$Family.members))*100)
plot(bank_data$Family.members)
# Education #
print(table(bank_data$Education))
print(prop.table(table(bank_data$Education))*100)
plot(bank_data$Education,main = "Education")
# Securities Account #
print(table(bank_data$Securities.Account))
print(prop.table(table(bank_data$Securities.Account))*100)
plot(bank_data$Securities.Account,main = "Securities Account")
# Cash Deposit #
print(table(bank_data$CD.Account))
print(prop.table(table(bank_data$CD.Account))*100)
plot(bank_data$CD.Account,main = "Certificate of Deposit")
# Internet Banking #
print(table(bank_data$Online))
print(prop.table(table(bank_data$Online))*100)
plot(bank_data$Online,main = "Internet Banking")
# Credit Card issued by the bank #
print(table(bank_data$CreditCard))
print(prop.table(table(bank_data$CreditCard))*100)
plot(bank_data$CD.Account,main = "Credit issued by the bank")
```

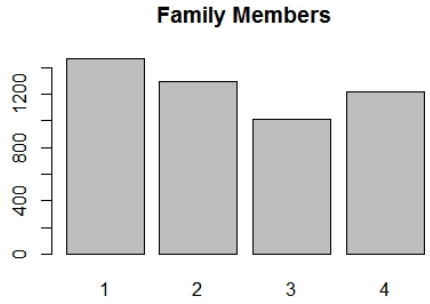
Personal Loan:

```
> print(table(bank_data$Personal.Loan))

 Yes   No
 480 4520
> print(prop.table(table(bank_data$Personal.Loan))*100)

 Yes   No
 9.6 90.4
> plot(bank_data$Personal.Loan,main = "Personal Loan")
>
```



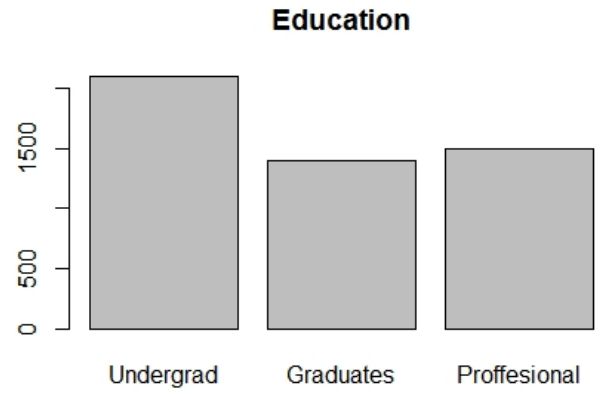
Family Members:

```
> print(table(bank_data$Family.members))

   1    2    3    4
1464 1292 1009 1217
> print(prop.table(table(bank_data$Family.members))*100)

       1        2        3        4
29.38579 25.93336 20.25291 24.42794
```



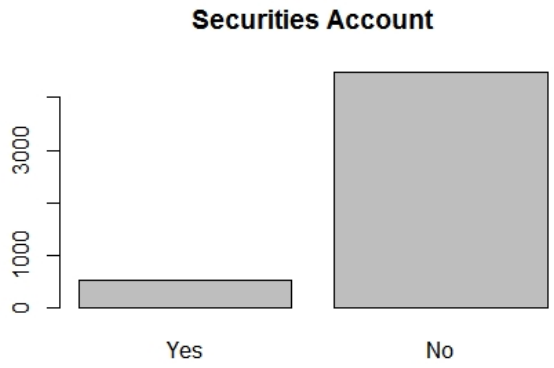
Education:

```
> print(table(bank_data$Education))

   Undergrad    Graduates Proffesional
        2096         1403         1501
> print(prop.table(table(bank_data$Education))*100)

   Undergrad    Graduates Proffesional
       41.92        28.06        30.02
```



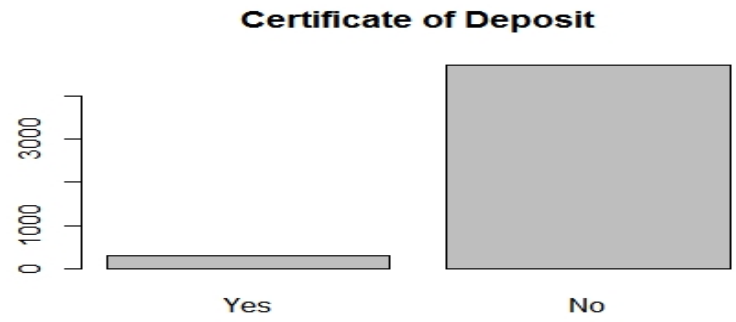
Securities Account:

```
> print(table(bank_data$Securities.Account))

 Yes   No
 522 4478
> print(prop.table(table(bank_data$Securities.Account))*100)

  Yes    No
10.44 89.56
> plot(bank_data$Securities.Account,main = "Securities Account")
```



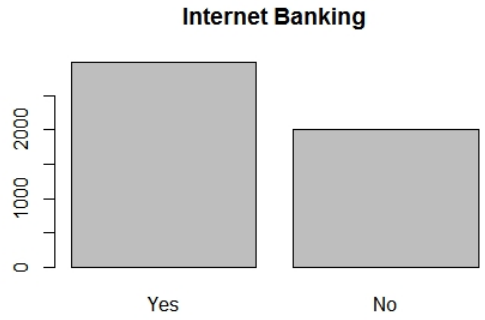
Certificate of Deposit:

```
plot(bank_data$Securities.Account,main = "Securities Account")
# Cash Deposit #
print(table(bank_data$CD.Account))
print(prop.table(table(bank_data$CD.Account))*100)
plot(bank_data$CD.Account,main = "Certificate of Deposit")
```



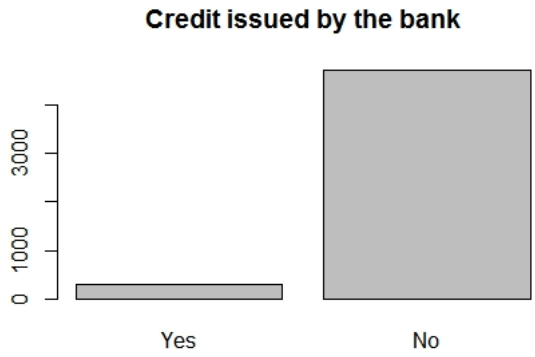
Internet Banking:

```
> print(table(bank_data$Online))

 Yes   No
2984 2016
> print(prop.table(table(bank_data$Online))*100)

  Yes    No
59.68 40.32
> plot(bank_data$Online,main = "Internet Banking")
```



Credit Card:

```
> print(table(bank_data$CreditCard))

 Yes   No
1470 3530
> print(prop.table(table(bank_data$CreditCard))*100)

 Yes   No
29.4 70.6
> plot(bank_data$CD.Account,main = "Credit issued by the bank")
```



Inferences:

1) We can see that in the previous campaign, only 480(9.6%) of the customers have responded Yes to the **Personal Loan.**

2) We can see that **29.2%** of the customers have only **one family member** while **0.36%** of the customers **don't have a family member.**

3) We can see that **2096** Customers are undergraduates while only **1403** customers are **Graduates** showcasing the knowledge most of the customers possess.

4) Since we have **majority** of **Undergraduates,** we can see that only **10.44%** a **securities account** indicating that they don't know the benefits got by them and only afraid of taking **risk** given their low knowledge compared to **professional** education.

5) Most of the customers don't hold a **Certificate of Deposit** indicating that customers would go for **liquidity of the cash in their balance** over **long-time investments**

6) We can see that only **59.68%** of the customers have **Online accounts** which would make the customers easily access their **Loan** details when they respond **Yes.**

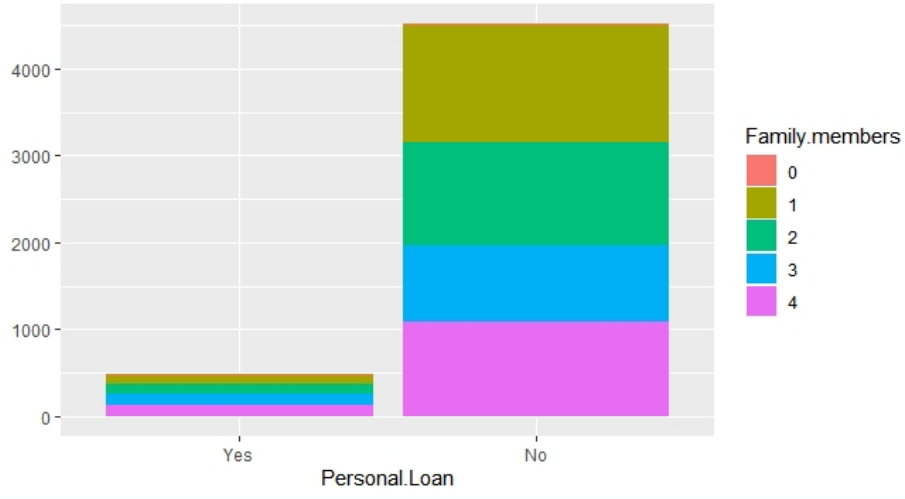
7) We can see that **70.6%** of the customers don't possess a **Credit Card** only contributing to **low assets** of the bank.

f.) Bi – Variate Analysis:

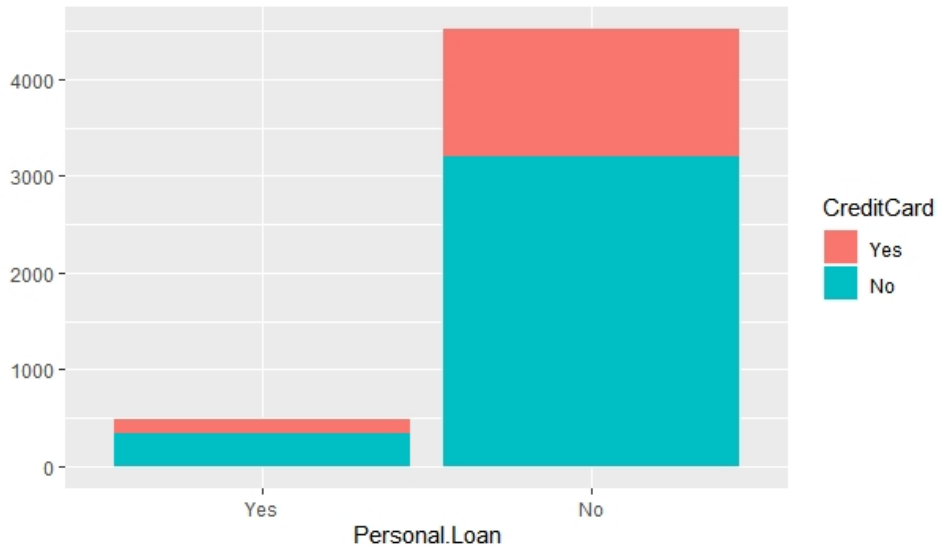
The Bi- Variate Analysis can be done by analysing both **Categorical and Numerical variable** with each other. For plotting these **Bi-variate graphs,** we can use the function **qplot()** function from the **ggplot2** library.

I. Plotting the response variable with other Categorical variables:

```
> ### Response variable with Categorical variable ###
> table(bank_data$Personal.Loan,bank_data$Family.members)

        0    1    2    3    4
Yes     2  106  106  133  133
No     16 1358 1186  876 1084
```

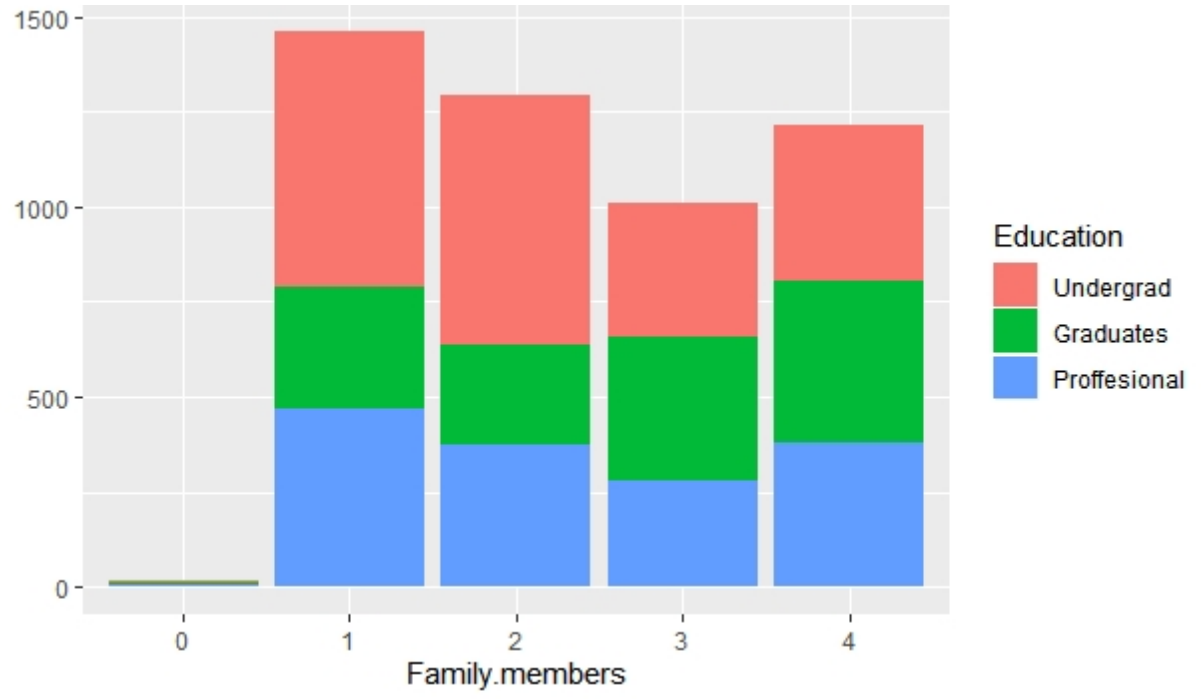


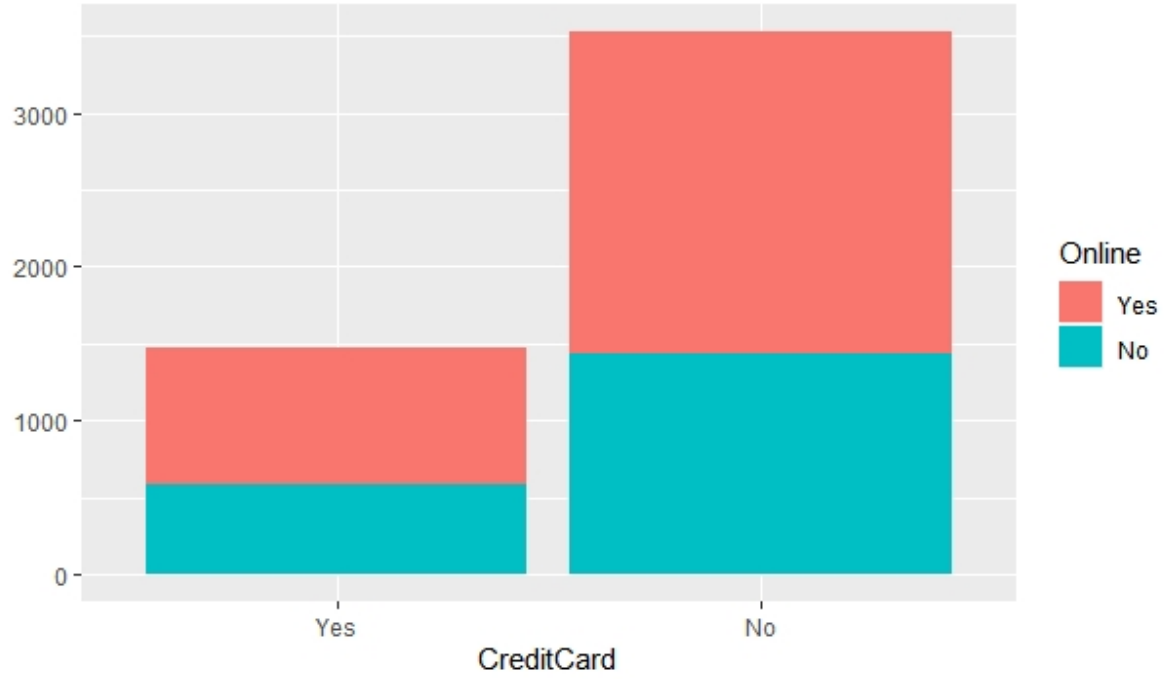

```
> table(bank_data$Personal.Loan,bank_data$CreditCard)

      Yes   No
Yes   143  337
No   1327 3193
> qplot(Personal.Loan,fill = CreditCard,data = bank_data)
```



II. Plotting the Categorical variables with other Categorical variables:

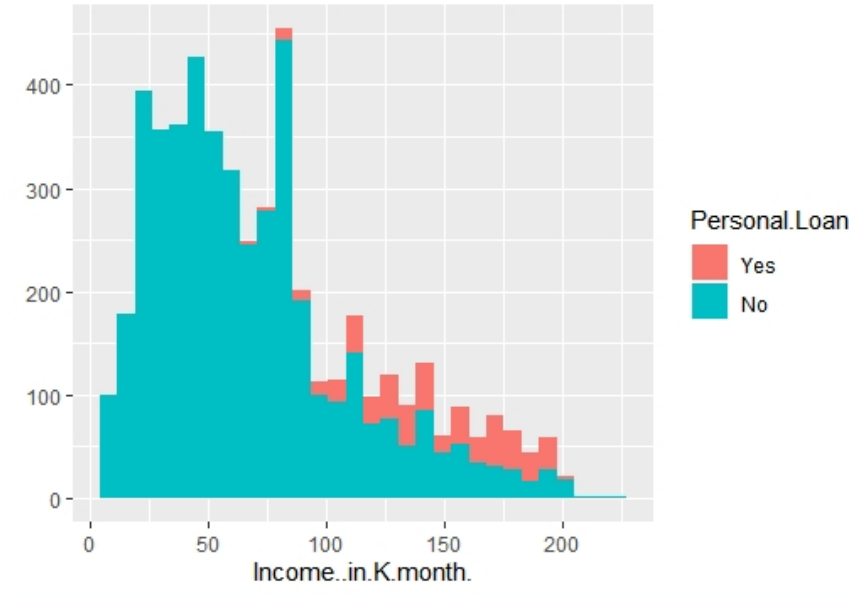
```
> qplot(Family.members,fill = Education,data = bank_data)
> qplot(CreditCard,fill = Online,data = bank_data)
```

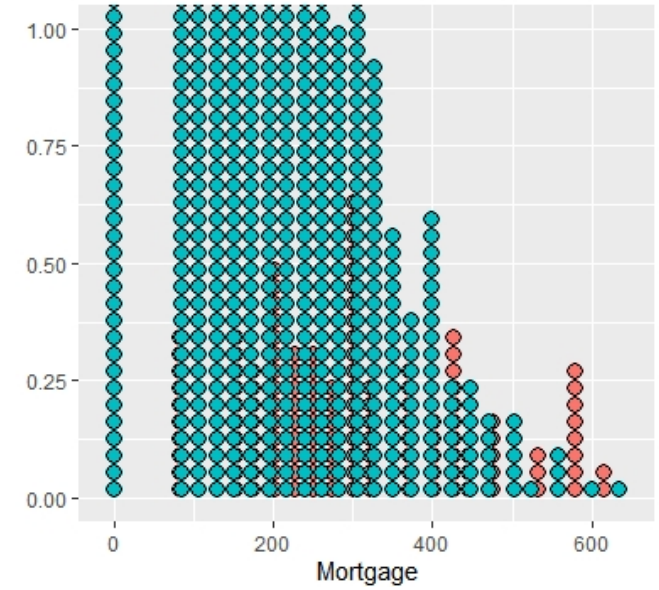


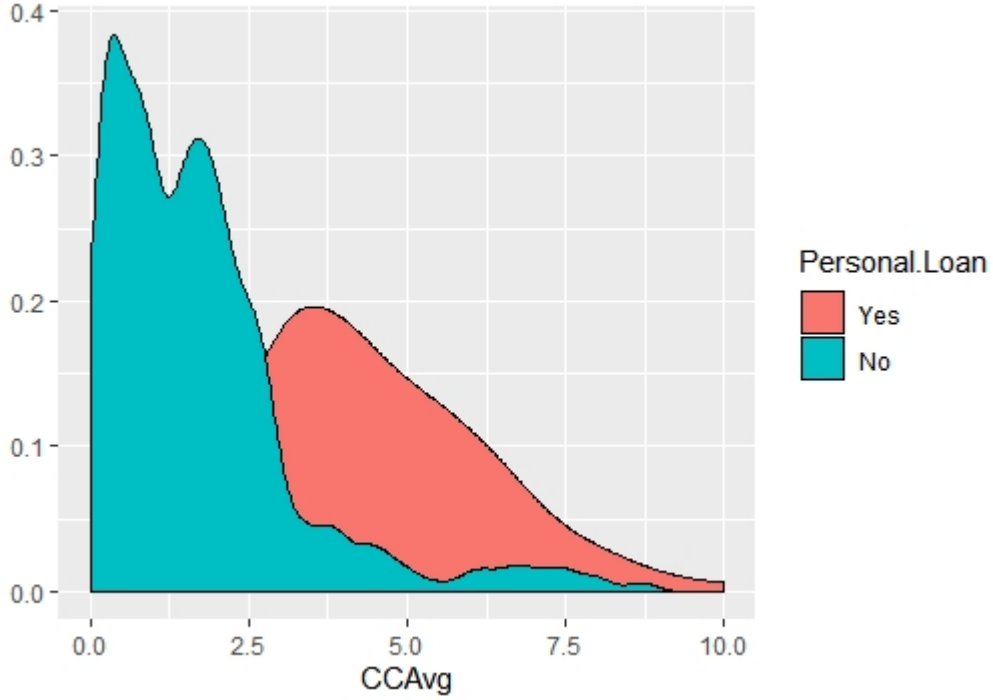


III. *Plotting the Response variables with other numerical variables:*

```
qplot(Income..in.K.month.,fill = Personal.Loan,data = bank_data)
qplot(Mortgage,fill = Personal.Loan,data = bank_data,geom = "dotplot")
qplot(CCAvg,fill = Personal.Loan,data = bank_data,geom = "density")
```

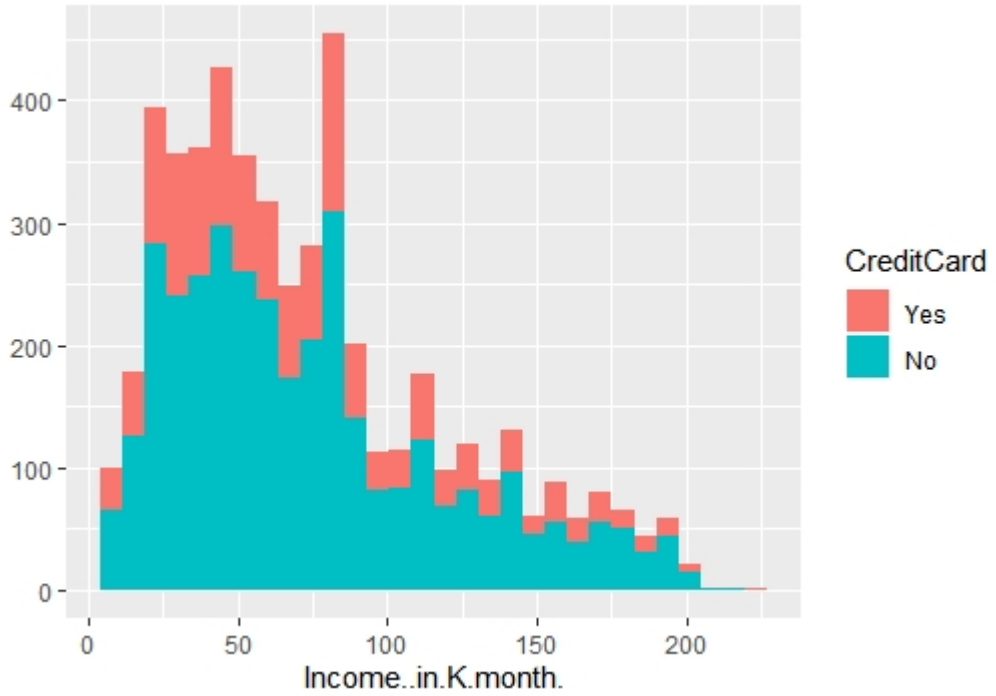


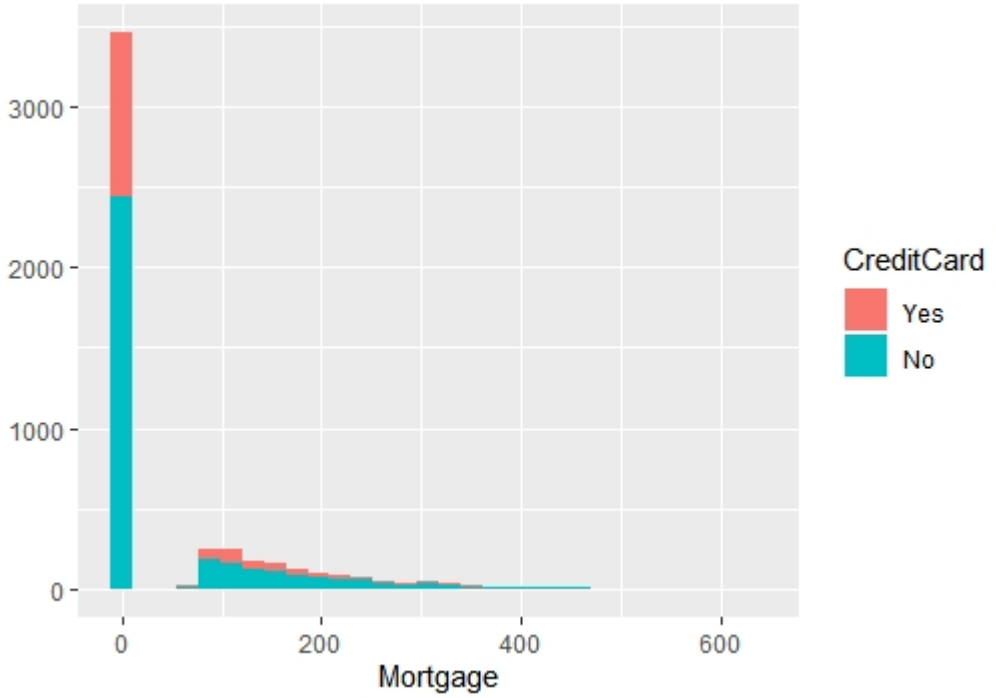




IV. Plotting the Numerical variables with other numerical variables:

```
qplot(Income..in.K.month.,fill = CreditCard,data = bank_data)
qplot(Mortgage,fill = CreditCard,data = bank_data)
```





Inferences:

1) We can see that the **Response Rate** doesn't depend on the **number of the family members** customer has because customers with **more family members** tend to have more **liabilities** and will tend to take a **Personal Loan** which is **not the case here**.

2) Even though not much, but we can see that in those who have responded **Yes,** majority of them comprise of the customers **who don't possess a credit card** meaning they are open to **Liability** than the ones who already possess a **Liability in the form of Credit Card.**
3) We can see that customers with **Under-graduation and having only one family member** contribute the **highest** to the **customer base.**
4) We can see that for most of the customers who **have responded Yes,** majority of them are **Customers who have a net banking facility** because customers who have a **Net Banking facility** can easily keep track of their **Loan amounts and pay the dues on time.**
5) We can see that majority of the customers who have **responded No** are the customers whose **Income is on the lower side.** But the customers who have their **incomes on the higher side** have applied for **Personal Loan** because they are sure that they can pay back the loan taken due to their **higher income.**
6) The customers who have responded **Yes** have a **higher Credit Card Usage** meaning that customers who are **unable** to pay their **Credit Card bills turn** towards **acquiring Personal Loan.**
7) The more customers who contribute to the **customers having credit card** consist of customers whose **Income is on the lower side.**
8) We can see that **majority of the customers who possess a Credit Card** have either **minimum or zero Mortgages to pay.**

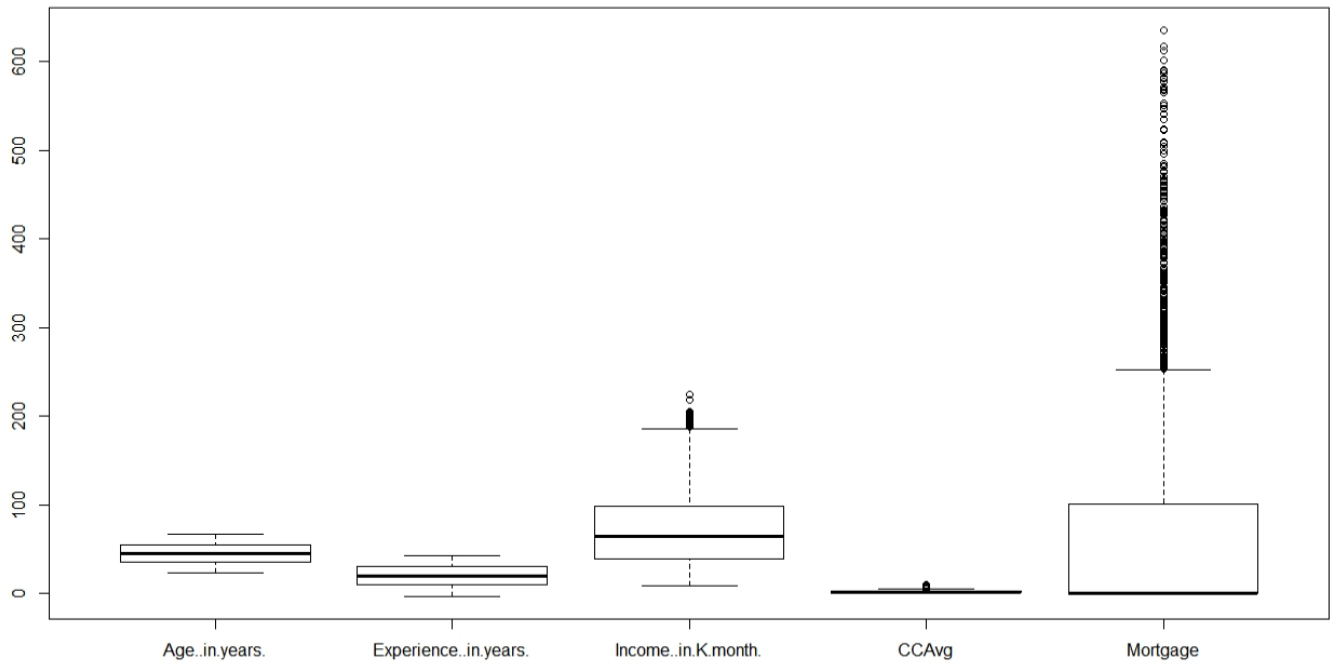
e.) Missing Value treatment:

We saw that in the **Variable Identification**, we saw that **Family Members variable** contained missing values. We select those values using **is.na()** function. Since they are present only in the **Family Members** variable, we can convert them to **Zero** because when bank couldn't find any **Family member associated with the customer,** it might have termed it **NA.** And also converting them to **Zero** would have the **least effect** on the dataset.

```
> ## Identifying and Changing the NA's from the family Column #####
> bank_data$Family.members = as.numeric(bank_data$Family.members)
> sum(is.na(bank_data))
[1] 18
> bank_data[is.na(bank_data)] = 0
> sum(is.na(bank_data))
[1] 0
> bank_data$Family.members = as.factor(bank_data$Family.members)
```

f.) Outlier Identification:

We can find out the variables having **outliers** by plotting a **Boxplot. Boxplot can be plotted** by using the function **Boxplot()**



Inferences:

As we can see from the above **Boxplots, Mortgage** has the **highest number of outliers,** followed by **Income variable** and then **CCAvg** has the least number of **Outliers.**

7) Clustering:

Clustering is an **Unsupervised Learning** which aims at making groups of **Homogenous items** in an **unlabelled data. A Cluster** can be defined as group of **Homogenous items** which are clearly **distinct** from the other **Clusters. The Homogeneity** can be determined either by studying the **closeness of the items (**Measurable or Non-Measurable) or by **distinction between the homogeneous items of a cluster.** There are two methods of **Clustering. Hierarchical and K-Means Clustering.**

Before performing **Clustering,** we can trim the dataset by removing the **ID and ZIP Code variables** as they don't contribute in any way to the analysis of the **dataset.**

```
> bank_data = bank_data[,-c(1,5)]
> colnames(bank_data)
 [1] "Age..in.years."        "Experience..in.years." "Income..in.K.month."
 [4] "Family.members"        "CCAvg"                 "Education"
 [7] "Mortgage"              "Personal.Loan"         "Securities.Account"
[10] "CD.Account"            "Online"                "CreditCard"
```

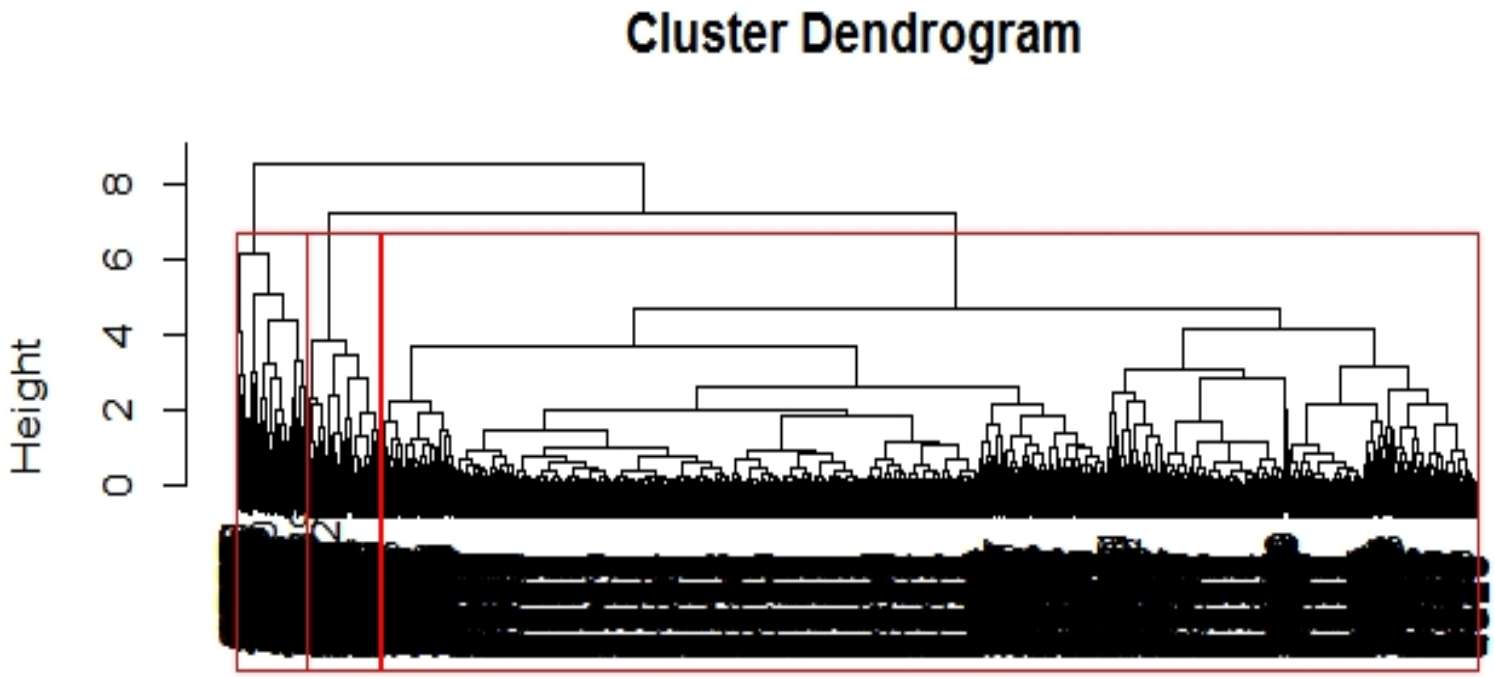
a.) Hierarchical Clustering:

In this method of **Clustering,** the items are **clustered** based on the distance between each of the items. This **measure of distance** can either be **Euclidian, Manhattan and Chessboard.** This method differs from the fact that **Clustering** is done and then a **rough number of Clusters** is estimated**.** From EDA, we can see that the **number of clusters required would be around three.** Before starting the **Clustering,** we first need to **scale the dataset and then extract distance matrix** which can be used as part of **Hierarchical clustering.**

These can be achieved by using the functions **scale()** and **dist()** and argument as **“Euclidean”, “Manhattan” or “Chessboard”** depending on our preference. We can then create the **Clusters** using the function

hclust() using the distance matrix created and plot them using the functions **plot()** and **rect.hclust()**. Then we can use these **Clusters** on to the dataset using the function **cuttree()** function where we determine the number of **clusters we need in the argument of that function.**Then the dataset can be **Aggregated** using the function **aggregate()** function.

```
> ### Hierarchial Clustering ###
> scale_bank_data = scale(bank_data[,c(3,5,7)])
> ## Taking the distance matrix ##
> dist_bank_data = dist(scale_bank_data)
> ## Creating Hierarchial Clusters ####
> hclust_bank_data = hclust(dist_bank_data,method = "complete")
> plot(hclust_bank_data)
> rect.hclust(hclust_bank_data,k = 3)
> ### Clustering the dataset using the Cluster object created ###
> bank_data_hclust = bank_data[,c(3,5,7)]
> bank_data_hclust$cluster = cutree(hclust_bank_data,k = 3)
> bank_data_hclust = aggregate(bank_data_hclust,list(bank_data_hclust$cluster),FUN = "mean")
> print(bank_data_hclust)
```



Clustered Dataframe:

	Group.1	Income..in.K.month.	CCAvg	Mortgage	cluster
1	1	63.71496	1.530697	39.64456	1
2	2	161.53242	6.519249	25.55290	2
3	3	138.32069	3.511931	344.47241	3

Inferences:

From the above results, we can see that the whole **Customer Base** has been divided into **3 Groups.** The **3 Clusters** are as follows:

Group	Income	Credit Card Usage	Mortgage
1	Low	Low	Moderate
2	High	High	Low
3	Moderate	Moderate	High

1) **Group 1** has **Low Income, Low Credit Card Usage and Moderate Mortgage.** So this group would try best to avoid **going for loans** as their **Incomes are low but still good enough to pay off the Moderate Mortgage** since their **Credit Card Usage is low**
2) **Group 2** has **High Income, High Credit Card Usage and Low Mortgage.** This Group would **never ever approach the bank for a loan** because these belong to the **richer section** customer base **who have more than enough Income** and have **never ever incurred mortgages** and **so would never require a Personal Loan.**
3) **Group 3 has Moderate Income, Moderate Credit Card Usage and High Mortgage.** This group is **more likely** to go for **Personal Loan** since their **Mortgages are high** but have **enough income** to pay back the **dues** considering their **moderate Credit Card Usage.**

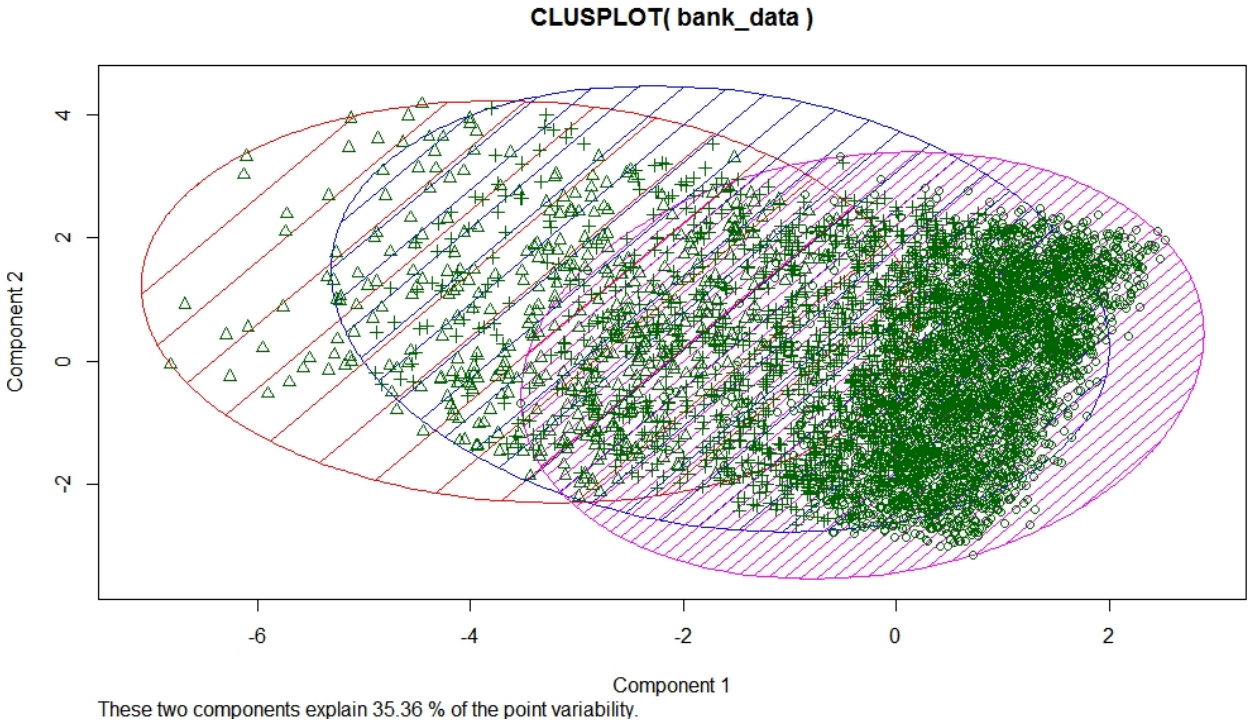
*We must note that the **number of clusters** was roughly estimated based on the **Scatter Plots** plotted between **Numerical Variables** and hence it may or may not be appropriate **number of clusters.***

b.) K-Means Clustering:

In this technique, the **Clustering** is done based on **distance of items** from a **common centroids.** The **number of centroids** needed to put them into **appropriate clusters** determines the **Number of Clusters.** By deciding the **number of Centroids,** we can determine the number of **Clusters** before starting the process of **Clustering.** Before running the **kmeans()** function, we must make sure run the **set.seed()** function to make sure the results are producible again and again.

To perform **Clustering,** the function **kmeans()** with the **parameters, centers(**Number of Centroids or Clusters) and **nstart(**Number of random sets be made for each set. After creation of the **Cluster object,** we can use the function **clusplot()** to plot how the **items were placed in the Clusters.**

```
### K-Means Clustering  ###
#### Randomly Clustering with the infered number of clusters ####
set.seed(77)
library(cluster)
k.cluster = kmeans(dist_bank_data,centers = 3,nstart = 5)
clusplot(bank_data,k.cluster$cluster,lines = 1,color = TRUE,shade = TRUE)
```

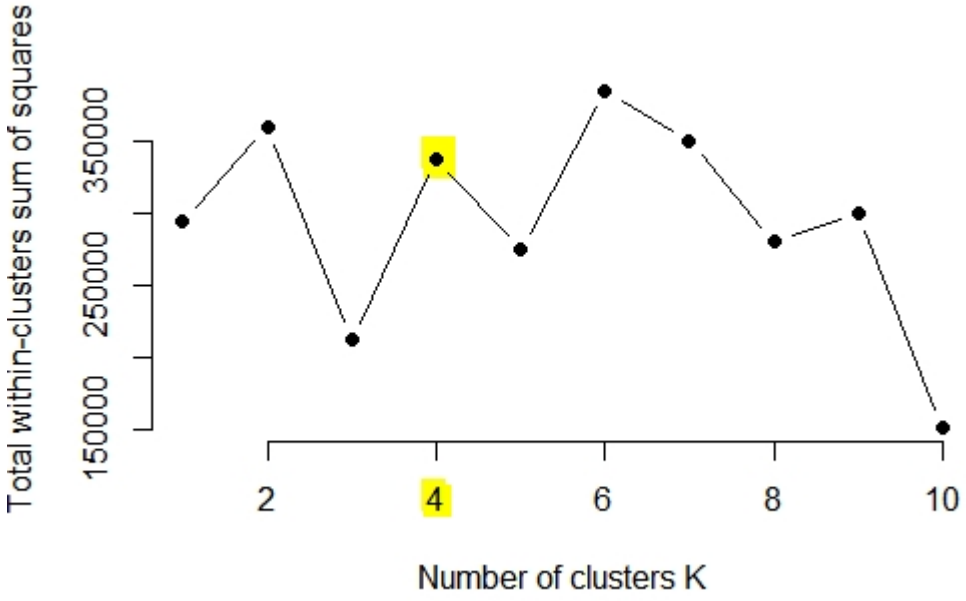


Even though we have given the **Number of Clusters** beforehand, we have various methods to find the **right number of Clusters.** Finding the right number of clusters helps in **explains much of the variance of the data** and also helps in getting **more distinct clusters.**

Plotting a graph between Number of Clusters and Variance between Clusters:

A graph can be plotted stating the relationship between the number of **clusters** and the **variance between the Clusters.** The right number of Clusters can be decided by choosing the number **which gives out the highest variance.** But the number of clusters must be optimal as **very low number or very high number** of clusters might result in **over fitting or under fitting.**

```
### Trying to find the right number of Clusters by calculating variance ###
set.seed(77)
k.values = c(1:10)
kcluster2 = kmeans(dist_bank_data,centers = 9,nstart = 13)
plot(k.values, kcluster2$withinss,
     type="b", pch = 19, frame = FALSE,
     xlab="Number of clusters K",
     ylab="Total within-clusters sum of squares")
```



From the above graph, we can see that we choose **4** as the right number of Clusters as **2** would result in way **too less** of clusters and **6** would result in way **too many** Clusters.

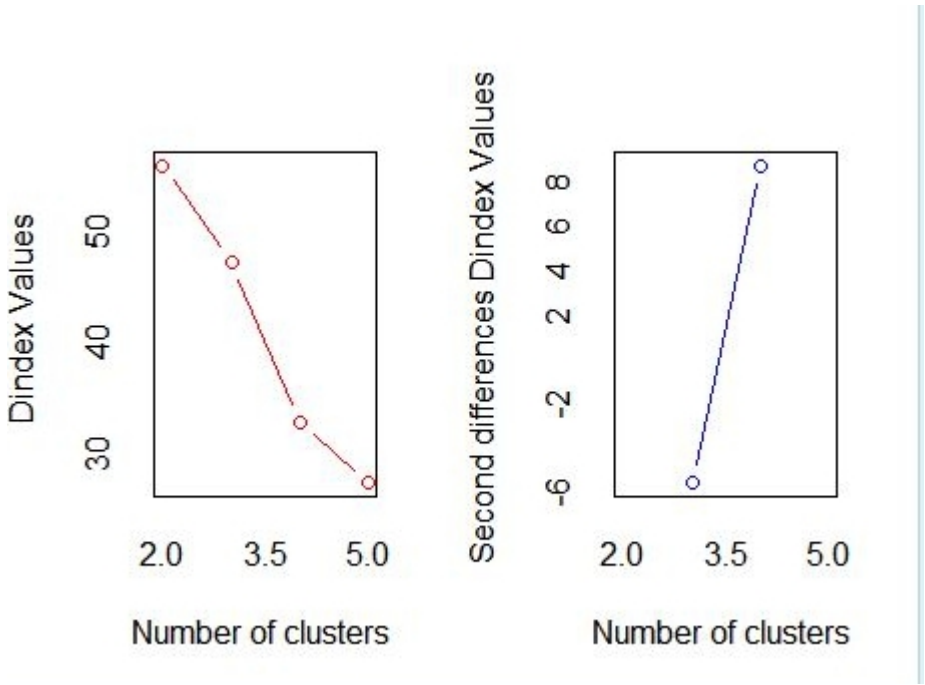
Using the NbClust package to determine the number of clusters:

The **NbClust** function helps in us in determining the number of clusters by **using 30 indices** which help in determining the **appropriate number of clusters** given the **Dataframe.** The arguments of the **NbClust()** function are **dataframe, minimum number of clusters and maximum number of clusters.**

```
### K means Clustering using the number of clusters obtained after the experiment ##
set.seed(77)
nc1 = NbClust(bank_data[,c(3,5,7)],min.nc = 2,max.nc = 5,method = "kmeans")
```

```
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

*** : The D index is a graphical method of determining the number of clusters.
                In the plot of D index, we seek a significant knee (the significant peak in Dindex
                second differences plot) that corresponds to a significant increase of the value of
                the measure.

*******************************************************************
* Among all indices:
* 9 proposed 2 as the best number of clusters
* 2 proposed 3 as the best number of clusters
* 10 proposed 4 as the best number of clusters
* 3 proposed 5 as the best number of clusters

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  4

*******************************************************************
```

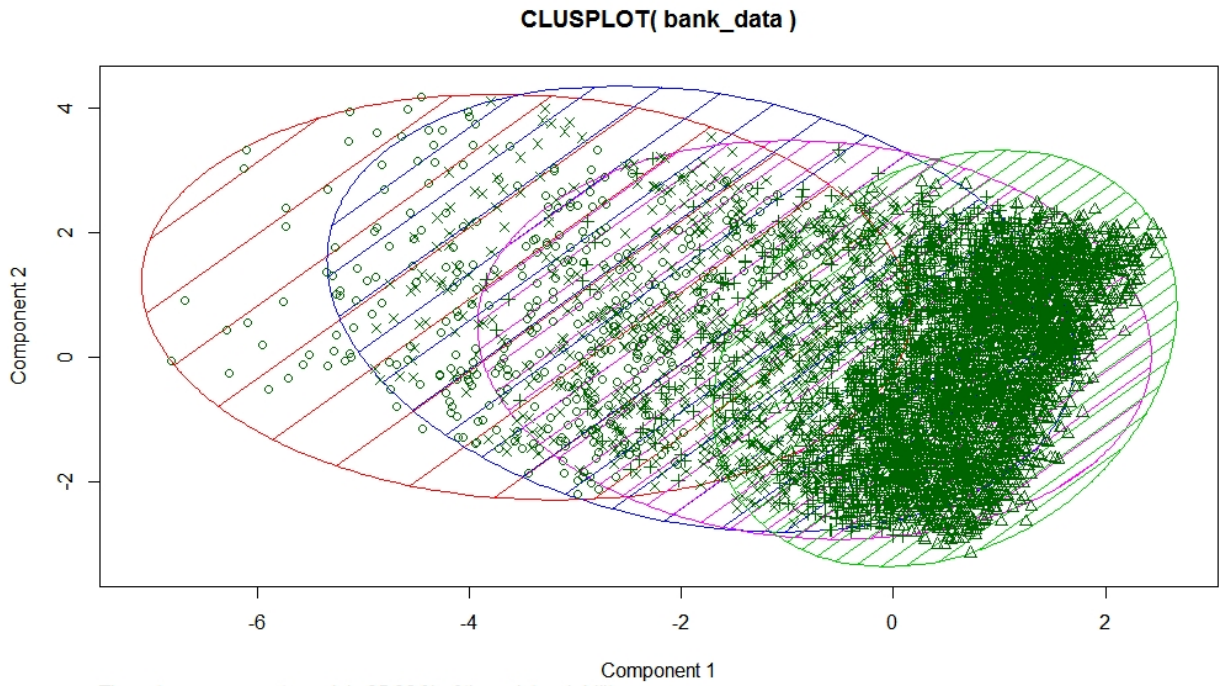
```
' According to the majority rule, the best number of clusters is  4
```



Hence, from the above results, we can see that **4** is the appropriate number of clusters.

By choosing the number of clusters as **4,** we again create a **Kmeans Clustering object** and use it to **Cluster** our dataset.

```
### Perfroming the K means Clustering with number of clusters as 4 ####
set.seed(77)
kcluster = kmeans(dist_bank_data,centers = 4,nstart = 8)
clusplot(bank_data,kcluster$cluster,lines = 1,color = TRUE,shade = TRUE)
```




```
> ### Using the clustering object on the dataset ####
> bank_data_kclust = bank_data[,c(3,5,7)]
> bank_data_kclust$Cluster = kcluster$cluster
> bank_data_kclust = aggregate(bank_data_kclust,list(bank_data_kclust$Cluster),FUN = mean)
> print(bank_data_kclust)
```

	Group.1	Income..in.K.month.	CCAvg	Mortgage	Cluster
1	1	159.86792	5.5687500	183.62736	1
2	2	33.87980	0.7826189	44.51816	2
3	3	75.59621	1.8876436	10.11723	3
4	4	114.98927	2.8007403	107.84871	4

Inferences:

From the above results, we can see that the whole **Customer Base** has been divided into **4 Groups.** The **4 Clusters** are as follows:

Group	Income	Credit Card usage	Mortgage
1	Very High	Very High	Very High
2	Low	Low	Moderate
3	Moderate	Moderate	Low
4	High	High	High

1) **Group 1** has more **Very High Income, Very High Credit Card usage and Very High Mortgage.** Therefore, there are **more chances** of this group **going for a Personal Loan** due to their **High Mortgages** and **High Credit Card Usage** and are confident that they would be able to pay it back due to their **Very High Income.**

2) **Group 2** has **Low Income, Low Credit Card Usage and Moderate Mortgage.** This group has the **least chance** of **going for Personal Loan** as their **Mortgages are not that much** and even if they did take a loan, **their low income** wouldn't be **enough** to pay back their loan.

3) **Group 3** has **Moderate Income, Moderate Credit Card usage and Low Mortgage.** This group would **never go for a loan** as their **Income is Moderate** and **Mortgages are very low** and **do not have the need to take a loan** as their **incomes would be enough** to pay off the **mortgages.**

4) **Group 4** has **High Income, High Credit Card Usage and High Mortgage.** This group **also has chances of going for personal loan** due to their **High Mortgages and High Credit Card Usage** and the fact that **their incomes are high** results in confidence that they can **pay back the loan if taken.**

4) Decision Tree (Classification) and Performance measures of the Models:

*Before running some of the functions in the Model Building process, we must make sure run the **set.seed()** function to make sure the results are producible again and again.*

Decision Trees help build **Classification and Regression models** in the form of a **branches and leaves** structure. **Root node** is the beginning of a **Decision tree.** From there, Splitting of data is done at every **branch** and the final **predictions and probabilities** are present in present in the **final leaf nodes or decision nodes.** There are various methods to **build a classification Decision Tree** two of which are **CART** and **Random Forest.**

Before we start with the building process, we must follow the basic principle of **Machine Learning** which states that we must have two samples of data namely **Training data** and **Testing data.** The **Training data** must have a **larger sample size** than the **Testing data.** The **Training data** must be used in **model building** and that **model** must be checked for **Performance** on the **Testing data.** But since we are not provided with **separate training and testing data**, we must split the existing data into **train** and **test** data in the ratio of **0.7**(Industry Standard Splitting Ratio). This can be achieved using the **sample.split()** function from the library **caTools()**.

```
> ### Splitting the dataset into training and testing dataset ####
> ####(Industry standard splitting ratio of 0.7) ####
> set.seed(77)
> split <- sample.split(bank_data$Personal.Loan, SplitRatio = 0.7)
> train<- subset(bank_data, split == TRUE)
> test<- subset(bank_data, split == FALSE)
> dim(train)
[1] 3500   12
> dim(test)
[1] 1500   12
```

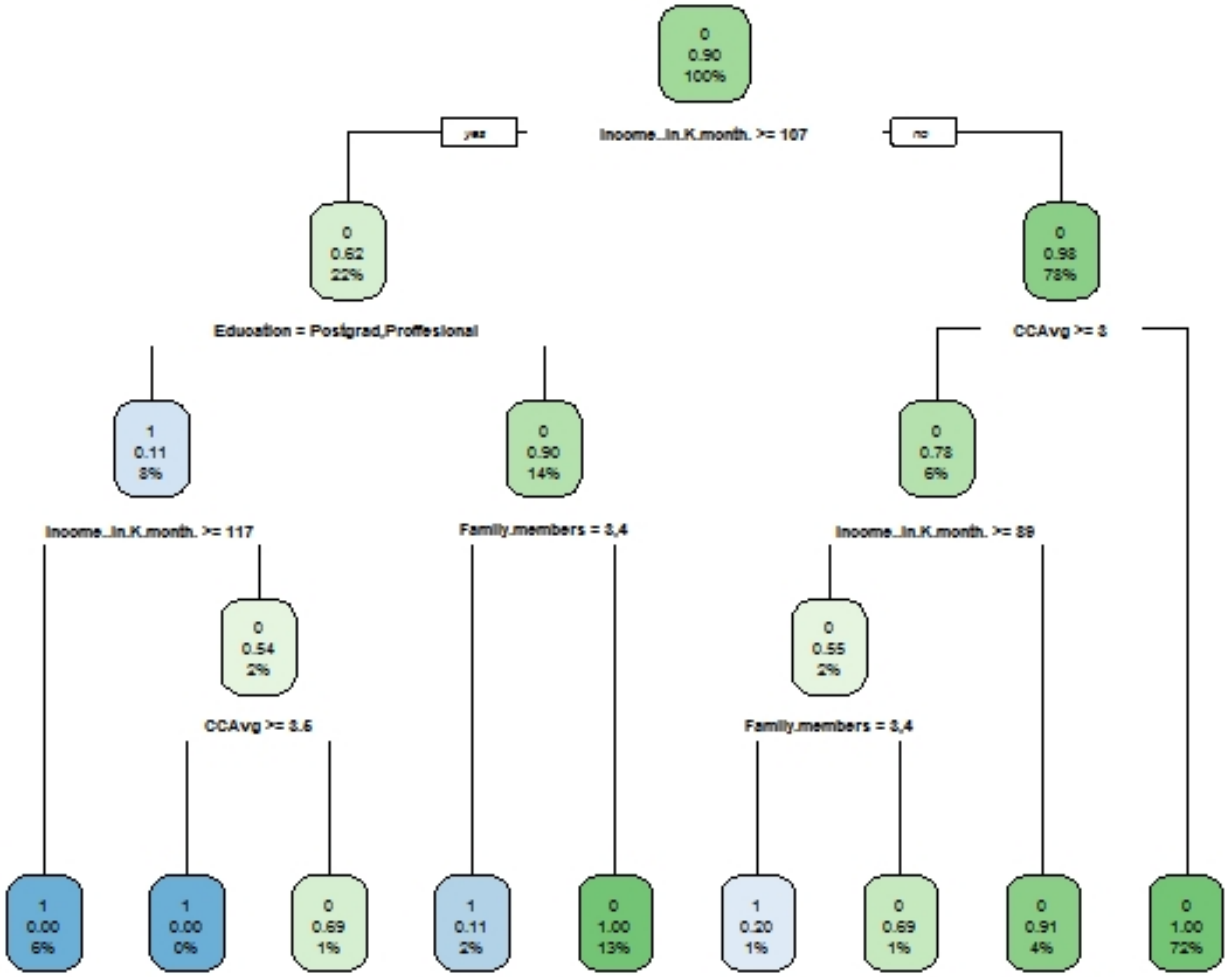
a.) CART Decision tree:

CART (Classification and Regression Tree) is one of the methods of preparing a **Classification predictive model** which uses **Tree structure** to explain the predications of the **Classification or Regression model.** It is one of the standard techniques to build a **classification model.**

CART model can be built for a database using the function **rpart() with arguments minbucket and minsplit** from the library **rpart.** Then the tree can be plotted using the function **rpart.plot()** from the package **rpart.plot.**

```
####CART MODEL ####
### For the process of creating a CART Basesd Model on train data and using it on train data, ###
### we create a dataframe named  "tntrain1" similar to the original train dataset ###
tntrain1 = train
tntrain1$Personal.Loan = factor(tntrain1$Personal.Loan,levels = c("Yes","No"),labels = c(1,0))
### Creating a CART model based on train data ###
set.seed(77)
rttree = rpart(tntrain1$Personal.Loan~.,data = tntrain1,minsplit = 10,minbucket = 10)
print(rttree)
rpart.plot(rttree)
```

```
n= 3500

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 3500 336 0 (0.09600000 0.90400000)
   2) Income..in.K.month.>=106.5 779 294 0 (0.37740693 0.62259307)
     4) Education=Postgrad,Proffesional 276  31 1 (0.88768116 0.11231884)
       8) Income..in.K.month.>=116.5 219   0 1 (1.00000000 0.00000000) *
       9) Income..in.K.month.< 116.5 57  26 0 (0.45614035 0.54385965)
        18) CCAvg>=3.535 12   0 1 (1.00000000 0.00000000) *
        19) CCAvg< 3.535 45  14 0 (0.31111111 0.68888889) *
     5) Education=Undergrad 503  49 0 (0.09741551 0.90258449)
      10) Family.members=3,4 55   6 1 (0.89090909 0.10909091) *
      11) Family.members=0,1,2 448   0 0 (0.00000000 1.00000000) *
   3) Income..in.K.month.< 106.5 2721  42 0 (0.01543550 0.98456450)
     6) CCAvg>=2.95 193  42 0 (0.21761658 0.78238342)
      12) Income..in.K.month.>=88.5 69  31 0 (0.44927536 0.55072464)
        24) Family.members=3,4 20   4 1 (0.80000000 0.20000000) *
        25) Family.members=1,2 49  15 0 (0.30612245 0.69387755) *
      13) Income..in.K.month.< 88.5 124  11 0 (0.08870968 0.91129032) *
     7) CCAvg< 2.95 2528   0 0 (0.00000000 1.00000000) *
```

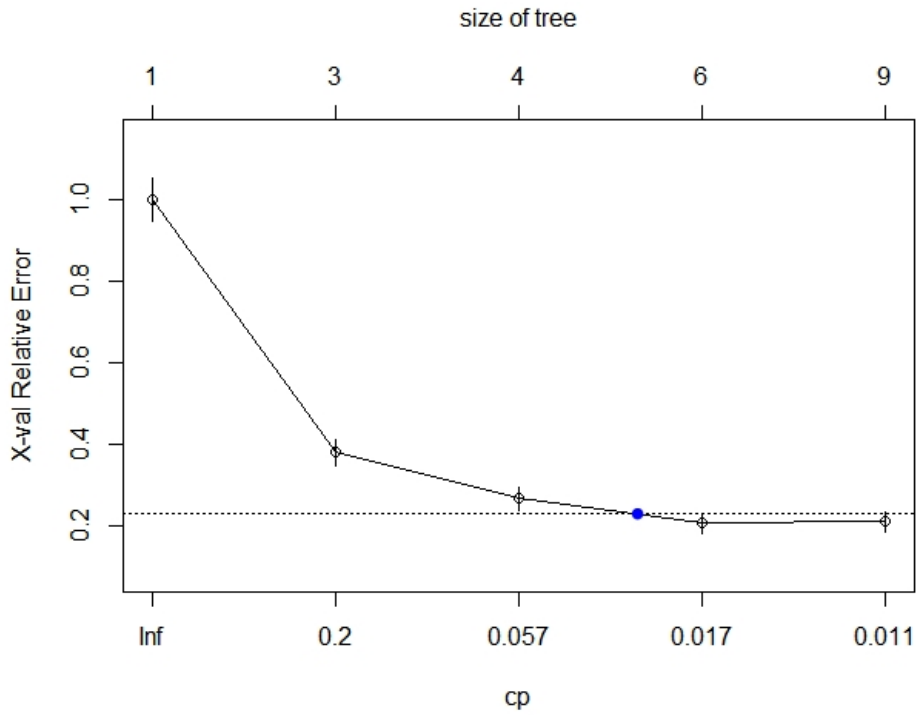


As we can see in the above plot, the **CART model** is quite long and complex meaning it also **accommodates noise along with the data.**

To avoid this, a process called **Pruning** must be done for the built tree in order to make the tree shorter and little less complex. Process of Pruning is based on the parameter **Complexity Parameter(CP)** of the tree. We must try to find the right **Complexity Parameter** making sure to neither **under fit the data** nor **overfed the data.** This can be decided by using the Thumb Rule on the **results of the Cross Validation tests.** We must select **CP** such that,

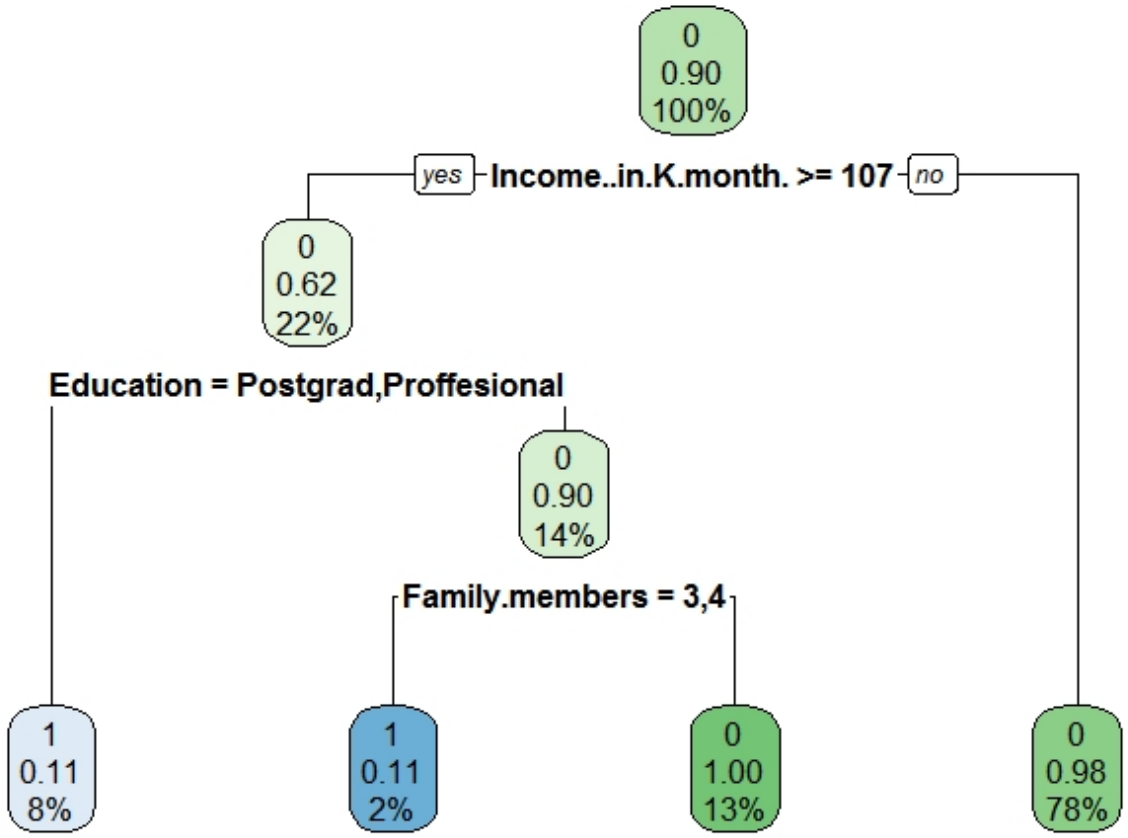
xstd + rel error < xerror

```
> ### Since we can see that the tree is complex, we need find the decide the right Complexity Parameter ##
> ### to prune the tree in the right way ###
> printcp(rttree)

Classification tree:
rpart(formula = tntrain1$Personal.Loan ~ ., data = tntrain1,
    minsplit = 10, minbucket = 10)

Variables actually used in tree construction:
[1] CCAvg               Education           Family.members      Income..in.K.month.

Root node error: 336/3500 = 0.096

n= 3500

        CP nsplit rel error  xerror     xstd
1 0.318452      0   1.00000 1.00000 0.051870
2 0.127976      2   0.36310 0.38095 0.033050
3 0.025298      3   0.23512 0.26786 0.027869
4 0.011905      5   0.18452 0.20833 0.024650
5 0.010000      8   0.14881 0.21131 0.024822
> plotcp(rttree)
```



As we can see from the above results, the right **CP** to prune the **Tree** is **0.025298.** We can create a **Pruned tree** from the original tree using the **prune() function.**

```
> ### Pruning the tree using the CP, 0.025298 ####
> prttree = prttree = prune(rttree,cp = 0.025298)
> print(prttree)
n= 3500

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 3500 336 0 (0.09600000 0.90400000)
   2) Income..in.K.month.>=106.5 779 294 0 (0.37740693 0.62259307)
     4) Education=Postgrad,Proffesional 276  31 1 (0.88768116 0.11231884) *
     5) Education=Undergrad 503  49 0 (0.09741551 0.90258449)
      10) Family.members=3,4 55   6 1 (0.89090909 0.10909091) *
      11) Family.members=0,1,2 448   0 0 (0.00000000 1.00000000) *
   3) Income..in.K.month.< 106.5 2721  42 0 (0.01543550 0.98456450) *
> rpart.plot(prttree)
```



As we can see from the above plot, **the pruned tree is less complex.**

Now we can use the **predict()** function to get the **Predictions and Probabilities of** the **Response Variable Personal Loan.**

```
### Using the model to make predictions on the tntrain1 dataset ####
tntrain1$predict = predict(prttree,tntrain1,type = "class")
tntrain1$prob = predict(prttree,tntrain1,type = "prob")[,1]
View((head(tntrain1)))
```

Personal.Loan	Securities.Account	CD.Account	Online	CreditCard	predict	prob
0	Yes	No	No	No	0	0.0154355
0	No	No	Yes	No	0	0.0154355
0	No	No	Yes	No	0	0.0154355
1	No	No	No	No	1	0.8876812
0	No	No	No	No	0	0.0154355
0	No	No	Yes	No	0	0.0154355

Performance Model Measures:

To test the performance of **any Classification model,** we have various measures which are as follows:

1) **Concordance Ratio :** In this method, the **values of Response variables and Probabilities are taken as pairs** and then tested if the **probabilities** predicted actually hold true. And then the number of pairs are counted with respect to total number of **pairs.**
2) **Error Rate:** In this method, the frequency of all the **Falsely Predicted values** are counted and then compared with the total number of **values** in the dataset. This shows to what extent the **predictions done were wrong.**
3) **ROC & AUC:** Both of these measures help in determining the of separation of the different Categories present in the **Target and Prediction Variables.** AUC = Area Under the Curve, ROC = Receiver Operating Characteristics.
4) **Gini:** The **Gini Coefficient** be determined to test the **purity** of the classes divided in the **Target variable using the prediction model.**
5) **KS Value:** The **KS**(Kolmogorov-Smirnov) value is the **highest separation of classes** that has been achieved by the predictive model.

Testing the performance of model built using the Training data on Training Data(Training Model on Train data):

1) Concordance Ratio:

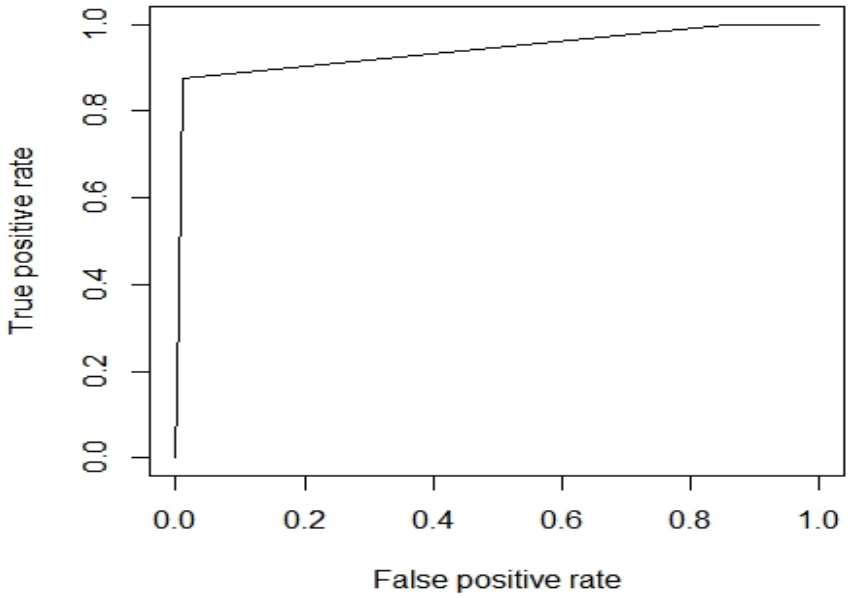
```
> ## CONCORDANCE RATIO (TRAINING MODEL ON TRAINING DATA) ##
> x = tntrain1$Personal.Loan
> y = tntrain1$prob
> Concordance(actuals = x,predictedScores = y)
$Concordance
[1] 0.8838956

$Discordance
[1] 0.1161044

$Tied
[1] 5.551115e-17

$Pairs
[1] 1063104
```

2) Error Rate:

```
> ## CONFUSION MATRIX ERROR RATE (TRAINING MODEL ON TRAINING DATA) ##
> cnf_tntrain1 = table(tntrain1$Personal.Loan,tntrain1$predict)
> ## CONFUSION MATRIX ERROR RATE (TRAINING MODEL ON TRAINING DATA) ##
> cnf_tntrain1 = table(tntrain1$Personal.Loan,tntrain1$predict)
> errrate_tntrain1 = (cnf_tntrain1[2,1]+cnf_tntrain1[1,2])/nrow(tntrain1)
> print(errrate_tntrain1)
[1] 0.02257143
> |
```

3) ROC:

```
## PLOTTING THE TPR AND FPR GRAPH AND FINDING ROC ###
##(TRAINING MODEL ON TRAINING DATA) ##
tntrain1.pred.obj = prediction(tntrain1$prob,tntrain1$Personal.Loan)
tntrain1.perf = performance(tntrain1.pred.obj,"tpr","fpr")
plot(tntrain1.perf)
```



4) KS-Value:

```
> ### KS VALUE ###
> print(max(tntrain1.perf@y.values[[1]] - tntrain1.perf@x.values[[1]]))
[1] 0.8633059
```

5) Gini:

```
> #### GINI COEFFICIENT (TRAINING MODEL ON TRAINING DATA) ####
> gini1 = ineq(tntrain1$prob,"gini")
> print(gini1)
[1] 0.7964702
```

6) AUC:

```
> #### AUC (TRAINING MODEL ON TRAINING DATA) ####
> tntrain1.auc = performance(tntrain1.pred.obj,"auc")
> tntrain1.auc = as.numeric(tntrain1.auc@y.values)
> print(tntrain1.auc)
[1] 0.9405256
```

Making predictions on the test data using the Predictive model created using training data (Train model on Test Data)

```
> ### For the process of testing the CART Based Model on train data and using it on test data, ###
> ### we create a dataframe named  "tntest1" similar to the original test dataset ###
> tntest1 = test
> tntest1$Personal.Loan = factor(tntest1$Personal.Loan,levels = c("Yes","No"),labels = c(1,0))
> ### Using the training data CART model to make predicitions on the test data ###
> tntest1$predict = predict(prttree,tntest1,type = "class")
> tntest1$prob = predict(prttree,tntest1,type = "prob")[,1]
> View((head(tntest1)))
```

Personal.Loan	Securities.Account	CD.Account	Online	CreditCard	predict	prob
0	Yes	No	No	No	0	0.0154355
0	No	No	No	No	0	0.0154355
0	No	No	No	No	0	0.0154355
0	No	No	No	Yes	0	0.0154355
0	No	No	Yes	No	0	0.0154355
0	No	No	No	Yes	0	0.0154355

Testing the performance of model built using the Training data on Testing Data(Training Model on Test data):

1) Concordance Ratio:

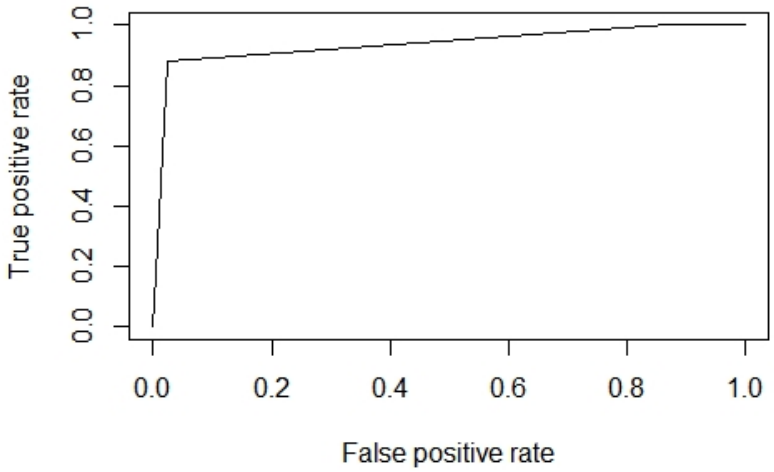
```
> ## CONCORDANCE RATIO (TRAINING MODEL ON TESTING DATA) #
> a = tntest1$Personal.Loan
> b = tntest1$prob
> Concordance(actuals = a,predictedScores = b)
$Concordance
[1] 0.8793019

$Discordance
[1] 0.1206981

$Tied
[1] 4.163336e-17

$Pairs
[1] 195264
```

2) Error Rate:

```
> ## CONFUSION MATRIX ERROR RATE (TRAINING MODEL ON TESTING DATA) ##
> cnf_tntest1 = table(tntest1$Personal.Loan,tntest1$predict)
> errrate_tntest1 = (cnf_tntest1[2,1]+cnf_tntest1[1,2])/nrow(tntest1)
> print(errrate_tntest1)
[1] 0.03333333
```

3) ROC:

```
## PLOTTING THE TPR AND FPR GRAPH TO FIND ROC (TRAINING MODEL ON TESTING DATA) ##
tntest1.pred.obj = prediction(tntest1$prob,tntest1$Personal.Loan)
tntest1.perf = performance(tntest1.pred.obj,"tpr","fpr")
plot(tntest1.perf)
```



4) KS-Value:

```
> #### KS VALUE (TRAINING MODEL ON TESTING DATA) ####
> print(max(tntest1.perf@y.values[[1]] - tntest1.perf@x.values[[1]]))
[1] 0.8576082
```

5) Gini:

```
> #### GINI INDEX (TRAINING MODEL ON TESTING DATA)####
> gini2 = ineq(tntest1$prob,"gini")
> print(gini2)
[1] 0.7941752
```

6) AUC:

```
> #### AUC (TRAINING MODEL ON TESTING DATA) ####
> tntest1.auc = performance(tntest1.pred.obj,"auc")
> tntest1.auc = as.numeric(tntest1.auc@y.values)
> print(tntest1.auc)
[1] 0.9369085
```

Inferences:

We can see that after building a **Classification CART Model**, we used it to make **predictions and also find the probabilities** on both the **Training** and **Testing data.** We also performed **Performance measures** on the **model** and **checked them** with both the **Training** and **Testing** data. The results are as follows:

Performance Measure	(Training model on Training Data)	Training model on Testing Data)
Concordance(More the better)	0.883	0.879
Error Rate(Lesser the better)	0.022	0.033
ROC(More the better)	0.85	0.85
KS-Value(More the better)	0.863	0.857
Gini(More the better)	0.796	0.794
AUC(More the better)	0.940	0.936

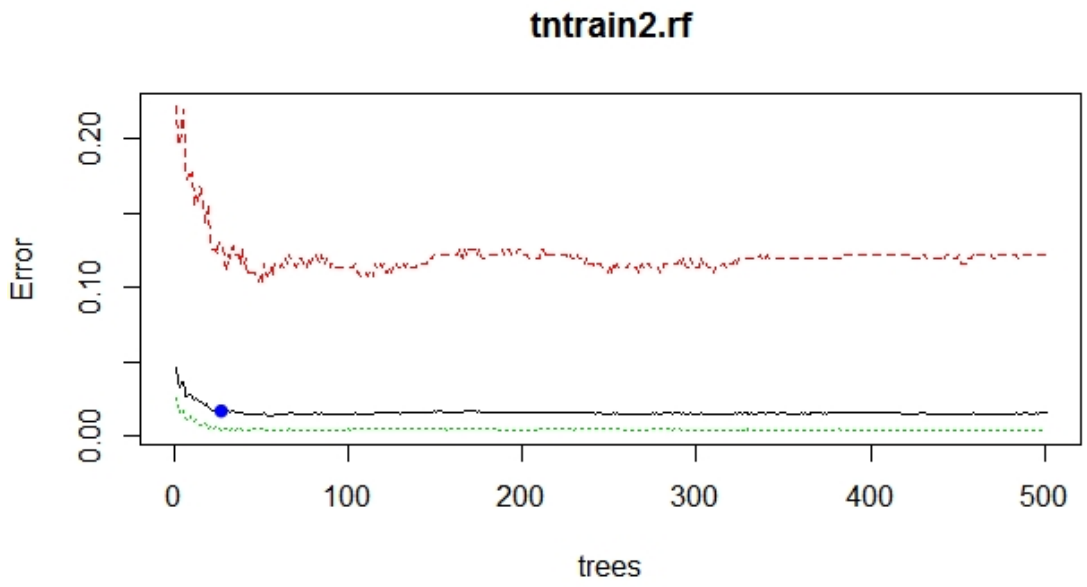
From the above results, we can see that results of **Training model on testing data** do not differentiate much from the results of **Training model on Training data.** And also we can see that all the values for the **Performance measures** for **Training model on Testing data** to tell that the **Predictive model** hence created is a **Legible and Good model** and can be further used for analysis.

b.) Random Forest:

Random Forest is an **ensemble learning technique** where **multiple decision trees** are built and after analysing all the predictions by each of those **multiple trees**, the **best model tree** is selected. This is

done to get a better model. For **classification models**, the **average of all the trees** is considered as the best tree. Random Forest can be built using the function **randomForest()** from the **randomForest library.**

```
> ### For the process of creating a Random Forest Model on train data and using it on train data, ###
> ### we create a dataframe named  "tntrain2" similar to the original train dataset ###
> tntrain2 = train
> tntrain2$Personal.Loan = factor(tntrain2$Personal.Loan,levels = c("Yes","No"),labels = c(1,0))
> ### Creating a Random Forest model on the train data ###
> #### Building a random forest using the random number of trees for the random forest ####
> set.seed(77)
> tntrain2.rf = randomForest(tntrain2$Personal.Loan~.,data = tntrain2,
+                            ntree =501,mtry = 3,nodesize = 10,importance = TRUE)
> plot(tntrain2.rf)
> print(tntrain2.rf)


Call:
 randomForest(formula = tntrain2$Personal.Loan ~ ., data = tntrain2,      ntree = 501, mtry = 3, nodesize = 10, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 501
No. of variables tried at each split: 3

        OOB estimate of  error rate: 1.54%
Confusion matrix:
     1    0 class.error
1 295   41 0.122023810
0  13 3151 0.004108723
```



As we can see above, the parameter **ntree(**Number of trees**)** decides the **OOB(**Out Of Bag) errors. We must make sure that **ntree** is an **optimal value** since if the number is **too high**, we may get lesser OOB error but computation costs become higher and if the number is **too**

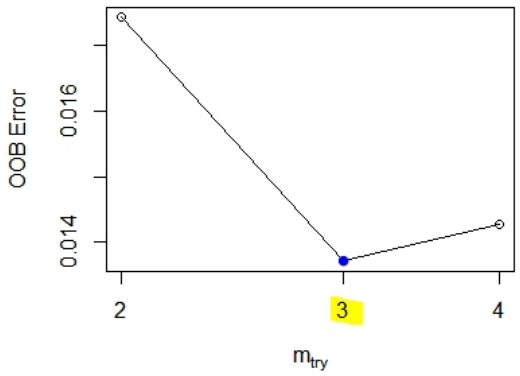
low, we may be increasing the **OOB error.** To find the optimal number, we must plot the object **err.rate** from the **Random Forest** and find out which least **ntree** value has the least **OOB error.** From the plot, we can see that the optimal number of **ntree** can lie around **40-60**.

```
> error = tntrain2.rf$err.rate
> error = as.data.frame(error)
> error$ntree = c(1:nrow(error))
> s.error = error[order(error$OOB,error$ntree),]
> s.error$ntree[1]
[1] 57
```

From the above, we can exactly say that for **ntree = 57,** we get the least **OOB error** while reducing the computational costs.

Another parameter which must be taken into consideration is **mtry** which refers to the number of variables must be selected at each tree during the process of ensemble learning. The number be must be an optimal number such that it is not too much to cause **overfitting** and not too less such that **OOB** error is **more.** This optimal number can be found out using the function **tuneRF()** function. The function checks for optimal number and goes ahead and builds a tuned **Random Forest** using those parameters.

```
> ### From the above, we can determine that the right number of trees is 57 ####
> ### Finidng the right number of mtry ###
> set.seed(77)
> tuned.tntrain2.rf = tuneRF(x = tntrain2[,-c(8)],y = tntrain2$Personal.Loan,mtryStart = 3,stepFactor=1.5,
+                            nodesize = 10,ntreeTry = 57,importance = TRUE,
+                            improve = 0.01,trace = TRUE,plot = TRUE,doBest = TRUE)
mtry = 3  OOB error = 1.37%
Searching left ...
mtry = 2        OOB error = 1.74%
-0.2708333 0.01
Searching right ...
mtry = 4        OOB error = 1.43%
-0.04166667 0.01
```




```
> print(tuned.tntrain2.rf)

Call:
 randomForest(x = x, y = y, mtry = res[which.min(res[, 2]), 1],      nodesize = 10, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 1.51%
Confusion matrix:
     1    0 class.error
1 295   41 0.122023810
0  12 3152 0.003792668
```

We can see that the **Random Forest Model** has been built with parameter **mtry = 3.**

Making predictions on the train data using the Predictive model created using training data (Train Random Forest model on Train Data)

```
> ### Using the tuned Random Forest on train data to make predictions ####
> tntrain2$predict = predict(tntrain2.rf,tntrain2,type = "class")
> tntrain2$prob = predict(tntrain2.rf,tntrain2,type = "prob")[,"1"]
> View(tntrain2)
```

Personal.Loan	Securities.Account	CD.Account	Online	CreditCard	predict	prob
0	Yes	No	No	No	0	0.000000000
0	No	No	Yes	No	0	0.000000000
0	No	No	Yes	No	0	0.000000000
1	No	No	No	No	1	0.976047904
0	No	No	No	No	0	0.033932136
0	No	No	Yes	No	0	0.001996008
0	Yes	No	No	No	0	0.035928144

Testing the performance of model built using the Training data on Train Data(Training Random Forest Model on Train data):

1) Concordance Ratio:

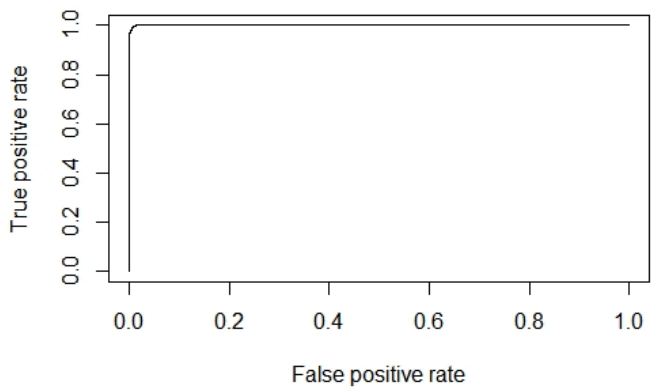
```
> ### Testing the Random Forest on train data ####
> ## CONCORDANCE RATIO (TRAINING RANDOM FOREST ON TRAINING DATA) ##
> c = tntrain2$Personal.Loan
> d = tntrain2$prob
> Concordance(actuals = c,predictedScores = d)
$Concordance
[1] 0.999826

$Discordance
[1] 0.0001740187

$Tied
[1] 4.786753e-17

$Pairs
[1] 1063104
```

2) Error Rate:

```
> ## CONFUSION MATRIX ERROR RATE (TRAINING RANDOM FOREST ON TRAINING DATA) ##
> cnf_tntrain2 = table(tntrain2$Personal.Loan,tntrain2$predict)
> errrate_tntrain2 = (cnf_tntrain2[2,1]+cnf_tntrain2[1,2])/nrow(tntrain2)
> print(errrate_tntrain2)
[1] 0.004571429
```

3) ROC:

```
> ## PLOTTING THE TPR AND FPR GRAPH TO FIND ROC (TRAINING RANDOM FOREST ON TRAINING DATA) ##
> tntrain2.pred.obj = prediction(tntrain2$prob,tntrain2$Personal.Loan)
> tntrain2.perf = performance(tntrain2.pred.obj,"tpr","fpr")
> plot(tntrain2.perf)
```



4) KS-Value:

```
> ## KS VALUE (TRAINING RANDOM FOREST ON TRAINING DATA) ##
> print(max(tntrain2.perf@y.values[[1]] - tntrain2.perf@x.values[[1]]))
[1] 0.9891224
```

5) Gini:

```
> #### AUC (TRAINING RANDOM FOREST ON TRAINING DATA) ####
> tntrain2.auc = performance(tntrain2.pred.obj,"auc")
> tntrain2.auc = as.numeric(tntrain2.auc@y.values)
> print(tntrain2.auc)
[1] 0.9998283
```

6) AUC:

```
> #### Gini Coefficient (TRAINING RANDOM FOREST ON TRAINING DATA) ####
> gini3 = ineq(tntrain2$prob,"gini")
> print(gini3)
[1] 0.8898675
```

Making predictions on the test data using the Predictive model created using training data (Train Random Forest model on Test Data)

```
### Using the tuned Random Forest on test data to make predictions ####
tntest2$predict = predict(tntrain2.rf,tntest2,type = "class")
tntest2$prob = predict(tntrain2.rf,tntest2,type = "prob")[,"1"]
View(tntest2)
```

Personal.Loan	Securities.Account	CD.Account	Online	CreditCard	predict	prob
0	Yes	No	No	No	0	0.000000000
0	No	No	No	No	0	0.000000000
0	No	No	No	No	0	0.015968064
0	No	No	No	Yes	0	0.000000000
0	No	No	Yes	No	0	0.000000000
0	No	No	No	Yes	0	0.000000000
0	No	No	Yes	No	0	0.001996008
0	Yes	No	No	No	1	0.732534930
0	No	No	Yes	No	0	0.003992016

Testing the performance of model built using the Training data on Testing Data(Training Random Forest Model on Test data):

1) Concordance Ratio:

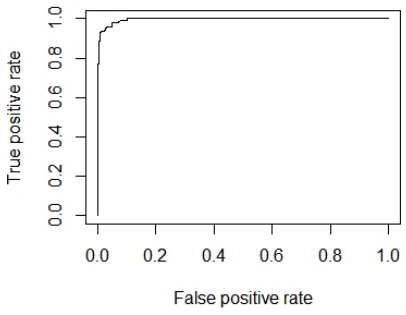
```
> ## CONCORDANCE RATIO (TRAINING RANDOM FOREST ON TESTING DATA) ##
> c = tntest2$Personal.Loan
> d = tntest2$prob
> Concordance(actuals = c,predictedScores = d)
$Concordance
[1] 0.9958466

$Discordance
[1] 0.004153351

$Tied
[1] 1.12757e-17

$Pairs
[1] 195264
```

2) Error Rate:

```
> ## CONFUSION MATRIX ERROR RATE (TRAINING RANDOM FOREST ON TESTING DATA) ##
> cnf_tntest2 = table(tntest2$Personal.Loan,tntest2$predict)
> errrate_tntest2 = (cnf_tntest2[2,1]+cnf_tntest2[1,2])/nrow(tntest2)
> print(errrate_tntest2)
[1] 0.01466667
```

3) ROC:

```
> ## PLOTTING THE TPR AND FPR GRAPH AND FINDING ROC(TRIAINING RANDOM FOREST ON TESTING DATA) ##
> tntest2.pred.obj = prediction(tntest2$prob,tntest2$Personal.Loan)
> tntest2.perf = performance(tntest2.pred.obj,"tpr","fpr")
> plot(tntest2.perf)
```



4) KS-Value:

```
> ## KS VALUE (TRIAINING RANDOM FOREST ON TESTING DATA) ##
> print(max(tntest2.perf@y.values[[1]] - tntest2.perf@x.values[[1]]))
[1] 0.9312316
```

5) Gini:

```
> #### AUC (TRAINING RANDOM FOREST ON TESTING DATA) ####
> tntest2.auc = performance(tntest2.pred.obj,"auc")
> tntest2.auc = as.numeric(tntest2.auc@y.values)
> print(tntest2.auc)
[1] 0.995862
>
```

6) AUC:

```
> #### GINI (TRAINING RANDOM FOREST ON TESTING DATA) ####
> gini4 = ineq(tntest2$prob,"gini")
> print(gini4)
[1] 0.8797265
```

Inferences:

We can see that after building a **Random Forest Model**, we used it to make **predictions and also find the probabilities** on both the **Training** and **Testing data.** We also performed **Performance measures** on the **model** and **checked them** with both the **Training** and **Testing** data. The results are as follows:

Performance Measure	(Training model on Training Data)	Training model on Testing Data)
Concordance(More the better)	0.999	0.995
Error Rate(Lesser the better)	0.004	0.014
ROC(More the better)	0.99	0.97
KS-Value(More the better)	0.989	0.931
Gini(More the better)	0.999	0.995
AUC(More the better)	0.889	0.879

From the above results, we can see that results of **Training model on testing data** do not differentiate much from the results of **Training model on Training data.** And also we can see that all the values for

the **Performance measures** for **Training model on Testing data** to tell that the **Random Forest** hence created is a **Legible** and **a Very Good model** and can be further used for analysis.

From all the above measures, to compare the **accuracy** between the **CART Model** and the **Random Forest Model**, we need to compare the **measures of performance** when these models are **tested on the Testing data.**

Performance Measure	CART Model (Test Data)	Random Forest (Test Data)	Difference
Concordance(More the better)	0.879	0.995	**0.116**
Error Rate(Lesser the better)	0.033	0.014	**0.019**
ROC(More the better)	0.85	0.97	**0.12**
KS-Value(More the better)	0.857	0.931	**0.074**
Gini(More the better)	0.794	0.995	**0.201**
AUC(More the better)	0.936	0.879	**0.057**

As we can see from the above, **Random Forest Model** gave the better results than **CART Model** most of the time. Only **once** when **CART Model** gave a better result, the **difference** in the result was very **low.** From this, we can say that **Random Forest Model** is a much **better and more legible model** than the **CART Model.**

8) Project Conclusion:

As an Analyst, the analysis of the dataset was done with one **objective in mind,** to build a **Classification Predictive Model** that would help the **Marketing Team** get more **Positive Responses** from its customers. Since the team had very low response rate the previous time, they had collected the data from their previous campaign and asked me to build a **Classification Predictive Model** so that they may change their **Marketing Strategy** and **target the right customers.** As asked, we were able to create a **very good predictive model** using which one could make predictions on whether that customer would give a **positive response or negative response.** The **accuracy of the** model was determined using different **Performance Measures** and selected **the best model** after careful **comparison** and **analysis** with various models. In addition to that, **EDA (Exploratory Data Analysis)** was done on the dataset and changes were made wherever necessary. The process of **Clustering** of the data was also done where the customer base was divided into **3-4 Meaningful groups** helping the **Marketing Team** understand the types of customers in their bank. During the due process of this project, following things were inferred:

- We could see that **Average Income** of the customer in the bank was **64k per Month.** The Bank must try to attract more number of customers with higher income due to which will increase in **higher range** of **Assets** instead of **liabilities** for the bank
- While doing **EDA**, we found out that there were a lot of discrepancies in the data provided which included **Negative Values, Missing Values, wrong formats,** etc. While collecting the data next time, it must be made sure that no discrepancies are present as it may tamper with the **performance of the model.**
- We have seen that majority of the customers have **Undergraduate level** education. The customer base must have majority of

Graduate or **Professional** level so that the standard of the **Bank** increases and also they will help other customers educate and make them invest various schemes of the bank.

- **Credit Card** schemes seem to benefit only those who have a very less **Mortgages** or **no Mortgages**. The schemes must be changed to benefit the customers of all ranges
- The **ID Variable** and **Zip Code variable** can be omitted while collecting the data next time as these variables do not in any contribute in giving us a proper analysis of the dataset.

---

9) Appendix – A(Source Code)

```
########################## THERA BANK ANALYSIS ##############################

#### Importing the necessary libraries #####
install.packages("readxl")
library(readxl)
install.packages("ggplot2")
library(ggplot2)
install.packages("Hmisc")
library(Hmisc)
install.packages("rpart")
library(rpart)
install.packages("rpart.plot")
library(rpart.plot)
install.packages("randomForest")
library(randomForest)
install.packages("caTools")
library(caTools)
install.packages("psych")
library(psych)
install.packages("ROCR")
library(ROCR)
install.packages("InformationValue")
library(InformationValue)
install.packages("ineq")
library(ineq)
install.packages("memisc")
library(memisc)
install.packages("cluster")
library(cluster)
install.packages("NbClust")
library(NbClust)

 #### Changing the working directory #####
   setwd("C:/R programs great lakes/DATA MINING/prjct")
```

```
   getwd()
[1] "C:/R programs great lakes/DATA MINING/prjct"

   #### Importing the dataset "THERA BANK" dataset #####
   bank_data = read_xlsx("Bank_data.xlsx")
   bank_data
# A tibble: 5,000 x 14
      ID `Age (in years)` `Experience (in~ `Income (in K/m~ `ZIP Code` `Fam
ily members` CCAvg
   <dbl>            <dbl>            <dbl>            <dbl>      <dbl>
<dbl> <dbl>
 1     1               25                1               49      91107
4   1.6
 2     2               45               19               34      90089
3   1.5
 3     3               39               15               11      94720
1   1
 4     4               35                9              100      94112
1   2.7
 5     5               35                8               45      91330
4   1
 6     6               37               13               29      92121
4   0.4
 7     7               53               27               72      91711
2   1.5
 8     8               50               24               22      93943
1   0.3
 9     9               35               10               81      90089
3   0.6
10    10               34                9              180      93023
1   8.9
# ... with 4,990 more rows, and 7 more variables: Education <dbl>, Mortgage
<dbl>, `Personal
#   Loan` <dbl>, `Securities Account` <dbl>, `CD Account` <dbl>, Online <db
l>, CreditCard <dbl>

   #### Performing EDA on the dataset #####
   dim(bank_data)
[1] 5000   14
   str(bank_data)
Classes ‘tbl_df’, ‘tbl’ and 'data.frame':     5000 obs. of  14 variables:
 $ ID                    : num  1 2 3 4 5 6 7 8 9 10 ...
 $ Age (in years)        : num  25 45 39 35 35 37 53 50 35 34 ...
 $ Experience (in years): num  1 19 15 9 8 13 27 24 10 9 ...
 $ Income (in K/month)   : num  49 34 11 100 45 29 72 22 81 180 ...
 $ ZIP Code              : num  91107 90089 94720 94112 91330 ...
 $ Family members        : num  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             : num  1 1 1 2 2 2 2 3 2 3 ...
 $ Mortgage              : num  0 0 0 0 0 155 0 0 104 0 ...
 $ Personal Loan         : num  0 0 0 0 0 0 0 0 0 1 ...
 $ Securities Account    : num  1 1 0 0 0 0 0 0 0 0 ...
 $ CD Account            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ Online                : num  0 0 0 0 0 1 1 0 1 0 ...
 $ CreditCard            : num  0 0 0 0 1 0 0 1 0 0 ...
   head(bank_data)
# A tibble: 6 x 14
     ID `Age (in years)` `Experience (in~ `Income (in K/m~ `ZIP Code` `Fami
ly members` CCAvg
  <dbl>            <dbl>            <dbl>            <dbl>      <dbl>
<dbl> <dbl>
```

```
1     1               25                1               49      91107
4   1.6
2     2               45               19               34      90089
3   1.5
3     3               39               15               11      94720
1   1
4     4               35                9              100      94112
1   2.7
5     5               35                8               45      91330
4   1
6     6               37               13               29      92121
4   0.4
# ... with 7 more variables: Education <dbl>, Mortgage <dbl>, `Personal Loa
n` <dbl>, `Securities
#   Account` <dbl>, `CD Account` <dbl>, Online <dbl>, CreditCard <dbl>
   tail(bank_data)
# A tibble: 6 x 14
     ID `Age (in years)` `Experience (in~ `Income (in K/m~ `ZIP Code` `Fami
ly members` CCAvg
  <dbl>            <dbl>            <dbl>            <dbl>      <dbl>
<dbl> <dbl>
1  4995               64               40               75      94588
3   2
2  4996               29                3               40      92697
1   1.9
3  4997               30                4               15      92037
4   0.4
4  4998               63               39               24      93023
2   0.3
5  4999               65               40               49      90034
3   0.5
6  5000               28                4               83      92612
3   0.8
# ... with 7 more variables: Education <dbl>, Mortgage <dbl>, `Personal Loa
n` <dbl>, `Securities
#   Account` <dbl>, `CD Account` <dbl>, Online <dbl>, CreditCard <dbl>
   summary(bank_data)
       ID        Age (in years)  Experience (in years) Income (in K/month)
ZIP Code
 Min.   :   1   Min.   :23.00   Min.   :-3.0          Min.   :  8.00      M
in.   : 9307
 1st Qu.:1251   1st Qu.:35.00   1st Qu.:10.0          1st Qu.: 39.00      1
st Qu.:91911
 Median :2500   Median :45.00   Median :20.0          Median : 64.00      M
edian :93437
 Mean   :2500   Mean   :45.34   Mean   :20.1          Mean   : 73.77      M
ean   :93153
 3rd Qu.:3750   3rd Qu.:55.00   3rd Qu.:30.0          3rd Qu.: 98.00      3
rd Qu.:94608
 Max.   :5000   Max.   :67.00   Max.   :43.0          Max.   :224.00      M
ax.   :96651

 Family members      CCAvg          Education        Mortgage     Personal
Loan
 Min.   :1.000   Min.   : 0.000   Min.   :1.000   Min.   :  0.0   Min.   :0
.000
 1st Qu.:1.000   1st Qu.: 0.700   1st Qu.:1.000   1st Qu.:  0.0   1st Qu.:0
.000
 Median :2.000   Median : 1.500   Median :2.000   Median :  0.0   Median :0
.000
```

```
 Mean   :2.397   Mean   : 1.938   Mean   :1.881   Mean   : 56.5   Mean   :0
.096
 3rd Qu.:3.000   3rd Qu.: 2.500   3rd Qu.:3.000   3rd Qu.:101.0   3rd Qu.:0
.000
 Max.   :4.000   Max.   :10.000   Max.   :3.000   Max.   :635.0   Max.   :1
.000
 NA's   :18
 Securities Account   CD Account         Online         CreditCard
 Min.   :0.0000     Min.   :0.0000   Min.   :0.0000   Min.   :0.000
 1st Qu.:0.0000     1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.000
 Median :0.0000     Median :0.0000   Median :1.0000   Median :0.000
 Mean   :0.1044     Mean   :0.0604   Mean   :0.5968   Mean   :0.294
 3rd Qu.:0.0000     3rd Qu.:0.0000   3rd Qu.:1.0000   3rd Qu.:1.000
 Max.   :1.0000     Max.   :1.0000   Max.   :1.0000   Max.   :1.000

   colnames(bank_data)
 [1] "ID"                    "Age (in years)"        "Experience (in years)
" "Income (in K/month)"
 [5] "ZIP Code"              "Family members"        "CCAvg"
"Education"
 [9] "Mortgage"              "Personal Loan"         "Securities Account"
"CD Account"
[13] "Online"                "CreditCard"

   ## Converting the variables into proper Columnnames #####
   colnames(bank_data) = make.names(colnames(bank_data))
   colnames(bank_data)
 [1] "ID"                    "Age..in.years."        "Experience..in.years.
" "Income..in.K.month."
 [5] "ZIP.Code"              "Family.members"        "CCAvg"
"Education"
 [9] "Mortgage"              "Personal.Loan"         "Securities.Account"
"CD.Account"
[13] "Online"                "CreditCard"

   ## Changing the negative values in the Experience Column to zeros ####
   sum(bank_data$Experience..in.years. < 0)
[1] 52
   bank_data[bank_data < 0] = 0
   sum(bank_data$Experience..in.years. < 0)
[1] 0
   ## Fixing the Pincodes ####
   pincode = bank_data
   sum(pincode$ZIP.Code < 10000)
[1] 1
   which(pincode$ZIP.Code < 10000)
[1] 385
   which(colnames(pincode) == "ZIP.Code")
[1] 5
   pincode[385,5] = pincode[385,5] * 10
   pincode[385,5] = pincode[385,5] + 5
   pincode[385,5]
# A tibble: 1 x 1
  ZIP.Code
     <dbl>
1    93075
   sum(pincode$ZIP.Code < 10000)
[1] 0

   #### Uni - Variate Analysis #####
   ### Converting the requiered variables into Categorical variable ###
```

```
   bank_data$Personal.Loan = factor(bank_data$Personal.Loan,levels = c(1,0)
,labels = c("Yes","No"))
   bank_data$Education = factor(bank_data$Education,levels = c(1,2,3),label
s = c("Undergrad","Graduates","Proffesional"))
   bank_data$Securities.Account = factor(bank_data$Securities.Account,level
s = c(1,0),labels = c("Yes","No"))
   bank_data$CD.Account = factor(bank_data$CD.Account,levels = c(1,0),label
s = c("Yes","No"))
   bank_data$Online = factor(bank_data$Online,levels = c(1,0),labels = c("Y
es","No"))
   bank_data$CreditCard = factor(bank_data$CreditCard,levels = c(1,0),label
s = c("Yes","No"))
   bank data$Family.members = as.factor(bank data$Family.members)
   ## Working with the response variable #####
   print(table(bank_data$Personal.Loan))

 Yes   No
 480 4520
   print(prop.table(table(bank_data$Personal.Loan))*100)

 Yes   No
 9.6 90.4
   plot(bank_data$Personal.Loan,main = "Personal Loan")
   ## Working with other categorical variables ##
   print(table(bank_data$Family.members))

   1    2    3    4
1464 1292 1009 1217
   print(prop.table(table(bank_data$Family.members))*100)

       1        2        3        4
29.38579 25.93336 20.25291 24.42794
   plot(bank_data$Family.members,main = "Family Members")
   # Education #
   print(table(bank_data$Education))

   Undergrad    Graduates Proffesional
        2096         1403         1501
   print(prop.table(table(bank_data$Education))*100)

   Undergrad    Graduates Proffesional
       41.92        28.06        30.02
   plot(bank_data$Education,main = "Education")
   # Securities Account #
   print(table(bank_data$Securities.Account))

 Yes   No
 522 4478
   print(prop.table(table(bank_data$Securities.Account))*100)

  Yes    No
10.44 89.56
   plot(bank_data$Securities.Account,main = "Securities Account")
   # Certificate of Deposit #
   print(table(bank_data$CD.Account))

 Yes   No
 302 4698
   print(prop.table(table(bank_data$CD.Account))*100)

  Yes    No
```

```
 6.04 93.96
   plot(bank_data$CD.Account,main = "Certificate of Deposit")
   # Internet Banking #
   print(table(bank_data$Online))

 Yes   No
2984 2016
   print(prop.table(table(bank_data$Online))*100)

  Yes    No
59.68 40.32
   plot(bank_data$Online,main = "Internet Banking")
   # Credit Card issued by the bank #
   print(table(bank_data$CreditCard))

 Yes   No
1470 3530
   print(prop.table(table(bank_data$CreditCard))*100)

 Yes   No
29.4 70.6
   plot(bank_data$CD.Account,main = "Credit issued by the bank")
   # Plotting all the numerical variables #
   hist.data.frame(bank_data[,c(2,3,4,7,9)])
   plot(bank_data[,c(2,3,4,7,9)])


   ##### Bi-Variate Analysis #####
   ### Response variable with Categorical variable ###
   table(bank_data$Personal.Loan,bank_data$Family.members)

        1    2    3    4
  Yes  106  106  133  133
  No  1358 1186  876 1084
   qplot(Personal.Loan,fill = Family.members,data = bank_data)
   table(bank_data$Personal.Loan,bank_data$CreditCard)

      Yes   No
  Yes  143  337
  No  1327 3193
   qplot(Personal.Loan,fill = CreditCard,data = bank_data)

   ### Independent variables with Independent variables ###
   qplot(Family.members,fill = Education,data = bank_data)
   qplot(CreditCard,fill = Online,data = bank_data)

   ### Dependent variables with Numerical variables ###
   qplot(Income..in.K.month.,fill = Personal.Loan,data = bank_data)
   qplot(Mortgage,fill = Personal.Loan,data = bank_data,geom = "dotplot")
   qplot(CCAvg,fill = Personal.Loan,data = bank_data,geom = "density")

   ### Other Dependent variables with the Numerical Variables ###
   qplot(Income..in.K.month.,fill = CreditCard,data = bank_data)
   qplot(Mortgage,fill = CreditCard,data = bank_data)

   ## Identifying and Changing the NA's from the family Column #####
   bank_data$Family.members = as.numeric(bank_data$Family.members)
   sum(is.na(bank_data))
[1] 18
   bank_data[is.na(bank_data)] = 0
   sum(is.na(bank_data))
```

```
[1] 0
   bank_data$Family.members = as.factor(bank_data$Family.members)


   #### Finding the Outliers ####
   boxplot(bank_data[,c(2,3,4,7,9)])

   #### Removing the ID variable and ZIP.CODES variable ####
   bank_data = bank_data[,-c(1,5)]
   colnames(bank_data)
 [1] "Age..in.years."        "Experience..in.years." "Income..in.K.month."
"Family.members"
 [5] "CCAvg"                 "Education"             "Mortgage"
"Personal.Loan"
 [9] "Securities.Account"    "CD.Account"            "Online"
"CreditCard"
   #### Clustering Analysis ####
   ### Hierarchial Clustering ###
   scale_bank_data = scale(bank_data[,c(3,5,7)])
   ## Taking the distance matrix ##
   dist_bank_data = dist(scale_bank_data)
   ## Creating Hierarchial Clusters ####
   hclust_bank_data = hclust(dist_bank_data,method = "complete")
   plot(hclust_bank_data)
   rect.hclust(hclust_bank_data,k = 3)
   ### Clustering the dataset using the Cluster object created ###
   bank_data_hclust = bank_data[,c(3,5,7)]
   bank_data_hclust$cluster = cutree(hclust_bank_data,k = 3)
   bank_data_hclust = aggregate(bank_data_hclust,list(bank_data_hclust$clus
ter),FUN = "mean")
   print(bank_data_hclust)
  Group.1 Income..in.K.month.    CCAvg  Mortgage cluster
1       1            63.71496 1.530697  39.64456       1
2       2           161.53242 6.519249  25.55290       2
3       3           138.32069 3.511931 344.47241       3

   ### K-Means Clustering  ###
   #### Randomly Clustering with the infered number of clusters ####
   set.seed(77)
   library(cluster)
   k.cluster = kmeans(dist_bank_data,centers = 3,nstart = 5)
   clusplot(bank_data,k.cluster$cluster,lines = 1,color = TRUE,shade = TRUE
)
   ### Trying to find the right number of Clusters by calculating variance
###
   set.seed(77)
   k.values = c(1:10)
   kcluster2 = kmeans(dist_bank_data,centers = 9,nstart = 13)

   plot(k.values, kcluster2$withinss,
+       type="b", pch = 19, frame = FALSE,
+       xlab="Number of clusters K",
+       ylab="Total within-clusters sum of squares")

   ### K means Clustering using the number of clusters obtained after the e
xperiment ###
   set.seed(77)
   nc1 = NbClust(bank_data[,c(3,5,7)],min.nc = 2,max.nc = 5,method = "kmean
s")

   ### Perfroming the K means Clustering with number of clusters as 4 ####
```

```
   set.seed(77)
   kcluster = kmeans(dist_bank_data,centers = 4,nstart = 8)
   clusplot(bank_data,kcluster$cluster,lines = 1,color = TRUE,shade = TRUE)
   ### Using the clustering object on the dataset ####
   bank_data_kclust = bank_data[,c(3,5,7)]
   bank_data_kclust$Cluster = kcluster$cluster
   bank_data_kclust = aggregate(bank_data_kclust,list(bank_data_kclust$Clus
ter),FUN = mean)
```

```
   print(bank data kclust)
  Group.1 Income..in.K.month.    CCAvg Mortgage Cluster
1       1            97.94524 2.401714 249.6119       1
2       2            73.80231 1.948486   0.0000       2
3       3            53.54116 1.481486 123.9839       3
4       4           153.63710 3.739113 436.8468       4
```

```
   #### CREATING DECISION TREES #####
   ### Splitting the dataset into training and testing dataset ####
   ####(Industry standard splitting ratio of 0.7) ####
   set.seed(77)
   split <- sample.split(bank_data$Personal.Loan, SplitRatio = 0.7)
   train<- subset(bank_data, split == TRUE)
   test<- subset(bank_data, split == FALSE)
   dim(train)
[1] 3500   12
   dim(test)
[1] 1500   12
```

```
   ####CART MODEL ####
   ### For the process of creating a CART Basesd Model on train data and us
ing it on train data, ###
   ### we create a dataframe named  "tntrain1" similar to the original trai
n dataset ###
   tntrain1 = train
   tntrain1$Personal.Loan = factor(tntrain1$Personal.Loan,levels = c("Yes",
"No"),labels = c(1,0))
   ### Creating a CART model based on train data ###
   set.seed(77)
   rttree = rpart(tntrain1$Personal.Loan~.,data = tntrain1,minsplit = 10,mi
nbucket = 10)
   print(rttree)
n= 3500

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 3500 336 0 (0.09600000 0.90400000)
   2) Income..in.K.month.>=106.5 779 294 0 (0.37740693 0.62259307)
     4) Education=Graduates,Proffesional 276  31 1 (0.88768116 0.11231884)
       8) Income..in.K.month.>=116.5 219   0 1 (1.00000000 0.00000000) *
       9) Income..in.K.month.< 116.5 57  26 0 (0.45614035 0.54385965)
        18) CCAvg>=3.535 12   0 1 (1.00000000 0.00000000) *
        19) CCAvg< 3.535 45  14 0 (0.31111111 0.68888889) *
     5) Education=Undergrad 503  49 0 (0.09741551 0.90258449)
      10) Family.members=3,4 55   6 1 (0.89090909 0.10909091) *
      11) Family.members=1,2 448   0 0 (0.00000000 1.00000000) *
   3) Income..in.K.month.< 106.5 2721  42 0 (0.01543550 0.98456450)
     6) CCAvg>=2.95 193  42 0 (0.21761658 0.78238342)
```

```
      12) Income..in.K.month.>=88.5 69  31 0 (0.44927536 0.55072464)
        24) Family.members=3,4 20   4 1 (0.80000000 0.20000000) *
        25) Family.members=1,2 49  15 0 (0.30612245 0.69387755) *
      13) Income..in.K.month.< 88.5 124  11 0 (0.08870968 0.91129032) *
     7) CCAvg< 2.95 2528   0 0 (0.00000000 1.00000000) *
   rpart.plot(rttree)
   ### Since we can see that the tree is complex, we need find the decide t
he right Complexity Parameter ###
   ### to prune the tree in the right way ###
   printcp(rttree)

Classification tree:
rpart(formula = tntrain1$Personal.Loan ~ ., data = tntrain1,
    minsplit = 10, minbucket = 10)

Variables actually used in tree construction:
[1] CCAvg               Education           Family.members      Income..in.
K.month.

Root node error: 336/3500 = 0.096

n= 3500

        CP nsplit rel error  xerror     xstd
1 0.318452      0   1.00000 1.00000 0.051870
2 0.127976      2   0.36310 0.38095 0.033050
3 0.025298      3   0.23512 0.26786 0.027869
4 0.011905      5   0.18452 0.20833 0.024650
5 0.010000      8   0.14881 0.21131 0.024822
   plotcp(rttree)
   ### From the CP graph and the CP table, we can see that, the right CP is
0.025298 #####
   ### Pruning the tree using the CP, 0.025298 ####
   prttree = prttree = prune(rttree,cp = 0.025298)
   print(prttree)
n= 3500

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 3500 336 0 (0.09600000 0.90400000)
   2) Income..in.K.month.>=106.5 779 294 0 (0.37740693 0.62259307)
     4) Education=Graduates,Proffesional 276  31 1 (0.88768116 0.11231884)
*
     5) Education=Undergrad 503  49 0 (0.09741551 0.90258449)
      10) Family.members=3,4 55   6 1 (0.89090909 0.10909091) *
      11) Family.members=1,2 448   0 0 (0.00000000 1.00000000) *
   3) Income..in.K.month.< 106.5 2721  42 0 (0.01543550 0.98456450) *
   rpart.plot(prttree)
   ### Using the model to make predictions on the tntrain1 dataset ####
   tntrain1$predict = predict(prttree,tntrain1,type = "class")
   tntrain1$prob = predict(prttree,tntrain1,type = "prob")[,1]
   View((head(tntrain1)))

   ##### Testing the Performance of the model on the tntrain1 dataset ####
   ## CONCORDANCE RATIO (TRAINING MODEL ON TRAINING DATA) ##
   x = tntrain1$Personal.Loan
   y = tntrain1$prob
   Concordance(actuals = x,predictedScores = y)
$Concordance
[1] 0.8838956
```

```
$Discordance
[1] 0.1161044

$Tied
[1] 5.551115e-17

$Pairs
[1] 1063104

   ## CONFUSION MATRIX ERROR RATE (TRAINING MODEL ON TRAINING DATA) ##
   cnf_tntrain1 = table(tntrain1$Personal.Loan,tntrain1$predict)
   errrate tntrain1 = (cnf tntrain1[2,1]+cnf tntrain1[1,2])/nrow(tntrain1)
   print(errrate_tntrain1)
[1] 0.02257143
   ## PLOTTING THE TPR AND FPR GRAPH AND FINDING ROC ###
   ##(TRAINING MODEL ON TRAINING DATA) ##
   tntrain1.pred.obj = prediction(tntrain1$prob,tntrain1$Personal.Loan)
   tntrain1.perf = performance(tntrain1.pred.obj,"tpr","fpr")
   plot(tntrain1.perf)
   ### KS VALUE ###
   print(max(tntrain1.perf@y.values[[1]] - tntrain1.perf@x.values[[1]]))
[1] 0.8633059
   #### AUC (TRAINING MODEL ON TRAINING DATA) ####
   tntrain1.auc = performance(tntrain1.pred.obj,"auc")
   tntrain1.auc = as.numeric(tntrain1.auc@y.values)
   print(tntrain1.auc)
[1] 0.9405256
   #### GINI COEFFICIENT (TRAINING MODEL ON TRAINING DATA) ####
   gini1 = ineq(tntrain1$prob,"gini")
   print(gini1)
[1] 0.7964702

   ### For the process of testing the CART Based Model on train data and us
ing it on test data, ###
   ### we create a dataframe named  "tntest1" similar to the original test
dataset ###
   tntest1 = test
   tntest1$Personal.Loan = factor(tntest1$Personal.Loan,levels = c("Yes","N
o"),labels = c(1,0))
   ### Using the training data CART model to make predicitions on the test
data ###
   tntest1$predict = predict(prttree,tntest1,type = "class")
   tntest1$prob = predict(prttree,tntest1,type = "prob")[,1]
   View((head(tntest1)))
   ## CONCORDANCE RATIO (TRAINING MODEL ON TESTING DATA) ##
   a = tntest1$Personal.Loan
   b = tntest1$prob
   Concordance(actuals = a,predictedScores = b)
$Concordance
[1] 0.8793019

$Discordance
[1] 0.1206981

$Tied
[1] 4.163336e-17

$Pairs
[1] 195264
```

```
   ## CONFUSION MATRIX ERROR RATE (TRAINING MODEL ON TESTING DATA) ##
   cnf_tntest1 = table(tntest1$Personal.Loan,tntest1$predict)
   errrate_tntest1 = (cnf_tntest1[2,1]+cnf_tntest1[1,2])/nrow(tntest1)
   print(errrate_tntest1)
[1] 0.03333333
   ## PLOTTING THE TPR AND FPR GRAPH TO FIND ROC (TRAINING MODEL ON TESTING
DATA) ##
   tntest1.pred.obj = prediction(tntest1$prob,tntest1$Personal.Loan)
   tntest1.perf = performance(tntest1.pred.obj,"tpr","fpr")
   plot(tntest1.perf)
   #### KS VALUE (TRAINING MODEL ON TESTING DATA) ####
   print(max(tntest1.perf@y.values[[1]] - tntest1.perf@x.values[[1]]))
[1] 0.8576082
   #### AUC (TRAINING MODEL ON TESTING DATA) ####
   tntest1.auc = performance(tntest1.pred.obj,"auc")
   tntest1.auc = as.numeric(tntest1.auc@y.values)
   print(tntest1.auc)
[1] 0.9369085
   #### GINI INDEX (TRAINING MODEL ON TESTING DATA)####
   gini2 = ineq(tntest1$prob,"gini")
   print(gini2)
[1] 0.7941752
   ### Creating a Random Forest model on the train data ###
   #### Building a random forest using the random number of trees for the r
andom forest ####
   set.seed(77)
   tntrain2.rf = randomForest(tntrain2$Personal.Loan~.,data = tntrain2,
+                              ntree =501,mtry = 3,nodesize = 10,importance =
TRUE)
   plot(tntrain2.rf)
   print(tntrain2.rf)

Call:
 randomForest(formula = tntrain2$Personal.Loan ~ ., data = tntrain2,      n
tree = 501, mtry = 3, nodesize = 10, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 501
No. of variables tried at each split: 3

        OOB estimate of  error rate: 1.54%
Confusion matrix:
    1    0 class.error
1 295   41 0.122023810
0  13 3151 0.004108723
   error = tntrain2.rf$err.rate
   error = as.data.frame(error)
   error$ntree = c(1:nrow(error))
   s.error = error[order(error$OOB,error$ntree),]
   s.error$ntree[1]
[1] 57
   ### From the above, we can determine that the right number of trees is 5
7 ####
   ### Finidng the right number of mtry ###
   set.seed(77)
   tuned.tntrain2.rf = tuneRF(x = tntrain2[,-c(8)],y = tntrain2$Personal.Lo
an,mtryStart = 3,stepFactor=1.5,
+                              nodesize = 10,ntreeTry = 57,importance = TRUE,
+                              improve = 0.01,trace = TRUE,plot = TRUE,doBest
= TRUE)
mtry = 3  OOB error = 1.37%
Searching left ...
```

```
mtry = 2       OOB error = 1.74%
-0.2708333 0.01
Searching right ...
mtry = 4       OOB error = 1.43%
-0.04166667 0.01
   print(tuned.tntrain2.rf)

Call:
 randomForest(x = x, y = y, mtry = res[which.min(res[, 2]), 1],      nodesi
ze = 10, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 1.51%
Confusion matrix:
    1    0 class.error
1 295   41 0.122023810
0  12 3152 0.003792668
   ### Using the tuned Random Forest on train data to make predictions ####
   tntrain2$predict = predict(tntrain2.rf,tntrain2,type = "class")
   tntrain2$prob = predict(tntrain2.rf,tntrain2,type = "prob")[,"1"]
   View(tntrain2)
   ### Testing the Random Forest on train data ####
   ## CONCORDANCE RATIO (TRAINING RANDOM FOREST ON TRAINING DATA) ##
   c = tntrain2$Personal.Loan
   d = tntrain2$prob
   Concordance(actuals = c,predictedScores = d)
$Concordance
[1] 0.999826

$Discordance
[1] 0.0001740187

$Tied
[1] 4.786753e-17

$Pairs
[1] 1063104

   ## CONFUSION MATRIX ERROR RATE (TRAINING RANDOM FOREST ON TRAINING DATA)
##
   cnf_tntrain2 = table(tntrain2$Personal.Loan,tntrain2$predict)
   errrate_tntrain2 = (cnf_tntrain2[2,1]+cnf_tntrain2[1,2])/nrow(tntrain2)
   print(errrate_tntrain2)
[1] 0.004571429
   ## PLOTTING THE TPR AND FPR GRAPH TO FIND ROC (TRAINING RANDOM FOREST ON
TRAINING DATA) ##
   tntrain2.pred.obj = prediction(tntrain2$prob,tntrain2$Personal.Loan)
   tntrain2.perf = performance(tntrain2.pred.obj,"tpr","fpr")
   plot(tntrain2.perf)
   ## KS VALUE (TRAINING RANDOM FOREST ON TRAINING DATA) ##
   print(max(tntrain2.perf@y.values[[1]] - tntrain2.perf@x.values[[1]]))
[1] 0.9891224
   #### AUC (TRAINING RANDOM FOREST ON TRAINING DATA) ####
   tntrain2.auc = performance(tntrain2.pred.obj,"auc")
   tntrain2.auc = as.numeric(tntrain2.auc@y.values)
   print(tntrain2.auc)
[1] 0.9998283
   #### Gini Coefficient (TRAINING RANDOM FOREST ON TRAINING DATA) ####
   gini3 = ineq(tntrain2$prob,"gini")
```

```
   print(gini3)
[1] 0.8898675
   ### For the process of testing the Random Forest Model on test data and
using it on test data, ###
   ### we create a dataframe named  "tntest2" similar to the original test
dataset ###
   tntest2 = test
   tntest2$Personal.Loan = factor(tntest2$Personal.Loan,levels = c("Yes","N
o"),labels = c(1,0))
   ### Using the tuned Random Forest on test data to make predictions ####
   tntest2$predict = predict(tntrain2.rf,tntest2,type = "class")
   tntest2$prob = predict(tntrain2.rf,tntest2,type = "prob")[,"1"]
   View(tntest2)
   ### Testing the Random Forest on test data ####
   ## CONCORDANCE RATIO (TRAINING RANDOM FOREST ON TRAINING DATA) ##
   c = tntest2$Personal.Loan
   d = tntest2$prob
   Concordance(actuals = c,predictedScores = d)
$Concordance
[1] 0.9958466

$Discordance
[1] 0.004153351

$Tied
[1] 1.12757e-17

$Pairs
[1] 195264

   ## CONFUSION MATRIX ERROR RATE (TRAINING RANDOM FOREST ON TESTING DATA)
##
   cnf_tntest2 = table(tntest2$Personal.Loan,tntest2$predict)
   errrate_tntest2 = (cnf_tntest2[2,1]+cnf_tntest2[1,2])/nrow(tntest2)
   print(errrate_tntest2)
[1] 0.01466667
   ## PLOTTING THE TPR AND FPR GRAPH AND FINDING ROC(TRIAINING RANDOM FORES
T ON TESTING DATA) ##
   tntest2.pred.obj = prediction(tntest2$prob,tntest2$Personal.Loan)
   tntest2.perf = performance(tntest2.pred.obj,"tpr","fpr")
   plot(tntest2.perf)
   ## KS VALUE (TRIAINING RANDOM FOREST ON TESTING DATA) ##
   print(max(tntest2.perf@y.values[[1]] - tntest2.perf@x.values[[1]]))
[1] 0.9312316
   #### AUC (TRAINING RANDOM FOREST ON TESTING DATA) ####
   tntest2.auc = performance(tntest2.pred.obj,"auc")
   tntest2.auc = as.numeric(tntest2.auc@y.values)
   print(tntest2.auc)
[1] 0.995862
   #### GINI (TRAINING RANDOM FOREST ON TESTING DATA) ####
   gini4 = ineq(tntest2$prob,"gini")
   print(gini4)
[1] 0.8797265
```