Project - Thera Bank Case Study

Table of Contents

Thera Bank - Loan Purchase Modeling

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.

You are brought in as a consultant and your job is to build the best model which can classify the right customers who have a higher probability of purchasing the loan. You are expected to do the following:

- EDA of the data available. Showcase the results using appropriate graphs - (10 Marks)
- 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) - (10 Marks)
- 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) - (20 Marks)
- 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. - (20 Marks)

Hint : split <- sample.split(Thera_Bank$Personal Loan, SplitRatio = 0.7)

#we are splitting the data such that we have 70% of the data is Train Data and 30% of the data is my Test Data

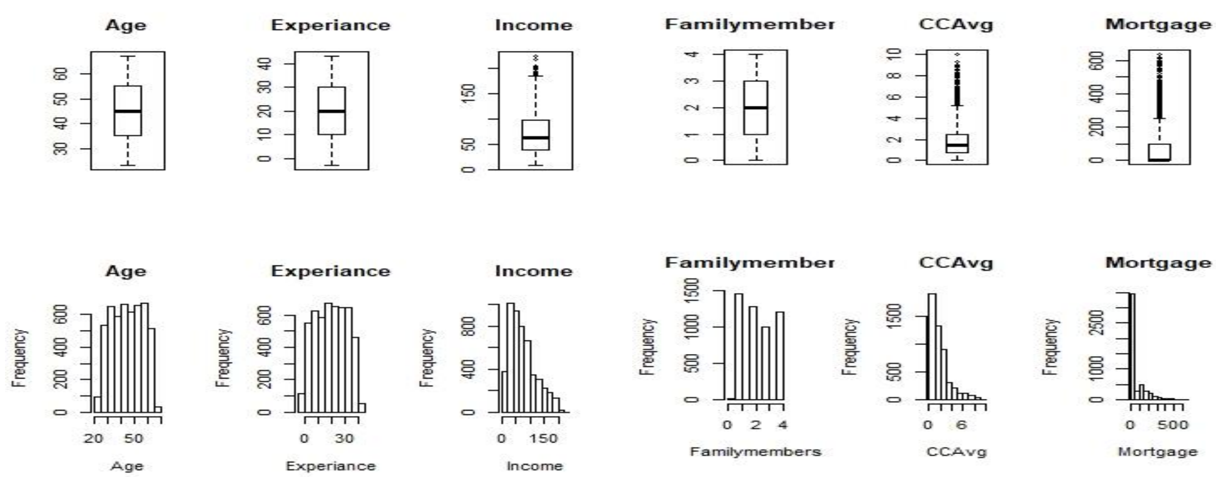
- train<- subset(Thera_Bank, split == TRUE)
- test<- subset(Thera_Bank, split == FALSE)

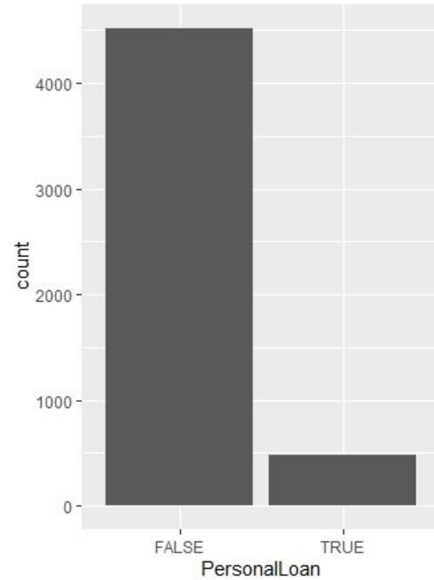
1. EDA - Basic data summary, Univariate, Bivariate analysis, graphs

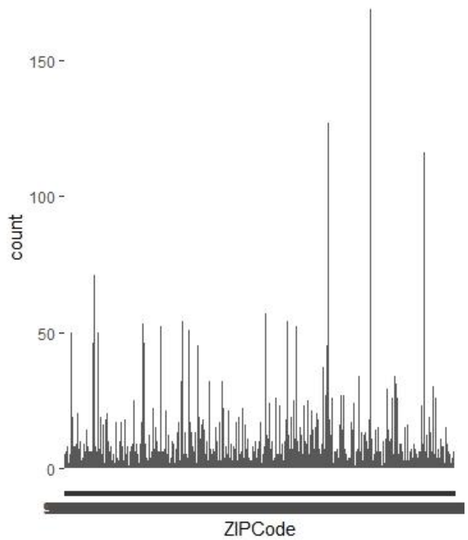
Basic data summary

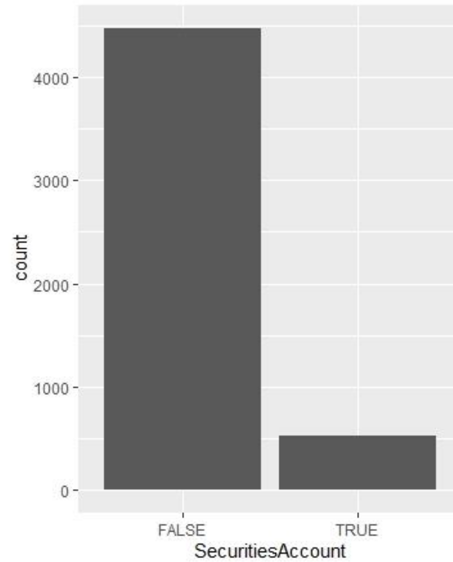
```
> summary(Thera_Bank_Personal_Loan_Modelling_Dataset)
       ID              Age          Experiance        Income          ZIPCode        Familymembers  
 Min.   :   1   Min.   :23.00   Min.   : 0.00   Min.   :  8.00   94720  : 169   Min.   :0.000  
 1st Qu.:1251   1st Qu.:35.00   1st Qu.:10.00   1st Qu.: 39.00   94305  : 127   1st Qu.:1.000  
 Median :2500   Median :45.00   Median :20.00   Median : 64.00   95616  : 116   Median :2.000  
 Mean   :2500   Mean   :45.34   Mean   :20.13   Mean   : 73.77   90095  :  71   Mean   :2.389  
 3rd Qu.:3750   3rd Qu.:55.00   3rd Qu.:30.00   3rd Qu.: 98.00   93106  :  57   3rd Qu.:3.000  
 Max.   :5000   Max.   :67.00   Max.   :43.00   Max.   :224.00   92037  :  54   Max.   :4.000  
                                                                 (Other):4406                  
     CCAvg        Education    Mortgage     PersonalLoan SecuritiesAccount CDAccount Online  
 Min.   : 0.000   1:2096    Min.   :  0.0   0:4520       0:4478            0:4698    0:2016  
 1st Qu.: 0.700   2:1403    1st Qu.:  0.0   1: 480       1: 522            1: 302    1:2984  
 Median : 1.500   3:1501    Median :  0.0                                                   
 Mean   : 1.938             Mean   : 56.5                                                   
 3rd Qu.: 2.500             3rd Qu.:101.0                                                   
 Max.   :10.000             Max.   :635.0                                                   

 CreditCard
 0:3530    
 1:1470    
```

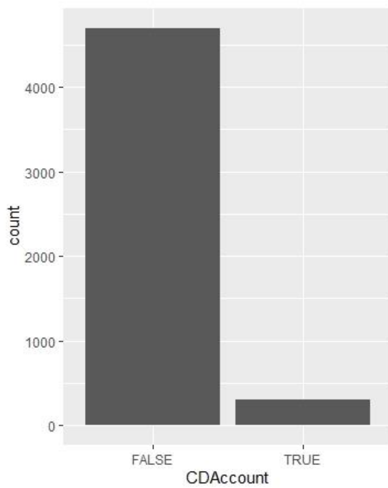
Univariate Analysis

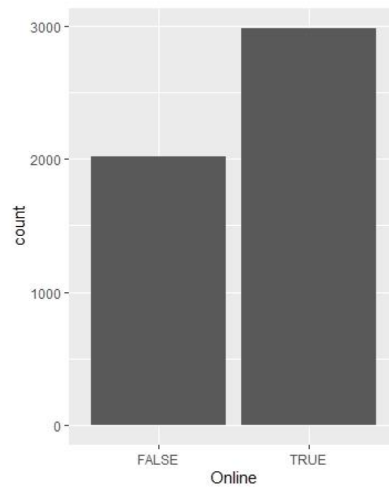


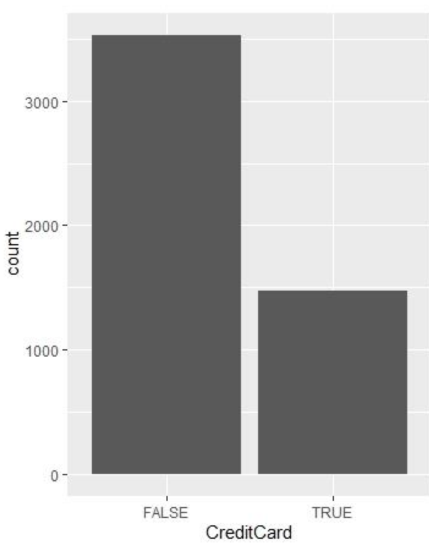

4000
3000
2000
1000
0
count
FALSE
TRUE
PersonalLoan

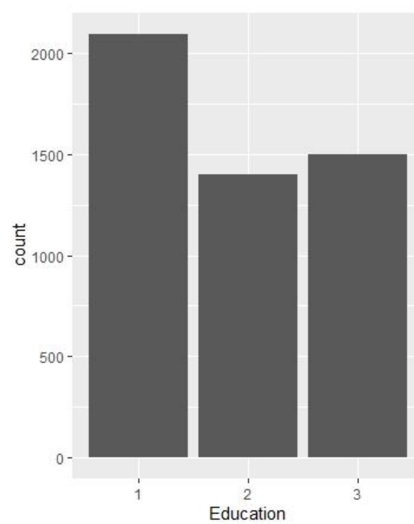

150
100
50
0
count
ZIPCode


4000
3000
2000
1000
0
count
FALSE
TRUE
SecuritiesAccount

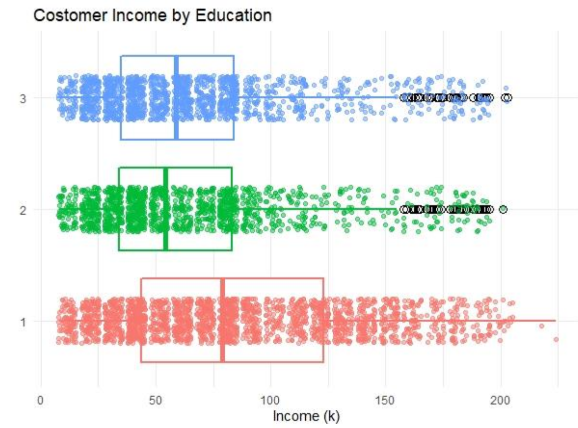

4000
3000
2000
1000
0
count
FALSE
TRUE
CDAccount

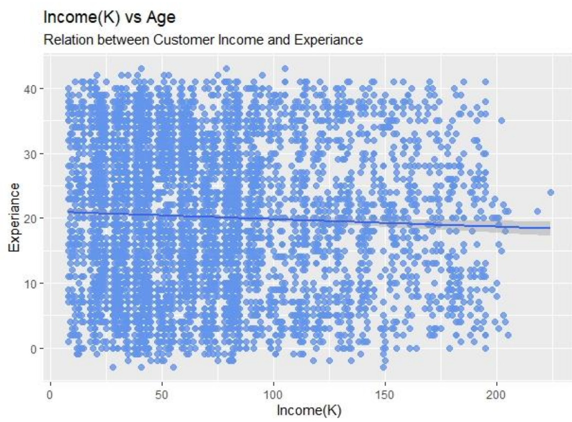

3000
2000
1000
0
count
FALSE
TRUE
Online

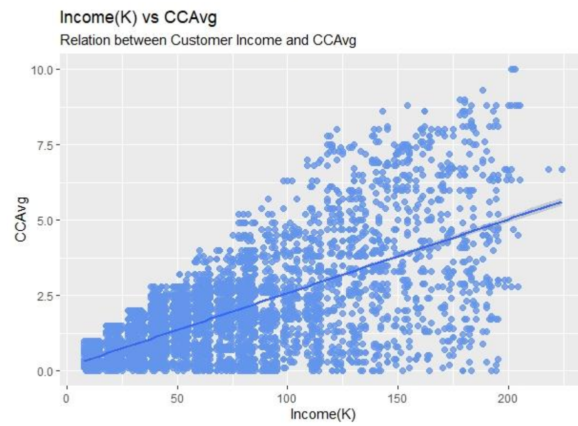

3000
2000
1000
0
count
FALSE
TRUE
CreditCard

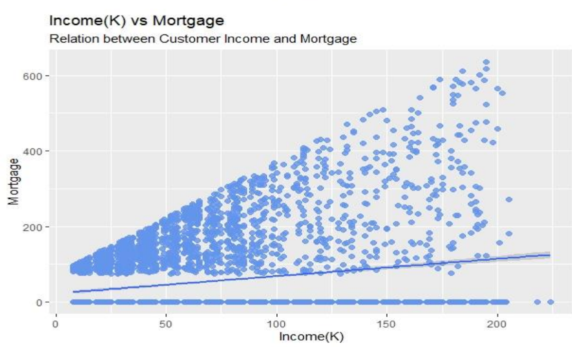

2000
1500
1000
500
0
count
1
2
3
Education

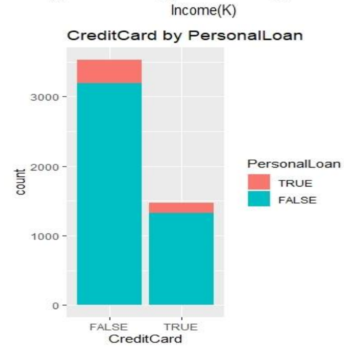
Bivariate Analysis

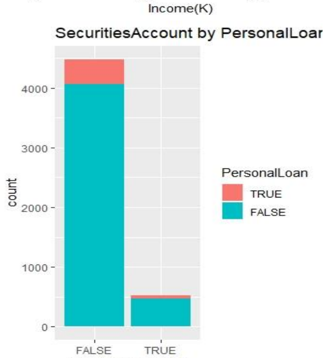


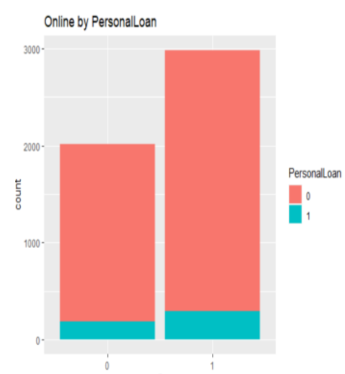






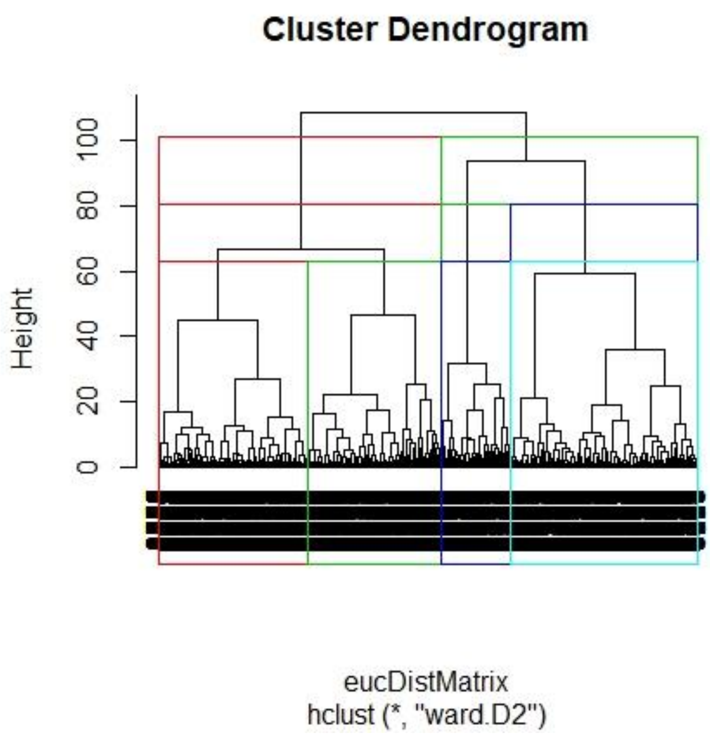


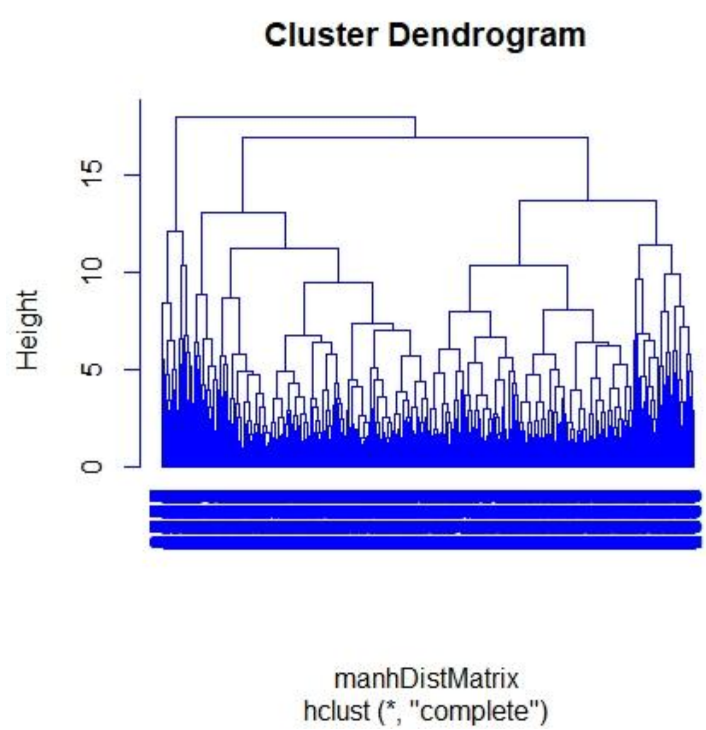


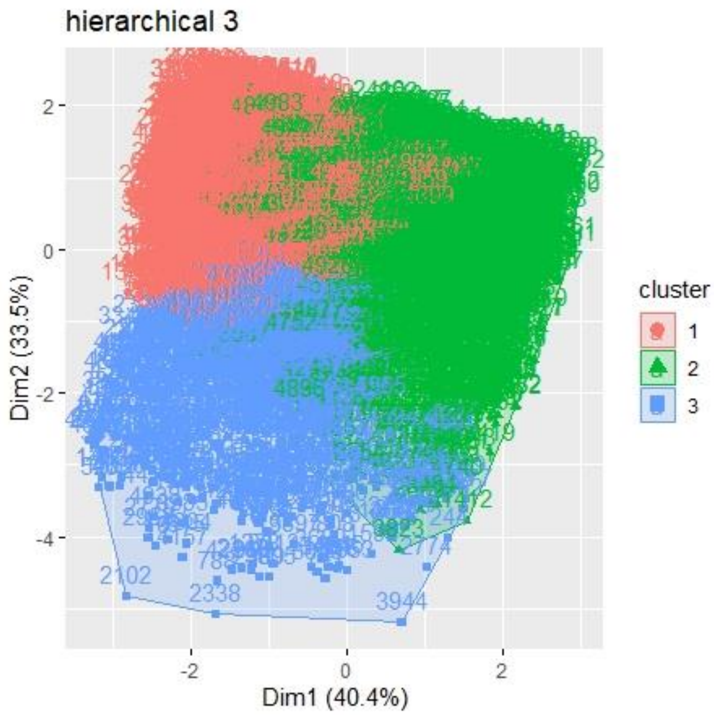


2.1 Apply Clustering algorithm < type, rationale>

Hierarchical clustering



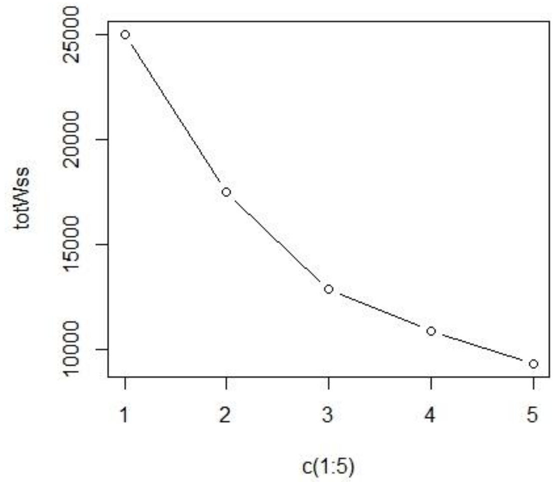




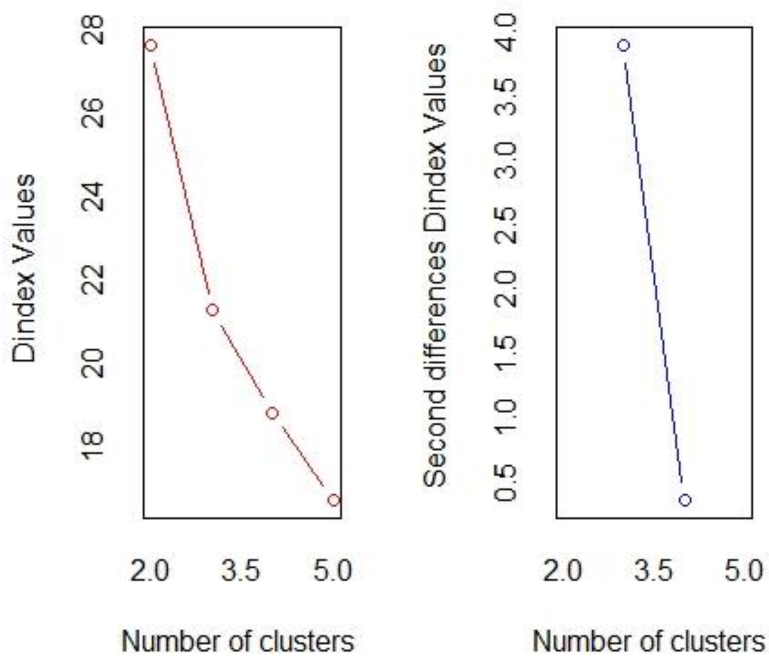

```
> hcluster.profile
  Cluster Number_of_Colleges        Age Experiance     Income Familymembers      CCAvg
1       1               1730 -0.9102506 -0.9107052 -0.3401925    -0.1118333 -0.3653484
2       2               2626  0.6925511  0.6869610 -0.1373049     0.1829813 -0.2374072
3       3                644 -0.3787353 -0.3547200  1.4737511    -0.4457100  1.9495094
```

K- Means clustering

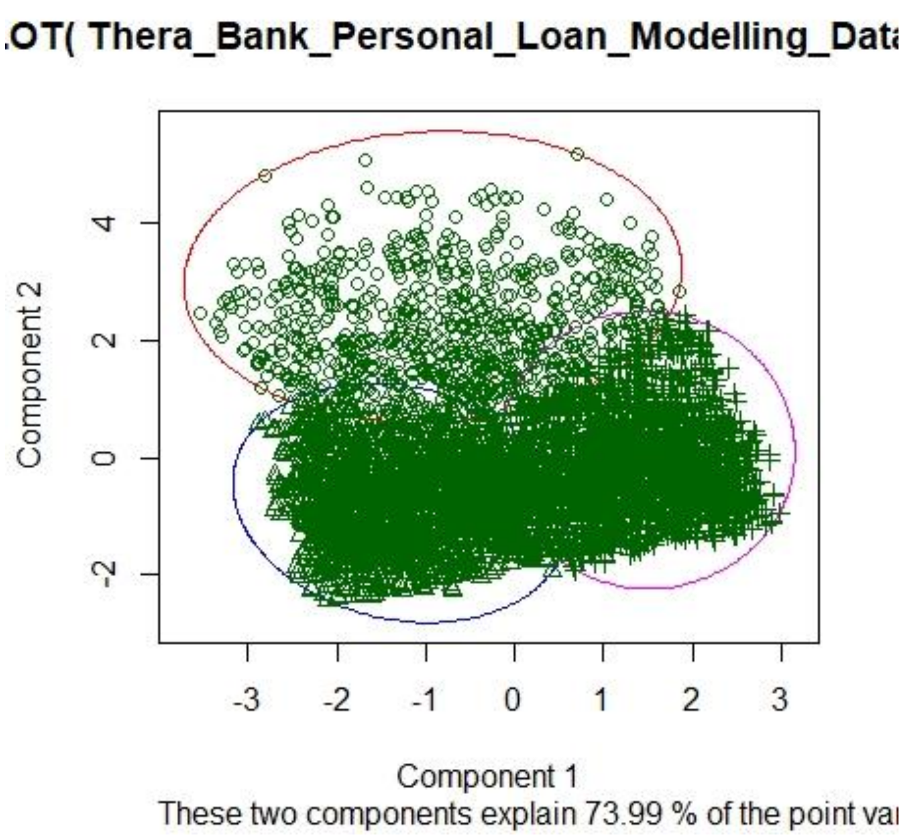
```
> clus
K-means clustering with 3 clusters of sizes 800, 2028, 2172

Cluster means:
         Age Experiance     Income Familymembers      CCAvg
1 -0.1648454 -0.1464603  1.5941746   -0.42879022  1.7263591
2 -0.8892334 -0.8908337 -0.2866121    0.19052831 -0.3275270
3  0.8909952  0.8857178 -0.3195627   -0.01996282 -0.3300472

Clustering vector:
   [1] 2 2 2 2 2 2 3 3 2 1 3 2 1 3 3 3 1 2 1 3 3 3 2 2 1 2 2 1 3 1 3 2 3 2 2 3 3 3 1 2 3 2 2 2 1 3 2 2 3 2 2 3 2 1
  [55] 2 1 3 3 2 1 3 1 2 2 1 1 3 3 3 3 1 3 1 1 1 1 3 3 3 3 3 3 1 2 3 2 2 3 3 2 1 2 2 3 3 2 1 1 3 3 3 3 3 2 3 2 2 2
 [109] 2 2 2 3 2 3 2 3 3 3 2 1 3 3 3 1 2 3 2 2 2 2 2 1 2 2 3 3 3 3 3 3 3 2 2 2 3 1 3 3 3 3 1 1 3 3 3 2 2 2 2 3 1 3
 [163] 2 2 3 2 2 2 3 2 2 3 1 3 1 2 3 2 3 3 3 2 2 1 3 2 3 1 3 3 3 3 3 3 1 2 1 3 2 1 2 2 2 3 3 2 3 2 2 3 3 2 3 1 3 2
 [217] 2 2 2 3 2 2 2 3 3 2 2 1 3 3 3 2 2 3 2 2 2 3 3 2 3 3 1 1 2 2 2 3 3 2 2 1 3 3 3 3 2 3 2 3 3 2 3 2 2 3 3 3 3 2
 [271] 3 2 2 2 2 3 2 2 3 1 2 3 2 3 2 2 3 2 1 2 1 2 2 2 2 3 2 3 2 1 2 1 1 1 3 3 3 2 1 3 3 1 2 2 3 2 3 1 2 3 3 1 3 3
 [325] 1 3 3 3 3 1 3 2 3 3 3 3 2 3 2 2 3 2 1 2 3 3 2 2 1 2 2 1 3 3 1 2 3 2 1 2 2 1 3 2 3 1 3 2 3 2 2 3 3 3 2 2 2 2
 [379] 3 2 3 3 3 2 3 2 2 2 3 1 3 3 3 3 2 3 3 2 3 2 1 2 1 3 3 1 1 3 3 3 1 3 2 2 3 2 2 3 2 3 3 2 1 1 3 2 2 2 3 2 3 2
 [433] 1 3 2 3 3 2 3 3 3 3 3 3 3 3 3 3 2 3 3 2 2 3 3 2 3 2 3 1 3 3 1 1 2 3 2 3 2 3 2 3 2 3 1 1 3 3 1 1 3 2 1 2 2 3
 [487] 3 2 2 3 2 2 3 1 1 2 3 3 2 3 3 3 2 2 2 2 2 3 3 3 1 3 2 2 2 2 2 3 3 2 2 3 3 2 3 2 3 1 2 3 2 3 2 3 2 3 3 2 1 2 3
 [541] 2 1 2 3 1 2 2 2 3 3 3 3 2 3 1 2 3 2 2 3 2 3 2 3 2 3 1 2 2 2 1 2 1 3 2 3 2 1 2 3 3 2 2 2 1 2 2 1 2 1 2 2 2 2
 [595] 3 2 1 1 3 2 3 3 2 3 2 3 2 2 2 2 3 3 3 3 1 1 2 2 3 3 2 2 2 2 2 3 2 2 3 2 2 2 3 3 3 3 1 3 2 3 1 1 3 1 3 2 3 3
 [649] 3 2 1 2 2 3 1 3 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 2 3 3 2 3 2 3 1 3 3 3 1 3 1 1 2 2 3 2 3 3 2 2 2 1 2 3 1 3 3 2 2
 [703] 1 1 1 3 1 3 2 2 2 3 2 2 3 3 2 3 3 3 3 3 2 3 3 1 3 3 1 3 1 2 2 3 3 2 3 3 2 3 1 3 2 3 1 2 3 3 2 3 1 3 3 3 2 3
 [757] 3 3 3 3 2 3 2 3 2 3 2 3 2 2 1 2 1 1 3 3 1 3 3 1 2 1 1 1 1 1 3 1 3 2 3 3 1 2 3 3 2 2 2 2 1 3 2 3 3 1 3 1 3 3
 [811] 2 3 2 3 2 3 3 2 3 3 3 2 3 2 2 2 3 3 2 3 2 3 2 3 1 1 2 2 2 2 2 1 2 3 3 2 3 2 3 2 3 2 2 2 3 3 3 3 2 1 3 3 3 3
 [865] 2 3 2 3 1 3 2 3 2 2 2 3 2 2 2 3 3 1 1 3 2 2 3 1 1 2 3 1 2 3 2 2 1 3 3 1 2 3 3 2 2 3 2 3 3 1 3 3 2 3 3 1 1 1
 [919] 2 3 2 2 2 3 3 2 2 3 2 3 2 2 3 3 3 1 3 2 3 3 1 1 3 2 2 3 2 3 2 2 2 3 1 3 1 3 2 3 3 3 1 2 3 2 2 1 3 3 3 2 3 1
 [973] 2 1 3 3 3 3 3 3 2 2 3 1 3 1 1 3 3 2 2 2 2 1 2 2 2 2 3 3
 [ reached getOption("max.print") -- omitted 4000 entries ]

Within cluster sum of squares by cluster:
[1] 2991.347 4679.670 5162.376
 (between_SS / total_SS =  48.7 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
```




```
> nc = NbClust(Thera_Bank_Personal_Loan_Modelling_Dataset %>% select(2, 3, 4, 6, 7), min.nc = 2, max.nc = 5, method =
 "kmeans")
*** : The Hubert index is a graphical method of determining the number of clusters.
                In the plot of Hubert index, we seek a significant knee that corresponds to a
                significant increase of the value of the measure i.e the significant peak in Hubert
                index second differences plot.

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

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

                   ***** Conclusion *****

* According to the majority rule, the best number of clusters is  3


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



Dindex Values
Number of clusters
Second differences Dindex Values
Number of clusters



.OT(Thera_Bank_Personal_Loan_Modelling_Data
Component 2
Component 1
These two components explain 73.99 % of the point va

2.2 Clustering Output interpretation < number of clusters, remarks to make it meaningful to understand>

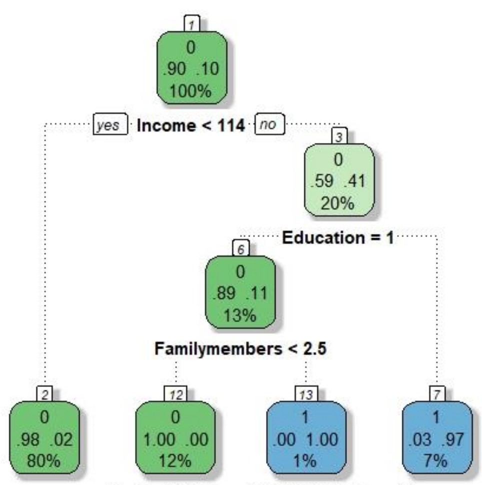
Its clear that the best number of clusters are 3, based on the variables which is :

- Age
- Experience
- Income
- Familymembers
- CCAvg
- Education

3.1 Applying CART <plot the tree>

```
# Building the CART model
# formula - response variable~predictor variables
# data - dataset
# method - "class" - for classification, "anova" for regression
# control - tree control parameters
cart_model1 <- rpart(formula = PersonalLoan~., data = train, method = "class", control = r.ctrl)
cart_model1
```

```
> cart_model1
n= 3461

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

 1) root 3461 338 0 (0.902340364 0.097659636)
   2) Income< 113.5 2755  52 0 (0.981125227 0.018874773)
     4) CCAvg< 2.95 2548   6 0 (0.997645212 0.002354788) *
     5) CCAvg>=2.95 207  46 0 (0.777777778 0.222222222)
      10) CDAccount< 0.5 185  30 0 (0.837837838 0.162162162) *
      11) CDAccount>=0.5 22   6 1 (0.272727273 0.727272727) *
   3) Income>=113.5 706 286 0 (0.594900850 0.405099150)
     6) Education< 1.5 463  51 0 (0.889848812 0.110151188)
      12) Familymembers< 2.5 411   0 0 (1.000000000 0.000000000) *
      13) Familymembers>=2.5 52   1 1 (0.019230769 0.980769231) *
     7) Education>=1.5 243   8 1 (0.032921811 0.967078189) *
>
```

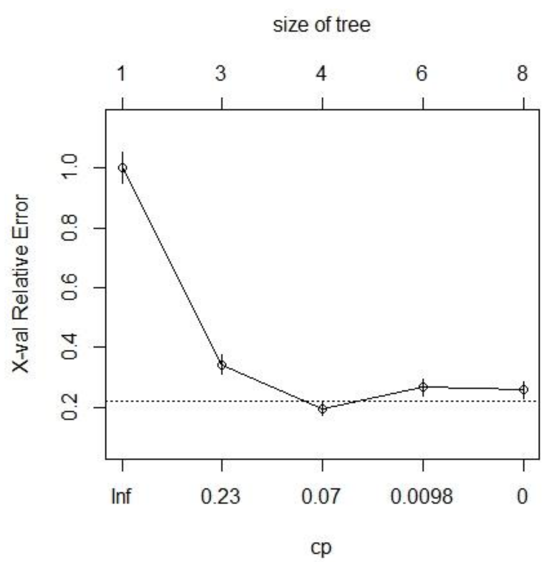


Rattle 2020-Jun-17 11:06:42 ahmasiri

3.2 Interpret the CART model output <pruning, remarks on pruning, plot the pruned tree>

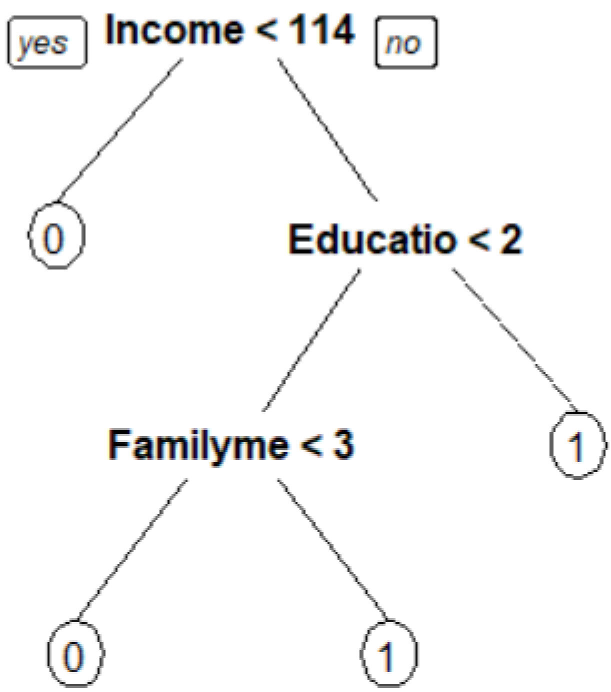
```
#Clearly we need to prune this tree
## Model Tuning
#The cost complexity table can be obtained using the printcp or plotcp functions
printcp(cart_model1)
plotcp(cart_model1)
```

```
> printcp(cart_model1)

Classification tree:
rpart(formula = PersonalLoan ~ ., data = train, method = "class",
    control = r.ctrl)

Variables actually used in tree construction:
[1] CCAvg         CDAccount     Education     Familymembers Income

Root node error: 338/3461 = 0.09766

n= 3461

        CP nsplit rel error  xerror     xstd
1 0.335799      0   1.00000 1.00000 0.051669
2 0.147929      2   0.32840 0.34320 0.031326
3 0.014793      3   0.18047 0.19822 0.023981
4 0.000000      5   0.15089 0.17160 0.022342
```




```
  # The unncessarily complex tree above can be pruned using a cost complexity threshold.
# Using a complexity threshold of 0.07 gives us a relatively simpler tree.
cart_model2 = prune(cart_model1, cp= 0.07 ,"CP")
printcp(cart_model2)
cart_model2
```

```
> cart_model2
n= 3461

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

 1) root 3461 338 0 (0.90234036 0.09765964)
   2) Income< 113.5 2755  52 0 (0.98112523 0.01887477) *
   3) Income>=113.5 706 286 0 (0.59490085 0.40509915)
     6) Education< 1.5 463  51 0 (0.88984881 0.11015119)
      12) Familymembers< 2.5 411   0 0 (1.00000000 0.00000000) *
      13) Familymembers>=2.5 52   1 1 (0.01923077 0.98076923) *
     7) Education>=1.5 243   8 1 (0.03292181 0.96707819) *
>
```




```
#Let us check the variable importance
cart_model1$variable.importance
```

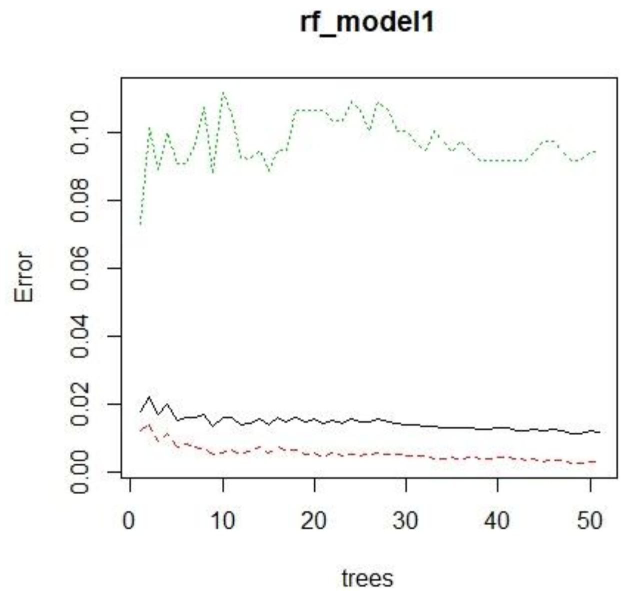
```
> cart_model1$variable.importance
    Education        Income Familymembers         CCAvg     CDAccount      Mortgage            ID
  234.0454563   167.6617764   145.6398112    87.9468309    55.1643674    20.7005453     3.8526001
      ZIPCode           Age    Experiance
    1.1416375     0.5708188     0.5708188
```

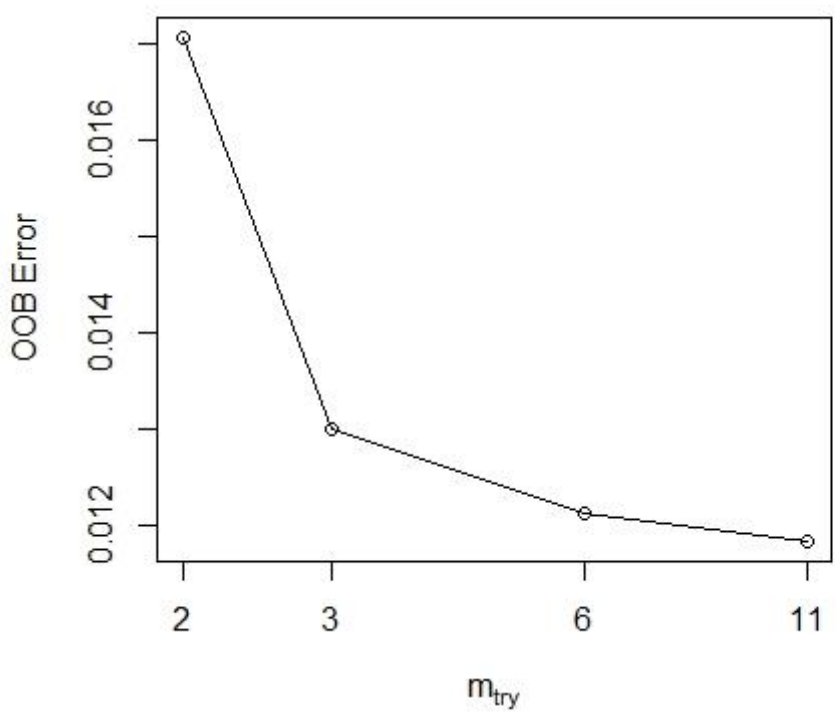
3.3 Applying Random Forests<plot the tree>

```
set.seed(500) # To ensure reproducibility

rf_model2 = tuneRF(x = select(train, -1,-5,-10), # matrix or data frame of predictor/independent
                   y = train$PersonalLoan, # response vector (factor for classification, numeric
                   mtrystart = 5, # starting value of mtry
                   stepfactor=1.5, # at each iteration, mtry is inflated (or deflated) by this v
                   ntree=51, # number of trees built for each mtry value
                   improve=0.0001, # the (relative) improvement in OOB error must be by this muc
                   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 #
)

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

```
mtry = 3  OOB error = 1.5%
Searching left ...
mtry = 2        OOB error = 2.14%
-0.4230769 1e-04
Searching right ...
mtry = 6        OOB error = 1.21%
0.1923077 1e-04
mtry = 11       OOB error = 1.33%
-0.0952381 1e-04
```

```
set.seed(1000) # To ensure reproducibility

rf_model1 = randomForest(
  PersonalLoan ~ .,
  data = select(train, -1,-5),
  ntree = 51,
  mtry = 6,
  nodesize = 10,
  importance = TRUE
)

#Plot the model to determine the optimum number of trees
plot(rf_model1)
```

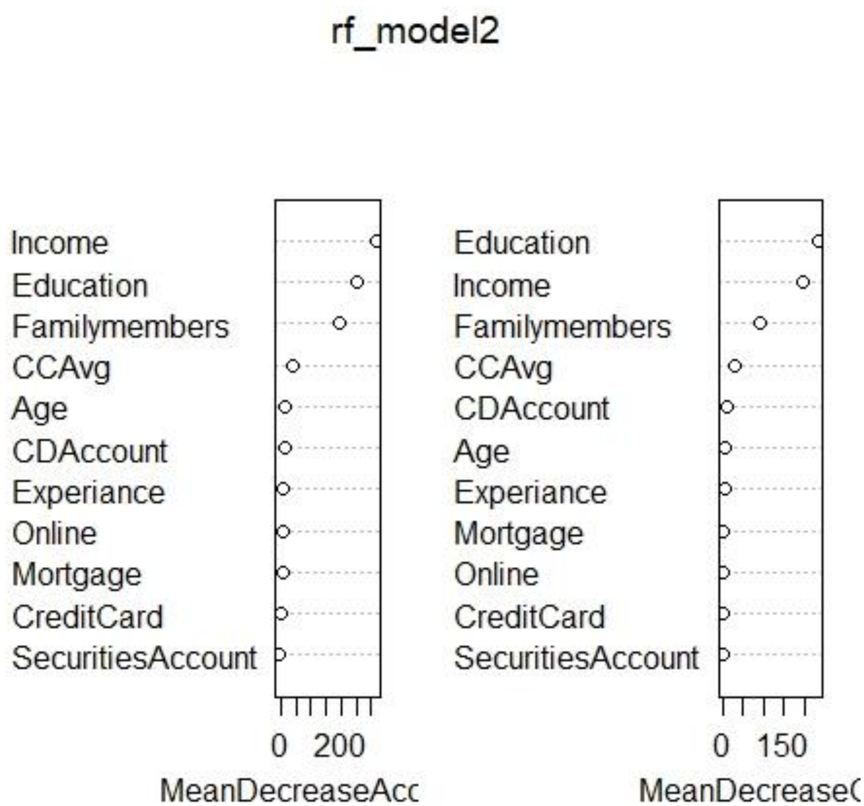

rf_model1
Error
0.00
0.02
0.04
0.06
0.08
0.10
0
10
20
30
40
50
trees


OOB Error
0.012
0.014
0.016
2
3
6
11
m_{try}

3.4 Interpret the RF model output <with remarks, making it meaningful for everybody>

```
> print(rf_model1$importance)
                              0             1 MeanDecreaseAccuracy MeanDecreaseGini
Age                0.0032501995  0.0009298683         0.0030322006         8.016937
Experiance         0.0029425875  0.0031841520         0.0029690604         8.870148
Income             0.1276652224  0.4670215697         0.1595792069       176.746149
Familymembers      0.0515593898  0.0687370806         0.0532224078        88.647178
CCAvg              0.0252924876  0.0856186281         0.0309823650        75.305774
Education          0.0703911040  0.1450534084         0.0774301047       167.096267
Mortgage           0.0007570444 -0.0028172208         0.0004457498         6.766673
SecuritiesAccount -0.0001355887  0.0001278486        -0.0001082247         1.055305
CDAccount          0.0030324419  0.0175637736         0.0043660577        21.806958
Online             0.0002712103  0.0004733529         0.0002926984         1.275832
CreditCard         0.0005491377  0.0017643225         0.0006528993         1.116115
```

```
# 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.

print(rf_model1$importance)
```



4.1 Confusion matrix interpretation

```r
# We will compare all the 4 models that we created earlier - rf_model1, rf_model2, cart_model1, cart_model2
# Predict PersonalLoan class and probability for all 4 models

# Predict on test data using cart_model1
cart_model1_predict_class = predict(cart_model1, test, type = 'class')
cart_model1_predict_score = predict(cart_model1, test, type = 'prob')

# Predict on test data using cart_model2
cart_model2_predict_class = predict(cart_model2, test, type = 'class')
cart_model2_predict_score = predict(cart_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_cart_model1 = table(test$PersonalLoan, cart_model1_predict_class)
conf_mat_cart_model1

conf_mat_cart_model2 = table(test$PersonalLoan, cart_model2_predict_class)
conf_mat_cart_model2

conf_mat_rf_model1 = table(test$PersonalLoan, rf_model1_predict_class)
conf_mat_rf_model1

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

```
> conf_mat_cart_model1
   cart_model1_predict_class
       0    1
  0 1377   20
  1   26  116
> conf_mat_cart_model2 = table(test$PersonalLoan, cart_model2_predict_class)
> conf_mat_cart_model2
   cart_model2_predict_class
       0    1
  0 1388    9
  1   32  110
> conf_mat_rf_model1 = table(test$PersonalLoan, rf_model1_predict_class)
> conf_mat_rf_model1
   rf_model1_predict_class
       0    1
  0 1392    5
  1   21  121
> conf_mat_rf_model2 = table(test$PersonalLoan, rf_model2_predict_class)
> conf_mat_rf_model2
   rf_model2_predict_class
       0    1
  0 1391    6
  1   23  119
```

4.2 Interpretation of other Model Performance Measures <KS, AUC, GINI>

```
# KS
# Using library ROCR functions prediction and performance
pred_cart_model1 = prediction(cart_model1_predict_score[, 2], test$PersonalLoan)
perf_cart_model1 = performance(pred_cart_model1,"tpr","fpr")
ks_cart_model1 = max(attr(perf_cart_model1,'y.values')[[1]] - attr(perf_cart_model1,'x.values')[[1]])

pred_cart_model2 = prediction(cart_model2_predict_score[, 2], test$PersonalLoan)
perf_cart_model2 = performance(pred_cart_model2,"tpr","fpr")
ks_cart_model2 = max(attr(perf_cart_model2,'y.values')[[1]] - attr(perf_cart_model2,'x.values')[[1]])

pred_rf_model1 = prediction(rf_model1_predict_score[, 2], test$PersonalLoan)
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$PersonalLoan)
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]])

# AUC
# Using library ROCR
auc_cart_model1 = performance(pred_cart_model1, measure = "auc")
auc_cart_model1 = auc_cart_model1@y.values[[1]]

auc_cart_model2 = performance(pred_cart_model2, measure = "auc")
auc_cart_model2 = auc_cart_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]]

# Gini
# Using library ineq
gini_cart_model1 = ineq(cart_model1_predict_score[, 2],"gini")

gini_cart_model2 = ineq(cart_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")
```

4.3 Remarks on Model validation exercise <Which model performed the best>

```
# Comparing models

cart_model1_metrics = c(accuracy_cart_model1, sensitivity_cart_model1, specificity_cart_model1,
                        precision_cart_model1, ks_cart_model1, auc_cart_model1, gini_cart_model1,
                        concordance_cart_model1$Concordance)

cart_model2_metrics = c(accuracy_cart_model2, sensitivity_cart_model2, specificity_cart_model2,
                        precision_cart_model2, ks_cart_model2, auc_cart_model2, gini_cart_model2,
                        concordance_cart_model2$Concordance)

rf_model1_metrics = c(accuracy_rf_model1, sensitivity_rf_model1, specificity_rf_model1,
                      precision_rf_model1, ks_rf_model1, auc_rf_model1, gini_rf_model1,
                      concordance_rf_model1$Concordance)

rf_model2_metrics = c(accuracy_rf_model2, sensitivity_rf_model2, specificity_rf_model2,
                      precision_rf_model2, ks_rf_model2, auc_rf_model2, gini_rf_model2,
                      concordance_rf_model2$Concordance)

comparison_table = data.frame(cart_model1_metrics, cart_model2_metrics, rf_model1_metrics, rf_model2_metrics)

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

comparison_table
```

```
> comparison_table
            cart_model1_metrics cart_model2_metrics rf_model1_metrics rf_model2_metrics
Accuracy              0.9701105           0.9733593         0.9831059         0.9811566
Sensitivity           0.8169014           0.7746479         0.8521127         0.8380282
Specificity           0.9856836           0.9935576         0.9964209         0.9957051
Precision             0.8529412           0.9243697         0.9603175         0.9520000
KS                    0.9019125           0.7682055         0.9221874         0.9266184
Auc                   0.9702456           0.8969321         0.9958034         0.9955236
Gini                  0.9100370           0.7688803         0.9052916         0.9088310
Concordance           0.8052063           0.7696573         0.8490629         0.8344289
```

```
#Random Forest model is a good predictor model for employee PersonalLoan prediction.
# We will use rf_model2 as our final model.
```

Appendix A – Source Code

```
#==============================================================================
#
# Project 3
#
#==============================================================================
#calling all libraries that we are going to use
library(readxl) # read xlsx
library(DataExplorer) # visual exploration of data
library(ggplot2)# data visualisation
library(gridExtra) # arrange multiple grid-based plots on a page
library(factoextra) # extract and visualize the results of multivariate data
analysis
library(NbClust) # to find optimal number of clusters
library(caTools)
library(rpart)
library(rpart.plot)
library(tidyverse)
library(cluster)
library(randomForest)
library(ROCR)
library(ineq)
library(InformationValue)
library(rattle)
require(caTools)

#setting up working directory
setwd("C:/Users/ahmasiri/Desktop/PGP DSBA/Data/Project 3 - Thera Bank Case
Study")
#reading data from csv file to Thera_Bank_Personal_Loan_Modelling_Dataset
variable and view it
Thera_Bank_Personal_Loan_Modelling_Dataset <- read_excel("Thera
Bank_Personal_Loan_Modelling-dataset.xlsx", sheet = 2)
#fexing negative value
Thera_Bank_Personal_Loan_Modelling_Dataset$`Experience (in years)`[
Thera_Bank_Personal_Loan_Modelling_Dataset$`Experience (in years)`<0]
a = abs(Thera_Bank_Personal_Loan_Modelling_Dataset$`Experience (in years)`)
Thera_Bank_Personal_Loan_Modelling_Dataset$`Experience (in years)` = a
# fix spaces in column names
spaceless <- function(x) {colnames(x) <- gsub(" ", "", colnames(x));x}
Thera_Bank_Personal_Loan_Modelling_Dataset <-
spaceless(Thera_Bank_Personal_Loan_Modelling_Dataset)
# renaming some columen
names(Thera_Bank_Personal_Loan_Modelling_Dataset)[2] <- "Age"
names(Thera_Bank_Personal_Loan_Modelling_Dataset)[3] <- "Experiance"
names(Thera_Bank_Personal_Loan_Modelling_Dataset)[4] <- "Income"
attach(Thera_Bank_Personal_Loan_Modelling_Dataset)
View(Thera_Bank_Personal_Loan_Modelling_Dataset)
#check if ther is any NA value in dataset
anyNA(Thera_Bank_Personal_Loan_Modelling_Dataset)
anyNA(ID)
anyNA(Age)
anyNA(Experiance)
anyNA(Income)
anyNA(ZIPCode)
```

```r
anyNA(Familymembers)
anyNA(CCAvg)
anyNA(Education)
anyNA(Mortgage)
anyNA(PersonalLoan)
anyNA(SecuritiesAccount)
anyNA(CDAccount)
anyNA(Online)
anyNA(CreditCard)
#dealing with NA values
Thera_Bank_Personal_Loan_Modelling_Dataset[is.na(Thera_Bank_Personal_Loan_Mod
elling_Dataset)] <- 0
#Converting variable tipes from int to logic and factor
Thera_Bank_Personal_Loan_Modelling_Dataset$ZIPCode =
as.factor(Thera_Bank_Personal_Loan_Modelling_Dataset$ZIPCode)
Thera_Bank_Personal_Loan_Modelling_Dataset$Education =
as.factor(Thera_Bank_Personal_Loan_Modelling_Dataset$Education)
Thera_Bank_Personal_Loan_Modelling_Dataset$PersonalLoan =
as.factor(Thera_Bank_Personal_Loan_Modelling_Dataset$PersonalLoan)
Thera_Bank_Personal_Loan_Modelling_Dataset$SecuritiesAccount =
as.factor(Thera_Bank_Personal_Loan_Modelling_Dataset$SecuritiesAccount)
Thera_Bank_Personal_Loan_Modelling_Dataset$CDAccount =
as.factor(Thera_Bank_Personal_Loan_Modelling_Dataset$CDAccount)
Thera_Bank_Personal_Loan_Modelling_Dataset$Online =
as.factor(Thera_Bank_Personal_Loan_Modelling_Dataset$Online)
Thera_Bank_Personal_Loan_Modelling_Dataset$CreditCard =
as.factor(Thera_Bank_Personal_Loan_Modelling_Dataset$CreditCard)
#attaching varioable names
attach(Thera_Bank_Personal_Loan_Modelling_Dataset)
#Return a summary of the dataset variables.
summary(Thera_Bank_Personal_Loan_Modelling_Dataset)
#Retrieve the dimension of an object.
dim(Thera_Bank_Personal_Loan_Modelling_Dataset)
#Get the names of an object.
names(Thera_Bank_Personal_Loan_Modelling_Dataset)
#Display the internal structure of an dataset.
str(Thera_Bank_Personal_Loan_Modelling_Dataset)
#Returns the first 10 rows of the dataset.
head(Thera_Bank_Personal_Loan_Modelling_Dataset, 10)
#Returns the last 10 rows of the dataset.
tail(Thera_Bank_Personal_Loan_Modelling_Dataset, 10)
# univariate Analysis
#graph for all variables
# Quantitative
par(mfrow=c(2,3))
boxplot(Age, main = "Age")
boxplot(Experiance, main = "Experiance")
boxplot(Income, main = "Income")
hist(Age, main = "Age")
hist(Experiance, main = "Experiance")
hist(Income, main = "Income")
par(mfrow=c(2,3))
boxplot(Familymembers,main = "Familymembers")
boxplot(CCAvg, main = "CCAvg")
boxplot(Mortgage,main = "Mortgage")
hist(Familymembers, main = "Familymembers")
```

```
hist(CCAvg, main = "CCAvg")
hist(Mortgage, main = "Mortgage")
# catagorical
par(mfrow=c(3,2))
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset) + geom_bar(aes(x =
ZIPCode))
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset) + geom_bar(aes(x =
Education))
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset) + geom_bar(aes(x =
PersonalLoan))
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset) + geom_bar(aes(x =
SecuritiesAccount))
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset) + geom_bar(aes(x =
CDAccount))
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset) + geom_bar(aes(x =
Online))
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset) + geom_bar(aes(x =
CreditCard))
#Bi-Variate Analysis
#jitter and box plots
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset,
       aes(x = factor(Education, #defining x axis a categorical
                      labels = c("1", "2", "3")),
           y = Income,
           color = Education)) + #specifying that coloring is to be based on
drive type
  geom_boxplot(size=1, #makes the lines thicker
               outlier.shape = 1, #specifies circles for outliers
               outlier.color = "black", #makes outliers black
               outlier.size = 3) + #increases the size of the outlier symbol
  geom_jitter(alpha = 0.5, #setting transparency of graph
              width=.2) + #decreases the amount of jitter (.4 is the default)
  labs(title = "Costomer Income by Education",
       x = "",
       y = "Income (k)") +
  theme_minimal() + #setting minimal theme (no background color)
  theme(legend.position = "none") + #hiding legend
  coord_flip() #x and y axes are reversed
#scatter plot
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset,aes(x = Income, y = Age)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "Income(K)", #specifying the labels of axes and title of plot
       y = "Age",
       title = "Income(K) vs Age",
       subtitle = "Relation between Customer Income and Age") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables
#scatter plot
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset,aes(x = Income, y =
Experiance)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
```

```
              alpha=.8) +
  labs(x = "Income(K)", #specifying the labels of axes and title of plot
       y = "Experiance",
       title = "Income(K) vs Age",
       subtitle = "Relation between Customer Income and Experiance") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables
#scatter plot
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset,aes(x = Income, y = CCAvg))
+
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "Income(K)", #specifying the labels of axes and title of plot
       y = "CCAvg",
       title = "Income(K) vs CCAvg",
       subtitle = "Relation between Customer Income and CCAvg") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables
#scatter plot
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset,aes(x = Income, y =
Mortgage)) +
  geom_point(color="cornflowerblue", #setting the colour, size and
transparency(alpha) of the points
             size = 2,
             alpha=.8) +
  labs(x = "Income(K)", #specifying the labels of axes and title of plot
       y = "Mortgage",
       title = "Income(K) vs Mortgage",
       subtitle = "Relation between Customer Income and Mortgage") +
  geom_smooth(method = "lm") # this adds a linear trend line which is useful
to summarize the relationship between the two variables
# stacked bar chart
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset,
       aes(x = CreditCard,
           fill = factor(PersonalLoan,
                         levels = c("0", "1"),
                         labels = c("0", "1")))) +
  labs(fill = "PersonalLoan", # setting title of legend
       x = "CreditCard",
       title = "CreditCard by PersonalLoan") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked
# stacked bar chart
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset,
       aes(x = SecuritiesAccount,
           fill = factor(PersonalLoan,
                         levels = c("0", "1"),
                         labels = c("0", "1")))) +
  labs(fill = "PersonalLoan", # setting title of legend
       x = "SecuritiesAccount",
       title = "SecuritiesAccount by PersonalLoan") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked
# stacked bar chart
ggplot(Thera_Bank_Personal_Loan_Modelling_Dataset,
       aes(x = Online,
```

```
           fill = factor(PersonalLoan,
                         levels = c("0", "1"),
                         labels = c("0", "1")))) +
  labs(fill = "PersonalLoan", # setting title of legend
       x = "online",
       title = "Online by PersonalLoan") +
  geom_bar(position = "stack") #specifying the type of bar chart as stacked
#========================================================================
#2.1 Apply Clustering algorithm < type, rationale>
Thera_Bank_Personal_Loan_Modelling_Dataset.scaled = scale(
  Thera_Bank_Personal_Loan_Modelling_Dataset %>% select(2, 3, 4, 6, 7, 8, 8))
#Calculate Euclidean Distance between data points
eucDistMatrix <- dist(x=Thera_Bank_Personal_Loan_Modelling_Dataset.scaled,
method = "euclidean")
print(eucDistMatrix, digits = 3)
# Create dissimilarity matric using hclust() and agglomeration method =
Ward's Method
h_cluster <- hclust(eucDistMatrix, method = "ward.D2" )
# Plot the dendrogram
plot(h_cluster, hang = -1)
# ** Dendrogram indicates 2/3 clusters of colleges in our data
# Plot rectangles for possible clusters
rect.hclust(h_cluster, k = 2, border = 2:5)
rect.hclust(h_cluster, k = 3, border = 2:5)
rect.hclust(h_cluster, k = 4, border = 2:5)
# Find optimal number of clusters by creating different dendrograms
# by varying agglomeration method
h_cluster_euc_comp <- hclust(eucDistMatrix, method = 'complete')
plot(h_cluster_euc_comp,
     hang = -1, col = 'green')
h_cluster_euc_avg <- hclust(eucDistMatrix, method = 'average')
plot(h_cluster_euc_avg,
     hang = -1, col = 'red')
manhDistMatrix <- dist(x=Thera_Bank_Personal_Loan_Modelling_Dataset.scaled,
method = "manhattan")
h_cluster_manh_comp <- hclust(manhDistMatrix, method = 'complete')
plot(h_cluster_manh_comp,
     hang = -1, col = 'blue')
# ** All the dendograms indicate a presence of 3 major clusters
# Add cluster membership to original dataset
cluster_name <- cutree(h_cluster, k = 3)
clg_data_hclusters <-
cbind(Thera_Bank_Personal_Loan_Modelling_Dataset.scaled,cluster_name)
# Visualise the clusters in two dimensions
h_clust_viz_3 <- fviz_cluster(list(data =
Thera_Bank_Personal_Loan_Modelling_Dataset.scaled,
                                cluster = clg_data_hclusters[, 6])) +
  ggtitle("hierarchical 3")
h_clust_viz_3
# Observe the differences between identified clusters
aggr_mean <- aggregate(Thera_Bank_Personal_Loan_Modelling_Dataset.scaled,
list(cluster_name), mean)
# Create cluster profiles
hcluster.profile <- data.frame(Cluster = aggr_mean[, 1],
                               Number_of_Colleges =
                                 as.vector(table(cluster_name)),
```

```
                                aggr_mean[, -1])
View(hcluster.profile)
hcluster.profile
#=========================kmeans=========================
Thera_Bank_Personal_Loan_Modelling_Dataset.scaled = scale(
  Thera_Bank_Personal_Loan_Modelling_Dataset %>% select(2, 3, 4, 6, 7, 8, 8))
seed = 1000
#finding best k
set.seed(seed)
nc = NbClust(Thera_Bank_Personal_Loan_Modelling_Dataset %>% select(2, 3, 4,
6, 7, 8)
             , min.nc = 2, max.nc = 5, method = "kmeans")
#it shows that the best value of k is 3
clus = kmeans(x=Thera_Bank_Personal_Loan_Modelling_Dataset.scaled, centers =
3, nstart = 5)
clusplot(Thera_Bank_Personal_Loan_Modelling_Dataset.scaled,clus$cluster,
color = T, shode = T, label = 1, lines = 1)
#3.1 Applying CART <plot the tree>
#===========================CART=====================================
set.seed(1000) # To ensure reproducibility
split <- sample.split(Thera_Bank_Personal_Loan_Modelling_Dataset[-1],
SplitRatio = 0.7)
train <- subset(Thera_Bank_Personal_Loan_Modelling_Dataset, split == TRUE)
test <- subset( Thera_Bank_Personal_Loan_Modelling_Dataset, split == FALSE)
nrow(train)
nrow(test)
# Check that the distribution of the dependent variable is similar in train
and test sets
prop.table(table(Thera_Bank_Personal_Loan_Modelling_Dataset$PersonalLoan))
prop.table(table(train$PersonalLoan))
prop.table(table(test$PersonalLoan))
## Build a CART model on the train dataset
# We will use the "rpart" and the "rattle" libraries to build decision trees.
# 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
cart_model1 <- rpart(formula = PersonalLoan~., data = train, method =
"class", control = r.ctrl)
cart_model1
#Clearly we need to prune this tree
## Model Tuning
#The cost complexity table can be obtained using the printcp or plotcp
functions
printcp(cart_model1)
plotcp(cart_model1)
  # The unncessarily complex tree above can be pruned using a cost complexity
threshold.
# Using a complexity threshold of 0.07 gives us a relatively simpler tree.
cart_model2 = prune(cart_model1, cp= 0.07 ,"CP")
printcp(cart_model2)
cart_model2
```

```
#Displaying the decision tree
prp(cart_model2)
#Let us check the variable importance
cart_model1$variable.importance
# Predicting on the train dataset
train_predict.class_CART <- predict(cart_model2, train, type="class") #
Predicted Classes
train_predict.score_CART <- predict(cart_model2, train) # Predicted
Probabilities
# Create confusion matrix for train data predictions
tab.train_CART = table(train$PersonalLoan, train_predict.class_CART)
tab.train_CART
# Accuracy on train data
accuracy.train_CART = sum(diag(tab.train_CART)) / sum(tab.train_CART)
accuracy.train_CART
# CART Model (cart_model2) has 98% accuracy on train data.
## Model Evaluation
# Predicting on the test dataset
test_predict.class_CART <- predict(cart_model2, test, type="class") #
Predicted Classes
test_predict.score_CART <- predict(cart_model2, test) # Predicted
Probabilities
# Create confusion matrix for test data predictions
tab.test_CART = table(test$PersonalLoan, test_predict.class_CART)
tab.test_CART
# Accuracy on test data
accuracy.test_CART = sum(diag(tab.test_CART)) / sum(tab.test_CART)
accuracy.test_CART
#The CART model accuracy on test data is 97%
fancyRpartPlot(cart_model2)
#3.1 Applying Random forest
#============================Random forest================================
set.seed(1000) # To ensure reproducibility
rf_model1 = randomForest(
  PersonalLoan ~ .,
  data = select(train, -1,-5),
  ntree = 51,
  mtry = 6,
  nodesize = 10,
  importance = TRUE
)
#Plot the model to determine the optimum number of trees
plot(rf_model1)
#The plot reveals that anything more than, say 50 trees, is really not that
valuable.
# List the importance of the variables.
# Larger the MeanDecrease values, the more important the variable.
# Look at the help files to get a better sense of how these are computed.
print(rf_model1$importance)
# Let us tune the randomforest model by trying diferent 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
## Tune the Random Forest Model
# Check the column number of the response variable
names(train)
```

```
set.seed(500) # To ensure reproducibility
rf_model2 = tuneRF(x = select(train, -1,-5,-10), # matrix or data frame of 
predictor/independent variables
                   y = train$PersonalLoan, # response vector (factor for 
classification, numeric for regression)
                   mtrystart = 5, # starting value of mtry
                   stepfactor=1.5, # at each iteration, mtry is inflated (or 
deflated) by this value
                   ntree=51, # number of trees built for each mtry value
                   improve=0.0001, # the (relative) improvement in OOB error 
must be by this much for the search to continue
                   nodesize=10, # Minimum size of terminal nodes
                   trace=TRUE, # prints the progress of the search
                   plot=TRUE, # to get the plot of the OOB error as function 
of mtr
                   doBest=TRUE, # return a forest using the optimal mtry 
found
                   importance=TRUE #
)
# The optimal number of mtry is 6.
# tuneRF returns rf_model2. It is the random forest of 51 trees built with m 
= 6
## Model Validation
# Predicting on the train dataset
train_predict.class_RF <- predict(rf_model2, train, type="class") # Predicted 
Classes
train_predict.score_RF <- predict(rf_model2, train) # Predicted Probabilities
# Create confusion matrix for train data predictions
tab.train_RF = table(train$PersonalLoan, 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
# RandomForest mode (rf_model2) has 99% accuracy on train data.
## 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) # Predicted Probabilities
# Create confusion matrix for test data predictions
tab.test_RF = table(test$PersonalLoan, 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
# The model has good performance on test data too.
# An accuracy of 99% on train data and 97% on test data indicates that this 
is a good model,
# neither overfit nor underfit.
## Comparing Models
Model_Name = c("CART", "Random Forest")
Train_Accuracy_perc = c( accuracy.train_CART*100, accuracy.train_RF*100)
Test_Accuracy_perc = c( accuracy.test_CART*100, accuracy.test_RF*100)
output = data.frame(Model_Name,Train_Accuracy_perc,Test_Accuracy_perc)
output
```

```
#Random Forest model is a good predictor model for employee PersonalLoan
prediction.
# We will use rf_model2 as our final model.
varImpPlot(rf_model2, sort = TRUE)
# Confusion Matrix
#=======================================================================
# We will compare all the 4 models that we created earlier - rf_model1,
rf_model2, cart_model1, cart_model2
# Predict PersonalLoan class and probability for all 4 models
# Predict on test data using cart_model1
cart_model1_predict_class = predict(cart_model1, test, type = 'class')
cart_model1_predict_score = predict(cart_model1, test, type = 'prob')
# Predict on test data using cart_model2
cart_model2_predict_class = predict(cart_model2, test, type = 'class')
cart_model2_predict_score = predict(cart_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_cart_model1 = table(test$PersonalLoan, cart_model1_predict_class)
conf_mat_cart_model1
conf_mat_cart_model2 = table(test$PersonalLoan, cart_model2_predict_class)
conf_mat_cart_model2
conf_mat_rf_model1 = table(test$PersonalLoan, rf_model1_predict_class)
conf_mat_rf_model1
conf_mat_rf_model2 = table(test$PersonalLoan, rf_model2_predict_class)
conf_mat_rf_model2
# Accuracy
# Accuracy of models on test data
accuracy_cart_model1 = sum(diag(conf_mat_cart_model1)) /
sum(conf_mat_cart_model1)
accuracy_cart_model2 = sum(diag(conf_mat_cart_model2)) /
sum(conf_mat_cart_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 / Recall
# Sensitivity of models on test data
sensitivity_cart_model1 = conf_mat_cart_model1[2,2] /
sum(conf_mat_cart_model1['1',])
sensitivity_cart_model2 = conf_mat_cart_model2[2,2] /
sum(conf_mat_cart_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
# Specificity of models on test data
specificity_cart_model1 = conf_mat_cart_model1[1,1] /
sum(conf_mat_cart_model1['0',])
specificity_cart_model2 = conf_mat_cart_model2[1,1] /
sum(conf_mat_cart_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
# Precision of models on test data
precision_cart_model1 = conf_mat_cart_model1[2,2] /
sum(conf_mat_cart_model1[,'1'])
precision_cart_model2 = conf_mat_cart_model2[2,2] /
sum(conf_mat_cart_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'])
# KS
# Using library ROCR functions prediction and performance
pred_cart_model1 = prediction(cart_model1_predict_score[, 2],
test$PersonalLoan)
perf_cart_model1 = performance(pred_cart_model1,"tpr","fpr")
ks_cart_model1 = max(attr(perf_cart_model1,'y.values')[[1]]
        - attr(perf_cart_model1,'x.values')[[1]])
pred_cart_model2 = prediction(cart_model2_predict_score[, 2],
test$PersonalLoan)
perf_cart_model2 = performance(pred_cart_model2,"tpr","fpr")
ks_cart_model2 = max(attr(perf_cart_model2,'y.values')[[1]]
        - attr(perf_cart_model2,'x.values')[[1]])
pred_rf_model1 = prediction(rf_model1_predict_score[, 2], test$PersonalLoan)
perf_rf_model1 = performance(pred_rf_model1,"tpr","fpr")
ks_rf_model1 = max(attr(perf_rf_model1,'y.values')[[1]]
        - attr(perf_rf_model1,'x.values')[[1]])
pred_rf_model2 = prediction(rf_model2_predict_score[, 2], test$PersonalLoan)
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]])
# AUC
# Using library ROCR
auc_cart_model1 = performance(pred_cart_model1, measure = "auc")
auc_cart_model1 = auc_cart_model1@y.values[[1]]
auc_cart_model2 = performance(pred_cart_model2, measure = "auc")
auc_cart_model2 = auc_cart_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]]
# Gini
# Using library ineq
gini_cart_model1 = ineq(cart_model1_predict_score[, 2],"gini")
gini_cart_model2 = ineq(cart_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 - Discordance
concordance_cart_model1 = Concordance(actuals = ifelse(test$PersonalLoan ==
'1', 1,0),
        predictedScores = ifelse(cart_model1_predict_class == '1', 1,0))
concordance_cart_model2 = Concordance(actuals = ifelse(test$PersonalLoan ==
'1', 1,0),
         predictedScores = ifelse(cart_model2_predict_class == '1', 1,0))
concordance_rf_model1 = Concordance(actuals = ifelse(test$PersonalLoan ==
'1', 1,0),
         predictedScores = ifelse(rf_model1_predict_class == '1', 1,0))
```

```
concordance_rf_model2 = Concordance(actuals = ifelse(test$PersonalLoan ==
'1', 1,0),
         predictedScores = ifelse(rf_model2_predict_class == '1', 1,0))
# Comparing models
cart_model1_metrics = c(accuracy_cart_model1, sensitivity_cart_model1,
specificity_cart_model1,
                      precision_cart_model1, ks_cart_model1, auc_cart_model1,
gini_cart_model1,
                      concordance_cart_model1$Concordance)
cart_model2_metrics = c(accuracy_cart_model2, sensitivity_cart_model2,
specificity_cart_model2,
                      precision_cart_model2, ks_cart_model2, auc_cart_model2,
gini_cart_model2,
                      concordance_cart_model2$Concordance)
rf_model1_metrics = c(accuracy_rf_model1, sensitivity_rf_model1,
specificity_rf_model1,
                    precision_rf_model1, ks_rf_model1, auc_rf_model1,
gini_rf_model1,
                    concordance_rf_model1$Concordance)
rf_model2_metrics = c(accuracy_rf_model2, sensitivity_rf_model2,
specificity_rf_model2,
                    precision_rf_model2, ks_rf_model2, auc_rf_model2,
gini_rf_model2,
                    concordance_rf_model2$Concordance)
comparison_table = data.frame(cart_model1_metrics, cart_model2_metrics,
         rf_model1_metrics, rf_model2_metrics)
rownames(comparison_table) = c("Accuracy", "Sensitivity", "Specificity",
         "Precision", "KS", "Auc", "Gini", "Concordance")
comparison_table
#========================================================================
#
# T H E - E N D
#
#========================================================================
```