project 5

loading DataSet for Thera Bank_Personal_Loan_Modelling :

```
library(readxl)
Thera_Bank <- read_excel("C:/Users/daoud/Downloads/PGP DSBA/ML/week 5/Thera Bank_Personal_Loan_Modelling-dataset-1.xlsx",
    sheet = "Bank_Personal_Loan_Modelling")
```

packages :

```
library(rpart)

## Warning: package 'rpart' was built under R version 3.6.3

library(caTools)

## Warning: package 'caTools' was built under R version 3.6.3

library(summarytools)

## Warning: package 'summarytools' was built under R version 3.6.3

## Registered S3 method overwritten by 'pryr':
##   method      from
##   print.bytes Rcpp

## For best results, restart R session and update pander using devtools:: or
remotes::install_github('rapporter/pander')

library(caret)

## Warning: package 'caret' was built under R version 3.6.3

## Loading required package: lattice

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 3.6.3

library(janitor)

## Warning: package 'janitor' was built under R version 3.6.3

##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test

library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.3

## randomForest 4.6-14

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

##
## Attaching package: 'randomForest'

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

library(InformationValue)

## Warning: package 'InformationValue' was built under R version 3.6.3

##
## Attaching package: 'InformationValue'

## The following objects are masked from 'package:caret':
##
##     confusionMatrix, precision, sensitivity, specificity

library(ROCR)

## Warning: package 'ROCR' was built under R version 3.6.3

library(ineq)
library(rattle)

## Warning: package 'rattle' was built under R version 3.6.3

## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.

##
## Attaching package: 'rattle'

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

library(stats)
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 3.6.3
```

Exploratory Data Analysis:

```
head(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     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>
```

use clean_name to rename all variables :

```
str(Thera_Bank)

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

Thera_Bank<-clean_names(dat = Thera_Bank)
```

data summary:

```
#view(dfSummary(Thera_Bank)) # is very helpful for summary
summary(Thera_Bank)

##        id         age_in_years   experience_in_years income_in_k_month
##  Min.   :   1   Min.   :23.00   Min.   :-3.0        Min.   :  8.00
##  1st Qu.:1251   1st Qu.:35.00   1st Qu.:10.0        1st Qu.: 39.00
##  Median :2500   Median :45.00   Median :20.0        Median : 64.00
##  Mean   :2500   Mean   :45.34   Mean   :20.1        Mean   : 73.77
##  3rd Qu.:3750   3rd Qu.:55.00   3rd Qu.:30.0        3rd Qu.: 98.00
##  Max.   :5000   Max.   :67.00   Max.   :43.0        Max.   :224.00
##
##     zip_code     family_members      cc_avg          education
##  Min.   : 9307   Min.   :1.000   Min.   : 0.000   Min.   :1.000
##  1st Qu.:91911   1st Qu.:1.000   1st Qu.: 0.700   1st Qu.:1.000
##  Median :93437   Median :2.000   Median : 1.500   Median :2.000
##  Mean   :93153   Mean   :2.397   Mean   : 1.938   Mean   :1.881
##  3rd Qu.:94608   3rd Qu.:3.000   3rd Qu.: 2.500   3rd Qu.:3.000
```

```
##  Max.   :96651   Max.   :4.000   Max.   :10.000   Max.   :3.000  
##                  NA's   :18                                       
##     mortgage     personal_loan   securities_account   cd_account    
##  Min.   :  0.0   Min.   :0.000   Min.   :0.0000      Min.   :0.0000  
##  1st Qu.:  0.0   1st Qu.:0.000   1st Qu.:0.0000      1st Qu.:0.0000  
##  Median :  0.0   Median :0.000   Median :0.0000      Median :0.0000  
##  Mean   : 56.5   Mean   :0.096   Mean   :0.1044      Mean   :0.0604  
##  3rd Qu.:101.0   3rd Qu.:0.000   3rd Qu.:0.0000      3rd Qu.:0.0000  
##  Max.   :635.0   Max.   :1.000   Max.   :1.0000      Max.   :1.0000  
##                                                                      
##      online        credit_card   
##  Min.   :0.0000   Min.   :0.000  
##  1st Qu.:0.0000   1st Qu.:0.000  
##  Median :1.0000   Median :0.000  
##  Mean   :0.5968   Mean   :0.294  
##  3rd Qu.:1.0000   3rd Qu.:1.000  
##  Max.   :1.0000   Max.   :1.000  
## 
```

Observation :

##1 we have 5000 customers data with 14 variables all numeric .

##2 we have 18 missing values on Family members .

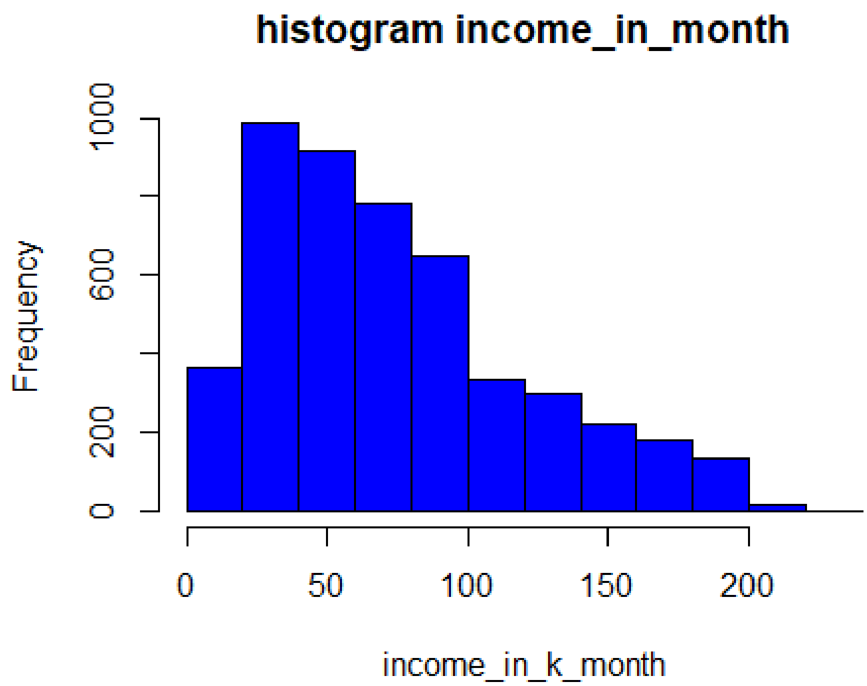
##3 we have negative values on experience_in_years will have to drop .

we replace the missing value with median and drop negative experience_in_years values :

```
Thera_Bank$family_members[is.na(Thera_Bank$family_members)] <-
median(Thera_Bank$family_members, na.rm=TRUE)
sum(is.na(Thera_Bank$family_members))

## [1] 0

Thera_Bank<-Thera_Bank[Thera_Bank$experience_in_years>0,]  # drop negative
values experience_in_years
```

lets plot some variables:

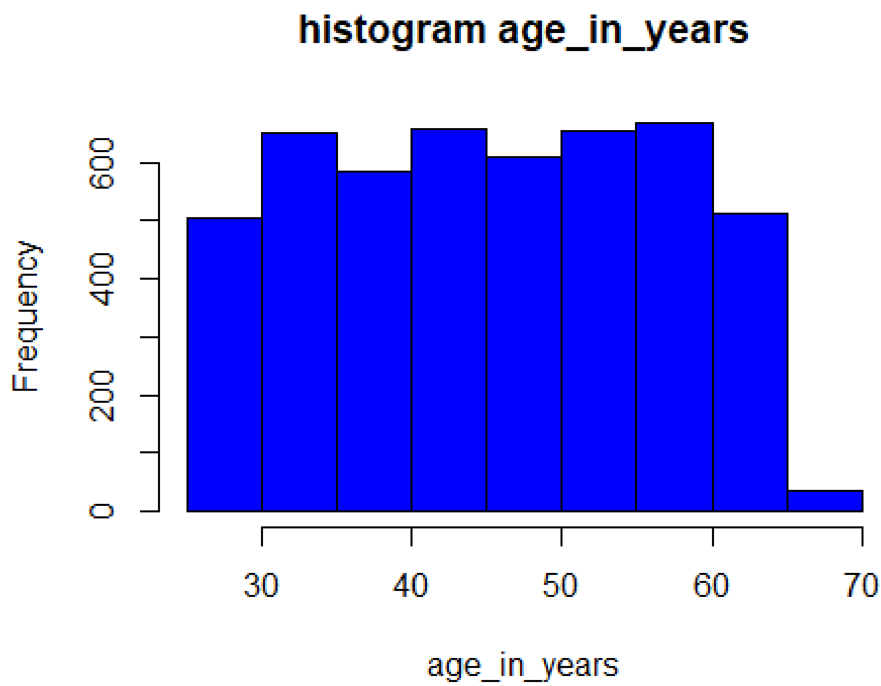
```
hist(Thera_Bank$income_in_k_month,col = "blue",xlab =
"income_in_k_month",main = "histogram income_in_month")
```



Observation :

The above distributionis is right skewed distribution.

```
hist(Thera_Bank$age_in_years,col = "blue",xlab = "age_in_years",main =
"histogram age_in_years")
```



Observation :

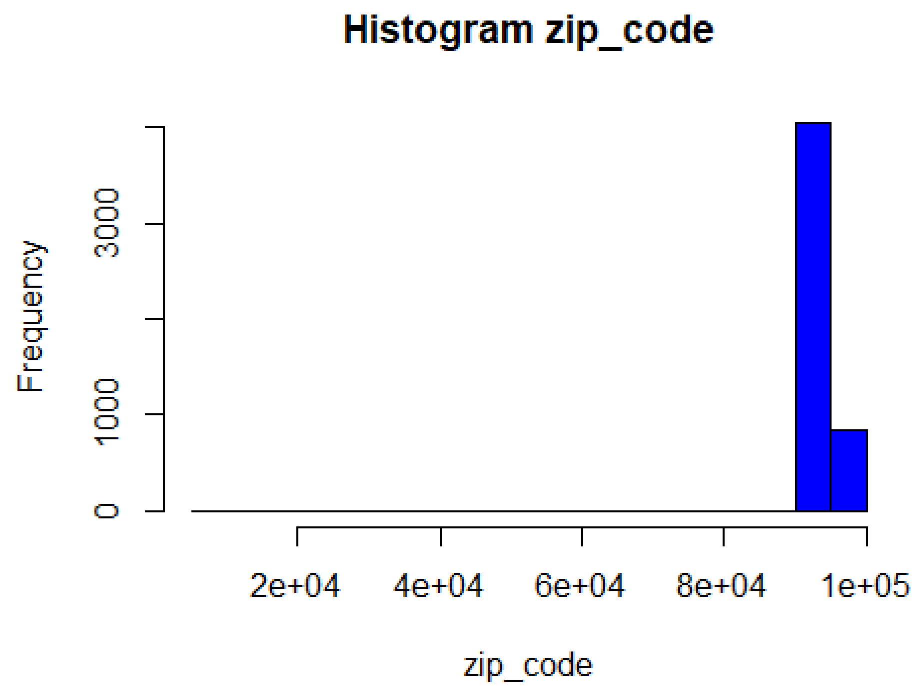
The Age is normal distribution.

```
hist(Thera_Bank$experience_in_years,col = "blue",xlab =
"experience_in_years",main = "Histogram experience_in_years")
```

Observation :

The Experience also is normal distribution.

```
hist(Thera_Bank$zip_code,col = "blue",xlab = "zip_code",main = "Histogram
zip_code")
```

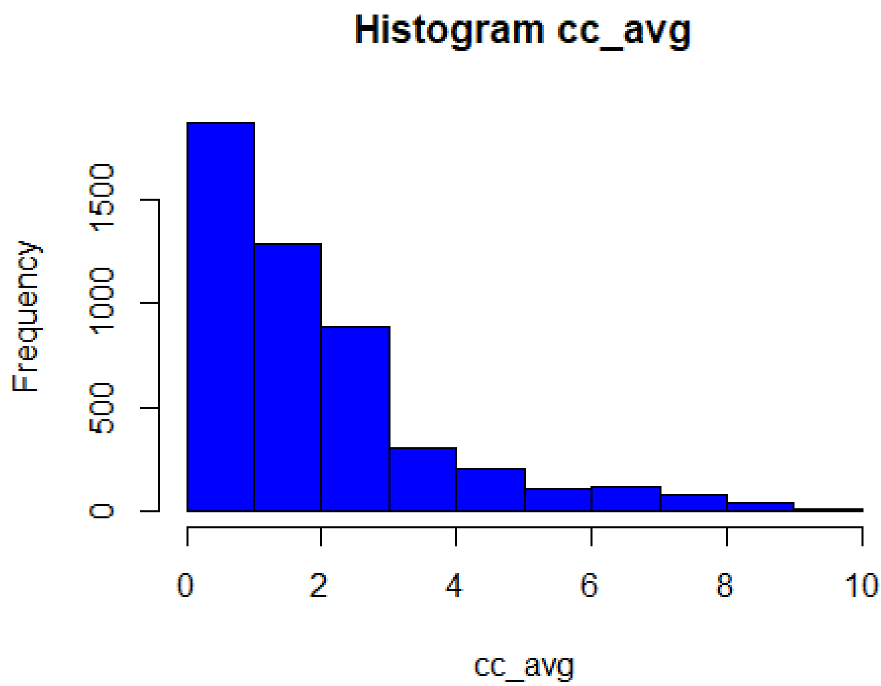


Observation :

zip code doesn't give any impact on personal loan.

we drop zip_code later .

```
hist(Thera_Bank$cc_avg,col = "blue",xlab = "cc_avg",main = "Histogram
cc_avg")
```



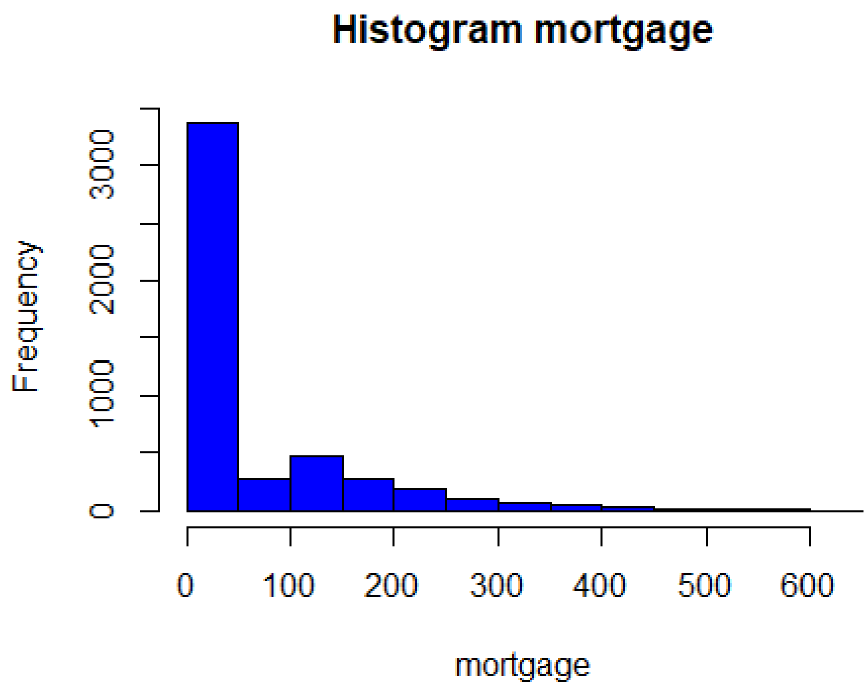
Observation :

The cc_avg is right skewed distribution because the tail goes to the right.

most of the customers spend on avg 1K to 2K per month on credit cards.

few customers spend more then 8K .

```
hist(Thera_Bank$mortgage,col = "blue",xlab = "mortgage",main = "Histogram
mortgage")
```



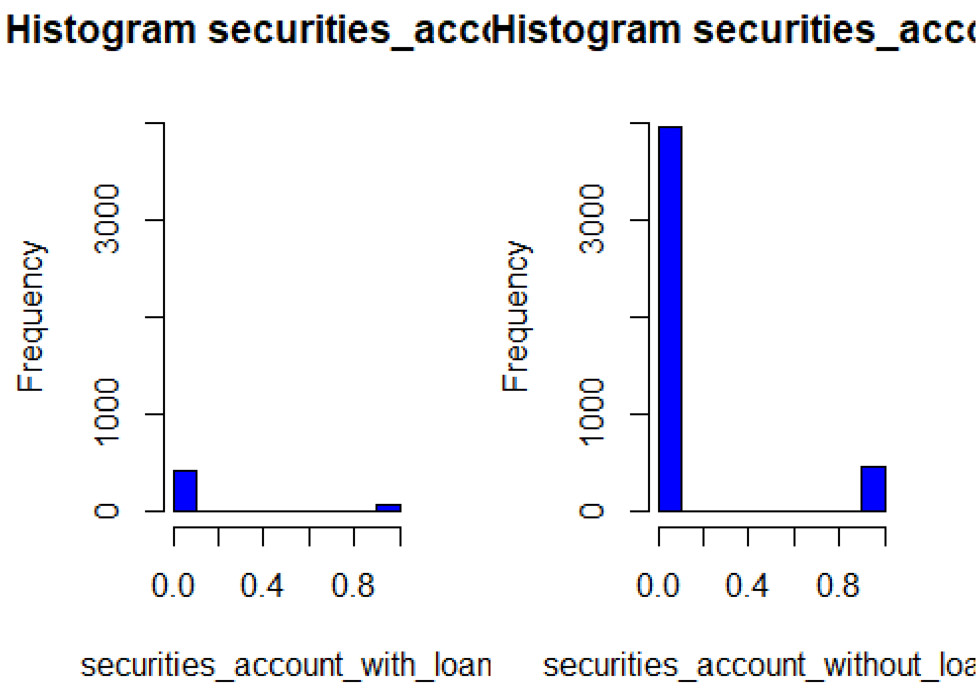
Observation :

The mortgage is right skewed distribution.

most of the customers mortgage 50K to 150K.

very few of customers mortgage above 400K.

```
par(mfrow=c(1,2))
hist(Thera_Bank$securities_account[Thera_Bank$personal_loan==1],col =
"blue",xlab = "securities_account_with_loan",main = "Histogram
securities_account",ylim = c(0,4000))
hist(Thera_Bank$securities_account[Thera_Bank$personal_loan==0],col =
"blue",xlab = "securities_account_without_loan",main = "Histogram
securities_account",ylim = c(0,4000))
```

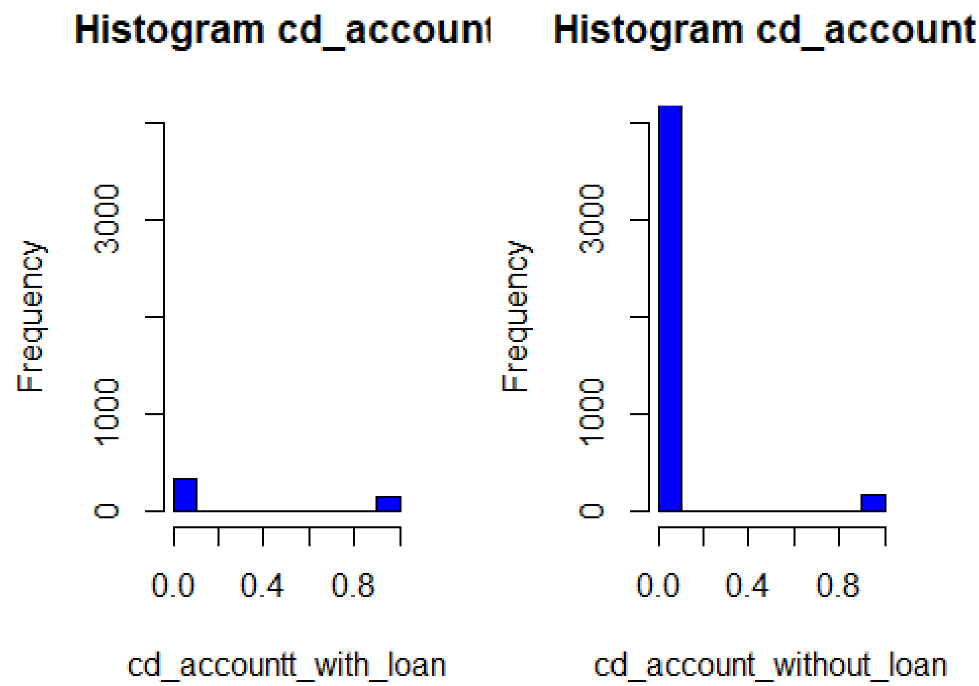


Observation :

The majorty of customer don’t have securities account.

the customers have securities account are more likly to loan.

```
par(mfrow=c(1,2))
hist(Thera_Bank$cd_account[Thera_Bank$personal_loan==1],col = "blue",xlab =
"cd_accountt_with_loan",main = "Histogram cd_account",ylim = c(0,4000))
hist(Thera_Bank$cd_account[Thera_Bank$personal_loan==0],col = "blue",xlab =
"cd_account_without_loan",main = "Histogram cd_accountt",ylim = c(0,4000))
```

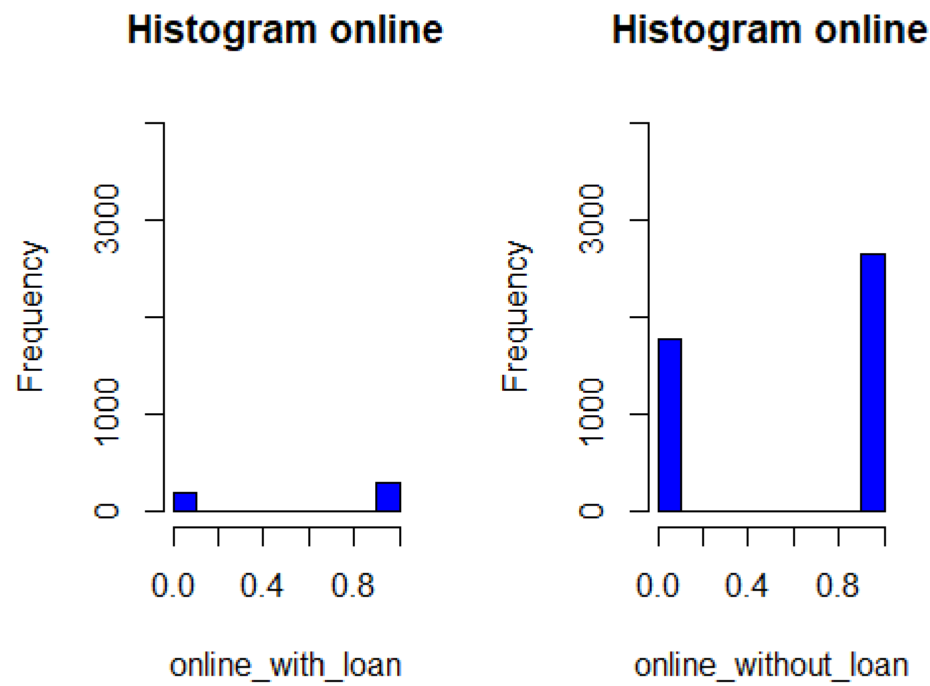


Observation :

The majorty of customer don't have cd account.

almost all customers have cd account has loan.

```
par(mfrow=c(1,2))
hist(Thera_Bank$online[Thera_Bank$personal_loan==1],col = "blue",xlab =
"online_with_loan",main = "Histogram online",ylim = c(0,4000))
hist(Thera_Bank$online[Thera_Bank$personal_loan==0],col = "blue",xlab =
"online_without_loan",main = "Histogram online",ylim = c(0,4000))
```

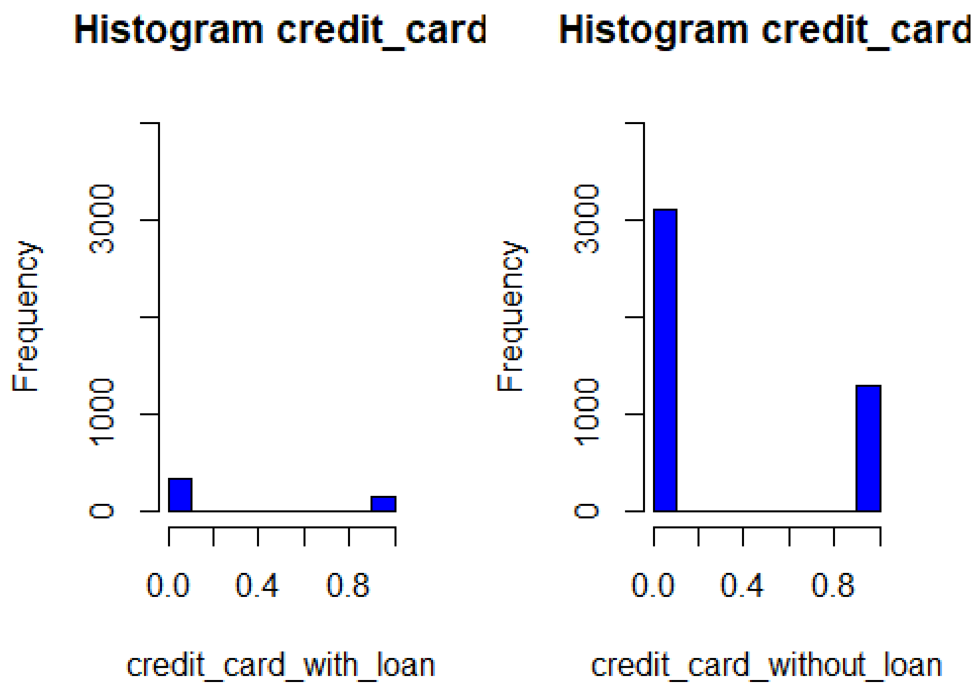


Observation :

The majorty of customer don’t use online banking.

all customers use online banking has loan as well.

```
par(mfrow=c(1,2))
hist(Thera_Bank$credit_card[Thera_Bank$personal_loan==1],col = "blue",xlab =
"credit_card_with_loan",main = "Histogram credit_card",ylim = c(0,4000))
hist(Thera_Bank$credit_card[Thera_Bank$personal_loan==0],col = "blue",xlab =
"credit_card_without_loan",main = "Histogram credit_card",ylim = c(0,4000))
```

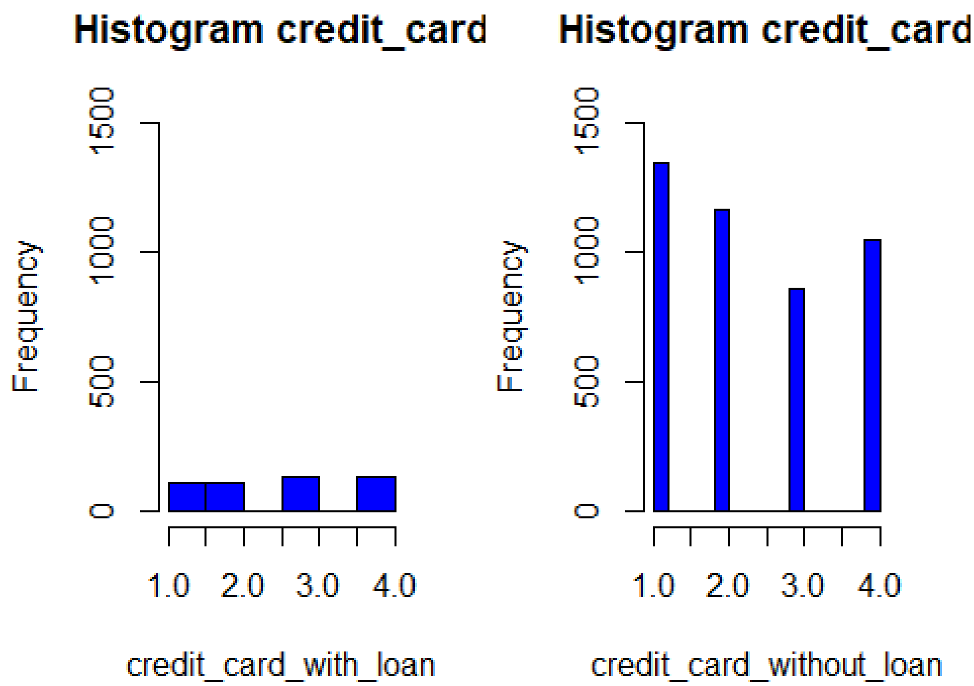


Observation :

The majorty of customer don’t use credit card.

almost all customers using credit card has loan as well.

```
par(mfrow=c(1,2))
hist(Thera_Bank$family_members[Thera_Bank$personal_loan==1],col = "blue",xlab
= "credit_card_with_loan",main = "Histogram credit_card",ylim = c(0,1500))
hist(Thera_Bank$family_members[Thera_Bank$personal_loan==0],col = "blue",xlab
= "credit_card_without_loan",main = "Histogram credit_card",ylim = c(0,1500))
```



Observation :

family nembers don't have any impact on personal loan.

drop the ID and zip_code colume:

```
Thera_Bank = Thera_Bank[,-c(1,5)]
Thera_Bank1<- Thera_Bank  # will use later for decision tree and Random forest
```

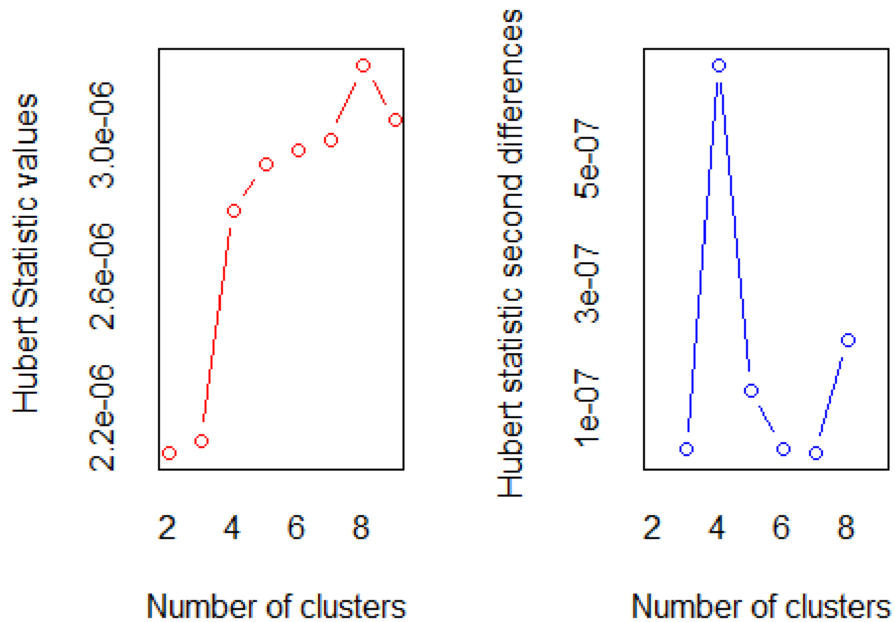
cluster:

Apply Clustering algorithm:

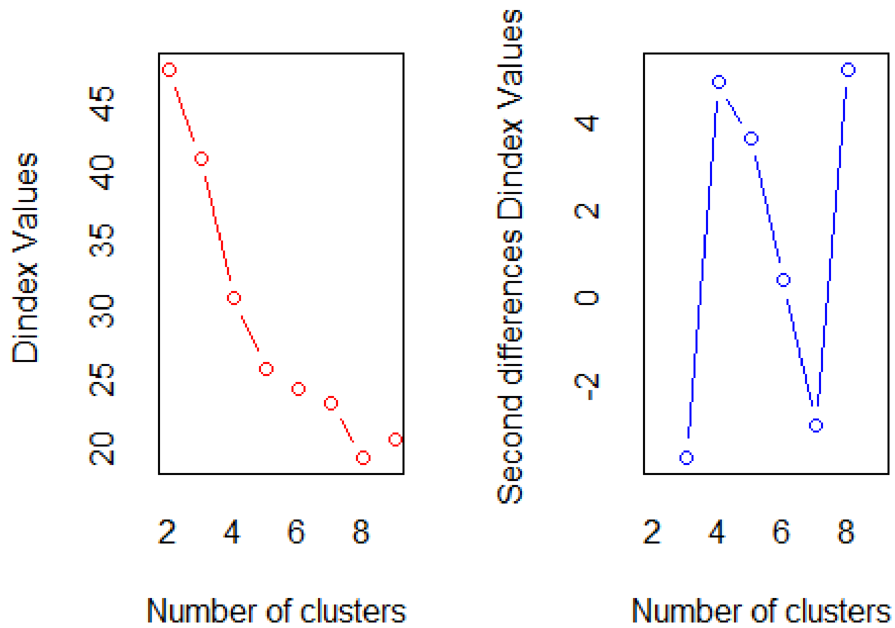
```
seed=1000
set.seed(seed)
levels(Thera_Bank$personal_loan) <- c("0", "1")
Thera_Bank$personal_loan<-as.numeric(Thera_Bank$personal_loan)
cluster_sample<-Thera_Bank
cluster_sample <- Thera_Bank[sample(nrow(Thera_Bank), 70), ] # we pick random sample to cluster using Kmeans
cluster_sample.scaled <- scale(cluster_sample)               # Scale the dataset
```

NbClust for the best K between 2 and 9 using Kmeans method :

```
library(NbClust)
seed=1000
set.seed(seed)
nc <- NbClust(cluster_sample[,c(-1)], min.nc=2, max.nc=9, method="kmeans")
```




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




```
## *** : The D index is a graphical method of determining the number of
clusters.
##                 In the plot of D index, we seek a significant knee (the
significant peak in Dindex
##                 second differences plot) that corresponds to a significant
increase of the value of
##                 the measure.
##
## *******************************************************************
## * Among all indices:
## * 5 proposed 2 as the best number of clusters
## * 4 proposed 3 as the best number of clusters
## * 5 proposed 4 as the best number of clusters
## * 1 proposed 5 as the best number of clusters
## * 2 proposed 6 as the best number of clusters
## * 6 proposed 8 as the best number of clusters
## * 1 proposed 9 as the best number of clusters
##
##                    ***** Conclusion *****
##
## * According to the majority rule, the best number of clusters is  8
##
##
## *******************************************************************
```

the best K is 4 .

note : it is random so every time is different .

nc now contains :

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

##
## 0 2 3 4 5 6 8 9
## 2 5 4 5 1 2 6 1
```

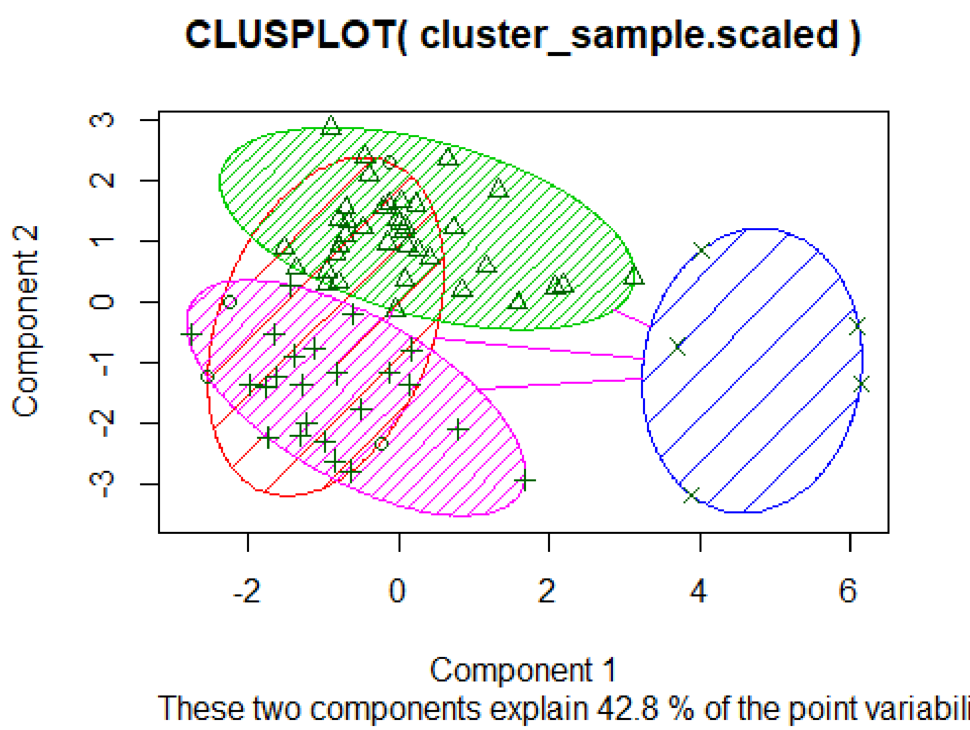
K=4 would be the best choice:

```
set.seed(seed)
clust3 = kmeans(x=cluster_sample.scaled, centers = 4, nstart = 5)
print(clust3)

## K-means clustering with 4 clusters of sizes 4, 38, 23, 5
##
## Cluster means:
##   age_in_years experience_in_years income_in_k_month family_members
cc_avg
## 1   -0.4692000          -0.5224265        -0.9067777     0.02538016 -
0.45451489
## 2    0.6660207           0.6692766        -0.2958379     0.07213308 -
0.20581658
## 3   -1.1387970          -1.1332814         0.1272683     0.08331313
0.05871114
## 4    0.5520687           0.5445336         2.3883558    -0.95175595
1.65774672
##    education    mortgage personal_loan securities_account cd_account
## 1  0.3856289 -0.55179792    -0.2753619          4.0329004  0.8249114
## 2  0.2039066  0.04066695    -0.2753619         -0.2444182 -0.1318572
## 3 -0.5287288 -0.12817778    -0.2753619         -0.2444182 -0.2444182
## 4  0.5739593  0.72198728     3.5797047         -0.2444182  1.4665092
##        online credit_card
## 1  0.05682608  -0.1374852
## 2  0.10916589   0.1157770
## 3 -0.41878349  -0.1142389
## 4  1.05128246  -0.2444182
##
## Clustering vector:
##  [1] 3 2 4 2 2 3 2 2 2 2 2 2 2 2 1 3 2 3 4 2 2 2 2 2 3 1 2 3 3 3 2 3 3 2 2
3 3 2
## [39] 3 2 3 3 2 2 3 2 1 2 2 3 3 2 2 2 3 4 3 2 2 2 2 2 3 4 1 2 3 3 4 2
##
## Within cluster sum of squares by cluster:
## [1]  36.13173 257.26378 135.16811  65.32334
##  (between_SS / total_SS =  40.4 %)
##
```

```
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

plot cluster :

```
library(cluster)
clusplot(cluster_sample.scaled, clust3$cluster, color=TRUE, shade=TRUE)
```



adding cluster number colume to dataset :

```
cluster_sample$Clusters = clust3$cluster
print(cluster_sample)

## # A tibble: 70 x 13
##    age_in_years experience_in_y~ income_in_k_mon~ family_members cc_avg
##           <dbl>            <dbl>            <dbl>          <dbl>  <dbl>
##  1           29                5              135              2    0.6
##  2           50               25               24              4    0.4
##  3           35               10              182              1    0.3
##  4           62               38              124              1    3.8
##  5           52               28               41              3    1.9
##  6           31                6               58              2    2.5
##  7           51               25               45              4    2.6
##  8           55               29               78              4    2.6
##  9           58               28               58              3    2
```

```
## 10           45                18                48               3    2.5
## # ... with 60 more rows, and 8 more variables: education <dbl>, mortgage
<dbl>,
## #   personal_loan <dbl>, securities_account <dbl>, cd_account <dbl>,
## #   online <dbl>, credit_card <dbl>, Clusters <int>
```

Aggregating:

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

##   Group.1 age_in_years experience_in_years income_in_k_month
family_members
## 1       1     41.50000            15.50000          31.00000
2.500000
## 2       2     55.39474            30.13158          57.92105
2.552632
## 3       3     33.30435             8.00000          76.56522
2.565217
## 4       4     54.00000            28.60000         176.20000
1.400000
##     cc_avg education  mortgage personal_loan securities_account cd_account
## 1 1.242500  2.250000   0.00000             0                  1 0.25000000
## 2 1.623684  2.105263  54.60526             0                  0 0.02631579
## 3 2.029130  1.521739  39.04348             0                  0 0.00000000
## 4 4.480000  2.400000 117.40000             1                  0 0.40000000
##      online credit_card Clusters
## 1 0.5000000   0.2500000        1
## 2 0.5263158   0.3684211        2
## 3 0.2608696   0.2608696        3
## 4 1.0000000   0.2000000        4
```

Observation :

every colume mean based on 4 cluster group.

prepare data for Train Models :

we ensure target varibal is factor :

```
head(Thera_Bank1)

## # A tibble: 6 x 12
##   age_in_years experience_in_y~ income_in_k_mon~ family_members cc_avg
education
##          <dbl>            <dbl>            <dbl>          <dbl>  <dbl>
<dbl>
## 1           25                1               49              4    1.6
```

```
1
## 2          45            19            34            3    1.5
1
## 3          39            15            11            1    1
1
## 4          35             9           100            1    2.7
2
## 5          35             8            45            4    1
2
## 6          37            13            29            4    0.4
2
## # ... with 6 more variables: mortgage <dbl>, personal_loan <dbl>,
## #   securities_account <dbl>, cd_account <dbl>, online <dbl>, credit_card
<dbl>
```

```
summary(Thera_Bank1)
```

```
##   age_in_years   experience_in_years income_in_k_month family_members
##  Min.   :25.00   Min.   : 1.00       Min.   :  8.00    Min.   :1.000
##  1st Qu.:36.00   1st Qu.:11.00       1st Qu.: 39.00    1st Qu.:1.000
##  Median :46.00   Median :21.00       Median : 64.00    Median :2.000
##  Mean   :45.83   Mean   :20.61       Mean   : 73.87    Mean   :2.385
##  3rd Qu.:55.00   3rd Qu.:30.00       3rd Qu.: 98.00    3rd Qu.:3.000
##  Max.   :67.00   Max.   :43.00       Max.   :224.00    Max.   :4.000
##      cc_avg          education        mortgage      personal_loan
##  Min.   : 0.000   Min.   :1.000   Min.   :  0.00   Min.   :0.00000
##  1st Qu.: 0.700   1st Qu.:1.000   1st Qu.:  0.00   1st Qu.:0.00000
##  Median : 1.500   Median :2.000   Median :  0.00   Median :0.00000
##  Mean   : 1.935   Mean   :1.875   Mean   : 56.84   Mean   :0.09689
##  3rd Qu.: 2.600   3rd Qu.:3.000   3rd Qu.:101.75   3rd Qu.:0.00000
##  Max.   :10.000   Max.   :3.000   Max.   :635.00   Max.   :1.00000
##  securities_account   cd_account          online        credit_card
##  Min.   :0.0000     Min.   :0.00000   Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.0000     1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.000
##  Median :0.0000     Median :0.00000   Median :1.0000   Median :0.000
##  Mean   :0.1041     Mean   :0.06145   Mean   :0.5987   Mean   :0.295
##  3rd Qu.:0.0000     3rd Qu.:0.00000   3rd Qu.:1.0000   3rd Qu.:1.000
##  Max.   :1.0000     Max.   :1.00000   Max.   :1.0000   Max.   :1.000
```

```
Thera_Bank1$personal_loan<- factor(ifelse(Thera_Bank1$personal_loan, 'Yes',
'No'))
```

spliting data Train and Test :

```
sample = sample.split(Thera_Bank1,SplitRatio = 0.7) # 70% train data , 30%
test data
d_train = subset(Thera_Bank1,sample == TRUE)

## Warning: Length of logical index must be 1 or 4882, not 12

d_test = subset(Thera_Bank1,sample == FALSE)
```

```
## Warning: Length of logical index must be 1 or 4882, not 12

nrow(d_train)

## [1] 3255

nrow(d_test)

## [1] 1627

prop.table(table(Thera_Bank1$personal_loan))

##
##         No        Yes
## 0.90311348 0.09688652

prop.table(table(d_train$personal_loan))

##
##         No        Yes
## 0.90752688 0.09247312

prop.table(table(d_test$personal_loan))

##
##       No      Yes
## 0.894284 0.105716
```

Ensure similar class distribution for train and test.

setting general paramater for training :

```
Ctrl <- trainControl(
              method = 'repeatedcv', # repeatedcv : repeated random sub-
sampling validation
              number = 5,                      # number of  k
              repeats = 3,                 # repeated k-fold cross-validation
              allowParallel = TRUE,
              classProbs = TRUE,               # Estimate class probabilities
              summaryFunction=twoClassSummary # should class probabilities be
returned
    )
```

Train a single decision tree:

```
set.seed(2000)
rpart_model <- train(personal_loan~ ., data =d_train,
                     method = "rpart",
                     minbucket = 50,
                     cp = 0.01,
                     tuneLength = 20,
                     trControl = Ctrl)
```

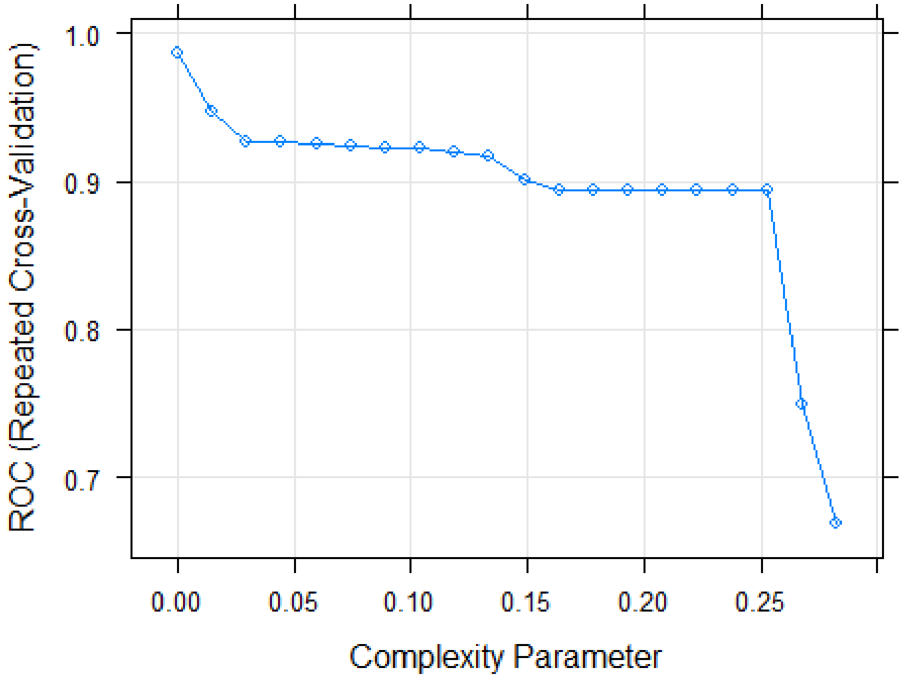
```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy"
was not
## in the result set. ROC will be used instead.
```

```
rpart_model
```

```
## CART
##
## 3255 samples
##   11 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 2604, 2604, 2605, 2603, 2604, 2604, ...
## Resampling results across tuning parameters:
##
##   cp          ROC        Sens       Spec
##   0.00000000  0.9876167  0.9943575  0.8982149
##   0.01486274  0.9472010  0.9952602  0.8560109
##   0.02972548  0.9273199  0.9963882  0.8230419
##   0.04458821  0.9272165  0.9963882  0.8119308
##   0.05945095  0.9257364  0.9946944  0.7897814
##   0.07431369  0.9247889  0.9925512  0.7986703
##   0.08917643  0.9229139  0.9881507  0.8208925
##   0.10403917  0.9224128  0.9871355  0.8286703
##   0.11890191  0.9202609  0.9856664  0.8174499
##   0.13376464  0.9168063  0.9847639  0.7985610
##   0.14862738  0.9015782  0.9864573  0.7188889
##   0.16349012  0.8934378  0.9865703  0.6901093
##   0.17835286  0.8934378  0.9865703  0.6901093
##   0.19321560  0.8934378  0.9865703  0.6901093
##   0.20807834  0.8934378  0.9865703  0.6901093
##   0.22294107  0.8934378  0.9865703  0.6901093
##   0.23780381  0.8934378  0.9865703  0.6901093
##   0.25266655  0.8934378  0.9865703  0.6901093
##   0.26752929  0.7493396  0.9918759  0.4276138
##   0.28239203  0.6679888  0.9957112  0.2820583
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.
```

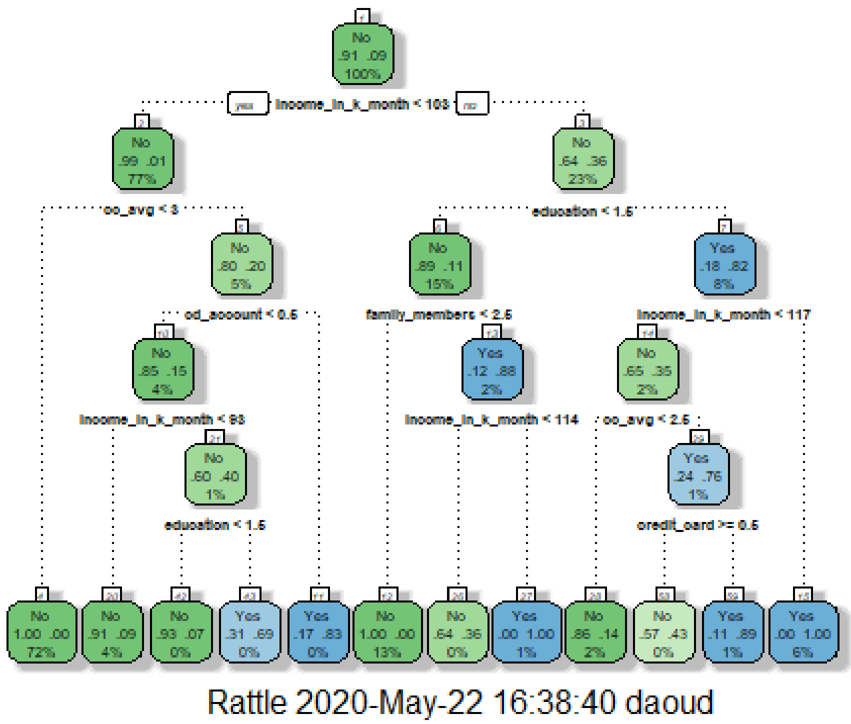
Plot the CP values and Tree:

```
plot(rpart_model)
```

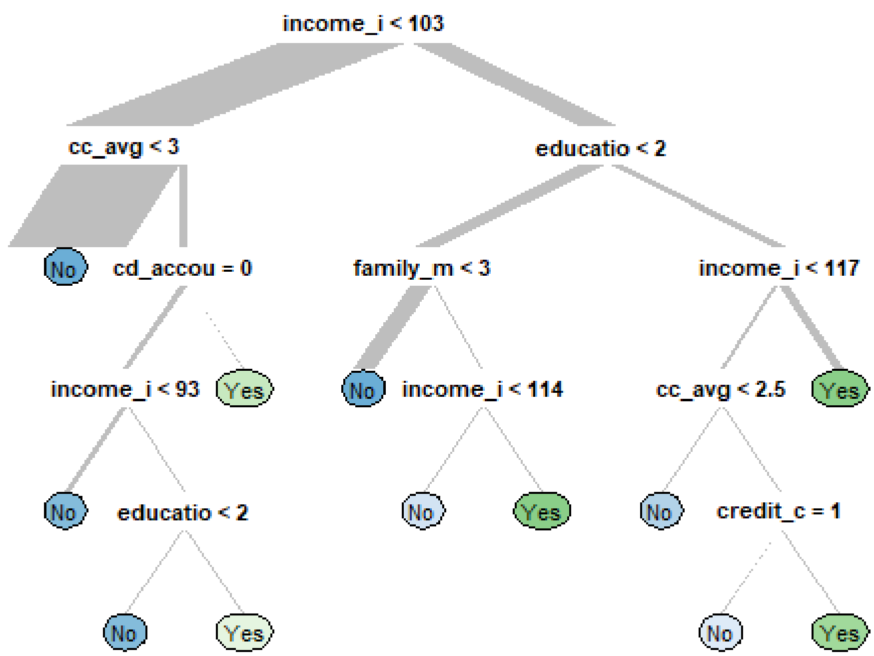



```
print(rpart_model$finalModel)

## n= 3255
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 3255 301 No (0.907526882 0.092473118)
##    2) income_in_k_month< 102.5 2504  32 No (0.987220447 0.012779553)
##      4) cc_avg< 2.95 2346   0 No (1.000000000 0.000000000) *
##      5) cc_avg>=2.95 158  32 No (0.797468354 0.202531646)
##       10) cd_account< 0.5 146  22 No (0.849315068 0.150684932)
##         20) income_in_k_month< 92.5 116  10 No (0.913793103 0.086206897) *
##         21) income_in_k_month>=92.5 30  12 No (0.600000000 0.400000000)
##           42) education< 1.5 14   1 No (0.928571429 0.071428571) *
##           43) education>=1.5 16   5 Yes (0.312500000 0.687500000) *
##       11) cd_account>=0.5 12   2 Yes (0.166666667 0.833333333) *
##    3) income_in_k_month>=102.5 751 269 No (0.641810919 0.358189081)
##      6) education< 1.5 485  51 No (0.894845361 0.105154639)
##       12) family_members< 2.5 429   2 No (0.995337995 0.004662005) *
##       13) family_members>=2.5 56   7 Yes (0.125000000 0.875000000)
##         26) income_in_k_month< 113.5 11   4 No (0.636363636 0.363636364) *
##         27) income_in_k_month>=113.5 45   0 Yes (0.000000000 1.000000000)
*
##      7) education>=1.5 266  48 Yes (0.180451128 0.819548872)
##       14) income_in_k_month< 116.5 74  26 No (0.648648649 0.351351351)
##         28) cc_avg< 2.45 49   7 No (0.857142857 0.142857143) *
```

```
##          29) cc_avg>=2.45 25   6 Yes (0.240000000 0.760000000)
##            58) credit_card>=0.5 7   3 No (0.571428571 0.428571429) *
##            59) credit_card< 0.5 18   2 Yes (0.111111111 0.888888889) *
##        15) income_in_k_month>=116.5 192   0 Yes (0.000000000 1.000000000) *

fancyRpartPlot(rpart_model$finalModel)
```




```
prp(rpart_model$finalModel, box.palette = "auto",branch.type = 5, yesno =
FALSE, faclen = 0)
```



Observation :

ROC reaches 1 when complexity parameter reaches 0 .

75% didn't loan had incom<103 and cc_avg <2.95 .

Predict both class and probabilities:

```
rpart_predict_test_prob <- predict(rpart_model, newdata = d_test, type =
"prob")
rpart_predict_test_class <- predict(rpart_model, newdata = d_test, type =
"raw")
```

The Confusion Matrix :

```
caret::confusionMatrix(rpart_predict_test_class, d_test$personal_loan,
positive = "Yes")

## Confusion Matrix and Statistics
##
##             Reference
## Prediction    No   Yes
##        No   1446    17
##        Yes     9   155
##
##                Accuracy : 0.984
##                  95% CI : (0.9767, 0.9895)
##     No Information Rate : 0.8943
```

```
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9137
##
##  Mcnemar's Test P-Value : 0.1698
##
##             Sensitivity : 0.90116
##             Specificity : 0.99381
##          Pos Pred Value : 0.94512
##          Neg Pred Value : 0.98838
##              Prevalence : 0.10572
##          Detection Rate : 0.09527
##    Detection Prevalence : 0.10080
##       Balanced Accuracy : 0.94749
##
##        'Positive' Class : Yes
##
```

Observation :

the Accuracy is 98.4% .

Sensitivity quite similar to Specificity.

Concordance - Discordance (overall : rarely used, specific to certain domains)

```
levels(d_test$personal_loan) <- c("0", "1")
Concordance(actuals = d_test$personal_loan, predictedScores =
rpart_predict_test_prob[,2])
```

```
## $Concordance
## [1] 0.9960521
##
## $Discordance
## [1] 0.003947894
##
## $Tied
## [1] 2.255141e-17
##
## $Pairs
## [1] 250260
```

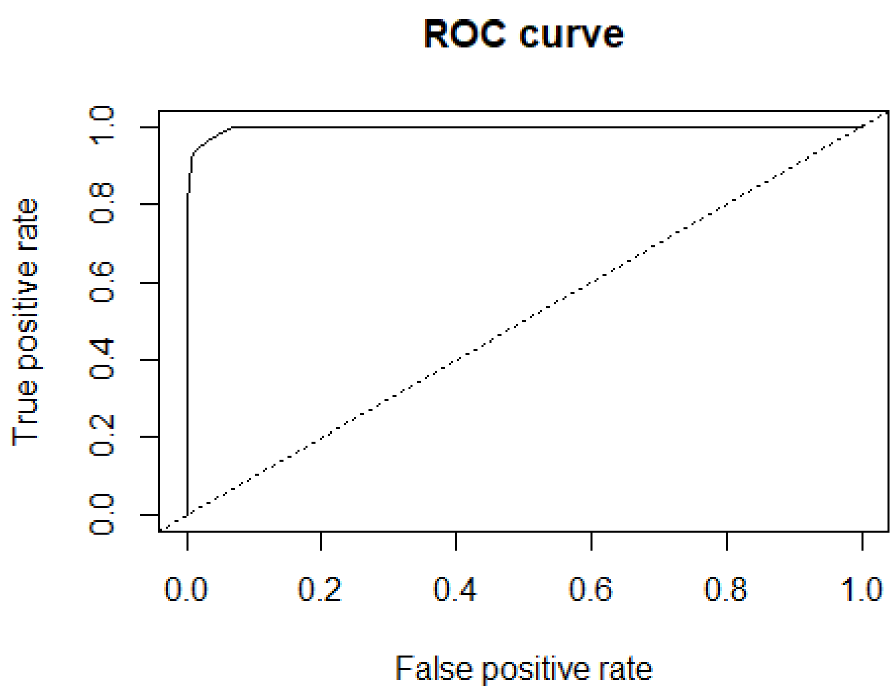
Observation :

Concordance is 99.60% , Probality of (Right) is 99.60% which is very Good.

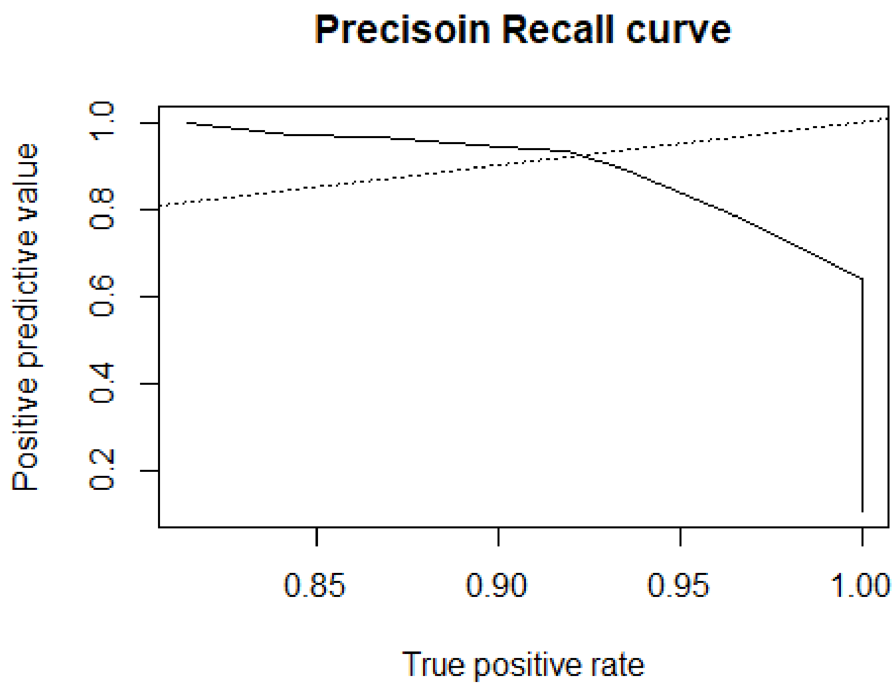
Discordance is 0.03%.

#3 ROC & Precision Recall Curves

```
# Creating the prediction object using ROCR library
levels(d_test$personal_loan) <- c("No", "Yes")
pred_obj_dtree = prediction(rpart_predict_test_prob[,2],
d_test$personal_loan)


# ROC curve
ROC_curve = performance(pred_obj_dtree, "tpr", "fpr")
plot(ROC_curve, main = "ROC curve")
abline(a=0, b= 1, lty = 3)
```




```
# Precision recall curve
precision_recall_dtree <- performance(pred_obj_dtree, "ppv", "tpr")
plot(precision_recall_dtree, main = "Precisoin Recall curve")
abline(a=0, b= 1, lty = 3)
```




```
# Computing the area under the curve
auc = performance(pred_obj_dtree,"auc");
auc = as.numeric(auc@y.values)
auc

## [1] 0.997075

# Computing Gini
gini = ineq(rpart_predict_test_prob[,2], type="Gini")
gini

## [1] 0.8866232
```

Observation :

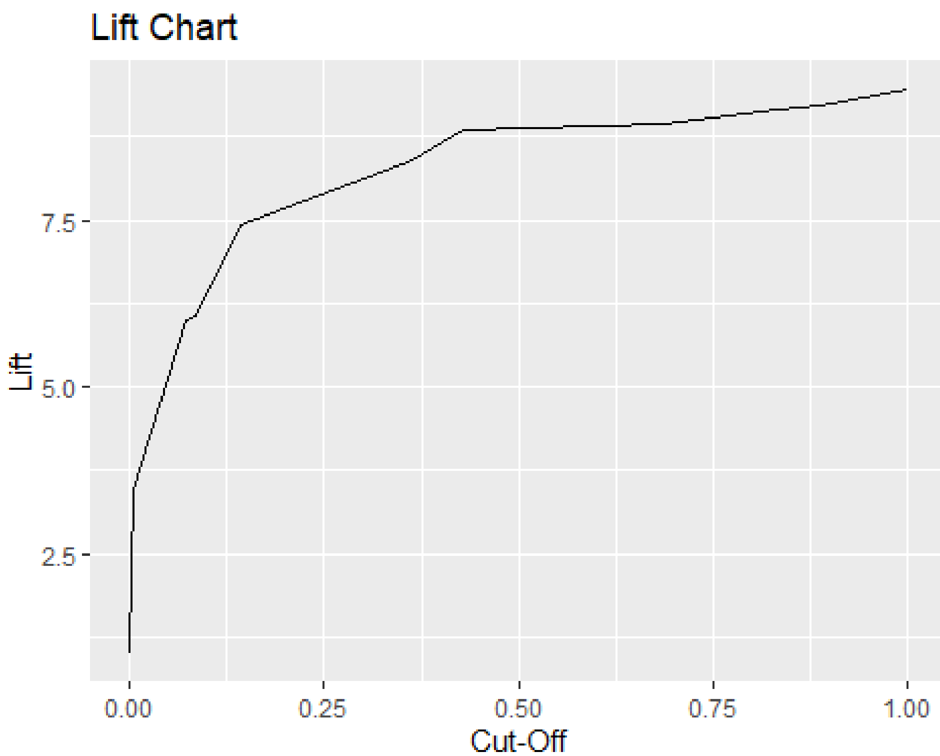
area under the curve AUC : 99.70%

gini :88.66%

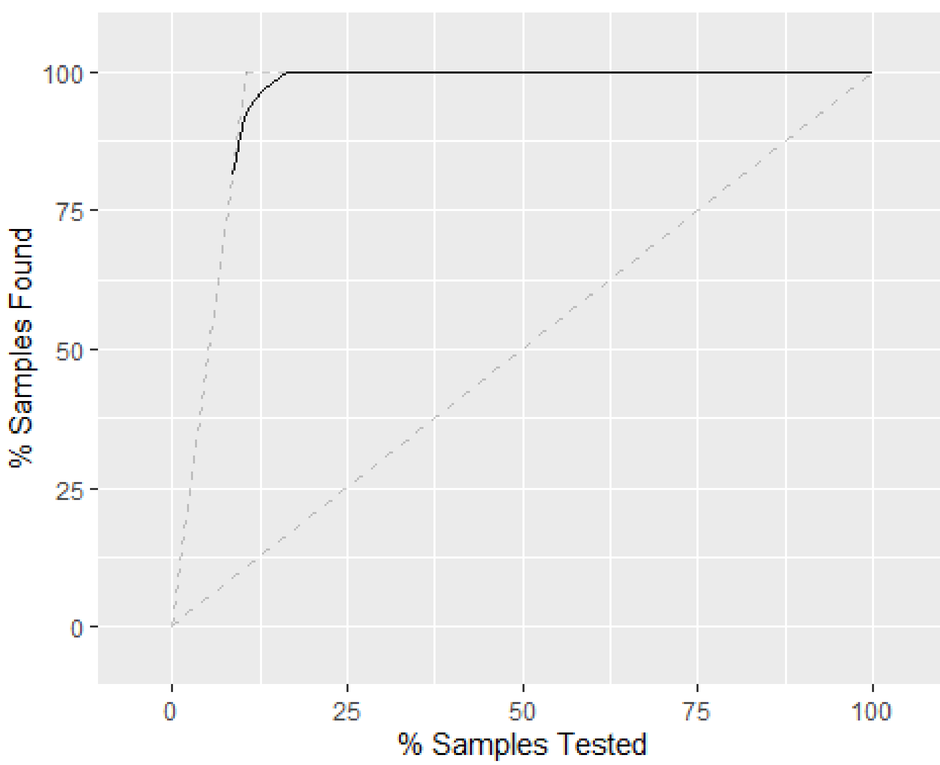
Gini score is merely a reformulation of the AUC: Gini= 2*AUC-1

Gain & Lift Chart

```
lift_dtree <- lift(d_test$personal_loan ~ rpart_predict_test_prob[,2], data =
d_test, class ="Yes")
ggplot(lift_dtree, plot = "lift")+ ggtitle("Lift Chart")
```




```
ggplot(lift_dtree, plot = "gain", valueitle("Gain Chart"))
```



KS table & KS plot

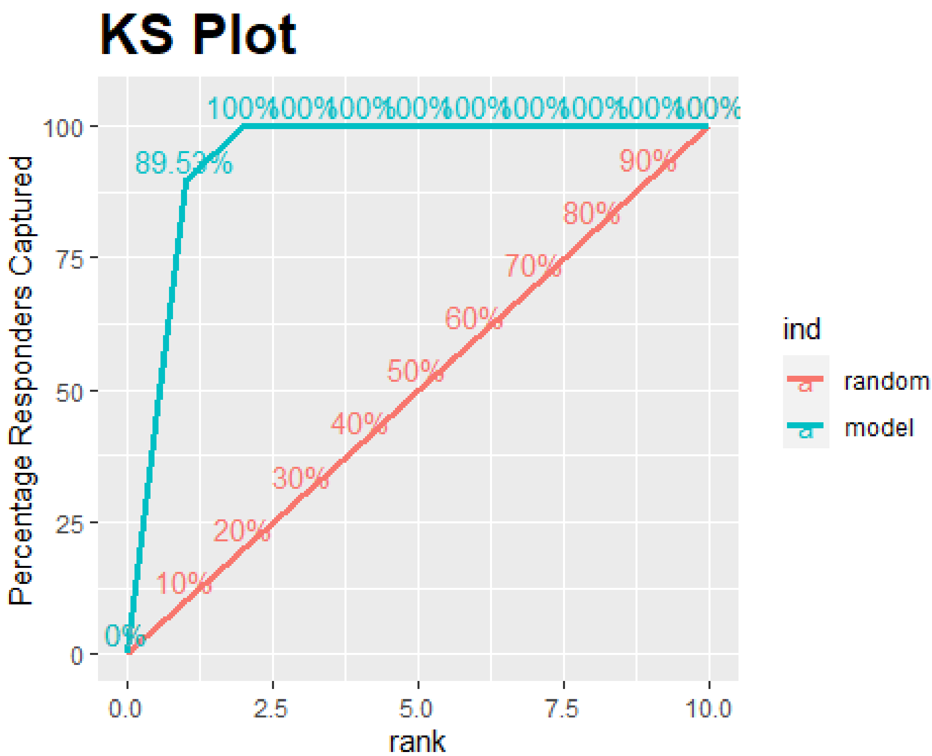
```r
ks_stat(d_test$personal_loan, rpart_predict_test_prob[,2]) # print KS
```

```
## [1] 0.8942
```

```r
ks_stat(d_test$personal_loan, rpart_predict_test_prob[,2], returnKSTable = T)
# print KS table
```

```
##    rank total_pop non_responders responders expected_responders_by_random
## 1     1       163              9        154                      17.23171
## 2     2       163            145         18                      17.23171
## 3     3       163            163          0                      17.23171
## 4     4       163            163          0                      17.23171
## 5     5       163            163          0                      17.23171
## 6     6       163            163          0                      17.23171
## 7     7       163            163          0                      17.23171
## 8     8       163            163          0                      17.23171
## 9     9       163            163          0                      17.23171
## 10   10       160            160          0                      16.91457
##    perc_responders perc_non_responders cum_perc_responders
## 1        0.8953488         0.006185567           0.8953488
## 2        0.1046512         0.099656357           1.0000000
## 3        0.0000000         0.112027491           1.0000000
## 4        0.0000000         0.112027491           1.0000000
## 5        0.0000000         0.112027491           1.0000000
## 6        0.0000000         0.112027491           1.0000000
## 7        0.0000000         0.112027491           1.0000000
## 8        0.0000000         0.112027491           1.0000000
## 9        0.0000000         0.112027491           1.0000000
## 10       0.0000000         0.109965636           1.0000000
##    cum_perc_non_responders difference
## 1              0.006185567  0.8891633
## 2              0.105841924  0.8941581
## 3              0.217869416  0.7821306
## 4              0.329896907  0.6701031
## 5              0.441924399  0.5580756
## 6              0.553951890  0.4460481
## 7              0.665979381  0.3340206
## 8              0.778006873  0.2219931
## 9              0.890034364  0.1099656
## 10             1.000000000  0.0000000
```

```r
ks_plot(d_test$personal_loan, rpart_predict_test_prob[,2]) # plot KS
```



Observation :

Ks : 94.15%.

by 20% you reache 99.30%.

train Random Forest :

```
set.seed(2020)
rf_model <- train(personal_loan ~ ., data = d_train,
                     method = "rf",
                     ntree = 301,    # number of tree
                     maxdepth = 15,
                     tuneLength = 15,
                     trControl = Ctrl)

## note: only 10 unique complexity parameters in default grid. Truncating the
grid to 10 .

## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy"
was not
## in the result set. ROC will be used instead.

rf_model

## Random Forest
##
```

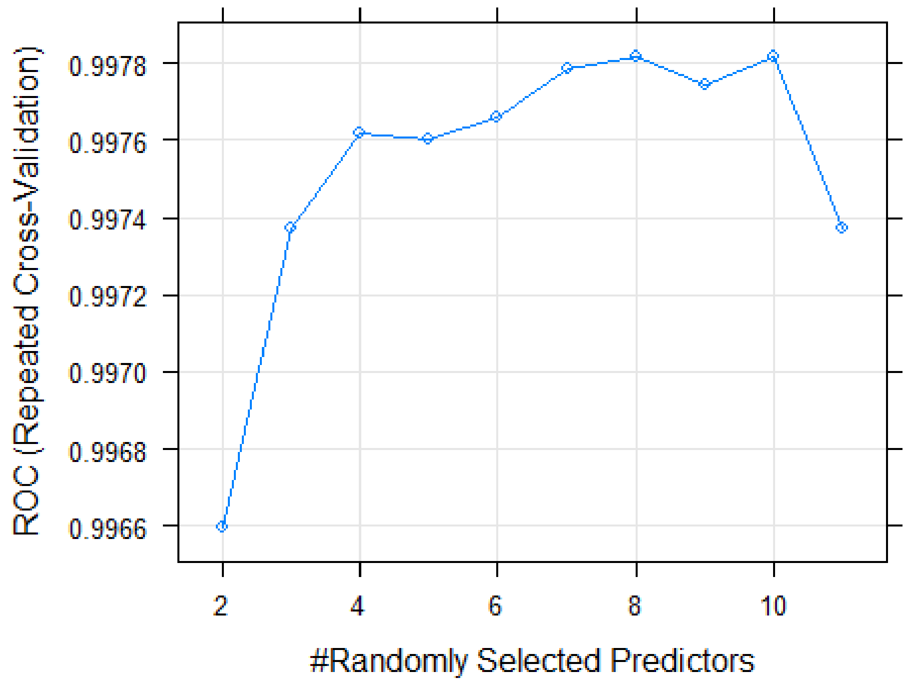
```
## 3255 samples
##   11 predictor
##    2 classes: 'No', 'Yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 2604, 2604, 2604, 2604, 2604, 2604, ...
## Resampling results across tuning parameters:
##
##   mtry  ROC        Sens       Spec
##    2    0.9965941  0.9986460  0.8372678
##    3    0.9973730  0.9983074  0.8781785
##    4    0.9976161  0.9979690  0.8870674
##    5    0.9976030  0.9978560  0.8903643
##    6    0.9976591  0.9975174  0.8914754
##    7    0.9977833  0.9972916  0.8948087
##    8    0.9978193  0.9971788  0.8992168
##    9    0.9977438  0.9968404  0.8959016
##   10    0.9978152  0.9969532  0.8936794
##   11    0.9973720  0.9970662  0.9003097
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 8.
```

Observation :

the best mtry value was 8.

Plot ROC and print the random forest:

```
plot(rf_model)
```




```
print(rf_model$finalModel)

##
## Call:
##  randomForest(x = x, y = y, ntree = 301, mtry = param$mtry, maxdepth = 15)
##                Type of random forest: classification
##                      Number of trees: 301
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 1.17%
## Confusion matrix:
##       No Yes  class.error
## No  2944  10  0.00338524
## Yes   28 273  0.09302326
```

Observation :

strongly support mtry=7 from the plot.

OOB error estimate rate: 1.17% which is good.

Predict both class and probabilities

```
rf_predict_test_prob <- predict(rf_model, newdata = d_test, type = "prob")
rf_predict_test_class <- predict(rf_model, newdata = d_test, type = "raw")
```

The Confusion Matrix (most common way of evaluating a model)

```
caret::confusionMatrix(rf_predict_test_class, d_test$personal_loan, positive
= "Yes")
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction   No  Yes
##        No  1445   12
##        Yes   10  160
##                                          
##                Accuracy : 0.9865         
##                  95% CI : (0.9796, 0.9915)
##     No Information Rate : 0.8943         
##     P-Value [Acc > NIR] : <2e-16         
##                                          
##                   Kappa : 0.9281         
##                                          
##  Mcnemar's Test P-Value : 0.8312         
##                                          
##             Sensitivity : 0.93023        
##             Specificity : 0.99313        
##          Pos Pred Value : 0.94118        
##          Neg Pred Value : 0.99176        
##              Prevalence : 0.10572        
##          Detection Rate : 0.09834        
##    Detection Prevalence : 0.10449        
##       Balanced Accuracy : 0.96168        
##                                          
##        'Positive' Class : Yes            
## 
```

Observation :

Accuracy is 98.65%

some difference between Sensitivity and Specificity.

Concordance - Discordance (overall : rarely used, specific to certain domains)

```
levels(d_test$personal_loan) <- c("0", "1")
Concordance(actuals = d_test$personal_loan, predictedScores =
rf_predict_test_prob[,2])
```

```
## $Concordance
## [1] 0.9973947
## 
```

```
## $Discordance
## [1] 0.00260529
##
## $Tied
## [1] -4.033232e-17
##
## $Pairs
## [1] 250260
```

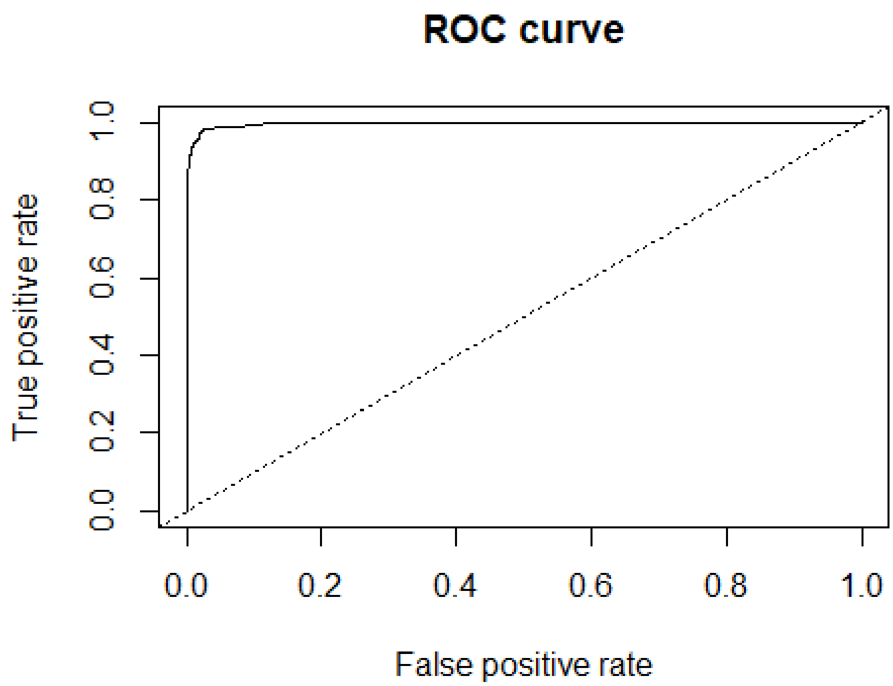
Observation :

Concordance is 99.73%, Probality of (Right) is 99.73% which is very Good.

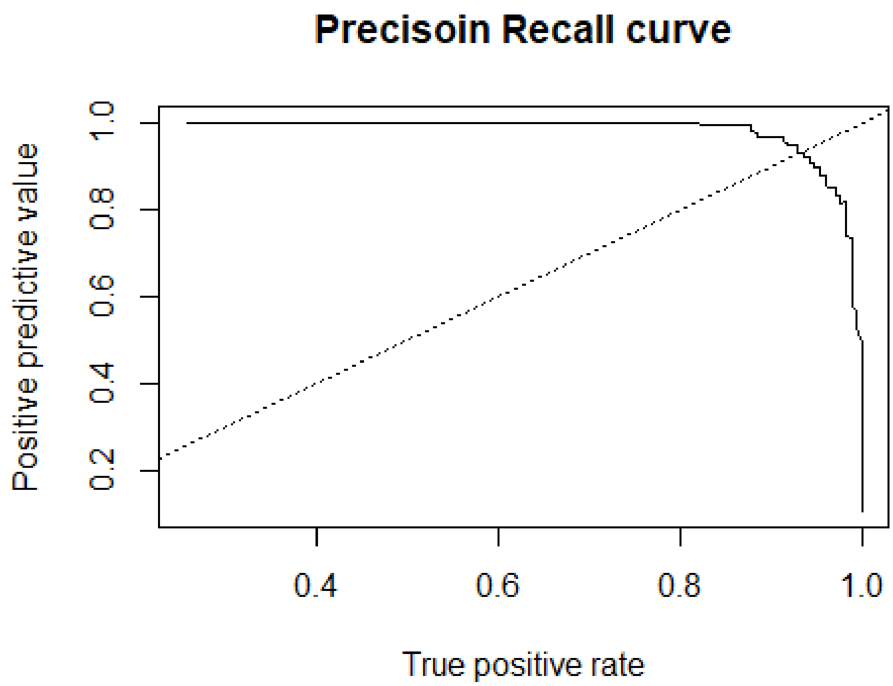
Discordance is 0.02%.

ROC & Precision Recall Curves:

```r
# Creating the prediction object using ROCR library
levels(d_test$personal_loan) <- c("No", "Yes")
pred_obj_rf = prediction(rf_predict_test_prob[,2], d_test$personal_loan)


# ROC curve
ROC_curve = performance(pred_obj_rf, "tpr", "fpr")
plot(ROC_curve, main = "ROC curve")
abline(a=0, b= 1, lty = 3)
```




```r
# Precision recall curve
precision_recall_rf <- performance(pred_obj_rf, "ppv", "tpr")
plot(precision_recall_rf, main = "Precisoin Recall curve")
abline(a=0, b= 1, lty = 3)
```




```
# Computing the area under the curve
auc = performance(pred_obj_rf,"auc");
auc = as.numeric(auc@y.values)
auc

## [1] 0.9974706

# Computing Gini
gini = ineq(rf_predict_test_prob[,2], type="Gini")
gini

## [1] 0.886092
```

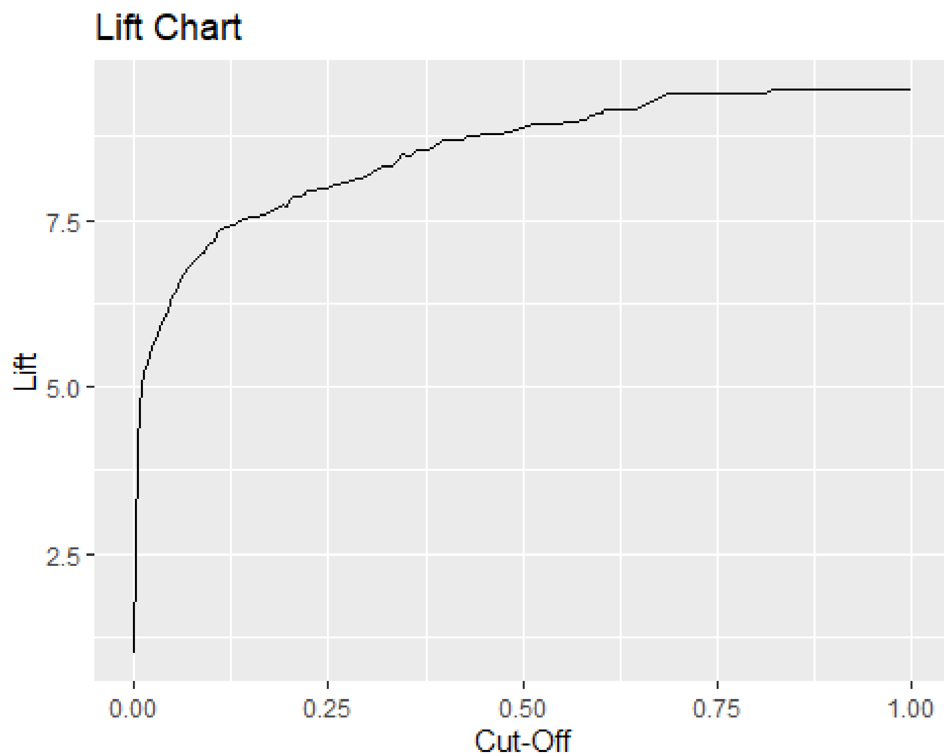
Observation :

the area under the curve : 99.74%

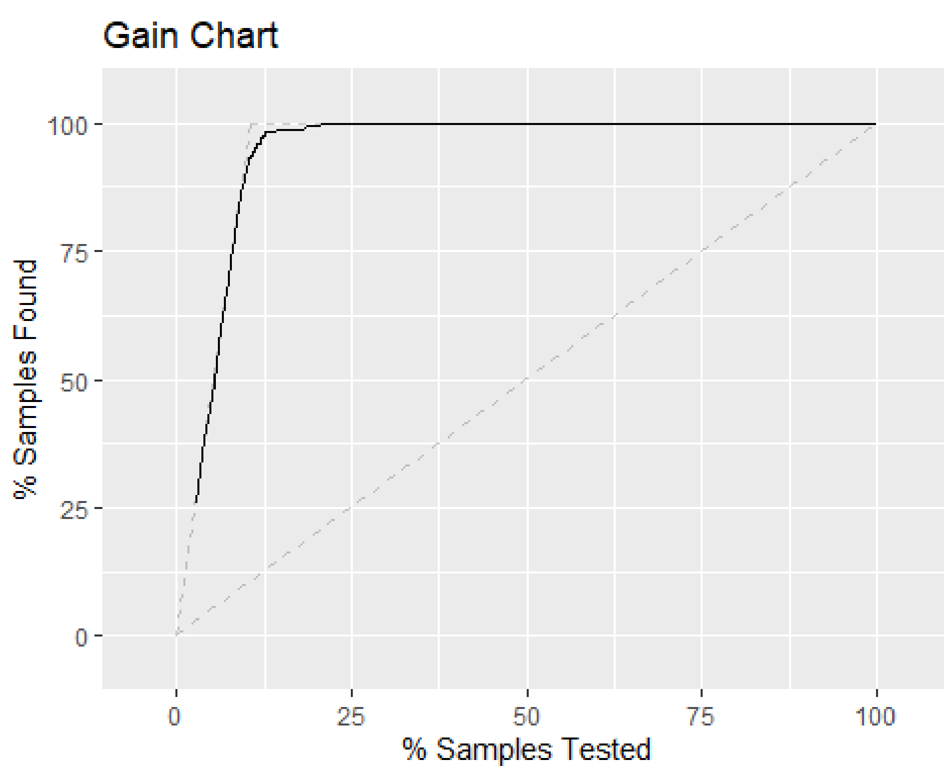
Gini : 88.60%

Gain & Lift Chart:

```
lift_rf <- lift(d_test$personal_loan ~ rf_predict_test_prob[,2], data =
d_test, class ="Yes")
ggplot(lift_rf, plot = "lift")+ ggtitle("Lift Chart")
```




```
ggplot(lift_rf, plot = "gain") + ggtitle("Gain Chart")
```



KS table & KS plot:

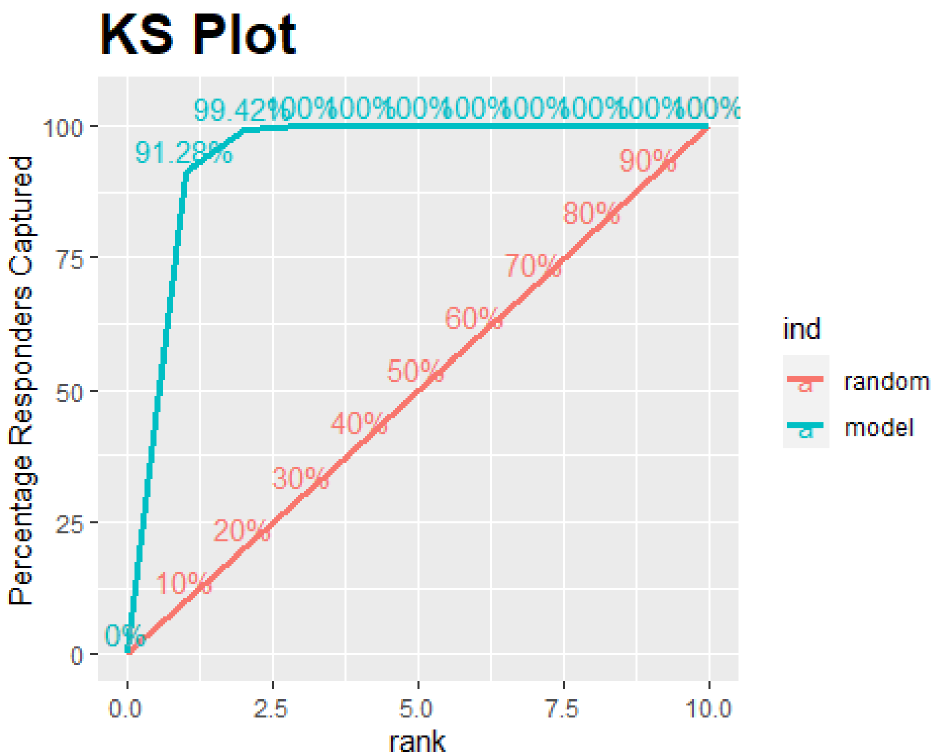
```r
ks_stat(d_test$personal_loan, rf_predict_test_prob[,2]) # print KS
```

```
## [1] 0.9087
```

```r
ks_stat(d_test$personal_loan, rf_predict_test_prob[,2], returnKSTable = T) #
print table KS
```

```
##    rank total_pop non_responders responders expected_responders_by_random
## 1     1       163              6        157                      17.23171
## 2     2       163            149         14                      17.23171
## 3     3       163            162          1                      17.23171
## 4     4       163            163          0                      17.23171
## 5     5       163            163          0                      17.23171
## 6     6       163            163          0                      17.23171
## 7     7       163            163          0                      17.23171
## 8     8       163            163          0                      17.23171
## 9     9       163            163          0                      17.23171
## 10   10       160            160          0                      16.91457
##    perc_responders perc_non_responders cum_perc_responders
## 1      0.912790698         0.004123711           0.9127907
## 2      0.081395349         0.102405498           0.9941860
## 3      0.005813953         0.111340206           1.0000000
## 4      0.000000000         0.112027491           1.0000000
## 5      0.000000000         0.112027491           1.0000000
## 6      0.000000000         0.112027491           1.0000000
## 7      0.000000000         0.112027491           1.0000000
## 8      0.000000000         0.112027491           1.0000000
## 9      0.000000000         0.112027491           1.0000000
## 10     0.000000000         0.109965636           1.0000000
##    cum_perc_non_responders difference
## 1              0.004123711  0.9086670
## 2              0.106529210  0.8876568
## 3              0.217869416  0.7821306
## 4              0.329896907  0.6701031
## 5              0.441924399  0.5580756
## 6              0.553951890  0.4460481
## 7              0.665979381  0.3340206
## 8              0.778006873  0.2219931
## 9              0.890034364  0.1099656
## 10             1.000000000  0.0000000
```

```r
ks_plot(d_test$personal_loan, rf_predict_test_prob[,2]) # plot KS
```



Observation :

Kolomogorov-Smirnov :KS : 96.43%.

10% of data give us 100% respond.

model validation:

```
RF_CM_train = table(d_test$personal_loan,rf_predict_test_class)
rf_ac<-(RF_CM_train[1,1]+RF_CM_train[2,2])/nrow(d_test)
print('the Accuracy for Random forest :')

## [1] "the Accuracy for Random forest :"

print(rf_ac*100,digits = 4)

## [1] 98.65

Rpart_CM_train = table(d_test$personal_loan,rpart_predict_test_class)
rpart_ac<-(Rpart_CM_train[1,1]+Rpart_CM_train[2,2])/nrow(d_test)
print('the Accuracy for decision tree :')

## [1] "the Accuracy for decision tree :"

print(rpart_ac*100,digits = 4)

## [1] 98.4
```

Remarks :

the best Accuracy model is Random Forest which is 98.65.

not to bad for decision tree too,very close accuracy to random forest .

but for decision tree we used random sample of 70 columes the datset , definitely training model for 3255 columes will not be the same as 70 columes.

I highly recommend Random Forest model.