Thera Bank Case Study

Benedict Egwuchukwu

8/22/2020

Contents

1. Project Objective

This case is about a bank (Thera Bank) which has a growing customer base. Majority of these customers are liability customers (depositors) with varying size of deposits. The number of customers who are also borrowers (asset customers) is quite small, and the bank is interested in expanding this base rapidly to bring in more loan business and in the process, earn more through the interest on loans. In particular, the management wants to explore ways of converting its liability customers to personal loan customers (while retaining them as depositors). A campaign that the bank ran last year for liability customers showed a healthy conversion rate of over 9% success. This has encouraged the retail marketing department to devise campaigns with better target marketing to increase the success ratio with a minimal budget. The department wants to build a model that will help them identify the potential customers who have a higher probability of purchasing the loan. This will increase the success ratio while at the same time reduce the cost of the campaign. The dataset has data on 5000 customers. The data include customer demographic information (age, income, etc.), the customer's relationship with the bank (mortgage, securities account, etc.), and the customer response to the last personal loan campaign (Personal Loan). Among these 5000 customers, only 480 (= 9.6%) accepted the personal loan that was offered to them in the earlier campaign.

As the consultant, I have a duty to build the best model which can classify the right customers who have a higher probability of purchasing the loan. In this project, I will be doing the following:

- EDA of the data available. Showcase the results using appropriate graphs
- Apply appropriate clustering on the data and interpret the output (Thera Bank wants to understand what kind of customers exist in their database and hence we need to do customer segmentation)
- Build appropriate models on both the test and train data (CART & Random Forest). Interpret all the model outputs and do the necessary modifications wherever eligible (such as pruning)
- Check the performance of all the models that you have built (test and train). Use all the model performance measures you have learned so far. Share your remarks on which model performs the best.

2. Assumptions

The Random Forest model should be able to accurately classify the right customers who have a higher probability of puraching the loan for Thera Bank compared to the CART model.

3. Exploratory Data Analysis (EDA) - Step by step approach

3.1 Environment Set up and Data Import

```
# Environment set up and data import

# Invoking libraries
library(readxl) # To import excel files
library(ggplot2) # To create plots
library(corrplot) # To plot correlation plot between numerical variables
library(dplyr) # To manipulate dataset
library(gridExtra) # To plot multiple ggplot graphs in a grid
library(DataExplorer) # visual exploration of data
library(mice) # Multivariate Imputation via Chained Equations; takes care of uncertainty in missing val
library(cluster)
library(factoextra) # extract and visualize the results of multivariate data analysis
library(NbClust) # to find optimal number of clusters
library(caTools) # Split Data into Test and Train Set
library(rpart) # To build CART decision tree
library(rattle) # To visualise decision tree
library(randomForest) # To build a Random Forest
library(ROCR) # To visualise the performance classifiers
library(ineq) # To calculate Gini
library(InformationValue) # For Concordance-Discordance
library(knitr) # Necessary to generate source codes from a .Rmd File
library(markdown) # To convert to HTML
library(rmarkdown) # To convret analyses into high quality documents
```

3.1.1 Install necessary packages and load libraries

```
# Set working directory
setwd("C:/Users/egwuc/Desktop/PGP-DSBA-UT Austin/Machine Learning/Week 5 - Project/")
```

3.1.2 Set up Working Directory

```
# Read input file
thera_bank <- read_excel("Thera Bank_Personal_Loan_Modelling-dataset-1.xlsx", sheet = 2)
```

3.1.3 Import and Read the Dataset

```
# Global options settings
options(scipen = 999) # turn off scientific notation like 1e+06
```

3.1.4 Global Options Settings

3.2 Variable Identification

In order for us to get familiar with the Cardio Good Fitness data, we would be using the following functions to get an overview

1. dim(): this gives us the dimension of the dataset provided. Knowing the data dimension gives us an idea of how large the data is. 2. head(): this shows the first 6 rows(observations) of the dataset. It is essential for us to get a glimpse of the dataset in a tabular format without revealing the entire dataset if we are to properly analyse the data.
2. tail(): this shows the last 6 rows(observations) of the dataset. Knowing what the dataset looks like at the end rows also helps us ensure the data is consistent.
3. str(): this shows us the structure of the dataset. It helps us determine the datatypes of the features and identify if there are datatype mismatches, so that we handle these ASAP to avoid inappropriate results from our analysis.
4. summary(): this provides statistical summaries of the dataset. This function is important as we can quickly get statistical summaries (mean,median, quartiles, min, frequencies/counts, max values etc.) which can help us derive insights even before diving deep into the data.
5. View(): helps to look at the entire dataset at a glance.

```
# Check dimension of dataset
dim(thera_bank)
```

3.2.1 Insight(s) from dim():

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

- The dataset has 5,000 rows and 14 columns.

```
# Check first 6 rows(observations) of dataset
head(thera_bank)
```

3.2.2 Insight(s) from head() and tail():

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

```
tail(thera_bank)
```

```
## # A tibble: 6 x 14
##      ID `Age (in years)` `Experience (in~ `Income (in K/m~ `ZIP Code`
##   <dbl>            <dbl>             <dbl>            <dbl>      <dbl>
## 1  4995               64                40               75      94588
## 2  4996               29                 3               40      92697
## 3  4997               30                 4               15      92037
## 4  4998               63                39               24      93023
## 5  4999               65                40               49      90034
```

```
## 6  5000               28                4               83      92612
## # ... with 9 more variables: `Family members` <dbl>, CCAvg <dbl>,
## #   Education <dbl>, Mortgage <dbl>, `Personal Loan` <dbl>, `Securities
## #   Account` <dbl>, `CD Account` <dbl>, Online <dbl>, CreditCard <dbl>
```

- Values in all fields are consistent in each column.

```
# Convert column names to appropriate column names
colnames(thera_bank) <- c("ID", "Age", "Experience", "Income", "ZIP_Code", "Family_members", "CCAvg", "
                          "Personal_Loan", "Securities_Account", "CD_Account", "Online", "CreditCard")
```

3.2.3 Convert column names

- Changed column names to appropriate names recognizable by rstudio.

```
# Check structure of dataset
str(thera_bank)
```

3.2.4 Insight(s) from str():

```
## tibble [5,000 x 14] (S3: tbl_df/tbl/data.frame)
##  $ ID                : num [1:5000] 1 2 3 4 5 6 7 8 9 10 ...
##  $ Age               : num [1:5000] 25 45 39 35 35 37 53 50 35 34 ...
##  $ Experience        : num [1:5000] 1 19 15 9 8 13 27 24 10 9 ...
##  $ Income            : num [1:5000] 49 34 11 100 45 29 72 22 81 180 ...
##  $ ZIP_Code          : num [1:5000] 91107 90089 94720 94112 91330 ...
##  $ Family_members    : num [1:5000] 4 3 1 1 4 4 2 1 3 1 ...
##  $ CCAvg             : num [1:5000] 1.6 1.5 1 2.7 1 0.4 1.5 0.3 0.6 8.9 ...
##  $ Education         : num [1:5000] 1 1 1 2 2 2 2 3 2 3 ...
##  $ Mortgage          : num [1:5000] 0 0 0 0 0 155 0 0 104 0 ...
##  $ Personal_Loan     : num [1:5000] 0 0 0 0 0 0 0 0 0 1 ...
##  $ Securities_Account: num [1:5000] 1 1 0 0 0 0 0 0 0 0 ...
##  $ CD_Account        : num [1:5000] 0 0 0 0 0 0 0 0 0 0 ...
##  $ Online            : num [1:5000] 0 0 0 0 0 1 1 0 1 0 ...
##  $ CreditCard        : num [1:5000] 0 0 0 0 1 0 0 1 0 0 ...
```

- Customer ID is an ID variable and not useful for predictive modelling.
- Zip code is not useful as well for predictive modelling.
- Personal loan is the response variable and is of numeric type. This should be a factor type.
- Education, securities account, CD account, online and credit card should all ideally be ordered factor variables. However, they are numeric variables.
- All the other vriables are numeric as they should be.

```
# Change personal_loan to factor variable
thera_bank$Personal_Loan <- as.factor(thera_bank$Personal_Loan)
```

3.2.5 Convert to factor variable

- Personal_loan is a factor variable and provides more meaning to the dataset.

```
# Get summary of dataset
summary(thera_bank)
```

3.2.6 Insight(s) from summary():

```
##        ID             Age           Experience       Income          ZIP_Code
##  Min.   :   1   Min.   :23.00   Min.   :-3.0   Min.   :  8.00   Min.   : 9307
##  1st Qu.:1251   1st Qu.:35.00   1st Qu.:10.0   1st Qu.: 39.00   1st Qu.:91911
##  Median :2500   Median :45.00   Median :20.0   Median : 64.00   Median :93437
##  Mean   :2500   Mean   :45.34   Mean   :20.1   Mean   : 73.77   Mean   :93153
##  3rd Qu.:3750   3rd Qu.:55.00   3rd Qu.:30.0   3rd Qu.: 98.00   3rd Qu.:94608
##  Max.   :5000   Max.   :67.00   Max.   :43.0   Max.   :224.00   Max.   :96651
##
##  Family_members      CCAvg          Education        Mortgage     Personal_Loan
##  Min.   :1.000   Min.   : 0.000   Min.   :1.000   Min.   :  0.0   0:4520
##  1st Qu.:1.000   1st Qu.: 0.700   1st Qu.:1.000   1st Qu.:  0.0   1: 480
##  Median :2.000   Median : 1.500   Median :2.000   Median :  0.0
##  Mean   :2.397   Mean   : 1.938   Mean   :1.881   Mean   : 56.5
##  3rd Qu.:3.000   3rd Qu.: 2.500   3rd Qu.:3.000   3rd Qu.:101.0
##  Max.   :4.000   Max.   :10.000   Max.   :3.000   Max.   :635.0
##  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
##
```

- Customer ID and zip code should be dropped.
- The age variable ranges from 23 to 67 with a mean and median of 45.34 and 45.00 respectively.
- The experience varible ranges from -3.0 to 43.0 years. Some customers have less than zero years of experience, bringing to question its possibility. Could this be an error?
- The income variable is expressed in $'000 and reflects annual income. It ranges from 8,000 to 224,000.
- The family members varaible is categorized from 1 through 4. The data contains 18 NA's that need to be treated.
- The CCAvg variable represents the average spending on credit cards per month, expressed in $'000. Ranges from 0 to 10,000.
- The education variable represents the education level of customers and is categorized as 1: Undergrad; 2: Graduate; 3: Advanced/Professional. The bank's customer base has more undergraduate students.
- The mortgage variable represents customer's value of house mortgage if any and is expressed in $'000. Ranges from 0 to 635,000.
- The personal loan varible represents customer response to the last personal loan campaign. Of the 5,000 customers, only 480 (9.6%) accepted the personal loan that was offered in the earlier campaign.
- Out of 5,000 customers, only 522 (10.4%) have securities account.
- Out of 5,000 customers, only 302 (6.0%) have certificate of deposit (CD) account.
- Out of 5,000 customers, 2,984 (59.7%) use the internet banking facilities.
- Out of 5,000 customers, only 1,470 (29.4%) use a credit card issued by the bank.

```
# Dropping ID and Zip Code column
thera_bank <- thera_bank[, -1]
thera_bank <- thera_bank[, -4]
```

3.2.7 Drop insignificant columns

- Removed the ID and Zip code column as it does not affect the analysis.

```r
# View the dataset
View(thera_bank)
```

3.2.8 Insight(s) from View():

- The dataset shows Thera Bank's growing customer base and gives detail of their customers.

3.3 Data Cleaning

```r
# Filter out values less than 0 in Experience
filter(thera_bank, Experience < 0)
```

3.3.1 Modifying Values

```
## # A tibble: 52 x 12
##      Age Experience Income Family_members CCAvg Education Mortgage Personal_Loan
##    <dbl>      <dbl>  <dbl>          <dbl> <dbl>     <dbl>    <dbl> <fct>
##  1    25         -1    113              4  2.3          3        0 0
##  2    24         -1     39              2  1.7          2        0 0
##  3    24         -2     51              3  0.3          3        0 0
##  4    28         -2     48              2  1.75         3       89 0
##  5    24         -1     75              4  0.2          1        0 0
##  6    25         -1     43              3  2.4          2      176 0
##  7    25         -1    109              4  2.3          3      314 0
##  8    25         -1     48              3  0.3          3        0 0
##  9    24         -1     38              2  1.7          2        0 0
## 10    24         -2    125              2  7.2          1        0 0
## # ... with 42 more rows, and 4 more variables: Securities_Account <dbl>,
## #   CD_Account <dbl>, Online <dbl>, CreditCard <dbl>
```

```r
# Replace values less than 0 in Experience with 0
thera_bank$Experience <- replace(thera_bank$Experience, thera_bank$Experience<0, 0)
```

```r
# Check if any values in less than 0 in Experience
filter(thera_bank, Experience < 0)
```

```
## # A tibble: 0 x 12
## # ... with 12 variables: Age <dbl>, Experience <dbl>, Income <dbl>,
## #   Family_members <dbl>, CCAvg <dbl>, Education <dbl>, Mortgage <dbl>,
## #   Personal_Loan <fct>, Securities_Account <dbl>, CD_Account <dbl>,
## #   Online <dbl>, CreditCard <dbl>
```

```r
# How many missing vaues do we have?
sum(is.na(thera_bank))
```

3.3.2 Missing Data Treatmemt

```
## [1] 18
```

```r
# What columns contain missing values?
colSums(is.na(thera_bank))
```

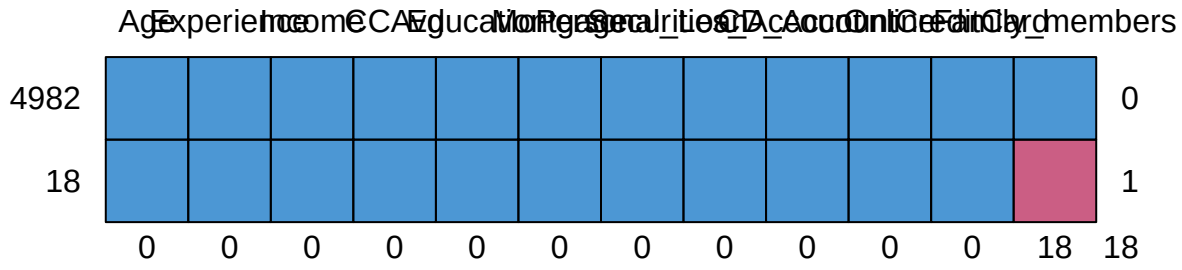
```
##                Age         Experience             Income     Family_members
##                  0                  0                  0                 18
##              CCAvg          Education           Mortgage      Personal_Loan
```

```
##                  0                  0                  0                  0
## Securities_Account         CD_Account             Online         CreditCard
##                  0                  0                  0                  0
```

```
# Use functions and algorithms to impute the missing values
data1 <- thera_bank
sum(is.na(data1))
```

```
## [1] 18
```

```
md.pattern(data1)
```




```
##      Age Experience Income CCAvg Education Mortgage Personal_Loan
## 4982   1          1      1     1         1        1             1
## 18     1          1      1     1         1        1             1
##        0          0      0     0         0        0             0
##      Securities_Account CD_Account Online CreditCard Family_members
## 4982                  1          1      1          1              1  0
## 18                    1          1      1          1              0  1
##                       0          0      0          0             18 18
```

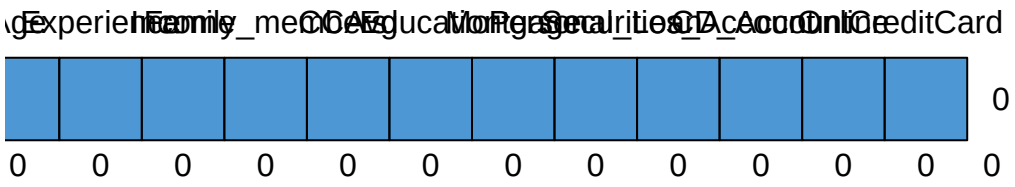
```
init.impute = mice(data1, m = 5, method = "pmm", seed = 1000)
```

```
##
##  iter imp variable
##   1   1  Family_members
##   1   2  Family_members
##   1   3  Family_members
##   1   4  Family_members
```

```
##   1   5  Family_members
##   2   1  Family_members
##   2   2  Family_members
##   2   3  Family_members
##   2   4  Family_members
##   2   5  Family_members
##   3   1  Family_members
##   3   2  Family_members
##   3   3  Family_members
##   3   4  Family_members
##   3   5  Family_members
##   4   1  Family_members
##   4   2  Family_members
##   4   3  Family_members
##   4   4  Family_members
##   4   5  Family_members
##   5   1  Family_members
##   5   2  Family_members
##   5   3  Family_members
##   5   4  Family_members
##   5   5  Family_members
```

```
thera_bank = complete(init.impute, 2)
md.pattern(thera_bank)
```

```
##  /\     /\
## {  `---'  }
## {  O   O  }
## ==>  V <==  No need for mice. This data set is completely observed.
##  \  \|/  /
##   `-----'
```




```
##      Age Experience Income Family_members CCAvg Education Mortgage
## 5000   1          1      1              1     1         1        1
##        0          0      0              0     0         0        0
##      Personal_Loan Securities_Account CD_Account Online CreditCard
## 5000             1                 1          1      1          1 0
##                  0                 0          0      0          0 0
```

```
sum(is.na(thera_bank))
```

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

3.4 Univariate Analysis

```
# Distribution of the dependent variable
prop.table(table(thera_bank$Personal_Loan))
```

```
##
##     0     1
## 0.904 0.096
```

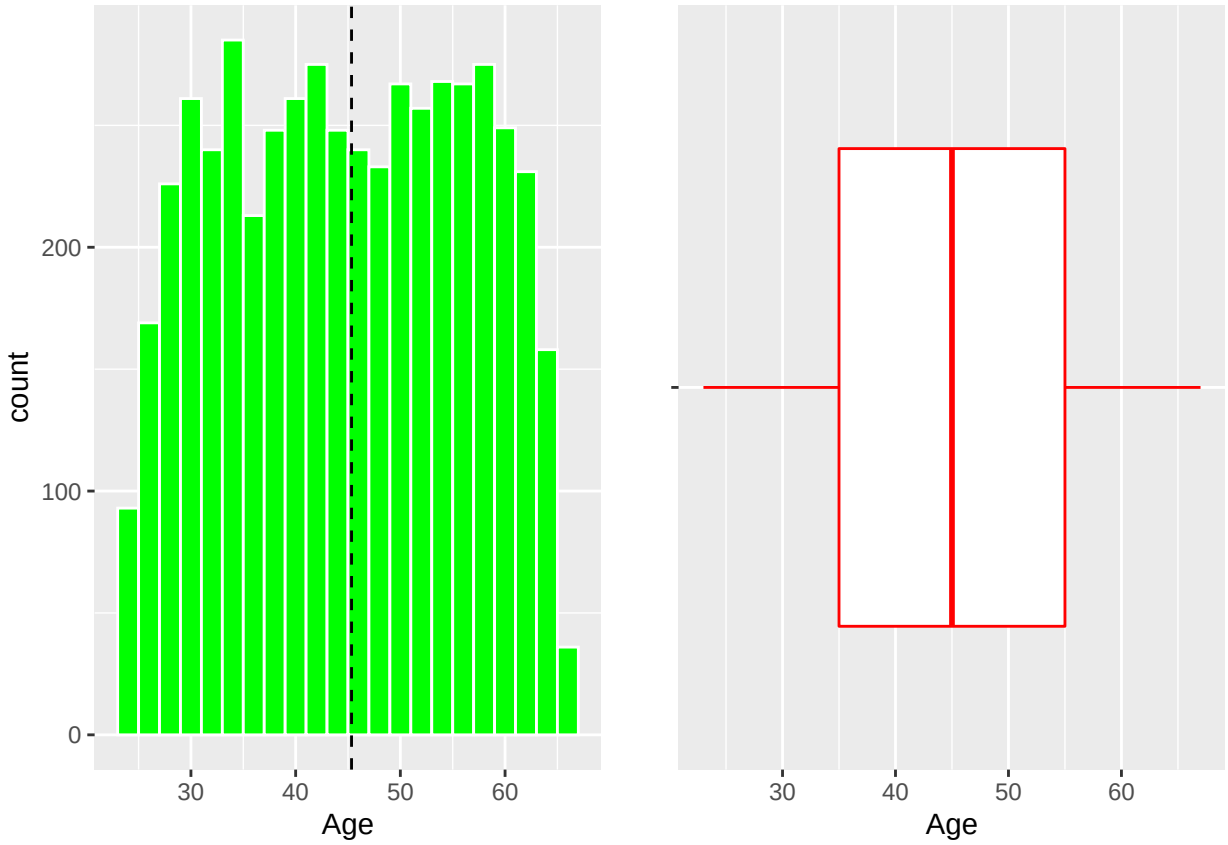
- 9.6% of the customers are borrowers and we need to determine factors that suggest ways of converting liability customers to personal loadn customers to bring in more loan business and in the process, earn more through the interest on loans.
- We will also build a model that will help them identify the potential customers who have a higher probability of purchasing the loan.

```
plot_histogram_n_boxplot = function(variable, variableNameString, binw){
```

```
  a = ggplot(data = thera_bank, aes(x= variable)) +
    labs(x = variableNameString,y ='count')+
    geom_histogram(fill = 'green',col = 'white', binwidth = binw) +
    geom_vline(aes(xintercept = mean(variable)),
               color = "black", linetype = "dashed", size = 0.5)

  b = ggplot(data = thera_bank, aes('',variable))+
    geom_boxplot(outlier.colour = 'red',col = 'red', outlier.shape = 19)+
    labs(x = '', y = variableNameString) + coord_flip()
  grid.arrange(a,b,ncol = 2)
}
```

1. Observations on Age

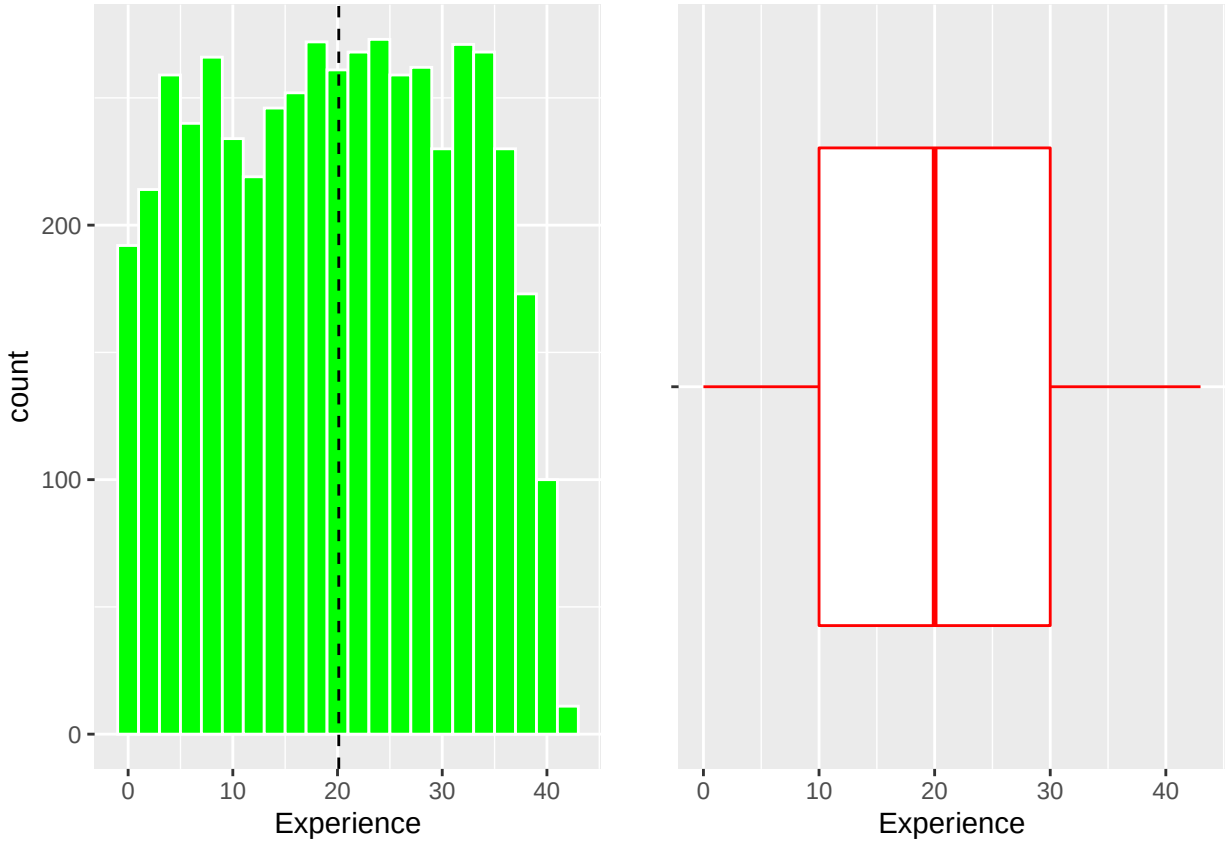
```
plot_histogram_n_boxplot(thera_bank$Age, 'Age', 2)
```



- There is a uniform distribution.

2. Observations on Experience

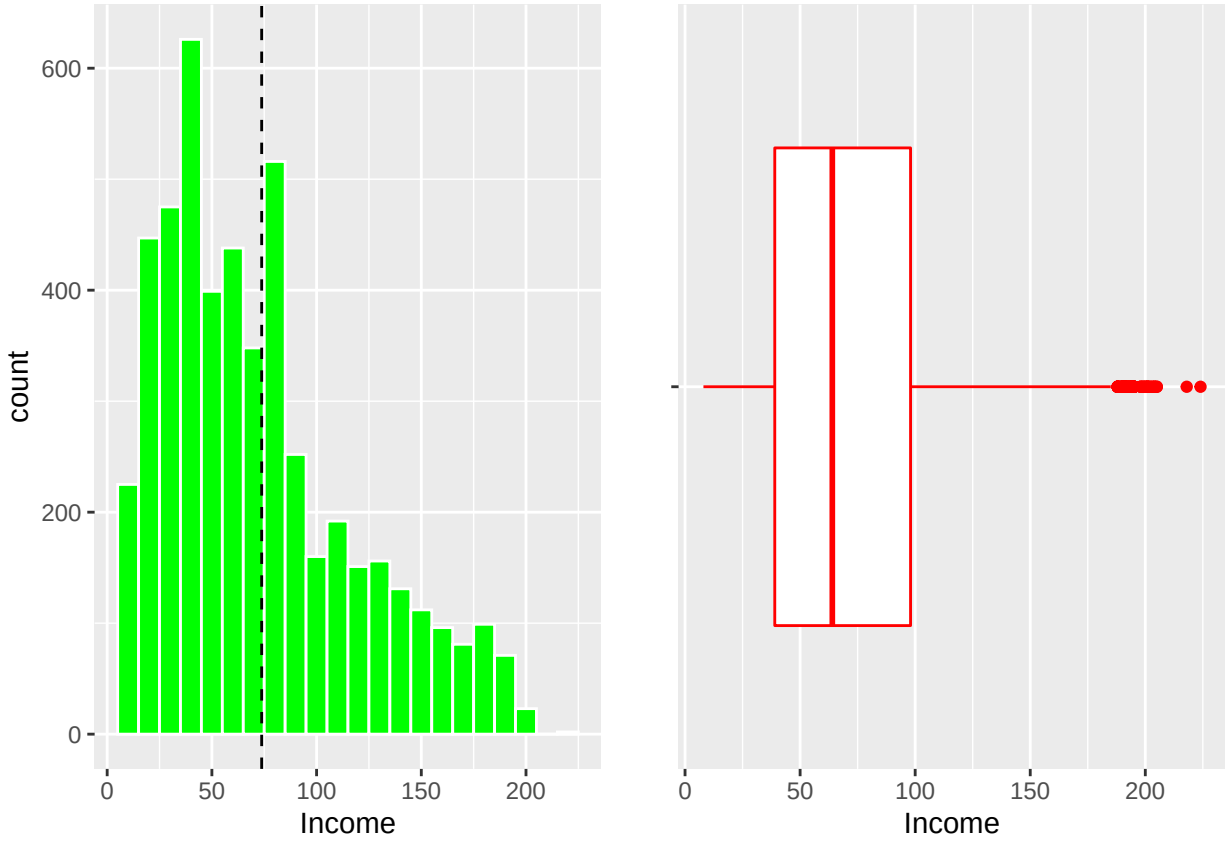
```
plot_histogram_n_boxplot(thera_bank$Experience, 'Experience', 2)
```



- There is a uniform distribution.

3. Observations on Income

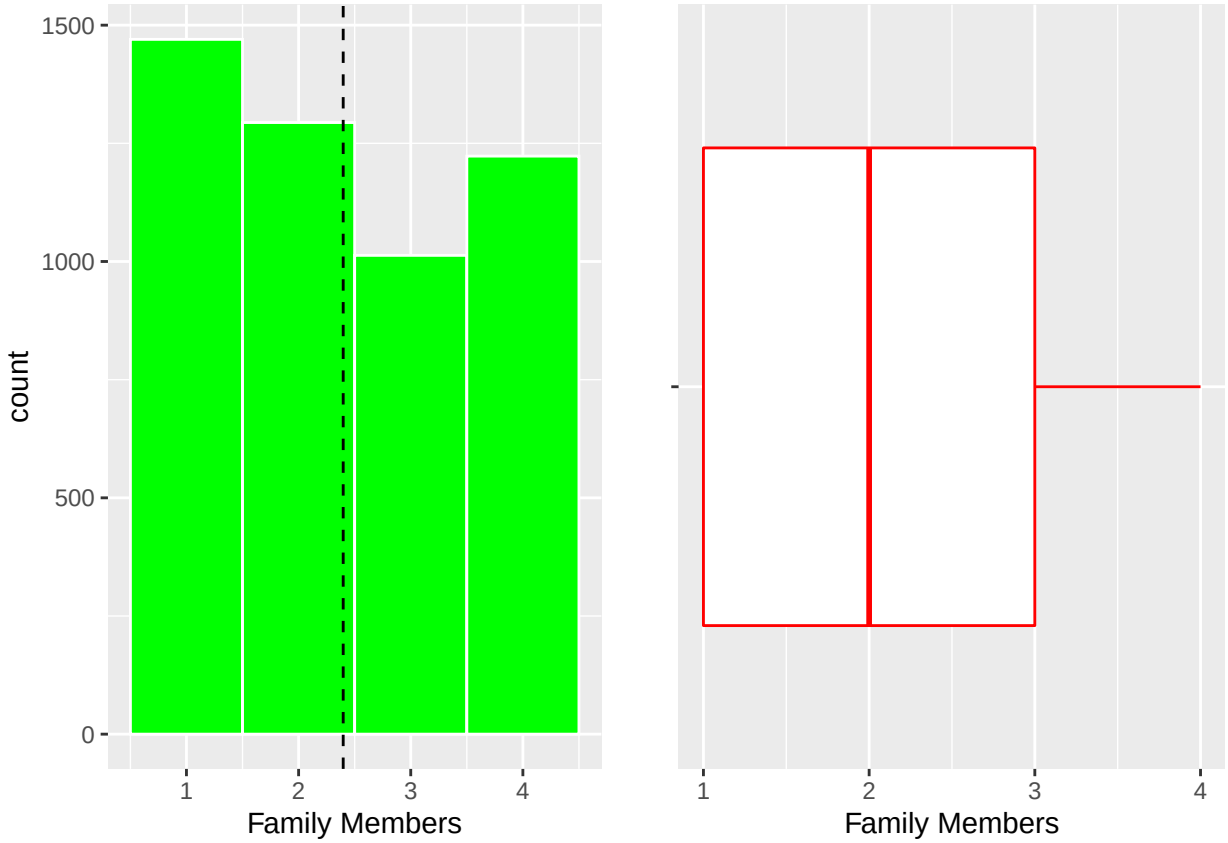
```
plot_histogram_n_boxplot(thera_bank$Income, 'Income', 10)
```



- The distribution is skewed to the right.
- There are however outliers towards the right, indicating that the bank has few customers with a higher level of income loans can be directed at.

4. Observations on Family Members

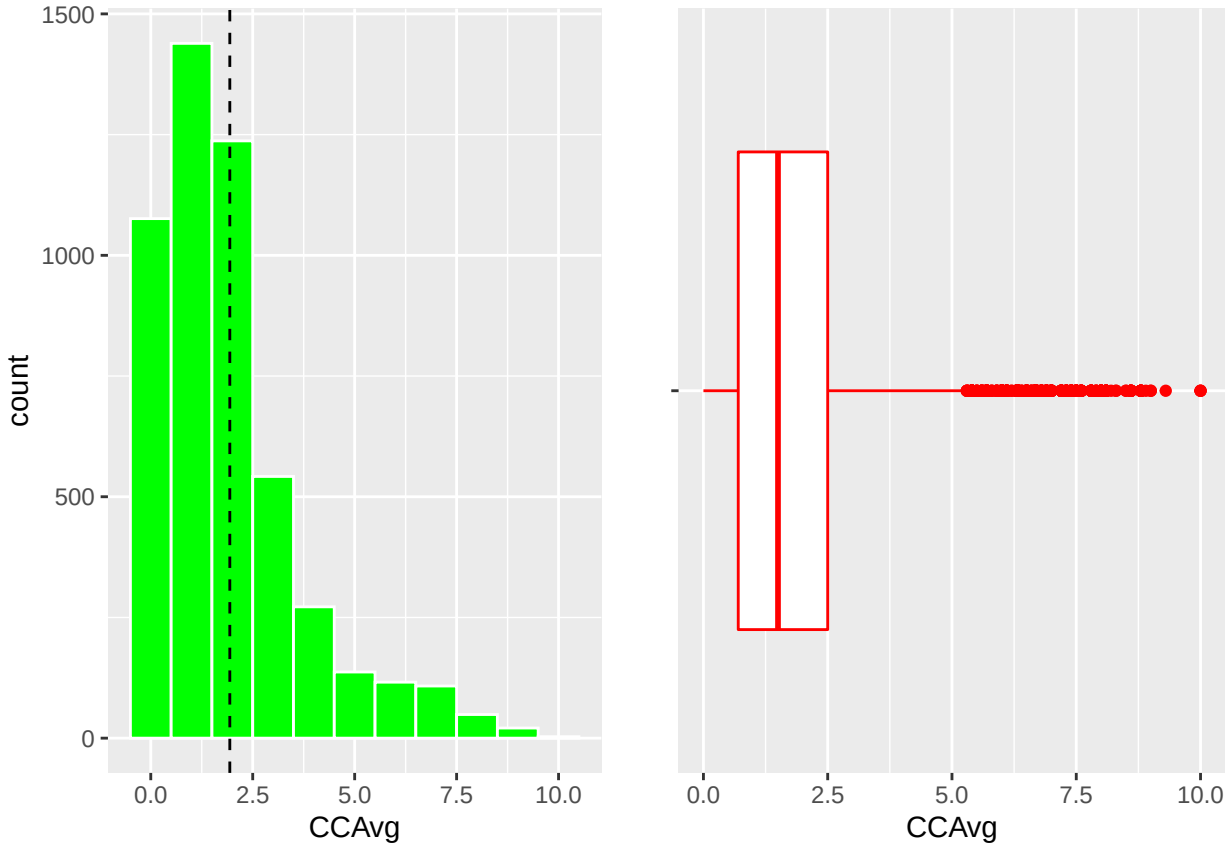
```
plot_histogram_n_boxplot(thera_bank$Family_members, 'Family Members', 1)
```



- There is a uniform distribution.
- The minimum and maximum number of family members each customer have is 1 and 4 respectively.

5. Observations on CCAvg

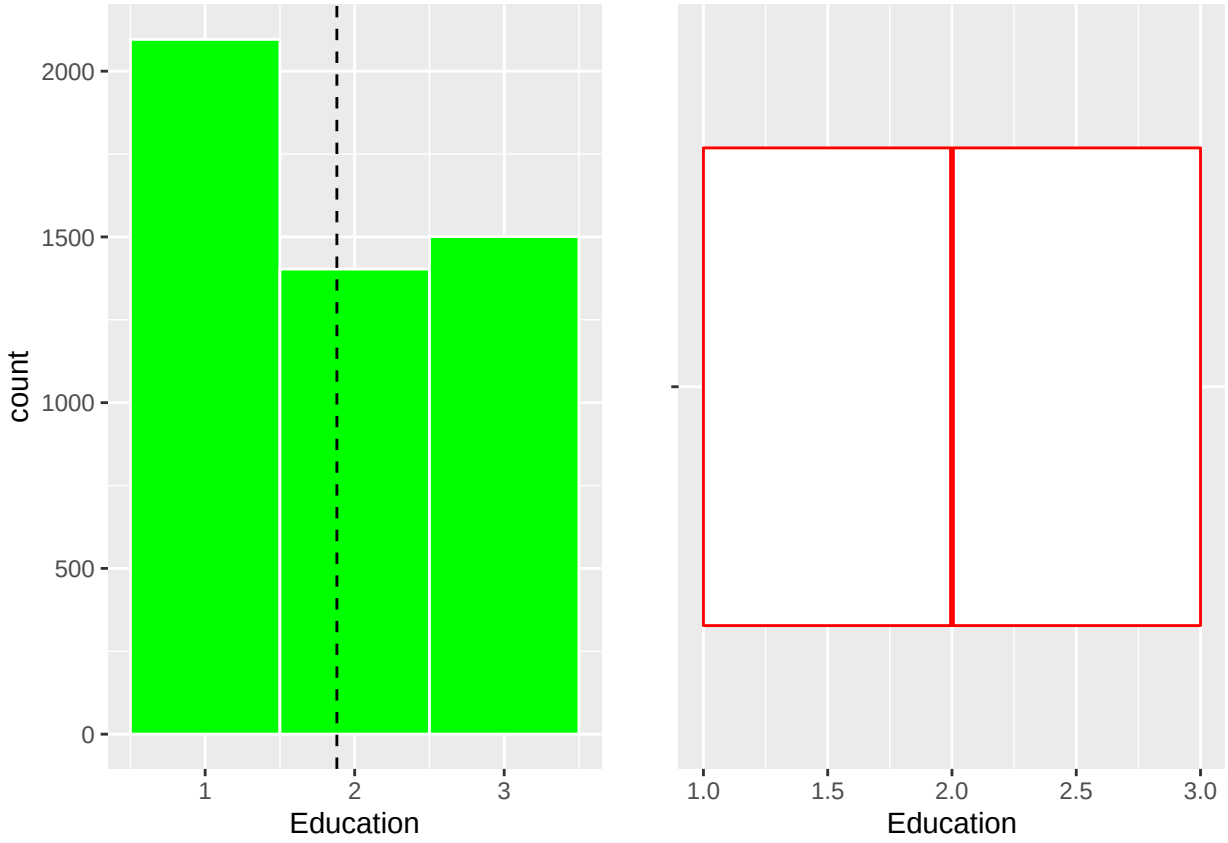
```
plot_histogram_n_boxplot(thera_bank$CCAvg, 'CCAvg', 1)
```



- The distribution is skewed to the right.
- There are a few number of outliers in the bank's customer base indicating individuals subsceptible to the loan program is limited.

6. Observations on Education

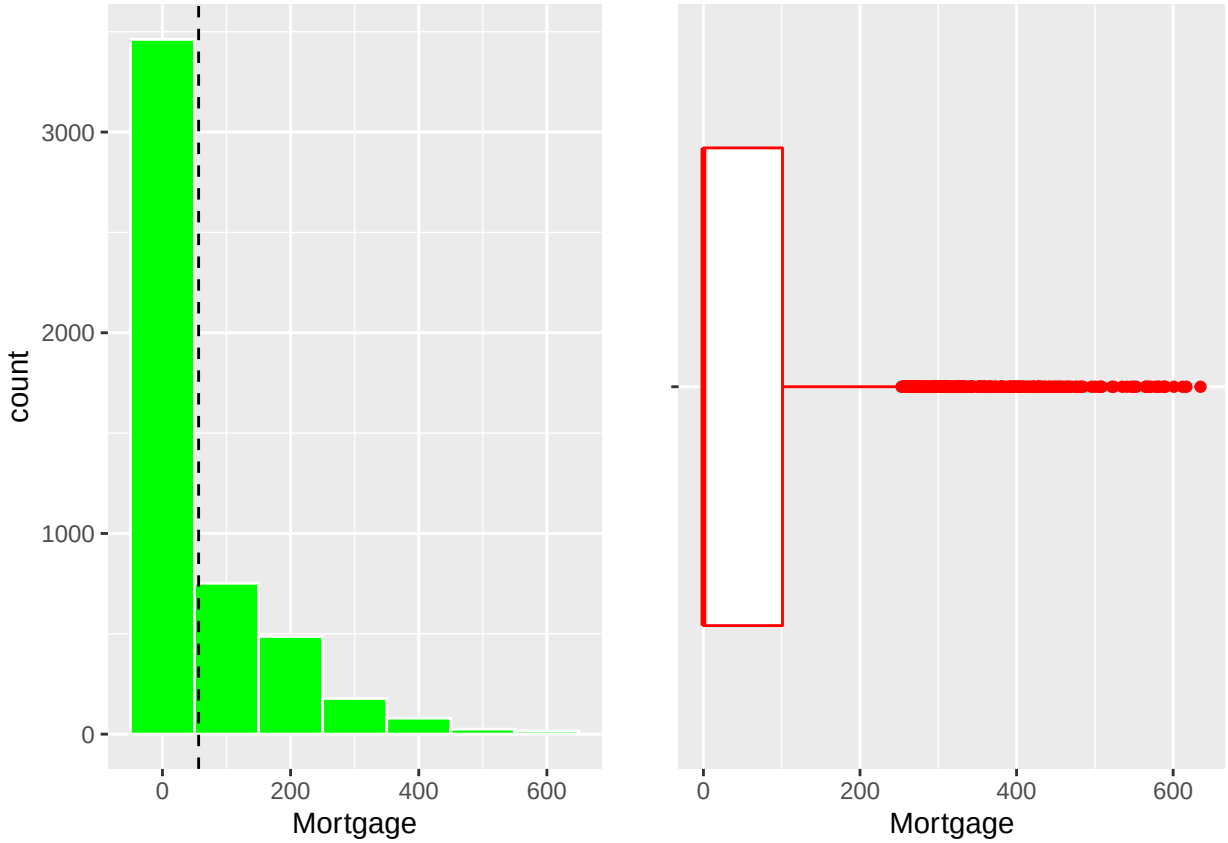
```
plot_histogram_n_boxplot(thera_bank$Education, 'Education', 1)
```



- This is a uniform distribution.
- Undergraduates make-up the greater part of the bank's customer base. Trailing, is the advanced/professional individuals and graduates respectively.

7. Observations on Mortgage

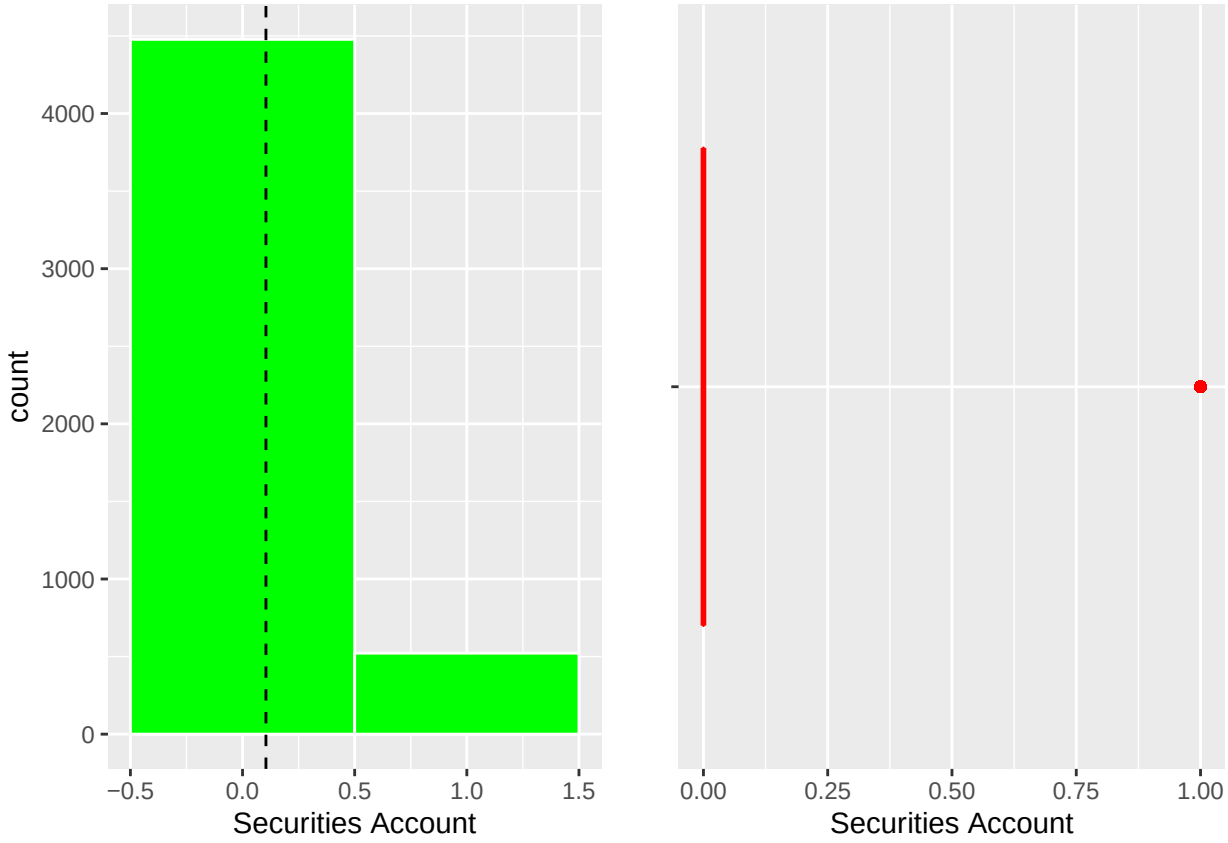
```
plot_histogram_n_boxplot(thera_bank$Mortgage, 'Mortgage', 100)
```



- The distribution is skewed to the right.
- The greater part of the bank's customer base have no mortgage indicating their lack of need for funding or financing.
- However, there are a few outliers which the loan program could be targeted at.

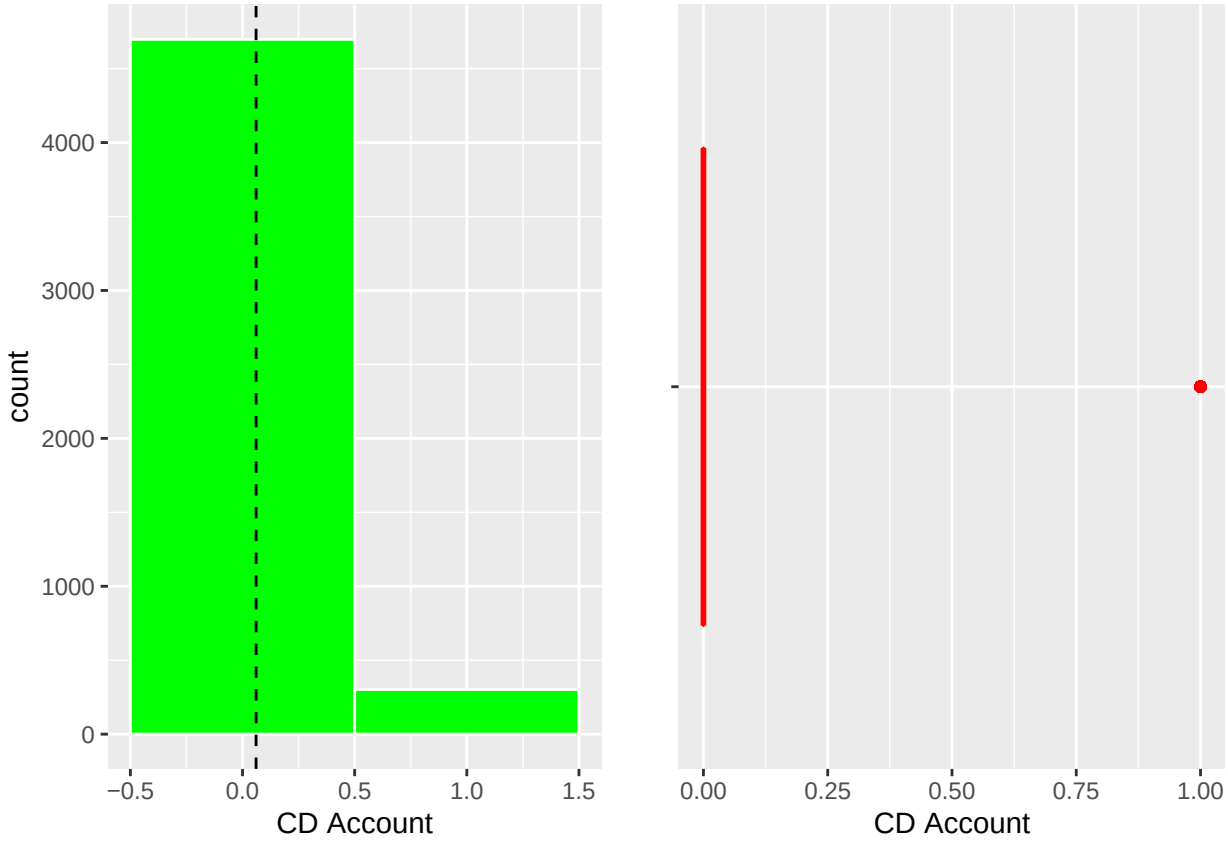
8. Observations on Securities Account

```
plot_histogram_n_boxplot(thera_bank$Securities_Account, 'Securities Account', 1)
```



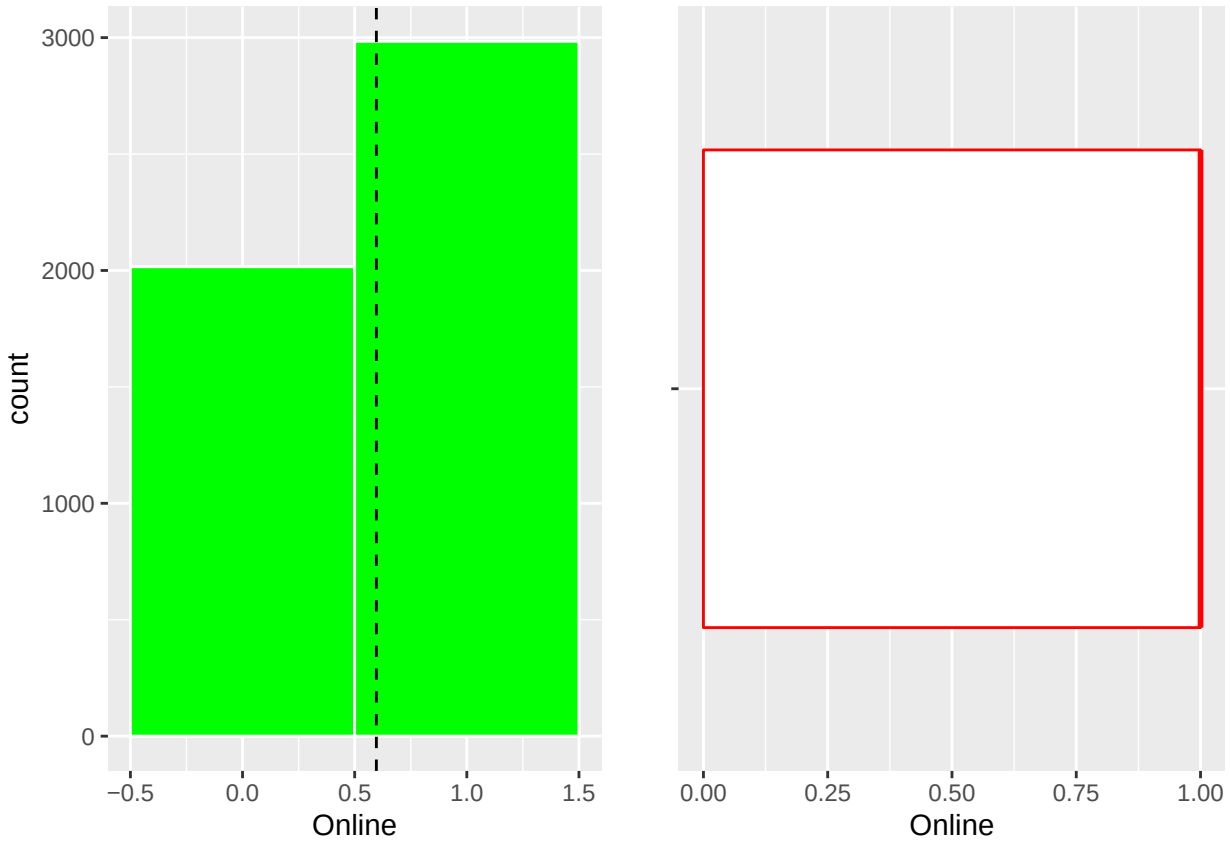
9. Observations on CD Account

```
plot_histogram_n_boxplot(thera_bank$CD_Account, 'CD Account', 1)
```



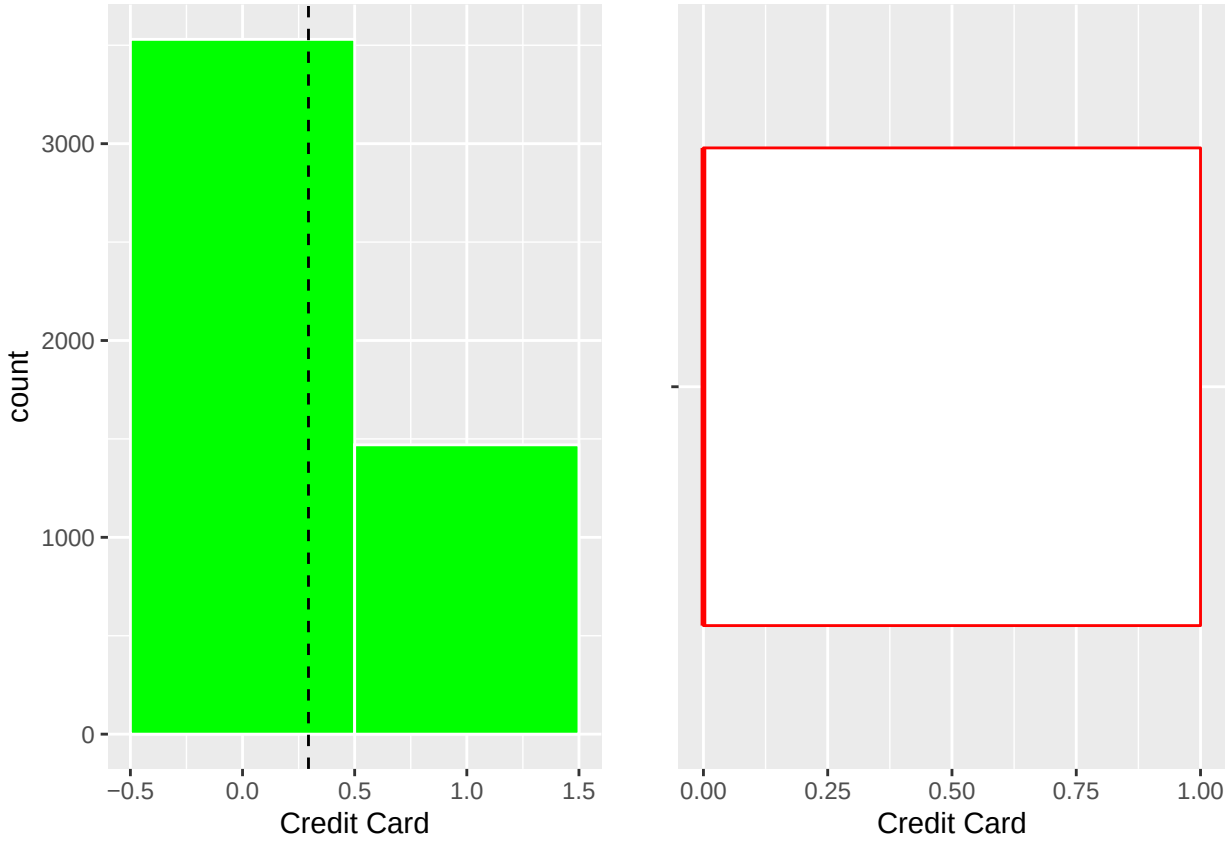
10. Observations on Online

```
plot_histogram_n_boxplot(thera_bank$Online, 'Online', 1)
```



11. Observations on Credit Card

```
plot_histogram_n_boxplot(thera_bank$CreditCard, 'Credit Card', 1)
```



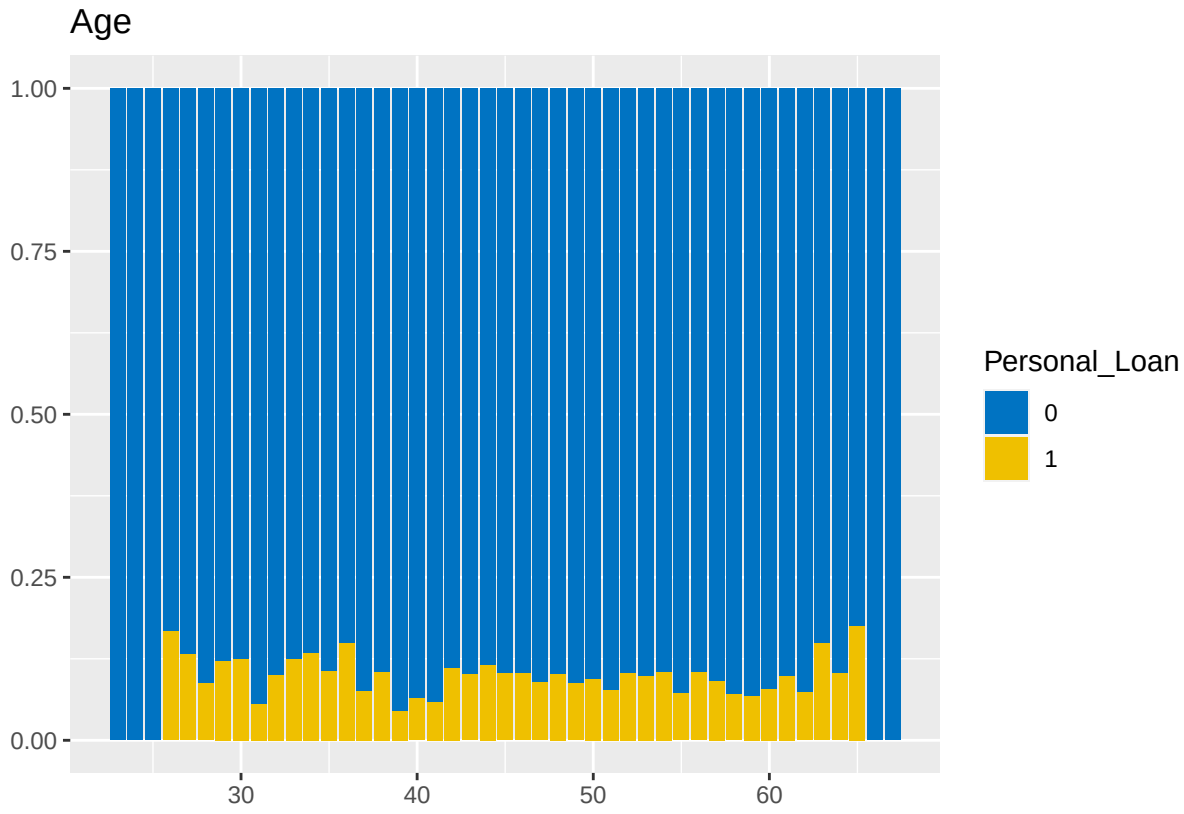
3.5 Bivariate Analysis

3.5.1 Bivariate Stacked Barchart Let us plot percent stacked barchart to see the effect of independent variables on the probability of personal loan

```
# Function to draw percent stacked barchart to see the effect of independent variables
# on the probability of personal loan using ggplot
plot_stacked_barchart = function(variable, variableNameString){
  ggplot(thera_bank, aes(fill = Personal_Loan, x = variable)) +
    geom_bar(position="fill")+
    labs(title = variableNameString, y = '', x = '')+
    scale_fill_manual(values=c("#0073C2FF", "#EFC000FF"))

}
```

1. Personal loan vs Age

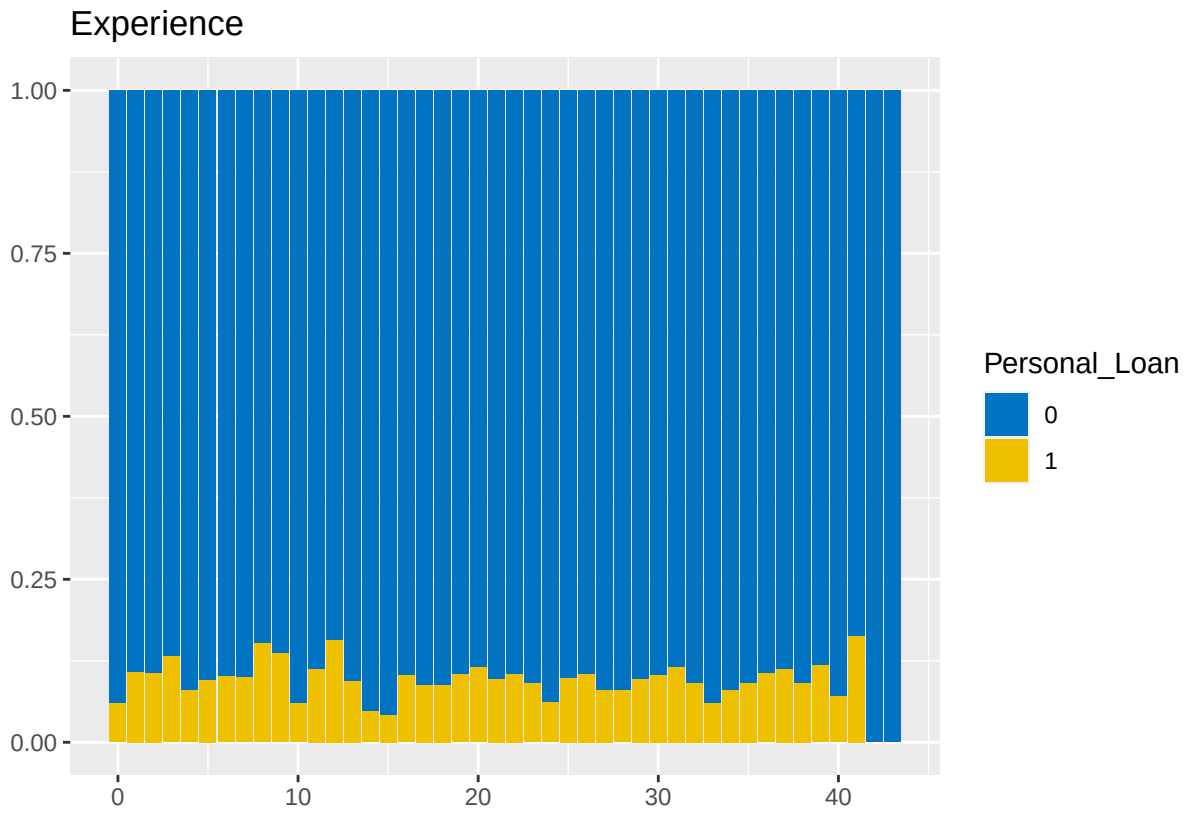
```
plot_stacked_barchart(thera_bank$Age, 'Age')
```



- Customers in the age range of 26 - 65 took a loan in the last campaign.

2. Personal loan vs Experience

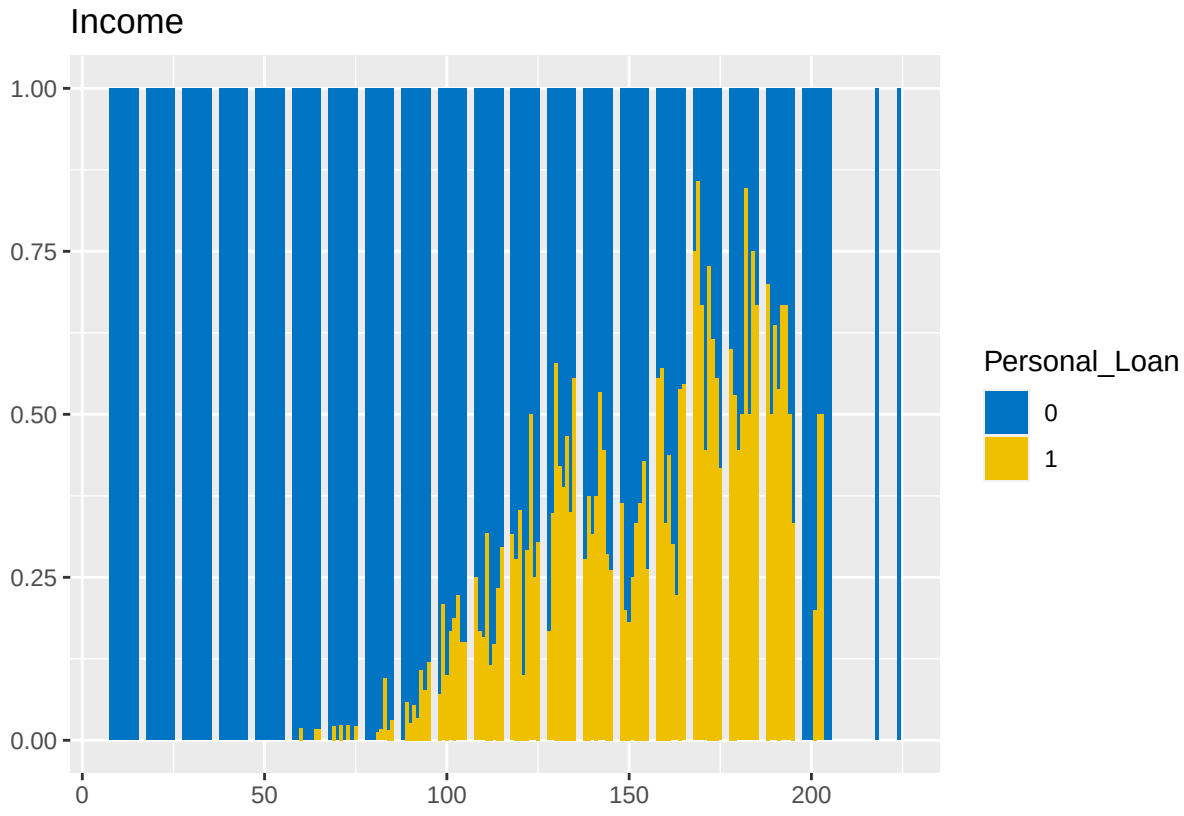
```
plot_stacked_barchart(thera_bank$Experience, 'Experience')
```



- Customers with or without professional experience took a loan in the last campaign.

3. Personal loan vs Income

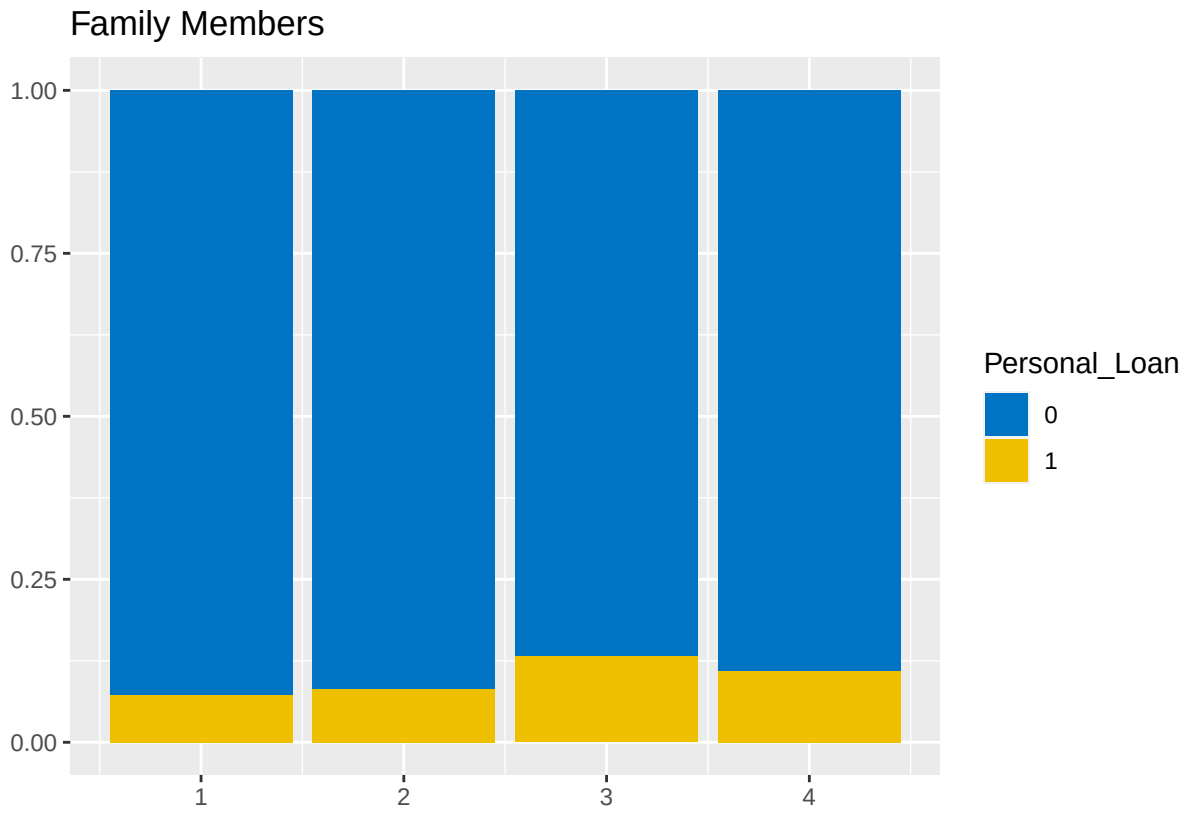
```
plot_stacked_barchart(thera_bank$Income, 'Income')
```



- In the last campaign, customers who earned more were likely to take a loan.

4. Personal loan vs Family members

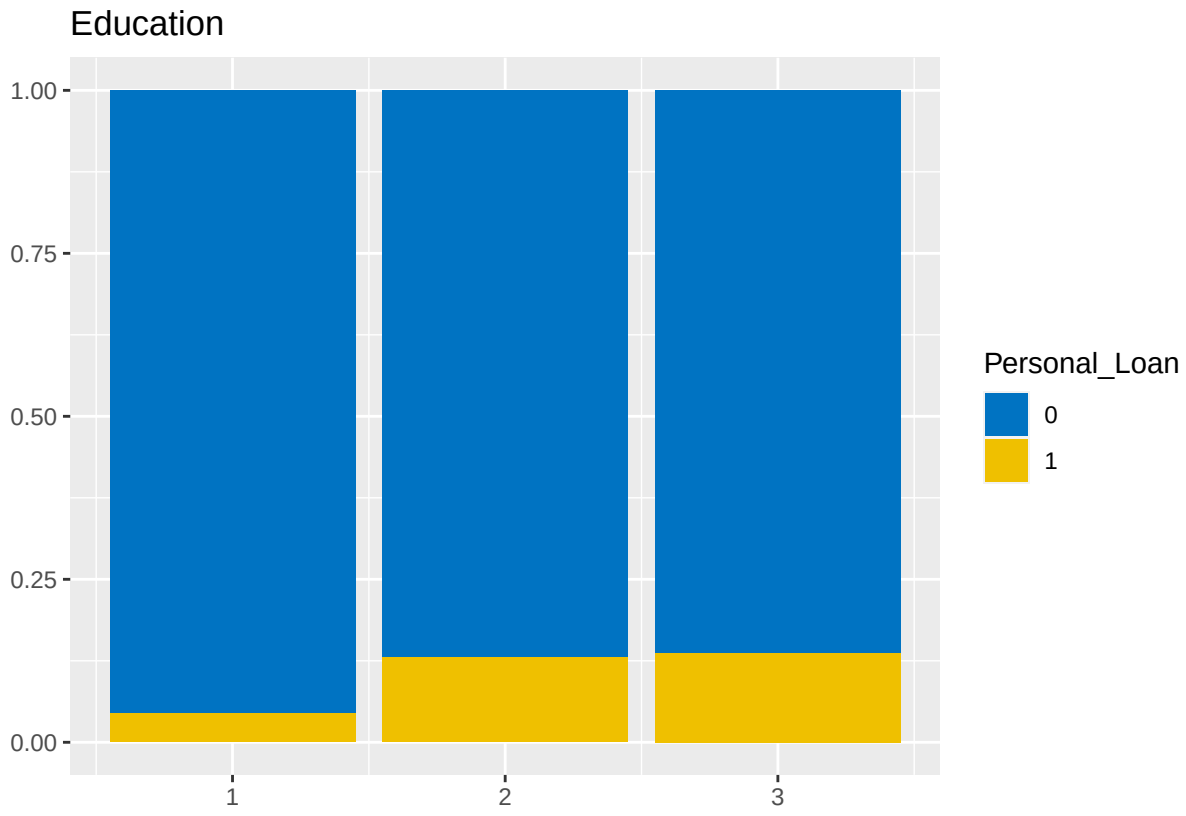
```
plot_stacked_barchart(thera_bank$Family_members, 'Family Members')
```



- There is little difference in family members and personal loan. However, customers with 3 or more family members have a higher tendency to take a loan.

5. Personal loan vs Education

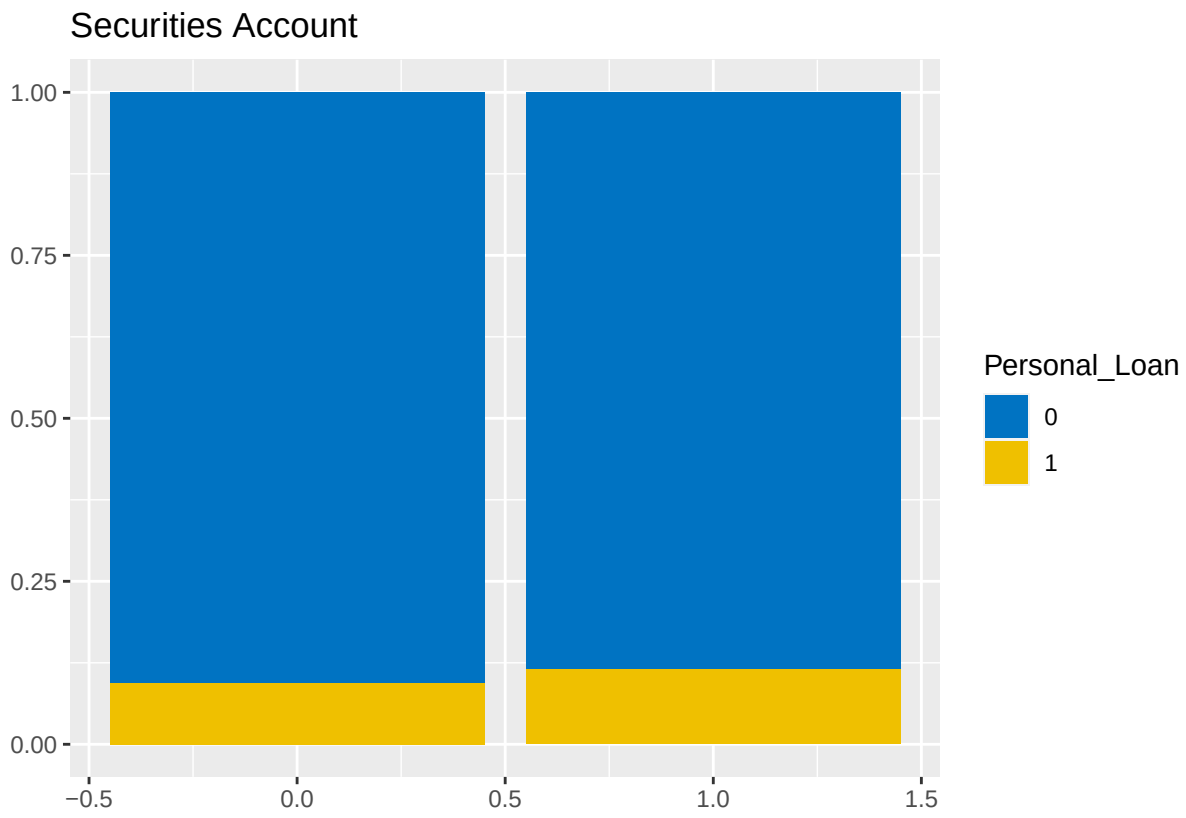
```
plot_stacked_barchart(thera_bank$Education, 'Education')
```



- Customers with a higher level of education have a higher tendency to take a loan.

6. Personal loan vs Securities Account

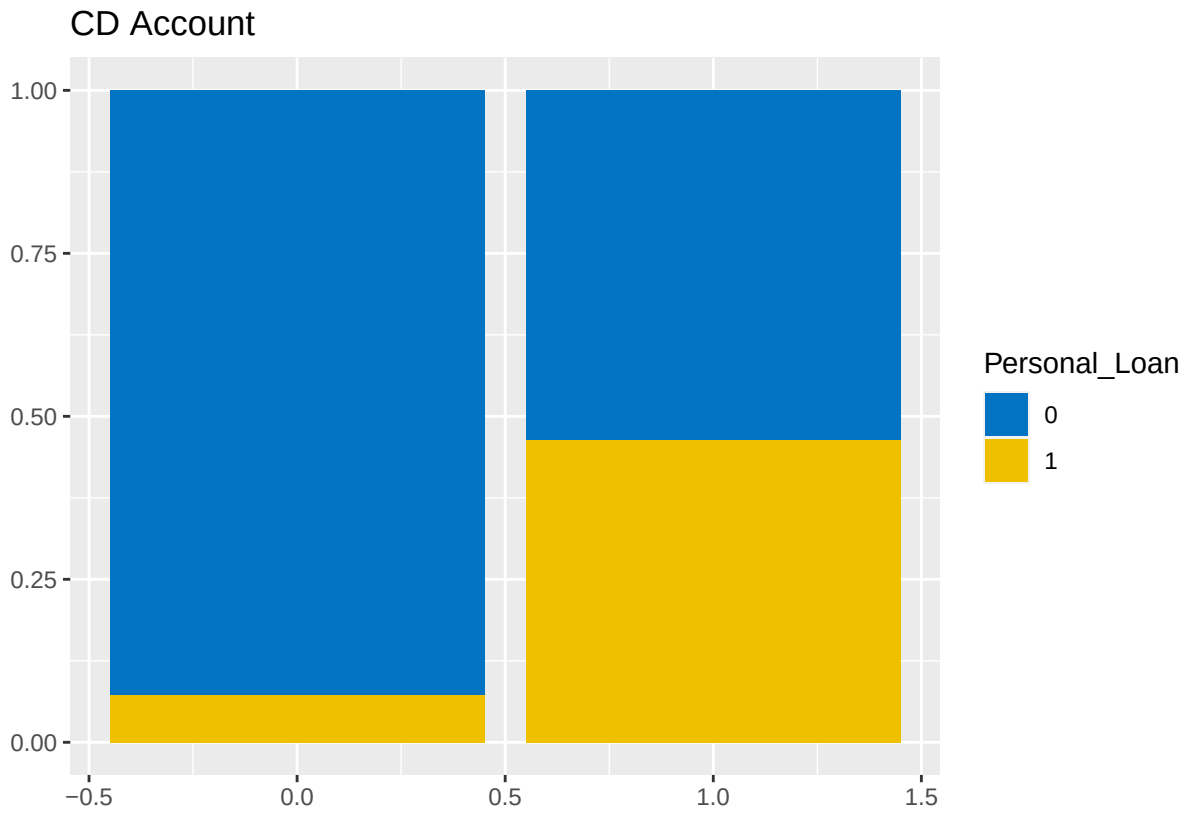
```
plot_stacked_barchart(thera_bank$Securities_Account, 'Securities Account')
```



- There is little or no difference between customers who have securities account and who took a personal loan.

7. Personal loan vs CD Account

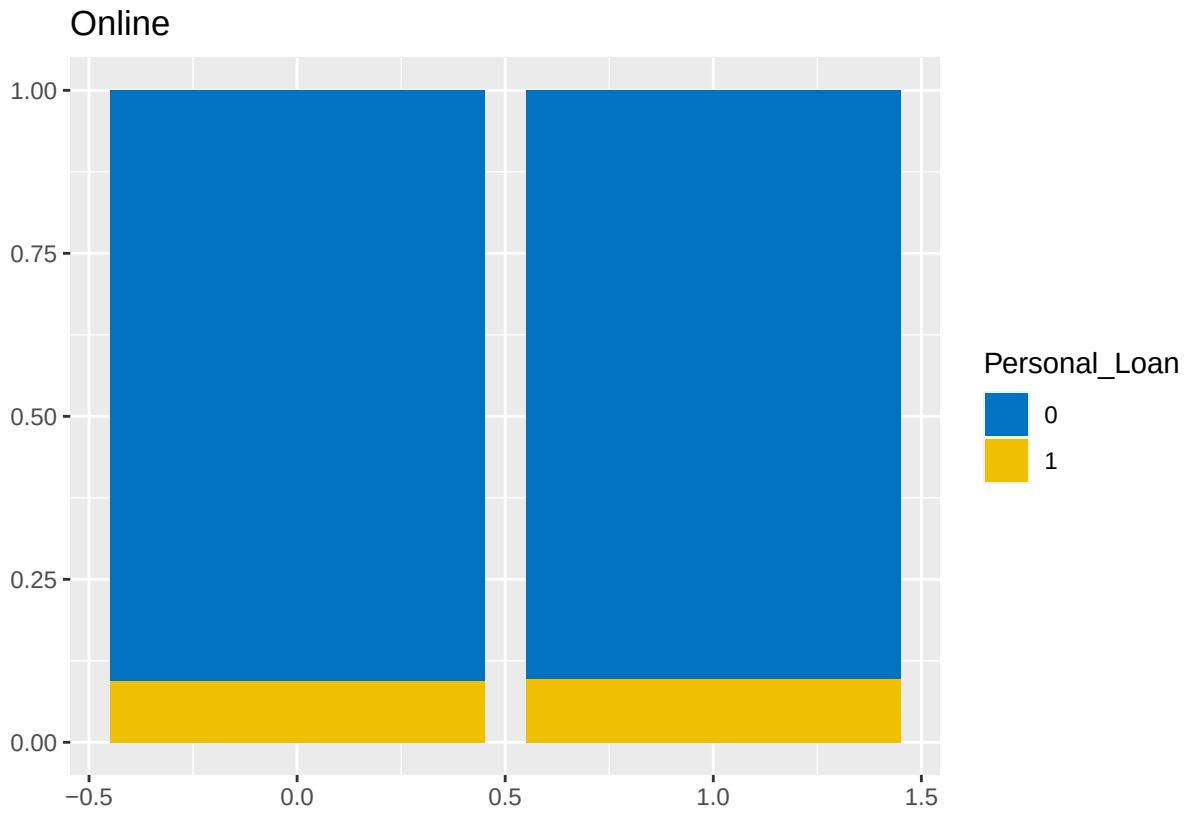
```
plot_stacked_barchart(thera_bank$CD_Account, 'CD Account')
```



- Customers with a CD account have a higher tendency to take a loan.

8. Personal loan vs Online

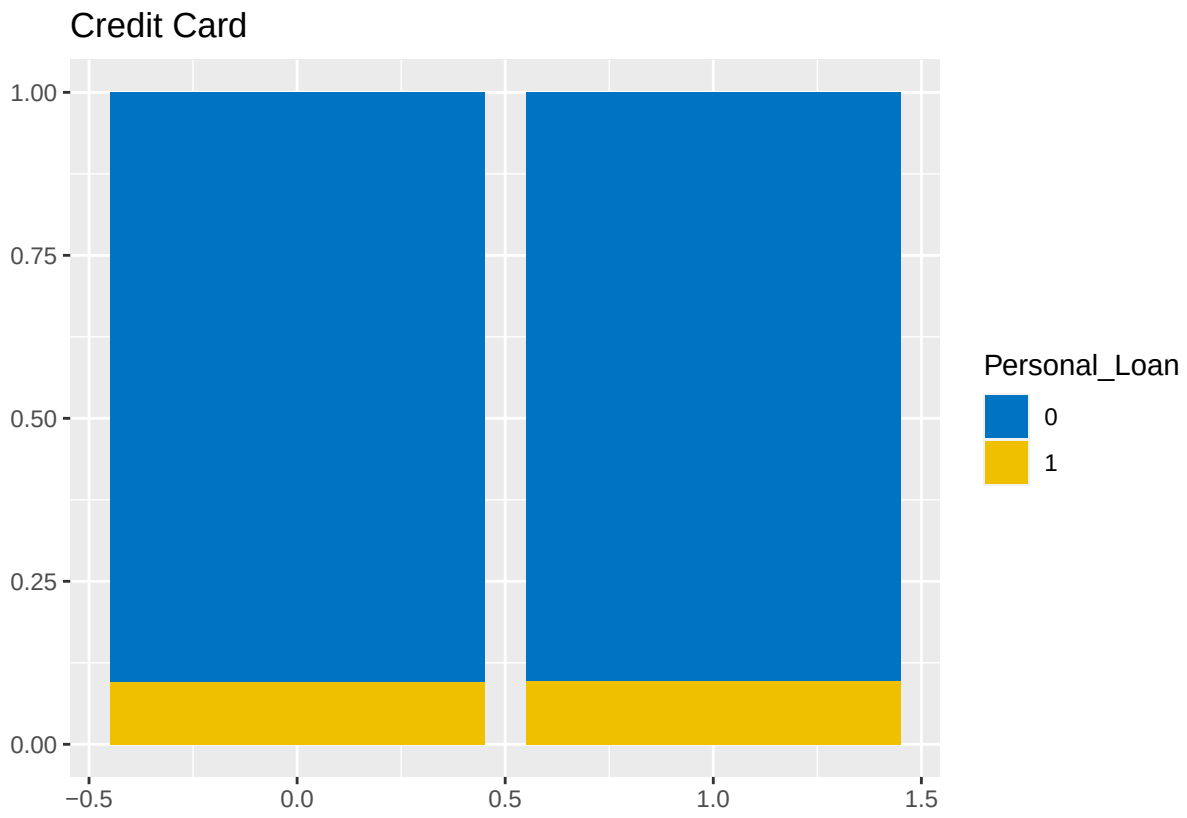
```
plot_stacked_barchart(thera_bank$Online, 'Online')
```



- There is little or no difference between customers who shop online and who took a personal loan.

9. Personal loan vs Credit Card

```
plot_stacked_barchart(thera_bank$CreditCard, 'Credit Card')
```

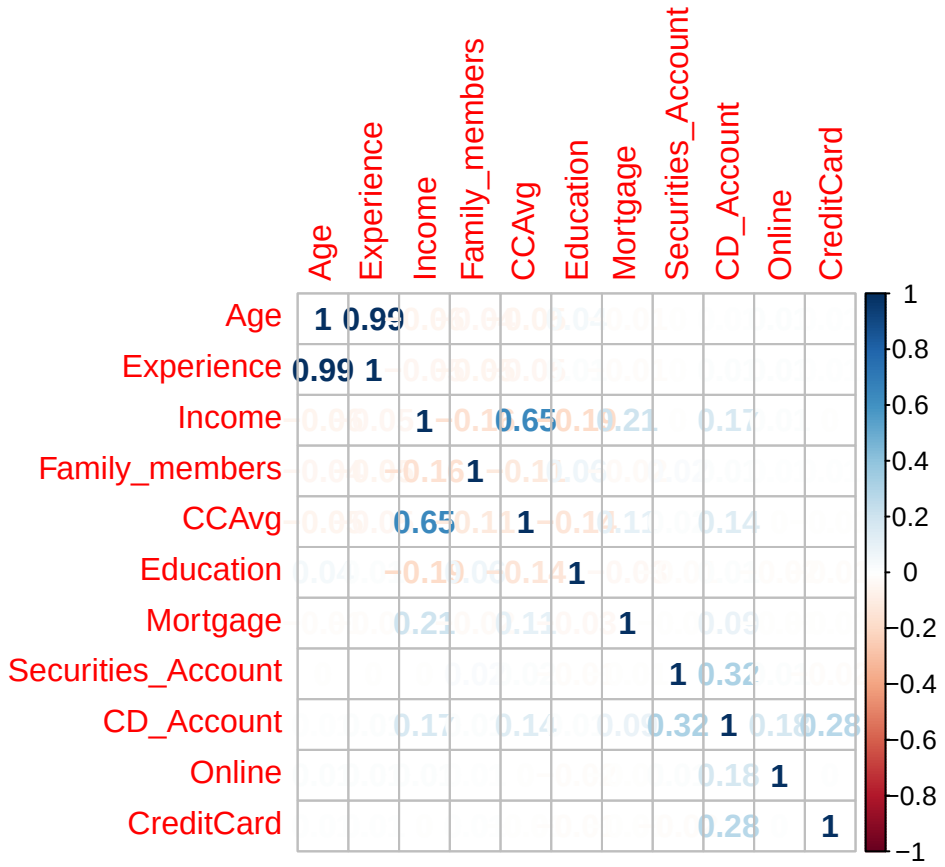


- There is little or no difference between customers who have credit card and who took a personal loan.

3.5.2 Correlation Plot between Numerical Variables Plot bivariate charts between variables to understand their relationship with each other.

Check for correlation among numerical variables

```
# Numeric variables in the data
num_vars = sapply(thera_bank, is.numeric)

# Correlation Plot
corrplot(cor(thera_bank[,num_vars]), method = 'number')
```



- There is a high correlation between age and experience, which is understandable.
- There is a moderate correlation between CCAvg and Income, suggesting that customer's level of income to an extent influence credit card spending pattern.

4. Data Modelling: Cluster Analysis

4.1 K-Means Clustering

For this problem statement, we choose to use K-Means clustering because of the following:

- First, the number of observations is large (5000), which will be computationally huge with Hierarchical clustering as it computes pairwise distance between every points of data. This is because the time complexity of K Means is linear i.e. O(n) while that of hierarchical clustering is quadratic i.e. O(n2), taking a quadratic amount if time.
- K-Means is found to work well when the shape of the clusters is spherical (like circle in 2D, sphere in 3D) and specifically when the dataset is huge.

```
# Scale the dataset to reduce the influence from variables with high values
data <- thera_bank

# Change Family_members, Education, Securities_Account, CD_Account, Online and CreditCard to factor var
thera_bank$Family_members <- as.factor(thera_bank$Family_members)
thera_bank$Education <- as.factor(thera_bank$Education)
thera_bank$Securities_Account <- as.factor(thera_bank$Securities_Account)
thera_bank$CD_Account <- as.factor(thera_bank$CD_Account)
thera_bank$Online <- as.factor(thera_bank$Online)
thera_bank$CreditCard <- as.factor(thera_bank$CreditCard)
```

```r
str(data)
```

```
## 'data.frame':    5000 obs. of  12 variables:
##  $ Age               : num  25 45 39 35 35 37 53 50 35 34 ...
##  $ Experience        : num  1 19 15 9 8 13 27 24 10 9 ...
##  $ Income            : num  49 34 11 100 45 29 72 22 81 180 ...
##  $ 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     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
##  $ 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 ...
```

```r
view(data)

thera_bank.scaled <- scale(data[, -c(4, 6, 8, 9, 10, 11, 12)])
```

```r
# Determine the optimum number of clusters (find optimal k)
seed <- 1000
set.seed(seed) # kmeans uses a randomized starting point for cluster centroids
clust2 = kmeans(thera_bank.scaled, centers = 2, nstart = 5)
print(clust2)
```

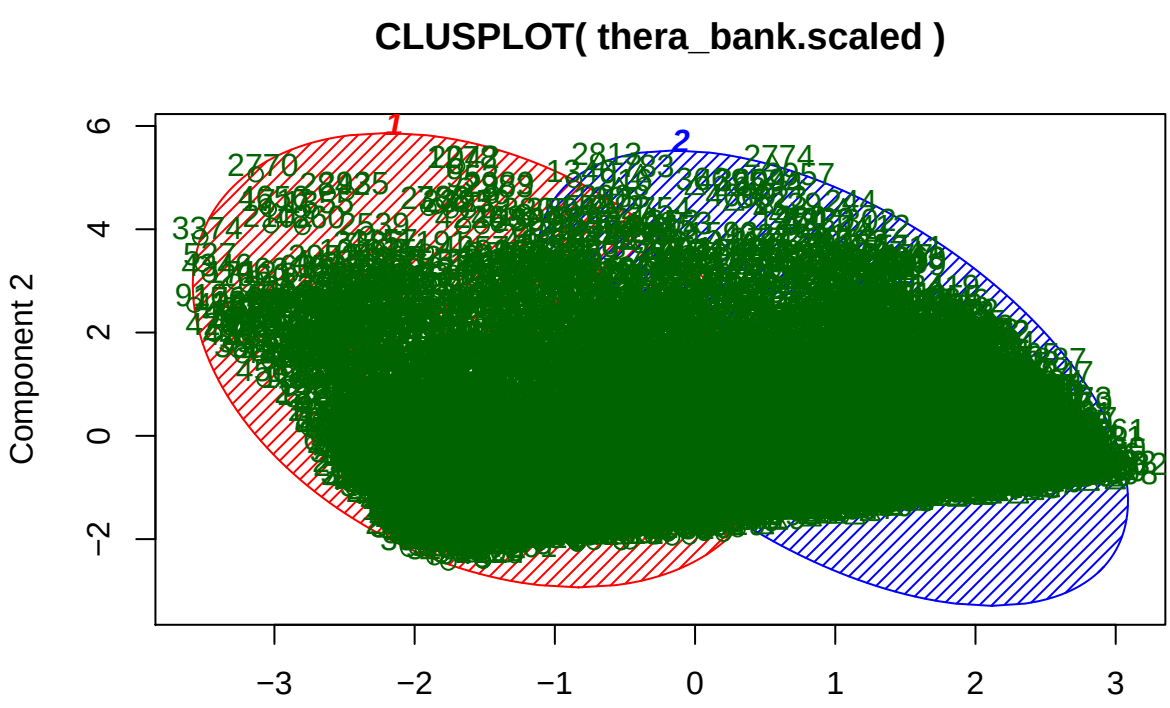
```
## K-means clustering with 2 clusters of sizes 2532, 2468
##
## Cluster means:
##          Age Experience     Income      CCAvg    Mortgage
## 1 -0.8463412 -0.8417361  0.1297191  0.1417990  0.02543704
## 2  0.8682884  0.8635640 -0.1330830 -0.1454762 -0.02609667
##
## Clustering vector:
##    [1] 1 2 1 1 1 1 2 2 1 1 2 1 2 2 2 2 1 1 1 2 2 2 1 1 1 1 1 1 2 1 2 1 2 1 1 2 2
##   [38] 2 1 1 2 1 1 1 1 2 1 1 2 1 1 2 1 2 1 1 2 2 1 1 2 1 1 1 2 2 2 2 2 2 1 2 1 1
##   [75] 1 1 2 2 2 2 2 2 1 1 2 1 1 2 2 1 2 1 1 2 2 1 1 2 2 2 2 2 2 1 2 1 1 1 1 1 1
##  [112] 2 1 2 1 2 2 2 1 1 2 2 2 1 1 2 1 1 1 1 1 2 1 1 2 2 2 2 2 2 2 1 1 1 2 2 1 2
##  [149] 2 2 1 1 2 2 2 1 1 1 1 2 1 2 1 1 2 1 1 1 2 1 1 2 1 2 1 1 2 1 2 2 2 1 1 1 2
##  [186] 1 2 1 2 2 2 2 2 2 2 1 1 2 1 1 1 1 1 2 2 1 2 1 1 2 2 1 2 2 2 1 1 1 2 2 1 1
##  [223] 1 2 2 1 1 1 2 2 2 1 1 2 1 1 1 2 2 1 2 2 1 2 1 1 1 2 2 1 1 2 2 2 2 2 1 2 1
##  [260] 2 2 1 2 1 2 2 2 2 2 1 2 1 1 1 1 2 1 1 2 1 1 2 1 2 1 1 2 1 1 1 2 1 1 1 1 2
##  [297] 1 2 1 1 1 2 1 2 2 2 2 1 1 2 2 2 1 1 2 1 2 1 1 2 2 1 2 2 2 2 2 2 2 1 2 1 2
##  [334] 2 2 2 1 2 1 1 2 1 1 1 2 2 1 1 1 1 1 1 2 2 1 1 2 1 1 1 1 1 2 1 2 2 2 1 2 1
##  [371] 1 2 2 2 1 1 2 1 2 1 2 2 2 1 2 1 1 1 2 1 2 2 2 2 1 2 2 1 2 1 1 1 2 2 2 1 1
##  [408] 2 2 2 2 2 1 1 2 1 1 2 1 2 2 1 1 1 2 1 1 1 2 1 2 1 1 2 1 2 2 1 2 2 2 2 2 2
##  [445] 2 2 2 2 1 2 2 1 1 2 2 1 2 1 2 1 2 2 1 1 1 2 1 2 1 2 1 2 1 2 2 1 2 2 1 2 2
##  [482] 1 2 1 1 2 2 1 1 2 1 1 2 2 1 1 2 2 1 2 2 2 1 1 1 1 2 2 2 2 2 1 1 1 1 1 2 2
##  [519] 1 1 2 2 1 2 1 2 1 1 2 1 2 1 2 1 2 2 1 1 1 2 1 1 1 2 1 1 1 1 2 2 2 2 1 2 1
##  [556] 1 2 1 1 2 1 2 1 2 1 2 2 1 1 1 1 1 1 2 1 2 1 2 1 2 2 1 1 1 1 1 1 2 1 1 1 1
##  [593] 1 1 2 1 1 1 2 1 2 2 1 2 1 2 1 1 1 1 2 2 2 2 1 2 1 1 2 2 1 1 1 2 1 2 1 1 2
##  [630] 1 1 1 2 2 2 2 1 2 1 2 1 1 2 1 2 1 2 2 2 1 1 1 1 2 2 2 1 1 2 2 1 2 2 2 2 2
##  [667] 2 2 2 2 1 2 2 1 2 1 2 1 2 2 2 1 2 1 1 1 1 2 1 2 2 1 1 1 1 1 2 1 2 1 1 1 1
##  [704] 1 2 2 2 2 2 1 1 1 2 1 1 2 2 1 2 2 2 2 2 1 2 2 1 2 2 1 2 2 1 2 1 1 1 2 2 1
##  [741] 2 2 1 2 1 1 2 2 1 2 1 2 2 2 2 1 2 2 2 2 2 2 1 2 1 2 1 2 1 2 1 1 1 1 2 1 2
```

```
##  [778] 2 2 2 1 2 2 1 1 1 2 1 2 1 2 2 1 1 2 2 1 1 1 1 1 2 1 2 2 2 2 2 2 2 1 2 1 2
##  [815] 1 2 2 1 2 2 2 1 2 1 1 1 2 2 1 2 1 2 1 2 1 2 1 1 2 1 1 2 1 2 2 1 2 1 2 1 2
##  [852] 1 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 1 2 1 1 1 2 1 1 1 2 2 1 2 2 1 1 2 1
##  [889] 2 1 2 1 1 2 1 1 2 2 2 1 1 2 2 1 1 2 1 2 2 1 2 2 1 2 2 2 1 1 1 2 1 1 1 2 2
##  [926] 1 1 2 1 2 1 1 2 2 2 1 2 1 2 2 2 1 2 1 1 2 1 2 1 1 1 2 1 2 1 2 1 2 2 2 1 1
##  [963] 2 1 1 2 2 2 2 1 2 1 1 1 2 2 2 2 2 2 1 1 2 1 2 1 2 2 2 1 1 1 1 1 1 1 1 2 2
## [1000] 2 2 2 2 1 2 1 2 1 1 1 1 2 2 1 2 2 1 1 1 1 2 1 1 1 2 2 1 1 1 1 2 2 1 2 2 1
## [1037] 2 1 2 1 1 2 2 2 2 1 2 2 2 1 2 1 1 2 2 1 1 1 2 1 2 2 1 2 1 1 1 2 1 1 1 1 2
## [1074] 1 1 1 1 1 2 2 2 2 1 1 2 2 2 1 2 2 1 1 1 1 2 2 1 2 2 1 1 1 1 1 2 1 1 2 2 2
## [1111] 2 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 2 2 2 1 2 1 1 1 1 2 1 1
## [1148] 1 1 2 2 2 2 2 2 1 2 2 1 2 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1 1 1 2
## [1185] 1 1 2 2 2 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 2 1 2 2 2 2 1 1 2 2 2 1 2 1 1
## [1222] 1 2 2 2 1 2 1 2 2 1 2 1 2 1 2 1 1 1 2 2 2 1 1 1 2 2 2 2 2 2 1 1 2 1 1 1 2
## [1259] 1 2 2 2 1 1 2 1 2 2 1 1 1 1 2 2 2 1 1 1 1 2 2 1 2 1 2 1 1 1 2 2 2 2 2 2 1
## [1296] 1 1 2 1 2 2 1 1 1 2 1 1 1 2 1 2 1 2 2 1 2 1 2 2 1 1 1 1 2 2 2 1 2 2 1 1 1
## [1333] 1 2 2 2 1 1 2 1 1 1 1 1 2 2 1 2 1 1 1 2 2 2 1 2 1 2 2 2 2 2 1 1 1 2 2 2 2
## [1370] 2 1 2 1 2 2 2 2 1 2 2 2 1 1 2 2 2 1 1 2 1 1 1 2 2 2 2 1 2 1 1 1 1 2 1 2 1
## [1407] 2 2 1 1 2 2 2 2 2 1 1 1 2 1 1 1 1 2 1 2 1 1 1 1 1 2 1 2 2 1 2 1 2 2 1 2 1
## [1444] 1 2 1 1 2 1 2 2 1 2 1 2 2 1 1 2 2 1 2 2 1 1 1 1 2 1 2 2 2 1 2 2 1 2 1 2 1
## [1481] 2 1 2 2 2 1 1 1 1 2 1 1 1 2 2 2 1 2 1 2 2 1 2 1 1 2 2 1 1 2 2 2 2 1 1 2 1
## [1518] 2 1 2 2 1 1 1 1 1 1 2 1 1 2 1 2 2 2 2 1 2 2 1 1 2 2 2 1 2 1 2 2 2 1 2 1 2
## [1555] 1 2 1 2 1 2 1 2 1 2 2 1 2 2 2 2 2 1 1 2 1 2 2 1 1 1 1 1 2 1 2 2 2 2 2 1 2 2
## [1592] 1 2 2 1 2 1 2 1 2 2 1 1 1 2 2 1 2 1 2 1 2 1 2 2 2 2 2 1 2 1 1 1 2 1 2 1 2
## [1629] 1 2 1 2 1 2 2 2 2 1 1 2 1 2 1 1 2 2 2 1 2 1 1 2 1 1 2 1 1 1 2 1 1 1 2 2 2
## [1666] 1 2 2 2 1 1 1 2 1 1 2 1 1 2 2 2 1 2 2 2 1 2 2 2 2 1 2 2 2 2 1 1 2 1 2 1 1
## [1703] 2 2 1 2 2 2 2 2 1 1 2 2 2 1 1 1 1 1 2 2 1 1 2 2 2 2 2 2 1 1 1 1 1 2 2 1 2
## [1740] 1 1 1 2 2 1 1 2 1 2 2 2 2 1 2 2 1 1 1 1 1 1 2 2 2 2 1 2 1 1 2 2 2 1 1 1 2
## [1777] 2 2 1 1 2 2 1 2 2 1 1 1 1 1 2 2 1 1 2 2 2 1 1 1 2 1 1 2 1 2 2 2 2 2 1 2 1 1
## [1814] 2 2 2 1 1 1 2 2 1 2 1 1 2 2 2 1 2 1 2 2 1 1 2 1 1 1 1 2 1 2 1 2 1 2 1 1 2
## [1851] 1 1 1 2 2 2 2 1 1 2 1 2 1 2 2 1 2 2 1 2 2 1 1 1 1 1 2 2 2 2 2 1 1 2 2 2 1 2
## [1888] 1 1 2 2 1 2 2 2 2 1 1 2 2 2 2 2 1 1 2 1 1 1 1 2 2 1 2 1 2 2 1 2 2 2 1 1 2 2 1 2
## [1925] 2 1 1 1 2 2 2 1 2 2 1 1 2 2 1 2 2 1 2 2 2 2 2 2 1 2 1 2 1 2 1 1 1 1 1 2 2
## [1962] 2 1 2 1 2 2 1 2 2 1 1 1 2 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 2 1 2 2 2 1 1 1 2 2
## [1999] 2 2 1 1 1 1 1 1 2 2 2 1 2 2 2 1 1 1 1 1 2 1 2 1 1 2 1 2 2 1 1 1 2 2 2 2 2
## [2036] 1 2 1 1 2 1 1 1 2 2 2 1 2 1 1 1 1 1 2 1 2 1 1 1 1 2 2 2 2 2 2 1 1 2 2 1 2 2
## [2073] 1 2 2 1 1 1 1 1 1 2 2 1 1 1 2 1 2 1 2 2 1 2 2 2 2 1 2 1 2 2 1 1 1 1 1 1 2 1 2
## [2110] 1 1 2 1 2 2 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 2 2 2 1 2 1 2 2 1 2 2 2 1 2 2 1 2
## [2147] 1 1 2 1 2 1 2 1 1 2 1 1 2 2 1 2 1 1 1 1 1 2 2 2 1 1 1 1 1 1 1 1 1 2 2 1 1
## [2184] 1 2 2 1 2 1 2 1 1 1 2 1 2 2 2 2 2 2 1 2 2 2 2 1 1 2 1 2 1 2 2 2 1 2 2 1 2
## [2221] 2 2 2 2 1 2 1 2 2 2 1 1 2 2 1 2 2 1 2 2 1 1 1 2 2 2 1 2 2 1 1 1 2 2 2 1 2
## [2258] 2 2 1 1 1 2 2 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1 2 1 2 2 2 2 1 1 2 1 2 2 1
## [2295] 1 2 1 2 2 2 2 1 1 2 1 1 1 2 1 1 1 2 2 2 1 2 2 1 2 1 2 1 2 1 2 1 1 2 1 2 1 1 1
## [2332] 2 1 2 2 1 1 1 1 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 2 1 1 1 1 1 2 1 1 1
## [2369] 2 2 1 1 1 1 1 2 2 1 1 1 1 1 1 2 2 1 1 1 2 1 1 1 1 2 1 1 1 2 2 2 2 1 2 1 1
## [2406] 2 1 1 2 2 1 2 2 2 1 2 2 1 1 2 2 1 2 2 1 2 2 1 1 1 1 2 2 1 1 1 2 2 2 2 1 2
## [2443] 1 1 2 2 1 1 2 1 1 2 1 1 2 1 2 1 2 2 1 1 2 1 2 2 1 1 1 1 1 1 2 2 2 2 2 1 1
## [2480] 2 1 2 1 1 2 2 2 2 1 1 2 1 1 1 1 2 2 1 1 2 1 1 2 1 2 1 2 2 1 1 2 2 2 2 1 1
## [2517] 1 2 2 2 2 2 2 2 2 1 1 1 2 1 2 2 2 2 2 2 2 2 1 1 2 1 2 2 2 1 2 1 2 1 1 2 1
## [2554] 1 2 2 1 1 1 1 1 1 2 1 1 1 1 2 2 1 1 1 2 2 2 1 2 2 1 2 1 2 1 1 2 2 1 2 2 2
## [2591] 1 1 2 1 2 1 1 1 2 1 1 2 2 2 1 2 2 2 2 1 1 1 2 2 1 2 2 1 1 1 2 1 2 1 2 2 2
## [2628] 2 1 2 2 2 2 2 1 1 1 2 1 2 1 1 2 2 1 1 1 2 1 1 2 1 1 1 2 2 1 1 1 2 1 2 2 2
## [2665] 2 1 1 2 2 1 2 2 1 2 1 1 1 1 2 2 2 1 2 2 1 1 2 2 2 1 2 2 2 2 2 1 1 2 2 1 1 1
## [2702] 2 1 2 1 2 1 1 2 1 2 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2 1 1 2 2 1 2 2
## [2739] 1 1 2 1 1 1 2 2 2 1 1 2 2 2 2 2 1 1 1 2 2 2 1 1 2 2 1 2 2 2 1 1 1 1 2 2 2
```

```
## [2776] 1 1 2 2 2 1 2 2 2 1 1 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 2 1 2 1
## [2813] 2 2 2 1 2 1 1 2 1 2 1 1 2 1 1 1 1 1 2 2 1 2 1 1 1 1 1 2 1 1 1 1 2 2 2 2 1
## [2850] 1 2 2 2 1 2 1 1 1 1 1 1 1 2 1 2 2 1 2 2 2 1 2 2 2 2 2 2 1 2 1 1 2 2 2 1 1 2
## [2887] 2 1 2 2 2 2 2 2 1 2 2 2 1 1 1 2 2 2 2 2 2 2 1 2 2 1 1 1 1 1 1 1 2 2 1 1 2 2 2
## [2924] 2 2 2 2 2 1 1 1 1 1 1 2 1 2 2 2 2 1 2 1 2 1 2 2 2 2 1 1 2 1 1 1 1 2 1 2 2 2 2 1
## [2961] 2 2 1 1 1 2 1 1 1 1 2 2 1 2 1 2 1 1 2 2 1 2 2 2 2 2 2 2 1 1 1 2 1 2 2 2 2 1
## [2998] 2 2 2 1 1 1 2 1 2 2 2 2 1 1 2 1 1 2 1 2 2 2 2 1 2 2 2 2 2 1 2 2 1 2 2 2 1
## [3035] 2 2 1 1 1 1 1 1 2 2 1 2 1 2 2 2 2 2 2 2 1 1 1 2 1 1 2 2 1 1 2 2 1 2 1 2 2 1
## [3072] 1 2 1 1 1 1 1 2 1 2 2 1 1 1 1 2 2 2 2 1 2 2 1 1 2 2 1 2 1 2 2 2 2 2 2 1 1 1
## [3109] 1 2 2 1 2 1 1 1 1 1 2 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1 2 1 2 2 1 2 1 2 1 2 1
## [3146] 1 1 1 2 2 1 2 1 1 1 2 2 1 2 2 1 1 1 2 1 2 1 2 2 2 1 1 1 1 2 1 2 1 2 1 1 1
## [3183] 2 1 1 1 1 1 2 1 2 1 2 1 1 2 1 1 1 1 2 1 1 1 2 2 1 2 2 1 1 1 2 1 2 1 1 2 1
## [3220] 1 2 1 2 1 2 2 1 1 1 1 2 2 2 2 1 2 1 1 2 1 2 1 1 2 2 2 1 1 1 2 1 2 2 2 2 1
## [3257] 1 2 1 1 2 2 1 1 2 1 2 2 1 2 2 2 1 1 1 1 2 1 1 1 2 2 1 2 1 1 2 1 2 2 2 2 1
## [3294] 1 1 1 2 2 2 2 2 2 1 2 1 1 2 1 2 2 2 2 1 2 1 2 2 2 1 2 2 1 1 2 2 2 2 1 2 1
## [3331] 1 2 1 1 1 1 2 2 1 1 1 1 1 2 1 1 1 2 2 2 1 2 1 2 1 2 2 1 2 1 2 1 1 2 1 1 1
## [3368] 2 1 1 1 1 2 1 2 1 1 1 1 2 2 1 2 1 1 1 1 2 1 1 1 2 1 1 1 1 2 1 1 2 2 1 2 2
## [3405] 1 2 1 2 2 1 1 2 2 2 2 1 2 1 2 1 2 2 2 2 2 2 1 1 1 2 1 2 2 2 1 2 1 2 2 1 1 1
## [3442] 2 1 1 2 1 2 2 1 2 1 2 2 1 2 1 1 2 1 1 2 2 2 1 2 2 1 2 1 1 2 2 2 2 2 2 2 1
## [3479] 1 1 2 2 2 2 1 1 1 1 1 1 1 2 1 2 1 1 1 2 1 2 2 2 1 1 2 2 1 2 1 1 1 1 1 1 1
## [3516] 2 2 1 2 1 2 1 2 1 2 2 2 1 1 1 2 1 1 2 1 2 2 2 1 2 1 1 1 1 1 2 2 2 1 1 1 2
## [3553] 2 1 1 1 1 1 2 2 1 1 1 2 1 1 2 2 1 1 2 1 1 2 2 2 2 2 1 1 1 1 1 2 1 2 1 1 1 2
## [3590] 1 1 2 1 2 1 1 1 2 1 2 1 1 2 2 2 2 1 1 2 1 1 2 2 1 1 2 1 1 1 2 2 2 2 2 1 2 2
## [3627] 1 1 1 2 1 2 1 2 2 2 1 1 2 2 2 2 2 2 2 1 1 1 1 2 2 2 1 2 2 2 1 2 2 1 1 1 1
## [3664] 1 2 1 2 1 1 1 1 2 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2 1 2 1 2 1 2 2 1 2 1 1 1 2
## [3701] 1 2 2 2 1 1 2 1 1 1 2 1 2 2 2 1 2 2 2 1 2 1 1 2 1 1 1 2 1 1 1 1 1 2 1 1 2
## [3738] 2 2 1 2 2 1 1 2 1 2 1 1 1 2 1 2 1 2 2 1 1 1 1 2 2 2 2 2 2 1 2 1 1 1 1 1 1 2
## [3775] 2 1 1 2 2 2 2 2 1 2 1 2 2 1 1 2 2 1 2 2 2 2 2 1 2 2 1 2 1 1 1 1 1 1 1 1 1 2
## [3812] 2 1 2 1 1 2 2 1 2 1 1 2 2 1 1 1 1 1 2 1 1 2 1 2 1 2 2 1 1 2 1 2 1 2 1 1 1
## [3849] 2 1 2 1 1 2 1 1 2 2 1 2 1 2 2 1 2 2 2 1 1 1 1 1 2 2 1 1 1 1 1 1 2 2 1 1 1
## [3886] 1 2 1 1 1 1 2 2 1 1 1 1 2 1 2 2 1 2 2 1 2 2 1 1 1 1 2 1 2 1 1 2 1 2 2 1 1
## [3923] 1 1 2 1 2 2 2 1 2 2 1 1 1 2 1 1 2 2 1 2 1 2 2 1 1 1 1 1 1 1 2 2 1 2 2 1 2
## [3960] 1 2 2 1 2 1 1 1 1 1 1 2 1 1 2 2 2 2 2 1 1 2 2 1 1 1 2 1 2 2 2 2 2 2 1 1 2
## [3997] 2 2 1 2 2 2 2 2 2 2 2 1 2 1 1 2 1 2 2 1 2 1 2 2 2 1 1 2 1 2 1 2 1 1 2 1 2
## [4034] 2 1 1 2 2 2 1 2 2 1 2 1 2 1 1 1 2 2 2 1 1 2 1 2 2 1 2 1 1 1 2 2 1 2 2 2 2
## [4071] 2 1 1 2 2 1 2 1 1 2 1 2 1 2 2 1 2 2 1 1 1 1 1 2 2 1 1 2 1 2 1 2 1 1 1 1 2
## [4108] 2 2 1 2 1 1 1 2 1 1 1 1 1 2 2 2 2 2 2 2 1 2 1 2 2 2 1 1 1 1 1 2 1 2 1 2 2
## [4145] 2 2 2 2 2 1 2 1 1 2 2 2 1 1 2 1 1 1 2 2 1 2 2 2 2 1 1 2 2 1 1 1 1 2 2 1 1
## [4182] 2 2 1 2 1 1 1 1 1 1 1 2 2 2 1 2 2 2 1 1 2 1 2 1 2 2 1 2 1 1 1 2 2 2 2 2 2
## [4219] 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 1 1 2 1 1 2 1 2 1 1 2 2 2 1 2 2 2 1 2 1 2 2 2
## [4256] 2 1 1 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 1 2 1 1 2 1 1 2 1 1 1 1 2 1 1 2 2 1 2 2 2
## [4293] 2 2 2 2 1 1 1 1 2 2 2 1 2 1 1 1 1 1 2 1 1 2 1 2 1 2 2 2 1 1 1 2 2 2 2 1 1 2
## [4330] 2 2 2 2 2 2 2 1 2 1 2 1 1 1 1 1 2 1 2 2 2 1 2 1 1 2 1 1 1 1 1 1 2 2 1 1 2 1
## [4367] 2 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 2 2 1 2 1 2 2 1 2 2 2 2 1 2 2
## [4404] 2 1 2 2 1 2 1 1 1 1 1 1 2 2 2 2 2 1 2 2 2 2 2 1 1 1 1 2 2 1 1 2 2 1 2 2 2 1 1
## [4441] 1 2 2 2 1 1 2 2 2 2 1 1 2 2 1 2 2 1 2 2 1 2 2 1 1 2 1 1 2 2 1 1 2 2 1 2 1 2
## [4478] 1 1 1 2 1 1 2 1 1 1 1 1 1 1 1 2 2 1 1 2 1 2 2 2 2 2 2 2 1 1 1 1 1 2 2 1 2 1
## [4515] 1 1 2 1 2 2 1 2 1 1 2 1 1 1 2 1 1 1 2 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2
## [4552] 1 2 2 1 1 2 1 1 1 1 2 2 1 2 1 1 2 1 2 1 2 1 2 1 2 2 2 1 2 2 1 1 2 1 1 2 1
## [4589] 1 1 2 1 1 2 2 1 1 1 2 2 2 1 2 1 1 2 1 2 2 2 1 1 1 2 2 1 2 1 1 2 2 2 2 2 1
## [4626] 1 2 1 1 2 2 1 2 1 1 1 1 1 1 2 1 1 2 1 2 1 1 2 1 2 2 2 1 1 1 1 2 1 1 1 2 1
## [4663] 2 1 2 1 1 2 1 1 2 1 2 2 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 2 2 1 2 2 1 1 2 2 2
## [4700] 2 1 1 1 2 2 2 2 2 2 1 1 2 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 2 1 2 1 1 2 2 1
## [4737] 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 2 1 1 2 2 2 1 1 2 2 2 2 2 1 2 2 2 1 1 1 1 1 1
```

```
## [4774] 2 2 1 2 1 2 1 2 1 1 1 2 1 1 2 1 2 1 2 1 1 2 2 1 1 2 2 1 1 1 2 2 1 2 1 1 1
## [4811] 2 1 1 2 2 2 2 1 1 1 1 1 2 1 1 2 2 1 2 1 1 1 1 2 2 2 2 1 2 1 1 2 1 2 1 1 1
## [4848] 1 2 2 2 2 1 2 1 2 2 1 2 1 2 2 1 2 1 2 1 1 2 2 2 2 1 2 1 2 1 2 1 1 2 2 1 1
## [4885] 2 2 2 1 1 2 2 2 1 1 2 1 1 1 2 2 1 1 1 1 2 2 2 1 1 1 2 1 2 1 2 2 1 1 2 1 1
## [4922] 1 1 1 1 2 1 1 2 2 2 2 2 2 1 2 2 1 2 2 2 1 2 1 2 1 2 1 2 2 1 2 2 2 2 1 1
## [4959] 2 2 2 1 1 1 2 1 1 1 2 1 1 2 2 1 2 1 1 1 2 2 1 1 1 2 1 2 1 2 1 1 2 2 1 1 2
## [4996] 1 1 2 2 1
##
## Within cluster sum of squares by cluster:
## [1] 9940.347 7553.069
##  (between_SS / total_SS =  30.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

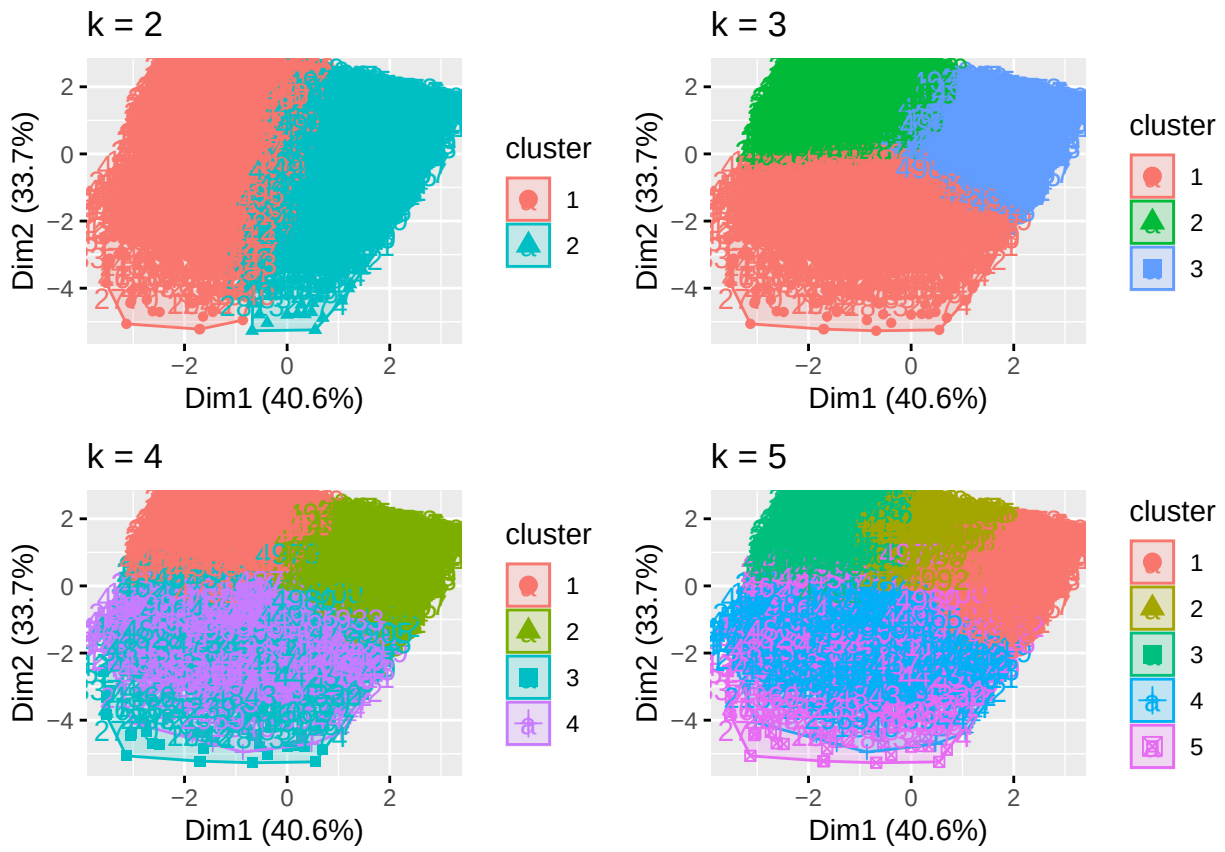
```r
# Visualise the cluster
clusplot(thera_bank.scaled, clust2$cluster,
         color=TRUE, shade=TRUE, labels=2, lines=1)
```




```r
# Create clusters for k=3, k=4 and k=5 for comparative analysis
clust3 <- kmeans(thera_bank.scaled, centers = 3, nstart = 5)
clust4 <- kmeans(thera_bank.scaled, centers = 4, nstart = 5)
clust5 <- kmeans(thera_bank.scaled, centers = 5, nstart = 5)
```

```
# Visualise clusters in 2 dimensions
k_clust_viz_2 = fviz_cluster(list(data = thera_bank.scaled,
                                  cluster = clust2$cluster)) +
  ggtitle("k = 2")
k_clust_viz_3 = fviz_cluster(list(data = thera_bank.scaled,
                                  cluster = clust3$cluster)) +
  ggtitle("k = 3")
k_clust_viz_4 = fviz_cluster(list(data = thera_bank.scaled,
                                  cluster = clust4$cluster)) +
  ggtitle("k = 4")
k_clust_viz_5 = fviz_cluster(list(data = thera_bank.scaled,
                                  cluster = clust5$cluster)) +
  ggtitle("k = 5")
```

```
# Visualise all 4 clustering plots together
grid.arrange(k_clust_viz_2, k_clust_viz_3, k_clust_viz_4, k_clust_viz_5, nrow = 2)
```

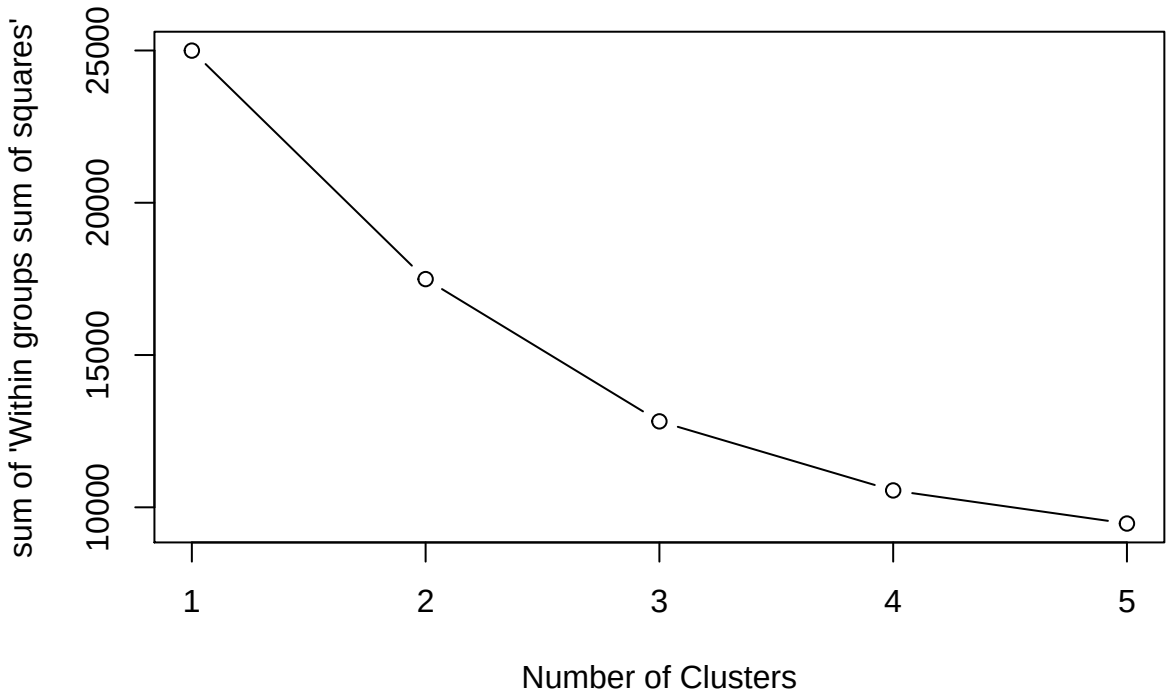


- To identify the optimal number of clusters, we can use the elbow algorithm method.

```
# To find the optimal numbner of clusters. Lets try K = 1 to 5 and for each plot the "sum of Within clu
totWss <- rep(0,5)
for(k in 1:5){
  set.seed(seed)
  clust <- kmeans(thera_bank.scaled, centers = k, nstart = 5)
  totWss[k] <- clust$tot.withinss
}
print(totWss)
```

```
## [1] 24995.000 17493.416 12821.894 10554.228  9467.579
```

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



- K = 3 might be a good choice using the elbow argument.
- "NbClust" is another package that the best clustering scheme using a number of experiments on the given data.

```
set.seed(seed)
nc <- NbClust(thera_bank.scaled, min.nc = 2, max.nc = 5, method="kmeans")
```

```
table(nc$Best.n[1,])
```

```
##
##  0  1  2  3  4  5
##  2  1  6 13  2  2
```

- Accordingly, NbClust suggests strongly that K = 3 would be the best choice.

```
# Adding the cluster numbers back to the dataset
data$Clusters = clust3$cluster
```

```
# Aggregate all columns except column 8 for each cluster by their means
custProfile = aggregate(data[, -c(4, 6, 8, 9, 10, 11, 12)],list(data$Cluster),FUN="mean")
print(custProfile)
```

```
##   Group.1      Age Experience    Income    CCAvg  Mortgage Clusters
## 1       1 43.68688  18.669554 147.69059 4.857463 116.42327        1
## 2       2 35.11489   9.904832  60.15138 1.383412  44.74951        2
```

```
## 3       3 55.53604  30.233826  58.94177 1.367514  45.13494        3
```

- Insights ——————————————————————————————-
- For clusters = 3, k-means demarcates the Thera bank's customer base into three tiers.
- K = 3 gives 3 clusters, out of which cluster 1 comprises of a high Income, CCAvg and Mortgage as well as mid-age and professional experience of 19. While cluster 2 and 3 customer base follows in that order in terms of income and CCAvg. However, there is a marginal difference in Mortgage for cluster 2 and 3.
- If we consider only 2 clusters we'll have a cluster of customers likely to take a loan and not likely to take a loan.
- A k = 4 would likely have given two high income groups and two low income groups
- We will use k = 3 cluster for deriving business insights.

5. Build CART to classify the right customers who have a higher probability of purchasing the loan

5.1 Model Building - Approach

1. Partition the data into train and test set.
2. Built a CART model on the train data. 2.1. Tune the model and prune the tree, if required. 2.2. Test the model on test set.

5.2 Split into train and test

```
set.seed(seed) # To ensure reproducibility
split <- sample.split(thera_bank$Personal_Loan, SplitRatio = 0.7)
train <- subset(thera_bank, split == TRUE)
test <- subset(thera_bank, split == FALSE)

nrow(train)
```

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

```
nrow(test)
```

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

```
# Check that the distribution of the dependent variable is similar in train and test sets
prop.table(table(thera_bank$Personal_Loan))
```

```
##
##     0     1
## 0.904 0.096
```

```
prop.table(table(thera_bank$Personal_Loan))
```

```
##
##     0     1
## 0.904 0.096
```

```
prop.table(table(thera_bank$Personal_Loan))
```

```
##
##     0     1
## 0.904 0.096
```

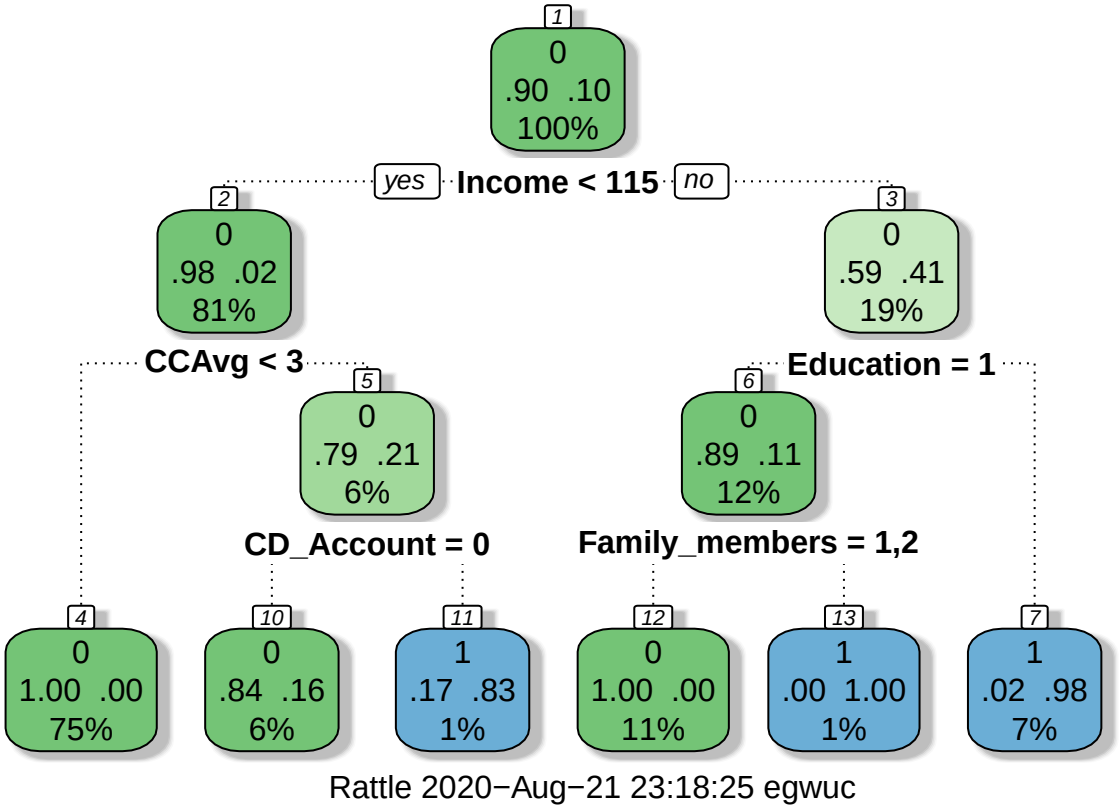
5.3 Build a CART model on the train dataset

```
# Setting the control parameters (to control the growth of the tree)
# Set the control parameters very low to let the tree grow deep

r.ctrl = rpart.control(minsplit = 50, minbucket = 10, cp = 0, xval = 10)

# Building the CART model

# formula - response variable~predictor variables
# data - dataset
# method - "class" - for classification, "anova" for regression
# control - tree control parameters

model1 <- rpart(formula = Personal_Loan ~ ., data = train, method = "class", control = r.ctrl)
model1
```

```
## n= 3500
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 3500 336 0 (0.904000000 0.096000000)
##    2) Income< 114.5 2827  57 0 (0.979837283 0.020162717)
##      4) CCAvg< 2.95 2613  11 0 (0.995790279 0.004209721) *
##      5) CCAvg>=2.95 214  46 0 (0.785046729 0.214953271)
##       10) CD_Account=0 196  31 0 (0.841836735 0.158163265) *
##       11) CD_Account=1 18   3 1 (0.166666667 0.833333333) *
##    3) Income>=114.5 673 279 0 (0.585438336 0.414561664)
##      6) Education=1 436  47 0 (0.892201835 0.107798165)
##       12) Family_members=1,2 389   0 0 (1.000000000 0.000000000) *
##       13) Family_members=3,4 47   0 1 (0.000000000 1.000000000) *
##      7) Education=2,3 237   5 1 (0.021097046 0.978902954) *
```

5.4 Visualise the decision tree on Model1

```
# Displaying the decision tree
fancyRpartPlot(model1)
```



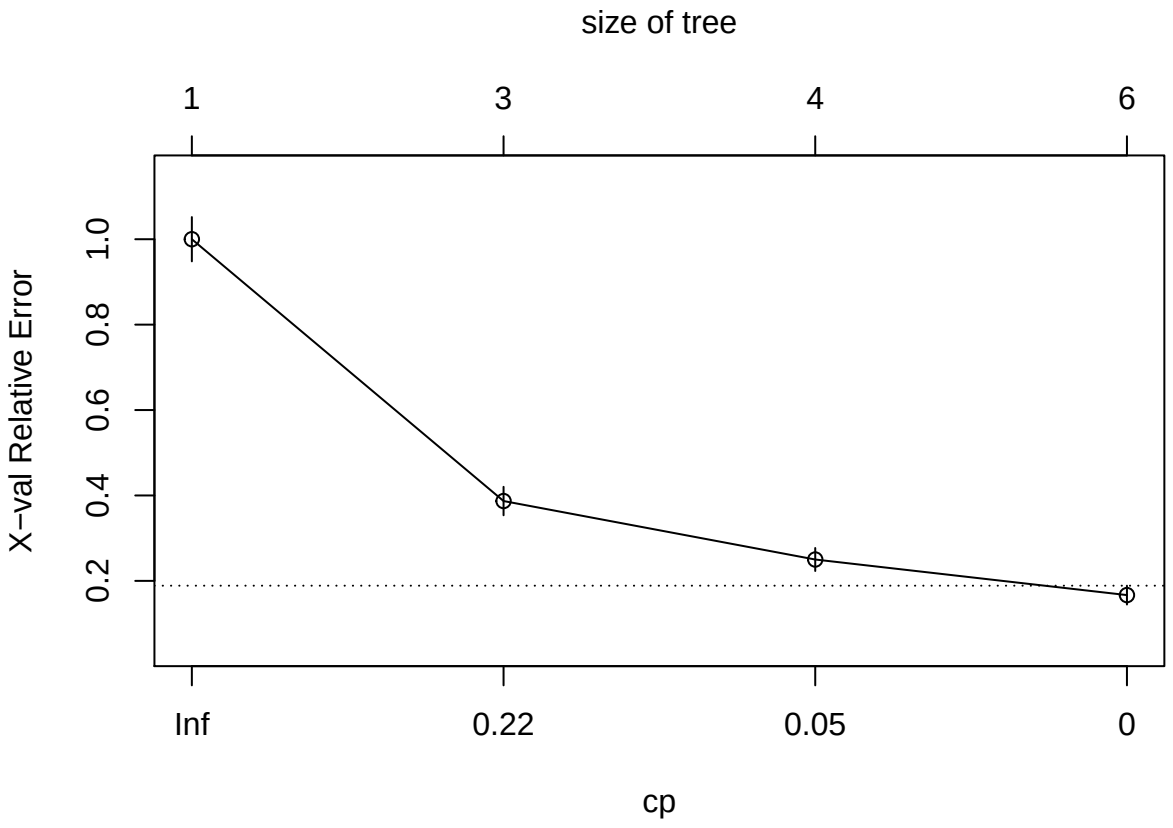
- We can still prune this tree.

5.5 Model Tuning

```
# The cost complexity table can be obtained using the printcp or plotcp functions
printcp(model1)
```

```
##
## Classification tree:
## rpart(formula = Personal_Loan ~ ., data = train, method = "class",
##     control = r.ctrl)
##
## Variables actually used in tree construction:
## [1] CCAvg          CD_Account     Education      Family_members Income
##
## Root node error: 336/3500 = 0.096
##
## n= 3500
##
##         CP nsplit rel error  xerror     xstd
## 1 0.337798      0   1.00000 1.00000 0.051870
## 2 0.139881      2   0.32440 0.38690 0.033298
## 3 0.017857      3   0.18452 0.25000 0.026948
## 4 0.000000      5   0.14881 0.16667 0.022093
```

```
plotcp(model1)
```



- The complex tree above can be pruned using a cost complexity threshold. Using a complexity threshold of 0.021 gives us a relatively simpler tree.

```
model2 = prune(model1, cp= 0.021, "CP")
printcp(model2)
```

5.5.1 Build a CART model for Model2

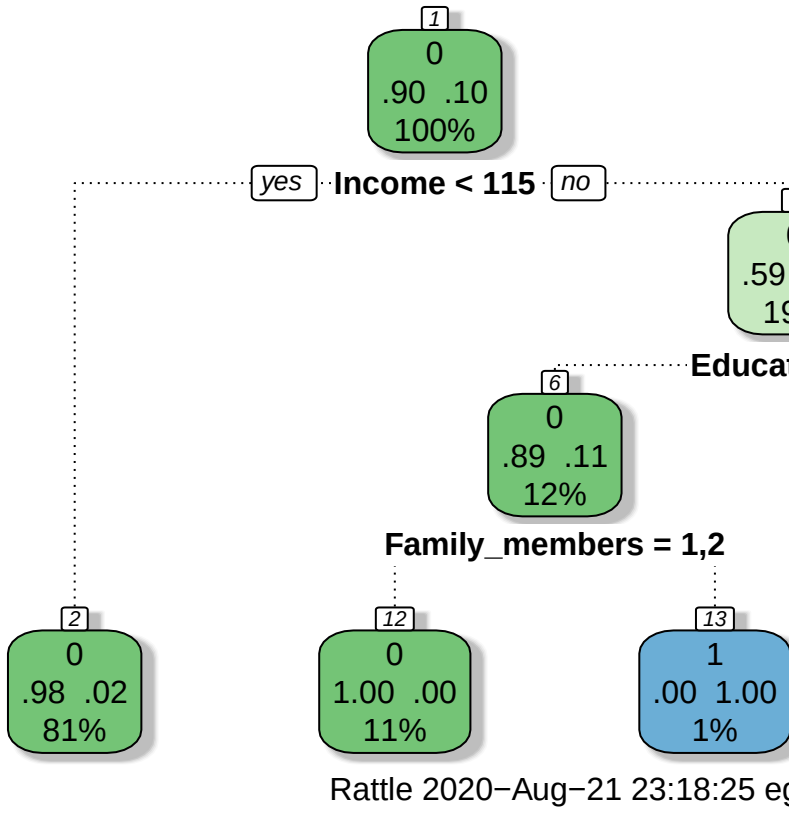
```
##
## Classification tree:
## rpart(formula = Personal_Loan ~ ., data = train, method = "class",
##     control = r.ctrl)
##
## Variables actually used in tree construction:
## [1] Education      Family_members Income
##
## Root node error: 336/3500 = 0.096
##
## n= 3500
##
##         CP nsplit rel error xerror     xstd
## 1 0.33780      0   1.00000 1.0000 0.051870
## 2 0.13988      2   0.32440 0.3869 0.033298
## 3 0.02100      3   0.18452 0.2500 0.026948
```

```
model2
```

```
## n= 3500
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 3500 336 0 (0.90400000 0.09600000)
##    2) Income< 114.5 2827  57 0 (0.97983728 0.02016272) *
##    3) Income>=114.5 673 279 0 (0.58543834 0.41456166)
##      6) Education=1 436  47 0 (0.89220183 0.10779817)
##       12) Family_members=1,2 389   0 0 (1.00000000 0.00000000) *
##       13) Family_members=3,4 47   0 1 (0.00000000 1.00000000) *
##      7) Education=2,3 237   5 1 (0.02109705 0.97890295) *
```

Variables actually used in tree construction: * Income
* Family members
* Education

```
#Displaying the decision tree
fancyRpartPlot(model2)
```



5.5.2 Visualise the decision tree on Model2

- Employees who do not have Income < 115 and whose education level is not less than 1.5 have a 98%.

5.6 Variable importance on Model1

```
# Variable importance is generally computed based on the corresponding reduction of predictive accuracy
# when the predictor of interest is removed.
model1$variable.importance
```

```
##      Education         Income Family_members          CCAvg     CD_Account
##     233.018589     169.111958     147.775024      86.857491      49.442449
##       Mortgage     Experience
##      21.271176       1.966402
```

5.6 Variable importance on Model2

```
# Variable importance is generally computed based on the corresponding reduction of predictive accuracy
# when the predictor of interest is removed.
model2$variable.importance
```

```
##      Education         Income Family_members          CCAvg     CD_Account
##     233.018589     169.111958     147.775024      69.287726      34.412028
##       Mortgage     Experience
##      21.271176       1.966402
```

- Variable importance is the same for both models.

5.7 Model Validation on Model1

```
# Predicting on the train dataset
train_predict.class1 <- predict(model1, train, type="class") # Predicted Classes
train_predict.score1 <- predict(model1, train) # Predicted Probabilities

# Create confusion matrix for train data predictions
tab.train1 = table(train$Personal_Loan, train_predict.class1)
tab.train1
```

```
##    train_predict.class1
##        0    1
##   0 3156    8
##   1   42  294
```

```
# Accuracy on train data
accuracy.train1 = sum(diag(tab.train1)) / sum(tab.train1)
accuracy.train1
```

```
## [1] 0.9857143
```

CART Model (model1) has 98.6% accuracy on train data. Baseline accuracy is 90.4% The model is an improvement on the baseline by 8.2% but let us see if we can improve it furthur using Random Forest Model.

Let us first see how the models performs on the test data

5.8 Model Validation on Model2

```
# Predicting on the train dataset
train_predict.class2 <- predict(model2, train, type="class") # Predicted Classes
train_predict.score2 <- predict(model2, train) # Predicted Probabilities

# Create confusion matrix for train data predictions
```

```
tab.train2 = table(train$Personal_Loan, train_predict.class2)
tab.train2
```

```
##    train_predict.class2
##        0    1
##   0 3159    5
##   1   57  279
```

```
# Accuracy on train data
accuracy.train2 = sum(diag(tab.train2)) / sum(tab.train2)
accuracy.train2
```

```
## [1] 0.9822857
```

CART Model (model2) has 98.2% accuracy on train data. Baseline accuracy is 90.4% The model is an improvement on the baseline by 7.8%, down 0.4% from model1.

5.9 Model Evaluation

```
# Predicting on the test dataset using MODEL 1
test_predict.class1 <- predict(model1, test, type="class") # Predicted Classes
test_predict.score1 <- predict(model1, test) # Predicted Probabilities

# Create confusion matrix for test data predictions (using MODEL 1)
tab.test1 = table(test$Personal_Loan, test_predict.class1)
tab.test1
```

5.9.1 MODEL 1

```
##    test_predict.class1
##        0    1
##   0 1352    4
##   1   28  116
```

```
# Accuracy on train data (MODEL 1 predictions)
accuracy.test1 = sum(diag(tab.test1)) / sum(tab.test1)
accuracy.test1
```

```
## [1] 0.9786667
```

```
# Predicting on the test dataset using MODEL 2
test_predict.class2 <- predict(model2, test, type="class") # Predicted Classes
test_predict.score2 <- predict(model2, test) # Predicted Probabilities

# Create confusion matrix for test data predictions (using MODEL 2)
tab.test2 = table(test$Personal_Loan, test_predict.class2)
tab.test2
```

5.9.2 MODEL 2

```
##    test_predict.class2
##        0    1
##   0 1355    1
##   1   34  110
```

```r
# Accuracy on train data (MODEL 2 predictions)
accuracy.test2 = sum(diag(tab.test2)) / sum(tab.test2)
accuracy.test2
```

```
## [1] 0.9766667
```

5.10 Comparing Models

```r
Model_Name = c("Baseline", "Model1", "Model2")
Train_Accuracy_perc = c(90, accuracy.train1*100, accuracy.train2*100)
Test_Accuracy_perc = c(90, accuracy.test1*100, accuracy.test2*100)
output = data.frame(Model_Name,Train_Accuracy_perc,Test_Accuracy_perc)
output
```

```
##   Model_Name Train_Accuracy_perc Test_Accuracy_perc
## 1   Baseline            90.00000           90.00000
## 2     Model1            98.57143           97.86667
## 3     Model2            98.22857           97.66667
```

5.11 Conclusion

- Model1 performs very well both on the test and train data.
- We will use model1 as the final model.
- Model1 Decision Tree has more accuracy than baseline model.
- Accuracy on the Training Data: 98.6%.
- Accuracy on the Test Data: 97.9%.
- Accuracy for test data is almost inline with training data.
- This tells that the model is neither underfit nor overfit.

6. Build a Random Forest Model to classify the right customers who have a higher probability of purchasing the loan

6.1 Random Forest Approach

1. Build a Random Forest model on the train set 2.1 Tune the model 2.2 Test the model performance on test set.
2. Compare the performance of the two models and pick the best model as your final model

6.2 Build the first RF model

```r
set.seed(seed)
# Formula - response variable ~ predictor variables
# To build a classification random forest the response variable should be converted to a factor if it i
# data -  dataset to train the model on
## Random Forest hyperparameters
# ntree - Total number of trees are to be constructed
# mtry - number of variables tried at each split
# importance - Set TRUE to assess variable importance

rf_model1 = randomForest(
  Personal_Loan ~ .,
```

```
  data = train,
  ntree = 501,
  mtry = 5,
  nodesize = 10,
  importance = TRUE
  )
```

Print the model to see the OOB and error rate

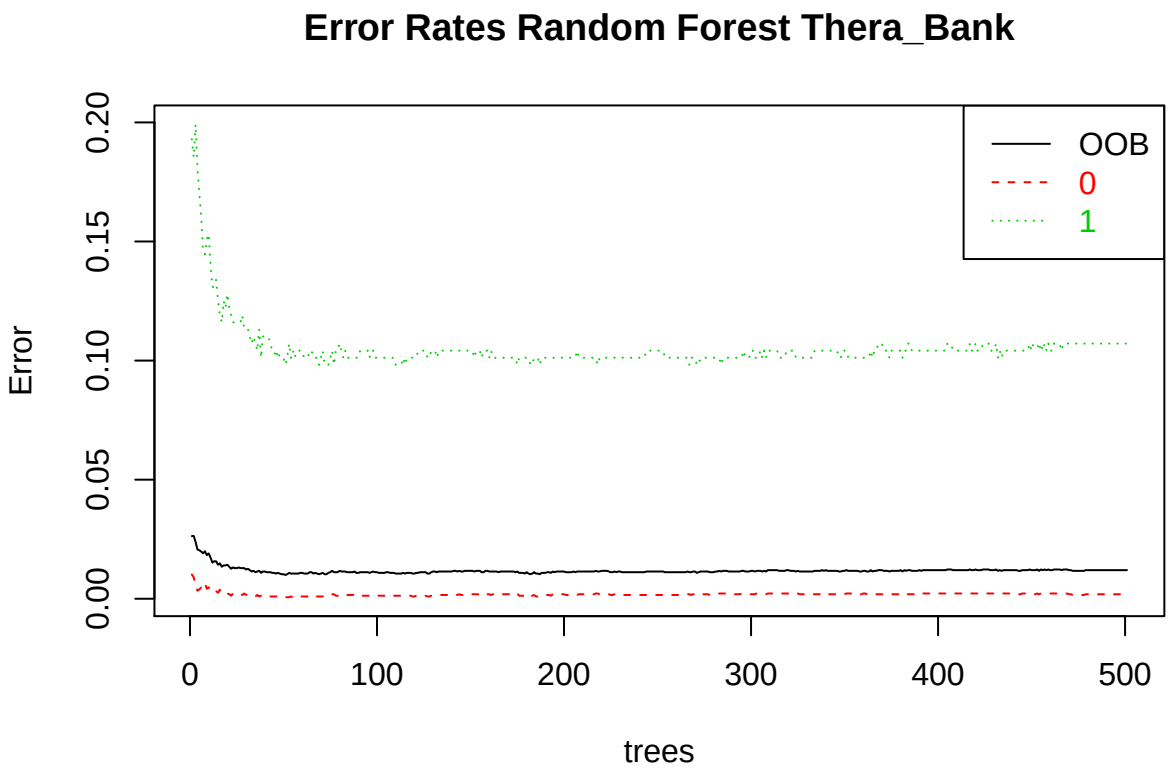
Out-of-bag refers to a scenario when the combined prediction from a set of trees is used to predict the dependent variable for all those sample not used to build these trees (hence the name out-of-bag).

```
print(rf_model1)
```

```
##
## Call:
##  randomForest(formula = Personal_Loan ~ ., data = train, ntree = 501,      mtry = 5, nodesize = 10, i
##                Type of random forest: classification
##                      Number of trees: 501
## No. of variables tried at each split: 5
##
##         OOB estimate of  error rate: 1.2%
## Confusion matrix:
##      0   1 class.error
## 0 3158   6 0.001896334
## 1   36 300 0.107142857
```

Plot the model to determine the optimum number of trees

```
plot(rf_model1, main="")
legend("topright", c("OOB", "0", "1"), text.col=1:6, lty=1:3, col=1:3)
title(main="Error Rates Random Forest Thera_Bank")
```



- The plot reveals that anything more than, say 50 trees, is really not that valuable.

List the importance of the variables. Larger the MeanDecrease values, the more important the variable. Look at the help files to get a better sense of how these are computed.

```
importance(rf_model1)
```

```
##                                0          1 MeanDecreaseAccuracy MeanDecreaseGini
## Age                   13.2278033  0.6459467             13.320794       10.6084864
## Experience            12.3384504 -0.1631246             12.342909        9.6981414
## Income               164.7273239 96.1097144            168.161263      186.3400430
## Family_members       105.2085112 50.2729051            105.603559       78.0840080
## CCAvg                 33.9305146 30.6238702             37.652751       74.5246431
## Education            183.7405009 87.0233998            189.058550      163.9789028
## Mortgage               7.4943457 -3.2754139              6.399859        8.7882173
## Securities_Account    -0.4003204 -0.8534174             -0.877141        0.7886145
## CD_Account            11.6266933 10.1516140             15.131853       27.8053658
## Online                 4.2243753  1.5607480              4.792808        1.2547299
## CreditCard             5.0893777  1.9373467              5.832846        1.7300173
```

The variables important include: * Education * Income * Family_members * CCAVG

6.3 Tune the Random Forest Model

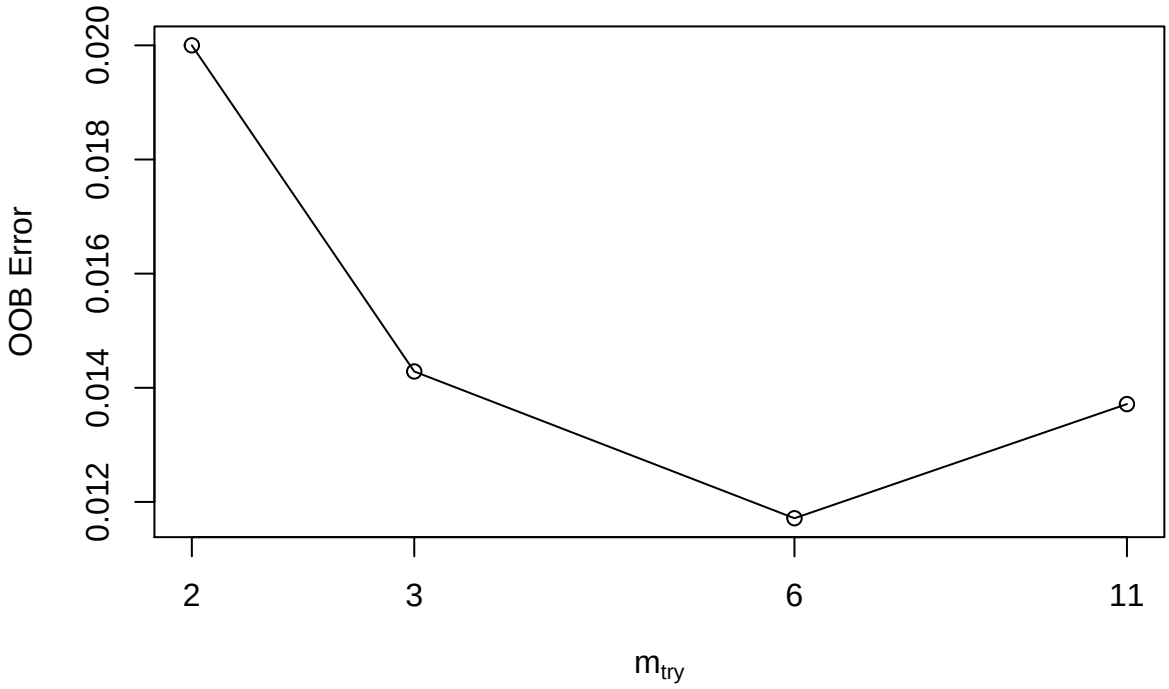
Let us tune the randomforest model by trying different m values Tune the RF model to find out the best mtry We will take ntree = 51 (odd number of trees are preferred) The returned forest, rf_model2 is the one corresponding to the best m

```r
# Check the column number of the response variable
names(train)
```

```
##  [1] "Age"                "Experience"         "Income"
##  [4] "Family_members"     "CCAvg"              "Education"
##  [7] "Mortgage"           "Personal_Loan"      "Securities_Account"
## [10] "CD_Account"         "Online"             "CreditCard"
```

```r
set.seed(seed) # To ensure reproducibility

rf_model2 = tuneRF(x = train[, -8], # matrix or data frame of predictor/independent variables
                   y = train$Personal_Loan, # response vector (factor for classification, numeric for re
                   mtrystart = 5, # starting value of mtry
                   stepfactor=1.5, # at each iteration, mtry is inflated (or deflated) by this value
                   ntree=51, # number of trees built for each mtry value
                   improve=0.0001, # the (relative) improvement in OOB error must be by this much for th
                   nodesize=10, # Minimum size of terminal nodes
                   trace=TRUE, # prints the progress of the search
                   plot=TRUE, # to get the plot of the OOB error as function of mtr
                   doBest=TRUE, # return a forest using the optimal mtry found
                   importance=TRUE #
                   )
```

```
## mtry = 3  OOB error = 1.43%
## Searching left ...
## mtry = 2     OOB error = 2%
## -0.4 0.0001
## Searching right ...
## mtry = 6     OOB error = 1.17%
## 0.18 0.0001
## mtry = 11    OOB error = 1.37%
## -0.1707317 0.0001
```



- The optimal number of mtry is 6.
- tuneRF returns rf_model2. It is the random forest of 51 trees built with m = 6.

6.4 Model Validation

```
# Predicting on the train dataset
train_predict.class_RF <- predict(rf_model2, train, type = "class") # Predicted Classes
train_predict.score_RF <- predict(rf_model2, train, type = 'prob') # Predicted Probabilities

# Create confusion matrix for train data predictions
tab.train_RF = table(train$Personal_Loan, train_predict.class_RF)
tab.train_RF
```

```
##    train_predict.class_RF
##        0    1
##   0 3162    2
##   1   23  313
```

```
# Accuracy on train data
accuracy.train_RF = sum(diag(tab.train_RF)) / sum(tab.train_RF)
accuracy.train_RF
```

```
## [1] 0.9928571
```

- RandomForest mode (rf_model2) has 99.3% accuracy on train data.
- Baseline accuracy is 90.4%
- This is an improvement over the baseline and the CART model!

Let us see how the models performs on the test data

6.5 Model Evaluation

```
# Predicting on the test dataset
test_predict.class_RF <- predict(rf_model2, test, type = "class") # Predicted Classes
test_predict.score_RF <- predict(rf_model2, test, type = 'prob') # Predicted Probabilities

# Create confusion matrix for test data predictions
tab.test_RF = table(test$Personal_Loan, test_predict.class_RF)
tab.test_RF
```

```
##    test_predict.class_RF
##        0    1
##   0 1352    4
##   1   23  121
```

```
# Accuracy on test data
accuracy.test_RF = sum(diag(tab.test_RF)) / sum(tab.test_RF)
accuracy.test_RF
```

```
## [1] 0.982
```

- The model has good performance on test data too.
- An accuracy of 99.3% on train data and 98.1% on test data indicates that this is a good model, neither overfit nor underfit.
- This model is an improvement on the baseline model and will help us classify the right customers who have a higher probability of purchasing the loan.

6.6 Comparing Models

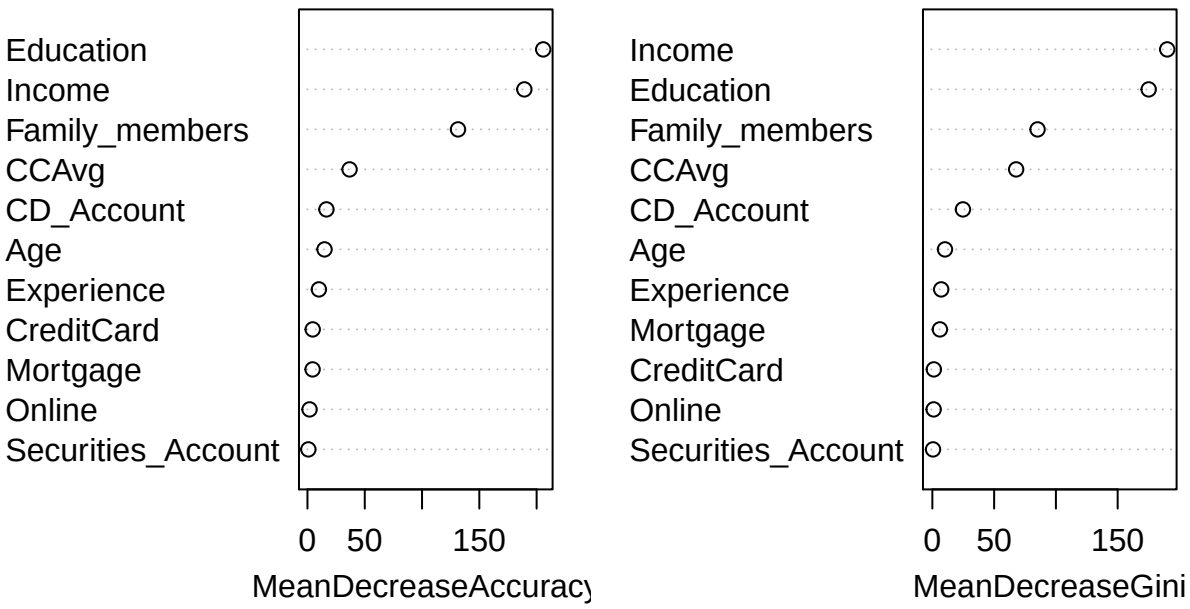
```
Model_Name = c("Baseline", "CART", "Random Forest")
Train_Accuracy_perc = c(90, accuracy.train1*100, accuracy.train_RF*100)
Test_Accuracy_perc = c(90, accuracy.test1*100, accuracy.test_RF*100)
output = data.frame(Model_Name, Train_Accuracy_perc, Test_Accuracy_perc)
output
```

```
##      Model_Name Train_Accuracy_perc Test_Accuracy_perc
## 1      Baseline            90.00000           90.00000
## 2          CART            98.57143           97.86667
## 3 Random Forest            99.28571           98.20000
```

- Random Forest model is a good predictor model for classifying customers who will purchase the loan.
- We will use rf_model2 as our final model.

Variable Importance Final Model

```
varImpPlot(rf_model2, sort = TRUE)
```



- It is worth noting that the most important variables indicated by the final CART model and final Random forest model are very similar.
- Random forest gives us a much improved model as compared to CART
- The decision tree model gives high importance to a particular set of features. However, the random forest chooses features randomly during the training process. Therefore, it does not depend highly on any specific set of features.
- The random forest can generalize over the data in a better way. This randomized feature selection makes random forest much more accurate than a decision tree.

For quick referencce: Variables actually used in CART tree construction-

- Education
- Income
- Family_members
- CCAvg
- CD_Account
- Mortgage
- Experience
- Age

7.Confusion Matrix

We will compare all the 4 models that we created earlier - model1, model2, rf_model1, rf_model2

Predict Attrition class and probability for all 4 models

```
# Predict on test data using cart_model1
model1_predict_class = predict(model1, test, type = 'class')
model1_predict_score = predict(model1, test, type = 'prob')

# Predict on test data using cart_model2
model2_predict_class = predict(model2, test, type = 'class')
model2_predict_score = predict(model2, test, type = 'prob')

# Predict on test data using rf_model1
rf_model1_predict_class = predict(rf_model1, test, type = 'class')
rf_model1_predict_score = predict(rf_model1, test, type = 'prob')

# Predict on test data using rf_model2
rf_model2_predict_class = predict(rf_model2, test, type = 'class')
rf_model2_predict_score = predict(rf_model2, test, type = 'prob')
```

```
# Create Confusion Matrix for all the four models
conf_mat_model1 = table(test$Personal_Loan, model1_predict_class)
conf_mat_model1
```

```
##    model1_predict_class
##        0    1
##   0 1352    4
##   1   28  116
```

```
conf_mat_model2 = table(test$Personal_Loan, model2_predict_class)
conf_mat_model2
```

```
##    model2_predict_class
##        0    1
##   0 1355    1
##   1   34  110
```

```
conf_mat_rf_model1 = table(test$Personal_Loan, rf_model1_predict_class)
conf_mat_rf_model1
```

```
##    rf_model1_predict_class
##        0    1
##   0 1353    3
##   1   25  119
```

```
conf_mat_rf_model2 = table(test$Personal_Loan, rf_model2_predict_class)
conf_mat_rf_model2
```

```
##    rf_model2_predict_class
##        0    1
##   0 1352    4
##   1   23  121
```

7.1 Accuracy

```
# Accuracy of models on test data
accuracy_model1 = sum(diag(conf_mat_model1)) / sum(conf_mat_model1)

accuracy_model2 = sum(diag(conf_mat_model2)) / sum(conf_mat_model2)

accuracy_rf_model1 = sum(diag(conf_mat_rf_model1)) / sum(conf_mat_rf_model1)

accuracy_rf_model2 = sum(diag(conf_mat_rf_model2)) / sum(conf_mat_rf_model2)
```

7.2 Sensitivity / Recall

```
# Sensitivity of models on test data
sensitivity_model1 = conf_mat_model1[2,2] / sum(conf_mat_model1['1',])

sensitivity_model2 = conf_mat_model2[2,2] / sum(conf_mat_model2['1',])

sensitivity_rf_model1 = conf_mat_rf_model1[2,2] / sum(conf_mat_rf_model1['1',])

sensitivity_rf_model2 = conf_mat_rf_model2[2,2] / sum(conf_mat_rf_model2['1',])
```

7.3 Specificity

```
# Specificity of models on test data
specificity_model1 = conf_mat_model1[1,1] / sum(conf_mat_model1['0',])

specificity_model2 = conf_mat_model2[1,1] / sum(conf_mat_model2['0',])

specificity_rf_model1 = conf_mat_rf_model1[1,1] / sum(conf_mat_rf_model1['0',])

specificity_rf_model2 = conf_mat_rf_model2[1,1] / sum(conf_mat_rf_model2['0',])
```

7.4 Precision

```
# Precision of models on test data
precision_model1 = conf_mat_model1[2,2] / sum(conf_mat_model1[,'1'])

precision_model2 = conf_mat_model2[2,2] / sum(conf_mat_model2[,'1'])

precision_rf_model1 = conf_mat_rf_model1[2,2] / sum(conf_mat_rf_model1[,'1'])

precision_rf_model2 = conf_mat_rf_model2[2,2] / sum(conf_mat_rf_model2[,'1'])
```

7.5 KS

```
# Using library ROCR functions prediction and performance
pred_model1 = prediction(model1_predict_score[, 2], test$Personal_Loan)
perf_model1 = performance(pred_model1,"tpr","fpr")
ks_model1 = max(attr(perf_model1,'y.values')[[1]] - attr(perf_model1,'x.values')[[1]])

pred_model2 = prediction(model2_predict_score[, 2], test$Personal_Loan)
perf_model2 = performance(pred_model2,"tpr","fpr")
ks_model2 = max(attr(perf_model2,'y.values')[[1]] - attr(perf_model2,'x.values')[[1]])
```

```
pred_rf_model1 = prediction(rf_model1_predict_score[, 2], test$Personal_Loan)
perf_rf_model1 = performance(pred_rf_model1,"tpr","fpr")
ks_rf_model1 = max(attr(perf_rf_model1,'y.values')[[1]] - attr(perf_rf_model1,'x.values')[[1]])

pred_rf_model2 = prediction(rf_model2_predict_score[, 2], test$Personal_Loan)
perf_rf_model2 = performance(pred_rf_model2,"tpr","fpr")
ks_rf_model2 = max(attr(perf_rf_model2,'y.values')[[1]] - attr(perf_rf_model2,'x.values')[[1]])
```

7.6 AUC

```
# Using library ROCR
auc_model1 = performance(pred_model1, measure = "auc")
auc_model1 = auc_model1@y.values[[1]]

auc_model2 = performance(pred_model2, measure = "auc")
auc_model2 = auc_model2@y.values[[1]]

auc_rf_model1 = performance(pred_rf_model1, measure = "auc")
auc_rf_model1 = auc_rf_model1@y.values[[1]]

auc_rf_model2 = performance(pred_rf_model2, measure = "auc")
auc_rf_model2 = auc_rf_model2@y.values[[1]]
```

7.7 Gini

```
# Using library ineq
gini_model1 = ineq(model1_predict_score[, 2],"gini")

gini_model2 = ineq(model2_predict_score[, 2],"gini")

gini_rf_model1 = ineq(rf_model1_predict_score[, 2],"gini")

gini_rf_model2 = ineq(rf_model2_predict_score[, 2],"gini")
```

7.8 Concordance - Discordance

```
concordance_model1 = Concordance(actuals = ifelse(test$Personal_Loan == '1', 1,0), predictedScores = if

concordance_model2 = Concordance(actuals = ifelse(test$Personal_Loan == '1', 1,0), predictedScores = if

concordance_rf_model1 = Concordance(actuals = ifelse(test$Personal_Loan == '1', 1,0), predictedScores =

concordance_rf_model2 = Concordance(actuals = ifelse(test$Personal_Loan == '1', 1,0), predictedScores =
```

7.9 Comparing models

```
model1_metrics = c(accuracy_model1, sensitivity_model1, specificity_model1, precision_model1, ks_model1

model2_metrics = c(accuracy_model2, sensitivity_model2, specificity_model2, precision_model2, ks_model2

rf_model1_metrics = c(accuracy_rf_model1, sensitivity_rf_model1, specificity_rf_model1, precision_rf_mo
```

```r
rf_model2_metrics = c(accuracy_rf_model2, sensitivity_rf_model2, specificity_rf_model2, precision_rf_mo

comparison_table = data.frame(model1_metrics, model2_metrics, rf_model1_metrics, rf_model2_metrics)

rownames(comparison_table) = c("Accuracy", "Sensitivity", "Specificity", "Precision", "KS", "Auc", "Gini

comparison_table
```

```
##             model1_metrics model2_metrics rf_model1_metrics rf_model2_metrics
## Accuracy         0.9786667      0.9766667         0.9813333         0.9820000
## Sensitivity      0.8055556      0.7638889         0.8263889         0.8402778
## Specificity      0.9970501      0.9992625         0.9977876         0.9970501
## Precision        0.9666667      0.9909910         0.9754098         0.9680000
## KS               0.9242871      0.7631514         0.9775688         0.9775688
## Auc              0.9825877      0.8959921         0.9985251         0.9985558
## Gini             0.8738842      0.7619826         0.9000792         0.9030452
## Concordance      0.8031793      0.7633255         0.8245606         0.8377991
```

7.10 Conclusion

- Accuracy is the percentage of correct predictions for the test data. Based on this, the Random Forest model2 (rf_model2) is the most accurate.
- Sensitivity is the proportion of total positives that were correctly identified. Based on this, the Random Forest model2 (rf_model2) is the most sensitivty. This model will help the bank to better predict susceptible customers willing to take a personal loan.
- Specificity is the proportion of total negatives that were correctly identified. Based on this, the CART model2 (model2) is the most specific.
- Precision is the fraction of relevant observations (true positives) among all of the observations which were predicted to belong in a certain class. Based on this, the CART model2 (model2) is the most precise.
- KS is used for decisions like how many to target - where we are more concerned about the predictive power of our best groups. Here, a higher KS is better and based on that, the Random Forest model1 and model2 (rf_model1 and rf_model2) is better.
- Area under the curve (AUC) is a measure of how good a model is, indicating a higher percentage is better. Based on this, the Random Forest model2 (rf_model2) is better.
- Gini is measured in values between 0 and 1, where a score of 1 means that the model is 100% accurate in predicting the outcome and vice versa. Based on this, the Random Forest model2 (rf_model2) is better.
- Concordance is also used to access the model's predictive power. Based on this, the Random Forest model2 (rf_model2) is better.
- Based on the confusion matrix, the Random Forest model2 (rf_model2) does a better job in predicting the probability of a customer taking a loan from Thera Bank since the Sensitvity of the Random Forest model is 84.0% and accuracy is 98.2%.
- Based on KS, AUC, Gini and Concordance the Random Forest model2 (rf_model2) does a better job in predicting the probability of a customer taking a loan from Thera Bank with a score of 97.8%, 99.8%, 90.3% and 83.8% respectively.
- From the above, we can see that the Random Forest model2 (rf_model2) is better in most parameters in comparison to model1, model2 and rf_model2. As a result, the rf_model2 performed the best.

8. Appendix A – Source Code

```
#=====================================================================
#
# Exploratory Data Analysis - CardioGoodFitness
#
#=====================================================================

# Environment set up and data import

# Invoking libraries
library(readxl) # To import excel files
library(ggplot2) # To create plots
library(corrplot) # To plot correlation plot between numerical variables
library(dplyr) # To manipulate dataset
library(gridExtra) # To plot multiple ggplot graphs in a grid
library(DataExplorer) # visual exploration of data
library(mice) # Multivariate Imputation via Chained Equations; takes care of uncertainty in missing valu
library(cluster)
library(factoextra) # extract and visualize the results of multivariate data analysis
library(NbClust) # to find optimal number of clusters
library(caTools) # Split Data into Test and Train Set
library(rpart) # To build CART decision tree
library(rattle) # To visualise decision tree
library(randomForest) # To build a Random Forest
library(ROCR) # To visualise the performance classifiers
library(ineq) # To calculate Gini
library(InformationValue) # For Concordance-Discordance
library(knitr) # Necessary to generate source codes from a .Rmd File
library(markdown) # To convert to HTML
library(rmarkdown) # To convret analyses into high quality documents

# Set working directory
setwd("C:/Users/egwuc/Desktop/PGP-DSBA-UT Austin/Machine Learning/Week 5 - Project/")

# Read input file
thera_bank <- read_excel("Thera Bank_Personal_Loan_Modelling-dataset-1.xlsx", sheet = 2)

# Global options settings
options(scipen = 999) # turn off scientific notation like 1e+06

# Check dimension of dataset
dim(thera_bank)

# Check first 6 rows(observations) of dataset
head(thera_bank)
tail(thera_bank)

# Convert column names to appropriate column names
colnames(thera_bank) <- c("ID", "Age", "Experience", "Income", "ZIP_Code", "Family_members", "CCAvg", "
                          "Personal_Loan", "Securities_Account", "CD_Account", "Online", "CreditCard")

# Check structure of dataset
str(thera_bank)
```

```
# Change personal_loan to factor variable
thera_bank$Personal_Loan <- as.factor(thera_bank$Personal_Loan)

# Get summary of dataset
summary(thera_bank)

# Dropping ID and Zip Code column
thera_bank <- thera_bank[, -1]
thera_bank <- thera_bank[, -4]

# View the dataset
View(thera_bank)

# Filter out values less than 0 in Experience
filter(thera_bank, Experience < 0)

# Replace values less than 0 in Experience with 0
thera_bank$Experience <- replace(thera_bank$Experience, thera_bank$Experience<0, 0)

# Check if any values in less than 0 in Experience
filter(thera_bank, Experience < 0)

# How many missing vaues do we have?
sum(is.na(thera_bank))

# What columns contain missing values?
colSums(is.na(thera_bank))

# Use functions and algorithms to impute the missing values
data1 <- thera_bank
sum(is.na(data1))
md.pattern(data1)
init.impute = mice(data1, m = 5, method = "pmm", seed = 1000)
thera_bank = complete(init.impute, 2)
md.pattern(thera_bank)
sum(is.na(thera_bank))

# Distribution of the dependent variable
prop.table(table(thera_bank$Personal_Loan))

plot_histogram_n_boxplot = function(variable, variableNameString, binw){

  a = ggplot(data = thera_bank, aes(x= variable)) +
    labs(x = variableNameString,y ='count')+
    geom_histogram(fill = 'green',col = 'white', binwidth = binw) +
    geom_vline(aes(xintercept = mean(variable)),
               color = "black", linetype = "dashed", size = 0.5)

  b = ggplot(data = thera_bank, aes('',variable))+
    geom_boxplot(outlier.colour = 'red',col = 'red', outlier.shape = 19)+
    labs(x = '', y = variableNameString) + coord_flip()
  grid.arrange(a,b,ncol = 2)
}
```

```
plot_histogram_n_boxplot(thera_bank$Age, 'Age', 2)

plot_histogram_n_boxplot(thera_bank$Experience, 'Experience', 2)

plot_histogram_n_boxplot(thera_bank$Income, 'Income', 10)

plot_histogram_n_boxplot(thera_bank$Family_members, 'Family Members', 1)

plot_histogram_n_boxplot(thera_bank$CCAvg, 'CCAvg', 1)

plot_histogram_n_boxplot(thera_bank$Education, 'Education', 1)

plot_histogram_n_boxplot(thera_bank$Mortgage, 'Mortgage', 100)

plot_histogram_n_boxplot(thera_bank$Securities_Account, 'Securities Account', 1)

plot_histogram_n_boxplot(thera_bank$CD_Account, 'CD Account', 1)

plot_histogram_n_boxplot(thera_bank$Online, 'Online', 1)

plot_histogram_n_boxplot(thera_bank$CreditCard, 'Credit Card', 1)

# Function to draw percent stacked barchart to see the effect of independent variables
# on the probability of personal loan using ggplot
plot_stacked_barchart = function(variable, variableNameString){
  ggplot(thera_bank, aes(fill = Personal_Loan, x = variable)) +
    geom_bar(position="fill")+
    labs(title = variableNameString, y = '', x = '')+
    scale_fill_manual(values=c("#0073C2FF", "#EFC000FF"))

}

plot_stacked_barchart(thera_bank$Age, 'Age')

plot_stacked_barchart(thera_bank$Experience, 'Experience')

plot_stacked_barchart(thera_bank$Income, 'Income')

plot_stacked_barchart(thera_bank$Family_members, 'Family Members')

plot_stacked_barchart(thera_bank$Education, 'Education')

plot_stacked_barchart(thera_bank$Securities_Account, 'Securities Account')

plot_stacked_barchart(thera_bank$CD_Account, 'CD Account')

plot_stacked_barchart(thera_bank$Online, 'Online')

plot_stacked_barchart(thera_bank$CreditCard, 'Credit Card')

# Numeric variables in the data
num_vars = sapply(thera_bank, is.numeric)
```

```
# Correlation Plot
corrplot(cor(thera_bank[,num_vars]), method = 'number')

# Scale the dataset to reduce the influence from variables with high values
data <- thera_bank

# Change Family_members, Education, Securities_Account, CD_Account, Online and CreditCard to factor vari
thera_bank$Family_members <- as.factor(thera_bank$Family_members)
thera_bank$Education <- as.factor(thera_bank$Education)
thera_bank$Securities_Account <- as.factor(thera_bank$Securities_Account)
thera_bank$CD_Account <- as.factor(thera_bank$CD_Account)
thera_bank$Online <- as.factor(thera_bank$Online)
thera_bank$CreditCard <- as.factor(thera_bank$CreditCard)

str(data)
view(data)

thera_bank.scaled <- scale(data[, -c(4, 6, 8, 9, 10, 11, 12)])

# Determine the optimum number of clusters (find optimal k)
seed <- 1000
set.seed(seed) # kmeans uses a randomized starting point for cluster centroids
clust2 = kmeans(thera_bank.scaled, centers = 2, nstart = 5)
print(clust2)

# Visualise the cluster
clusplot(thera_bank.scaled, clust2$cluster,
         color=TRUE, shade=TRUE, labels=2, lines=1)

# Create clusters for k=3, k=4 and k=5 for comparative analysis
clust3 <- kmeans(thera_bank.scaled, centers = 3, nstart = 5)
clust4 <- kmeans(thera_bank.scaled, centers = 4, nstart = 5)
clust5 <- kmeans(thera_bank.scaled, centers = 5, nstart = 5)

# Visualise clusters in 2 dimensions
k_clust_viz_2 = fviz_cluster(list(data = thera_bank.scaled,
                                  cluster = clust2$cluster)) +
  ggtitle("k = 2")
k_clust_viz_3 = fviz_cluster(list(data = thera_bank.scaled,
                                  cluster = clust3$cluster)) +
  ggtitle("k = 3")
k_clust_viz_4 = fviz_cluster(list(data = thera_bank.scaled,
                                  cluster = clust4$cluster)) +
  ggtitle("k = 4")
k_clust_viz_5 = fviz_cluster(list(data = thera_bank.scaled,
                                  cluster = clust5$cluster)) +
  ggtitle("k = 5")

# Visualise all 4 clustering plots together
grid.arrange(k_clust_viz_2, k_clust_viz_3, k_clust_viz_4, k_clust_viz_5, nrow = 2)

# To find the optimal numbner of clusters. Lets try K = 1 to 5 and for each plot the "sum of Within clus
totWss <- rep(0,5)
for(k in 1:5){
```

```
  set.seed(seed)
  clust <- kmeans(thera_bank.scaled, centers = k, nstart = 5)
  totWss[k] <- clust$tot.withinss
}
print(totWss)
plot(c(1:5), totWss, type="b", xlab="Number of Clusters",
       ylab="sum of 'Within groups sum of squares'")

set.seed(seed)
nc <- NbClust(thera_bank.scaled, min.nc = 2, max.nc = 5, method="kmeans")

table(nc$Best.n[1,])

# Adding the cluster numbers back to the dataset
data$Clusters = clust3$cluster

# Aggregate all columns except column 8 for each cluster by their means
custProfile = aggregate(data[, -c(4, 6, 8, 9, 10, 11, 12)],list(data$Cluster),FUN="mean")
print(custProfile)

set.seed(seed) # To ensure reproducibility
split <- sample.split(thera_bank$Personal_Loan, SplitRatio = 0.7)
train <- subset(thera_bank, split == TRUE)
test <- subset(thera_bank, split == FALSE)

nrow(train)
nrow(test)

# Check that the distribution of the dependent variable is similar in train and test sets
prop.table(table(thera_bank$Personal_Loan))
prop.table(table(thera_bank$Personal_Loan))
prop.table(table(thera_bank$Personal_Loan))

# Setting the control parameters (to control the growth of the tree)
# Set the control parameters very low to let the tree grow deep

r.ctrl = rpart.control(minsplit = 50, minbucket = 10, cp = 0, xval = 10)

# Building the CART model

# formula - response variable~predictor variables
# data - dataset
# method - "class" - for classification, "anova" for regression
# control - tree control parameters

model1 <- rpart(formula = Personal_Loan ~ ., data = train, method = "class", control = r.ctrl)
model1

# Displaying the decision tree
fancyRpartPlot(model1)

# The cost complexity table can be obtained using the printcp or plotcp functions
printcp(model1)
plotcp(model1)
```

```
model2 = prune(model1, cp= 0.021, "CP")
printcp(model2)
model2

#Displaying the decision tree
fancyRpartPlot(model2)

# Variable importance is generally computed based on the corresponding reduction of predictive accuracy
# when the predictor of interest is removed.
model1$variable.importance

# Variable importance is generally computed based on the corresponding reduction of predictive accuracy
# when the predictor of interest is removed.
model2$variable.importance

# Predicting on the train dataset
train_predict.class1 <- predict(model1, train, type="class") # Predicted Classes
train_predict.score1 <- predict(model1, train) # Predicted Probabilities

# Create confusion matrix for train data predictions
tab.train1 = table(train$Personal_Loan, train_predict.class1)
tab.train1

# Accuracy on train data
accuracy.train1 = sum(diag(tab.train1)) / sum(tab.train1)
accuracy.train1

# Predicting on the train dataset
train_predict.class2 <- predict(model2, train, type="class") # Predicted Classes
train_predict.score2 <- predict(model2, train) # Predicted Probabilities

# Create confusion matrix for train data predictions
tab.train2 = table(train$Personal_Loan, train_predict.class2)
tab.train2

# Accuracy on train data
accuracy.train2 = sum(diag(tab.train2)) / sum(tab.train2)
accuracy.train2

# Predicting on the test dataset using MODEL 1
test_predict.class1 <- predict(model1, test, type="class") # Predicted Classes
test_predict.score1 <- predict(model1, test) # Predicted Probabilities

# Create confusion matrix for test data predictions (using MODEL 1)
tab.test1 = table(test$Personal_Loan, test_predict.class1)
tab.test1

# Accuracy on train data (MODEL 1 predictions)
accuracy.test1 = sum(diag(tab.test1)) / sum(tab.test1)
accuracy.test1

# Predicting on the test dataset using MODEL 2
test_predict.class2 <- predict(model2, test, type="class") # Predicted Classes
```

```
test_predict.score2 <- predict(model2, test) # Predicted Probabilities

# Create confusion matrix for test data predictions (using MODEL 2)
tab.test2 = table(test$Personal_Loan, test_predict.class2)
tab.test2

# Accuracy on train data (MODEL 2 predictions)
accuracy.test2 = sum(diag(tab.test2)) / sum(tab.test2)
accuracy.test2

Model_Name = c("Baseline", "Model1", "Model2")
Train_Accuracy_perc = c(90, accuracy.train1*100, accuracy.train2*100)
Test_Accuracy_perc = c(90, accuracy.test1*100, accuracy.test2*100)
output = data.frame(Model_Name,Train_Accuracy_perc,Test_Accuracy_perc)
output

set.seed(seed)
# Formula - response variable ~ predictor variables
# To build a classification random forest the response variable should be converted to a factor if it is
# data -  dataset to train the model on
## Random Forest hyperparameters
# ntree - Total number of trees are to be constructed
# mtry - number of variables tried at each split
# importance - Set TRUE to assess variable importance

rf_model1 = randomForest(
  Personal_Loan ~ .,
  data = train,
  ntree = 501,
  mtry = 5,
  nodesize = 10,
  importance = TRUE
  )

print(rf_model1)

plot(rf_model1, main="")
legend("topright", c("OOB", "0", "1"), text.col=1:6, lty=1:3, col=1:3)
title(main="Error Rates Random Forest Thera_Bank")

importance(rf_model1)

# Check the column number of the response variable
names(train)

set.seed(seed) # To ensure reproducibility

rf_model2 = tuneRF(x = train[, -8], # matrix or data frame of predictor/independent variables
                   y = train$Personal_Loan, # response vector (factor for classification, numeric for reg
                   mtrystart = 5, # starting value of mtry
                   stepfactor=1.5, # at each iteration, mtry is inflated (or deflated) by this value
                   ntree=51, # number of trees built for each mtry value
                   improve=0.0001, # the (relative) improvement in OOB error must be by this much for the
                   nodesize=10, # Minimum size of terminal nodes
```

```
                  trace=TRUE, # prints the progress of the search
                  plot=TRUE, # to get the plot of the OOB error as function of mtr
                  doBest=TRUE, # return a forest using the optimal mtry found
                  importance=TRUE #
                  )

# Predicting on the train dataset
train_predict.class_RF <- predict(rf_model2, train, type = "class") # Predicted Classes
train_predict.score_RF <- predict(rf_model2, train, type = 'prob') # Predicted Probabilities

# Create confusion matrix for train data predictions
tab.train_RF = table(train$Personal_Loan, train_predict.class_RF)
tab.train_RF

# Accuracy on train data
accuracy.train_RF = sum(diag(tab.train_RF)) / sum(tab.train_RF)
accuracy.train_RF

# Predicting on the test dataset
test_predict.class_RF <- predict(rf_model2, test, type = "class") # Predicted Classes
test_predict.score_RF <- predict(rf_model2, test, type = 'prob') # Predicted Probabilities

# Create confusion matrix for test data predictions
tab.test_RF = table(test$Personal_Loan, test_predict.class_RF)
tab.test_RF

# Accuracy on test data
accuracy.test_RF = sum(diag(tab.test_RF)) / sum(tab.test_RF)
accuracy.test_RF

Model_Name = c("Baseline", "CART", "Random Forest")
Train_Accuracy_perc = c(90, accuracy.train1*100, accuracy.train_RF*100)
Test_Accuracy_perc = c(90, accuracy.test1*100, accuracy.test_RF*100)
output = data.frame(Model_Name, Train_Accuracy_perc, Test_Accuracy_perc)
output

varImpPlot(rf_model2, sort = TRUE)

# Predict on test data using cart_model1
model1_predict_class = predict(model1, test, type = 'class')
model1_predict_score = predict(model1, test, type = 'prob')

# Predict on test data using cart_model2
model2_predict_class = predict(model2, test, type = 'class')
model2_predict_score = predict(model2, test, type = 'prob')

# Predict on test data using rf_model1
rf_model1_predict_class = predict(rf_model1, test, type = 'class')
rf_model1_predict_score = predict(rf_model1, test, type = 'prob')

# Predict on test data using rf_model2
rf_model2_predict_class = predict(rf_model2, test, type = 'class')
rf_model2_predict_score = predict(rf_model2, test, type = 'prob')
```

```
# Create Confusion Matrix for all the four models
conf_mat_model1 = table(test$Personal_Loan, model1_predict_class)
conf_mat_model1

conf_mat_model2 = table(test$Personal_Loan, model2_predict_class)
conf_mat_model2

conf_mat_rf_model1 = table(test$Personal_Loan, rf_model1_predict_class)
conf_mat_rf_model1

conf_mat_rf_model2 = table(test$Personal_Loan, rf_model2_predict_class)
conf_mat_rf_model2

# Accuracy of models on test data
accuracy_model1 = sum(diag(conf_mat_model1)) / sum(conf_mat_model1)

accuracy_model2 = sum(diag(conf_mat_model2)) / sum(conf_mat_model2)

accuracy_rf_model1 = sum(diag(conf_mat_rf_model1)) / sum(conf_mat_rf_model1)

accuracy_rf_model2 = sum(diag(conf_mat_rf_model2)) / sum(conf_mat_rf_model2)

# Sensitivity of models on test data
sensitivity_model1 = conf_mat_model1[2,2] / sum(conf_mat_model1['1',])

sensitivity_model2 = conf_mat_model2[2,2] / sum(conf_mat_model2['1',])

sensitivity_rf_model1 = conf_mat_rf_model1[2,2] / sum(conf_mat_rf_model1['1',])

sensitivity_rf_model2 = conf_mat_rf_model2[2,2] / sum(conf_mat_rf_model2['1',])

# Specificity of models on test data
specificity_model1 = conf_mat_model1[1,1] / sum(conf_mat_model1['0',])

specificity_model2 = conf_mat_model2[1,1] / sum(conf_mat_model2['0',])

specificity_rf_model1 = conf_mat_rf_model1[1,1] / sum(conf_mat_rf_model1['0',])

specificity_rf_model2 = conf_mat_rf_model2[1,1] / sum(conf_mat_rf_model2['0',])

# Precision of models on test data
precision_model1 = conf_mat_model1[2,2] / sum(conf_mat_model1[,'1'])

precision_model2 = conf_mat_model2[2,2] / sum(conf_mat_model2[,'1'])

precision_rf_model1 = conf_mat_rf_model1[2,2] / sum(conf_mat_rf_model1[,'1'])

precision_rf_model2 = conf_mat_rf_model2[2,2] / sum(conf_mat_rf_model2[,'1'])

# Using library ROCR functions prediction and performance
pred_model1 = prediction(model1_predict_score[, 2], test$Personal_Loan)
perf_model1 = performance(pred_model1,"tpr","fpr")
ks_model1 = max(attr(perf_model1,'y.values')[[1]] - attr(perf_model1,'x.values')[[1]])
```

```
pred_model2 = prediction(model2_predict_score[, 2], test$Personal_Loan)
perf_model2 = performance(pred_model2,"tpr","fpr")
ks_model2 = max(attr(perf_model2,'y.values')[[1]] - attr(perf_model2,'x.values')[[1]])

pred_rf_model1 = prediction(rf_model1_predict_score[, 2], test$Personal_Loan)
perf_rf_model1 = performance(pred_rf_model1,"tpr","fpr")
ks_rf_model1 = max(attr(perf_rf_model1,'y.values')[[1]] - attr(perf_rf_model1,'x.values')[[1]])

pred_rf_model2 = prediction(rf_model2_predict_score[, 2], test$Personal_Loan)
perf_rf_model2 = performance(pred_rf_model2,"tpr","fpr")
ks_rf_model2 = max(attr(perf_rf_model2,'y.values')[[1]] - attr(perf_rf_model2,'x.values')[[1]])

# Using library ROCR
auc_model1 = performance(pred_model1, measure = "auc")
auc_model1 = auc_model1@y.values[[1]]

auc_model2 = performance(pred_model2, measure = "auc")
auc_model2 = auc_model2@y.values[[1]]

auc_rf_model1 = performance(pred_rf_model1, measure = "auc")
auc_rf_model1 = auc_rf_model1@y.values[[1]]

auc_rf_model2 = performance(pred_rf_model2, measure = "auc")
auc_rf_model2 = auc_rf_model2@y.values[[1]]

# Using library ineq
gini_model1 = ineq(model1_predict_score[, 2],"gini")

gini_model2 = ineq(model2_predict_score[, 2],"gini")

gini_rf_model1 = ineq(rf_model1_predict_score[, 2],"gini")

gini_rf_model2 = ineq(rf_model2_predict_score[, 2],"gini")

concordance_model1 = Concordance(actuals = ifelse(test$Personal_Loan == '1', 1,0), predictedScores = if

concordance_model2 = Concordance(actuals = ifelse(test$Personal_Loan == '1', 1,0), predictedScores = if

concordance_rf_model1 = Concordance(actuals = ifelse(test$Personal_Loan == '1', 1,0), predictedScores =

concordance_rf_model2 = Concordance(actuals = ifelse(test$Personal_Loan == '1', 1,0), predictedScores =


model1_metrics = c(accuracy_model1, sensitivity_model1, specificity_model1, precision_model1, ks_model1

model2_metrics = c(accuracy_model2, sensitivity_model2, specificity_model2, precision_model2, ks_model2

rf_model1_metrics = c(accuracy_rf_model1, sensitivity_rf_model1, specificity_rf_model1, precision_rf_mo

rf_model2_metrics = c(accuracy_rf_model2, sensitivity_rf_model2, specificity_rf_model2, precision_rf_mo

comparison_table = data.frame(model1_metrics, model2_metrics, rf_model1_metrics, rf_model2_metrics)

rownames(comparison_table) = c("Accuracy", "Sensitivity", "Specificity", "Precision", "KS", "Auc", "Gini
```

```
comparison_table

#=============================================================================
#
# T H E - E N D
#
#=============================================================================

# Generate the .R file from this .Rmd to hold the source code

purl("Thera Bank Project.Rmd", documentation = 0)
```

Generate .R file from this Rmd. The .R will contain only the R source code.

```r
# Generate the .R file from this .Rmd to hold the source code

purl("Thera Bank Project.Rmd", documentation = 0)
```

To create word or pdf report -> click on Knit in the toolbar above, select knit to pdf.