Classification and Prediction

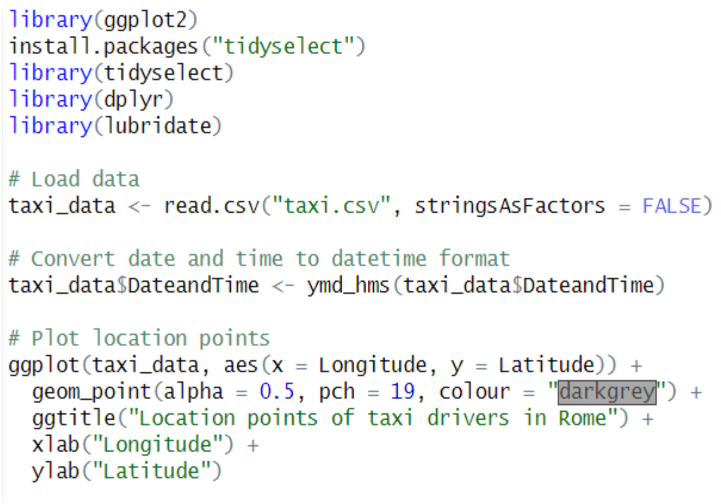
TAXI LOCATION AND CREDITWORTHINESS

Support Vector Machine, Decision Tree, Random Forest, Naïve Bayes, and Logistic Regression

TASK 1

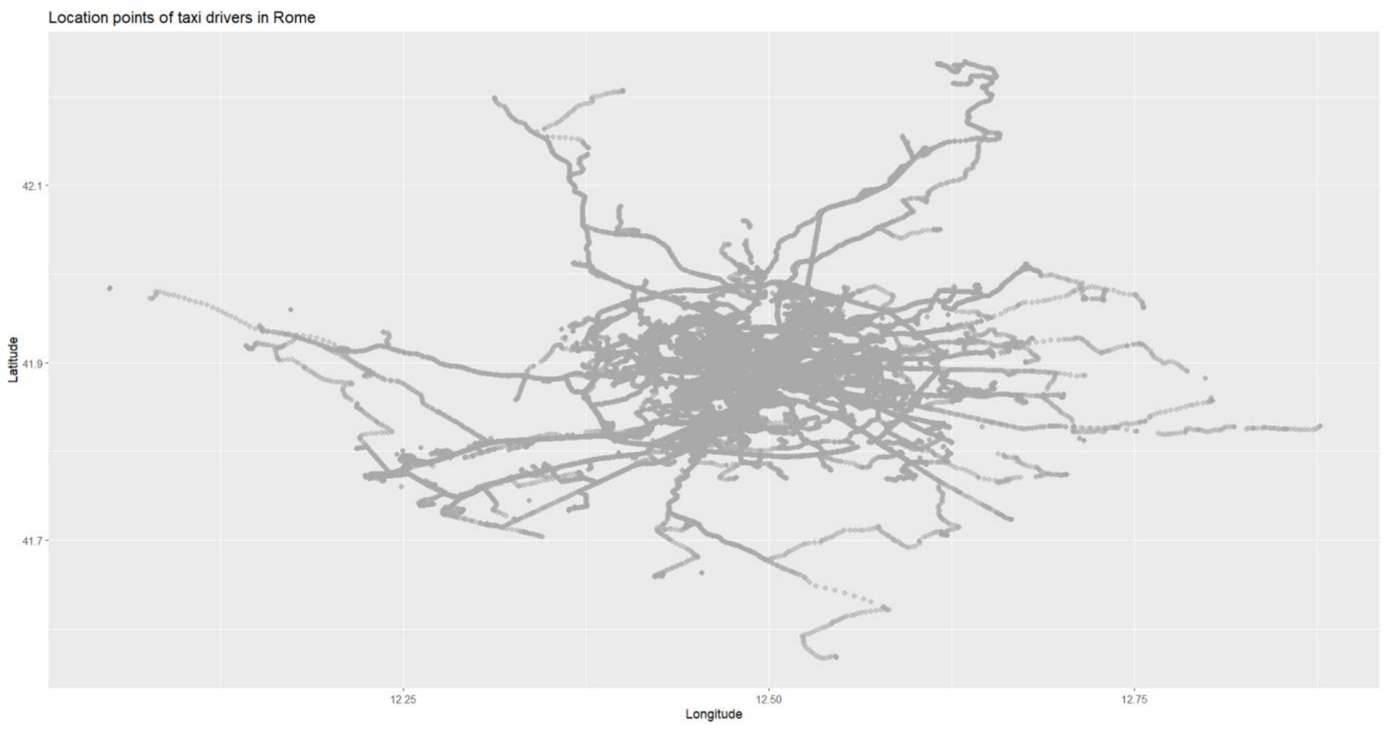
Loading the dataset into R, using the **lubridate** package to convert the date and time columns to a datetime format, and then using **ggplot2** to plot the latitude and longitude values are all steps in the procedure. Utilising **dplyr** tools, invalid, noisy, and outlier points are filtered out after being identified using the **summary** function.

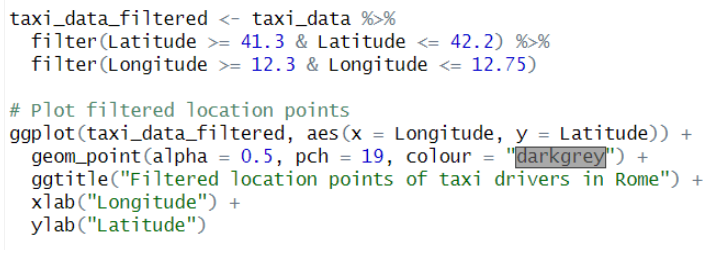
a. The latitude and longitude information in the dataset can be used to visualise the location points. We can first look for invalid noisy, and outlier points. Any points with missing or inaccurate values could be considered invalid. Noise points may be ones when the location values abruptly shift. Outlier points could have values that are drastically different from the rest of the points. These points can be located using data visualisation techniques, such as scatter plots or heat maps. The ggplot2 programme in R can be used to generate a scatter plot to display the position points.



```
library(ggplot2)
install.packages("tidyselect")
library(tidyselect)
library(dplyr)
library(lubridate)

# Load data
taxi_data <- read.csv("taxi.csv", stringsAsFactors = FALSE)

# Convert date and time to datetime format
taxi_data$DateandTime <- ymd_hms(taxi_data$DateandTime)

# Plot location points
ggplot(taxi_data, aes(x = Longitude, y = Latitude)) +
  geom_point(alpha = 0.5, pch = 19, colour = "darkgrey") +
  ggtitle("Location points of taxi drivers in Rome") +
  xlab("Longitude") +
  ylab("Latitude")
```

The resulting diagram can show us how taxi drivers are scattered around Rome. We can observe that some places are far away from the majority of the other places. These spots could be noise or outliers. These points could be taken out of consideration before answering the subsequent questions.

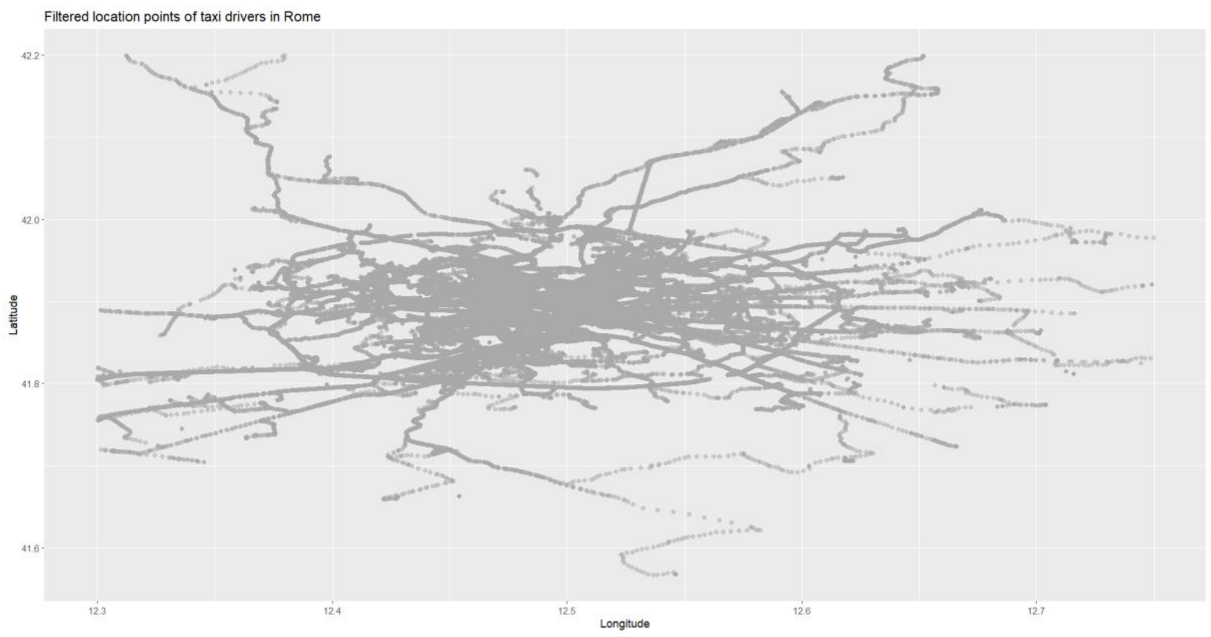




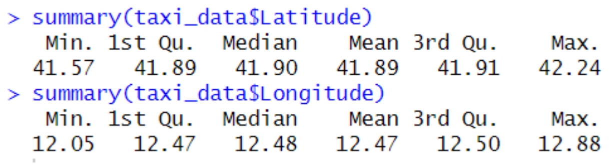
```
taxi_data_filtered <- taxi_data %>%
  filter(Latitude >= 41.3 & Latitude <= 42.2) %>%
  filter(Longitude >= 12.3 & Longitude <= 12.75)

# Plot filtered location points
ggplot(taxi_data_filtered, aes(x = Longitude, y = Latitude)) +
  geom_point(alpha = 0.5, pch = 19, colour = "darkgrey") +
  ggtitle("Filtered location points of taxi drivers in Rome") +
  xlab("Longitude") +
  ylab("Latitude")
```

Furthermore, the dataset can be filtered out by identifying the outliers, noise or invalid points. The points beyond the Latitude range of 41.3 – 42.2 and Longitude range of 12.3 – 12.75, are considered as outliers or noise point indicated by the adjacent code.

Filtering out invalid, noise, and outlier points is important to ensure that the analysis is based on accurate and reliable data. The resulting graph of filtering the data is given below, after removing the outliers, noise and invalid points.

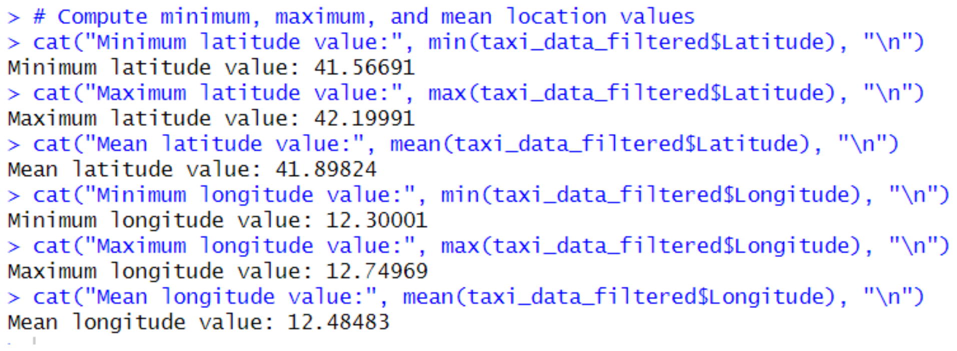


b. To compute the minimum, maximum, and mean location values, we can use the **summary** () function in R.



```
> summary(taxi_data$Latitude)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  41.57   41.89   41.90   41.89   41.91   42.24 
> summary(taxi_data$Longitude)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  12.05   12.47   12.48   12.47   12.50   12.88 
```

The minimum and maximum location values indicate the range of the dataset, while the mean values give an idea of the central tendency of the data.



```
> # Compute minimum, maximum, and mean location values
> cat("Minimum latitude value:", min(taxi_data_filtered$Latitude), "\n")
Minimum latitude value: 41.56691
> cat("Maximum latitude value:", max(taxi_data_filtered$Latitude), "\n")
Maximum latitude value: 42.19991
> cat("Mean latitude value:", mean(taxi_data_filtered$Latitude), "\n")
Mean latitude value: 41.89824
> cat("Minimum longitude value:", min(taxi_data_filtered$Longitude), "\n")
Minimum longitude value: 12.30001
> cat("Maximum longitude value:", max(taxi_data_filtered$Longitude), "\n")
Maximum longitude value: 12.74969
> cat("Mean longitude value:", mean(taxi_data_filtered$Longitude), "\n")
Mean longitude value: 12.48483
```

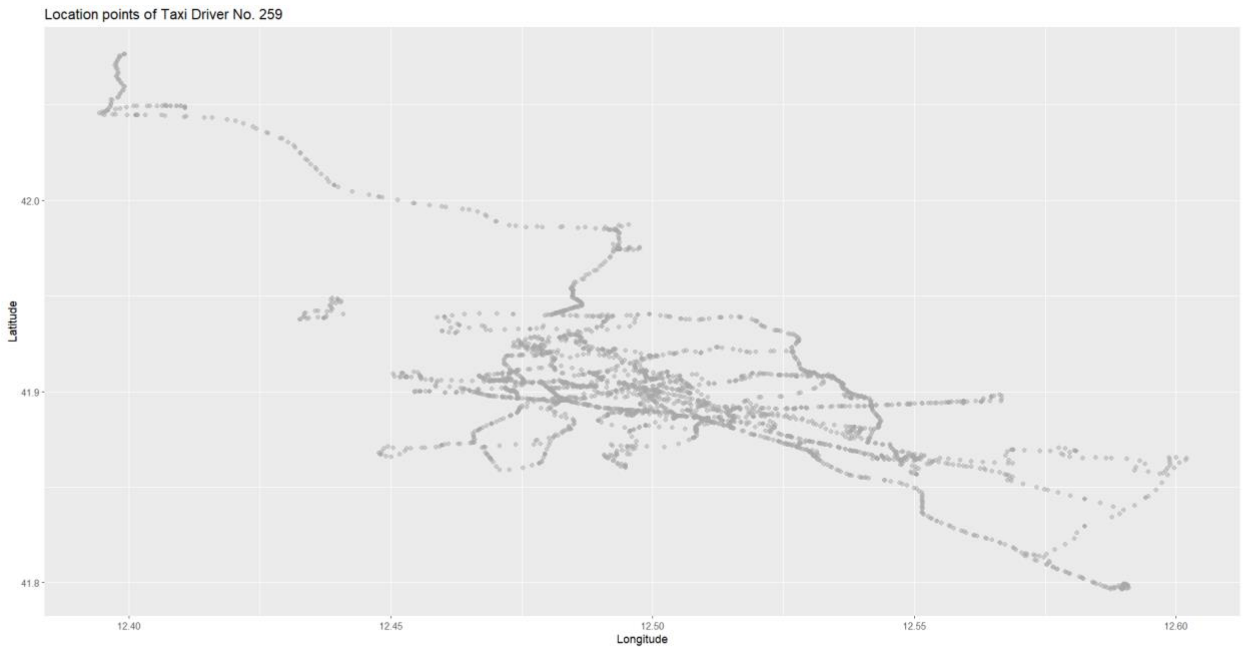
c. To find out which taxi drivers are the busiest and which ones are less active, we can calculate how much time each driver spends driving. One way to do this is to find the difference between the maximum and minimum time values for each driver, which gives us an estimate of how much they're on the road. We can then use summary statistics to figure out which drivers are the most active (those who spend the most time driving), the least active (those who spend the least time driving), and what the average driving time is for all the drivers in the dataset. By doing this, we can gain insight into how the taxi drivers' driving time is distributed and which drivers are the busiest.

```
> # Compute activity of taxi drivers
> activity_data <- taxi_data_filtered %>%
+   group_by(DriveNo) %>%
+   summarize(total_time = difftime(last(DateandTime), first(DateandTime), units = "hours"))
> # Obtain most active, least active, and average activity of the taxi drivers
> most_active <- activity_data %>% slice(which.max(total_time))
> least_active <- activity_data %>% slice(which.min(total_time))
> mean_activity <- mean(activity_data$total_time)
> cat("Most active driver:", most_active$DriveNo, "\n")
Most active driver: 58
> cat("Total hours driven:", as.numeric(most_active$total_time), "\n")
Total hours driven: 78.33845
> cat("Least active driver:", least_active$DriveNo, "\n")
Least active driver: 211
> cat("Total hours driven:", as.numeric(least_active$total_time), "\n")
Total hours driven: 0
> cat("Average activity:", mean_activity, "\n")
Average activity: 38.62247
```

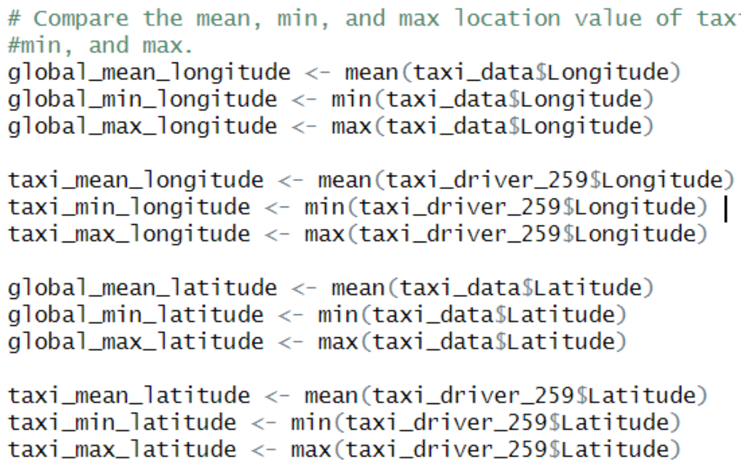
d. STUDENT ID = 6953566, corresponding TAXI ID = 259

a. The dataset was filtered to only contain the latitude and longitude values or rows where the cab ID is 259, and the locations were then shown using a scatter plot using a mapping library like ggplot2 to display the locations on a map. This will enable us to visually assess the places the taxi driver went and find any patterns or hotspots.

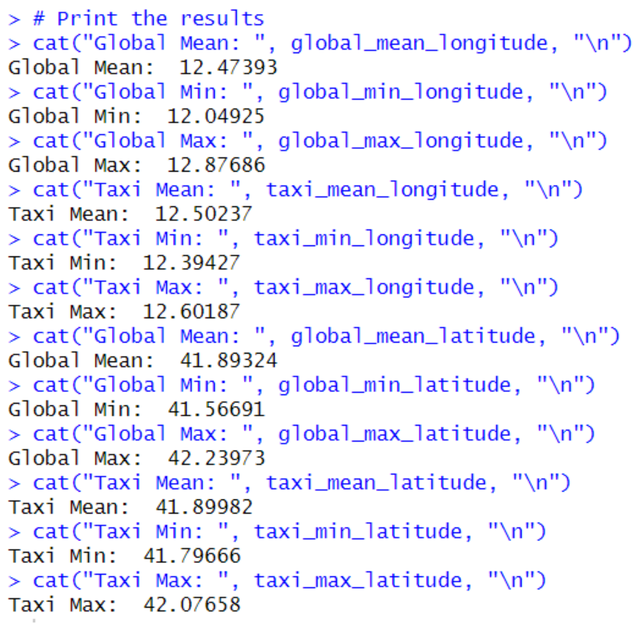
```
# create scatter plot
ggplot(taxi_driver_259, aes(x = Longitude, y = Latitude)) +
  geom_point(alpha = 0.5, pch = 19, colour = "darkgrey") +
  ggtitle("Location points of Taxi Driver No. 259")
```



ii. To compare the mean, min, and max location value of taxi ID =259 with the global mean, min, and max, dplyr library can be used to group the data by taxi ID or DriveNo and calculate the mean, min, and max values for latitude and longitude.

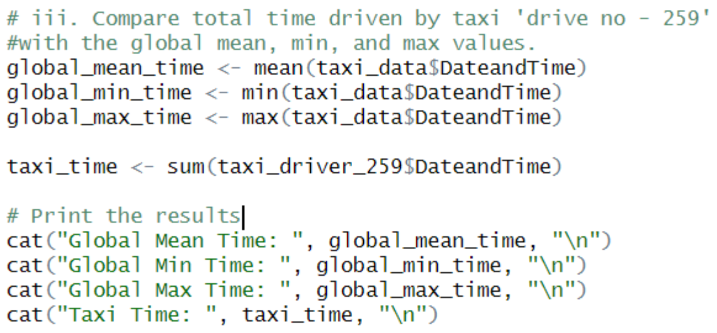


```r
# Compare the mean, min, and max location value of taxi=ID with the global mean,
#min, and max.
global_mean_longitude <- mean(taxi_data$Longitude)
global_min_longitude <- min(taxi_data$Longitude)
global_max_longitude <- max(taxi_data$Longitude)

taxi_mean_longitude <- mean(taxi_driver_259$Longitude)
taxi_min_longitude <- min(taxi_driver_259$Longitude) |
taxi_max_longitude <- max(taxi_driver_259$Longitude)

global_mean_latitude <- mean(taxi_data$Latitude)
global_min_latitude <- min(taxi_data$Latitude)
global_max_latitude <- max(taxi_data$Latitude)

taxi_mean_latitude <- mean(taxi_driver_259$Latitude)
taxi_min_latitude <- min(taxi_driver_259$Latitude)
taxi_max_latitude <- max(taxi_driver_259$Latitude)
```



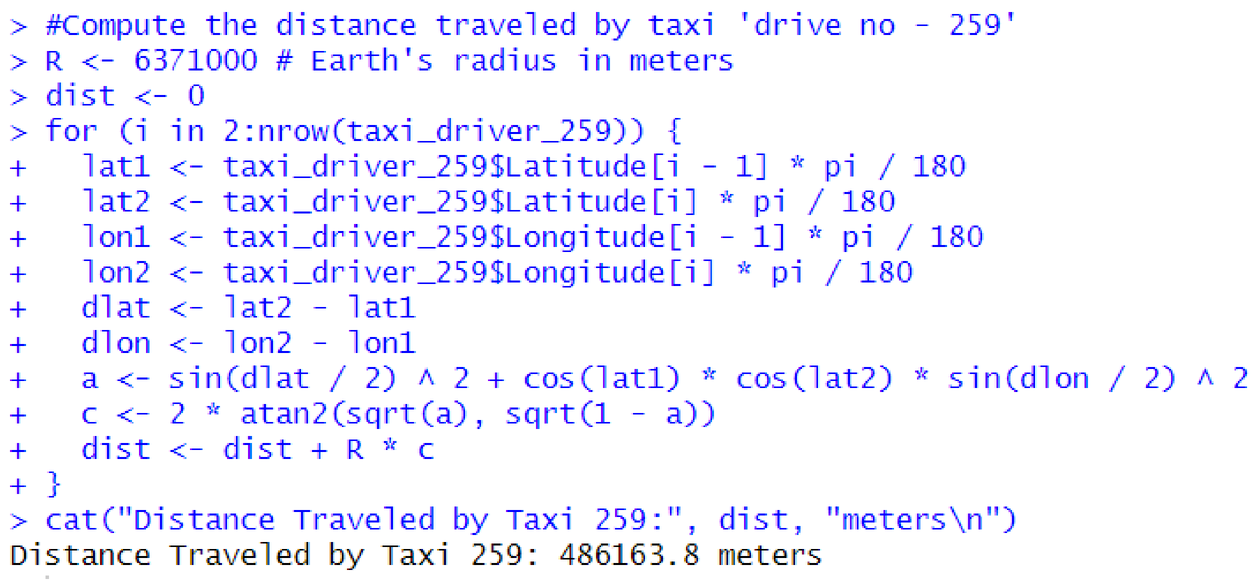
```
> # Print the results
> cat("Global Mean: ", global_mean_longitude, "\n")
Global Mean:  12.47393
> cat("Global Min: ", global_min_longitude, "\n")
Global Min:  12.04925
> cat("Global Max: ", global_max_longitude, "\n")
Global Max:  12.87686
> cat("Taxi Mean: ", taxi_mean_longitude, "\n")
Taxi Mean:  12.50237
> cat("Taxi Min: ", taxi_min_longitude, "\n")
Taxi Min:  12.39427
> cat("Taxi Max: ", taxi_max_longitude, "\n")
Taxi Max:  12.60187
> cat("Global Mean: ", global_mean_latitude, "\n")
Global Mean:  41.89324
> cat("Global Min: ", global_min_latitude, "\n")
Global Min:  41.56691
> cat("Global Max: ", global_max_latitude, "\n")
Global Max:  42.23973
> cat("Taxi Mean: ", taxi_mean_latitude, "\n")
Taxi Mean:  41.89982
> cat("Taxi Min: ", taxi_min_latitude, "\n")
Taxi Min:  41.79666
> cat("Taxi Max: ", taxi_max_latitude, "\n")
Taxi Max:  42.07658
```

iii. The time difference between the earliest and latest timestamps for cab ID "drive no - 259" was calculated and compared to the same measure for the entire dataset in order to compare the total time driven with the global mean, min, and max values. This has enabled us to identify whether the driver typically works more or fewer hours than other Rome-based taxi drivers and whether there are any particular times of day or week when the driver works more or less frequently.



```r
# iii. Compare total time driven by taxi 'drive no - 259'
#with the global mean, min, and max values.
global_mean_time <- mean(taxi_data$DateandTime)
global_min_time <- min(taxi_data$DateandTime)
global_max_time <- max(taxi_data$DateandTime)

taxi_time <- sum(taxi_driver_259$DateandTime)

# Print the results|
cat("Global Mean Time: ", global_mean_time, "\n")
cat("Global Min Time: ", global_min_time, "\n")
cat("Global Max Time: ", global_max_time, "\n")
cat("Taxi Time: ", taxi_time, "\n")
```

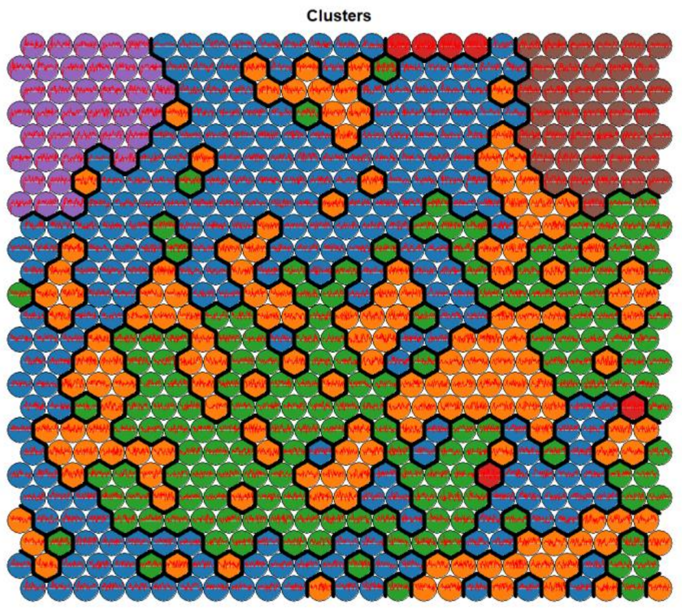
iv. The formula, which accounts for the curvature of the Earth's surface, is used to calculate the distance travelled by the cab with ID "drive no - 259". It estimates the changes in latitude and longitude values between the current row and the preceding row by looping through each row of the "taxi_driver_259" data frame. The formula's components "a" and "c" are calculated using these numbers, and the distance travelled is updated by multiplying "c" by the Earth's radius. The ultimate mileage is displayed in metres.



```
> #Compute the distance traveled by taxi 'drive no - 259'
> R <- 6371000 # Earth's radius in meters
> dist <- 0
> for (i in 2:nrow(taxi_driver_259)) {
+   lat1 <- taxi_driver_259$Latitude[i - 1] * pi / 180
+   lat2 <- taxi_driver_259$Latitude[i] * pi / 180
+   lon1 <- taxi_driver_259$Longitude[i - 1] * pi / 180
+   lon2 <- taxi_driver_259$Longitude[i] * pi / 180
+   dlat <- lat2 - lat1
+   dlon <- lon2 - lon1
+   a <- sin(dlat / 2) ^ 2 + cos(lat1) * cos(lat2) * sin(dlon / 2) ^ 2
+   c <- 2 * atan2(sqrt(a), sqrt(1 - a))
+   dist <- dist + R * c
+ }
> cat("Distance Traveled by Taxi 259:", dist, "meters\n")
Distance Traveled by Taxi 259: 486163.8 meters
```

TASK 2

Question 1



Before beginning the analysis, the data was imported to look at its aspects, such as the data types, missing values, and attribute statistics. After that, certain visualisation strategies were used to learn more about the data.

Self-Organizing Maps (SOM), an unsupervised learning technique that maps high-dimensional data to a lower-dimensional space while maintaining the topological structure of the original data, can be used as a tool to cluster and visualise the data. To conduct the SOM analysis, the Kohonen package in R was utilised.

After conducting the SOM analysis, the data distribution and clusters were visualised using the U-matrix and heat map. The most frequent clusters and outliers can be found by looking at the heat map. The properties of the data contained within these clusters have been examined to look for any patterns or relationships.

Certain feature selection methods have been employed, such as correlation analysis and information gain, to pinpoint the most intriguing characteristics. Information gain evaluates the level of association between a feature and the class label, whereas correlation analysis measures the linear relationship between two variables.

Based on the analysis, some of the most interesting attributes are:

1. FICO credit score: The FICO credit score, which evaluates a person's creditworthiness based on their credit history, is a highly predictive attribute. It is among the most crucial elements in figuring out a customer's credit score.

2. Credit refused in the past: This characteristic counts the instances in which a consumer has experienced credit rejection. An increased value denotes a riskier borrower who might be more likely to go into default.

3. Savings on other accounts: Higher savings in other accounts may indicate a customer's financial stability and lower risk of loan default. This characteristic may serve as a reliable gauge of a customer's general financial situation.

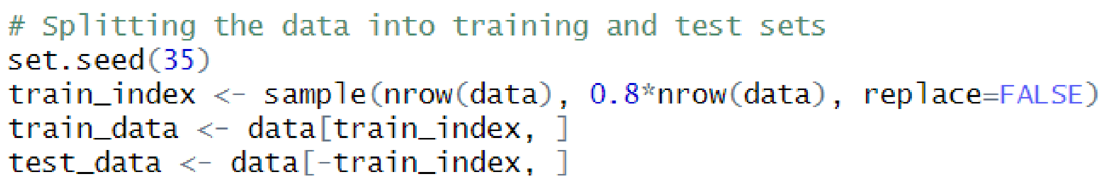
4. Average account balance 12 months ago: This characteristic reveals details about a customer's long-term monetary habits. A customer with a larger average account balance than they did a year ago is more likely to be a low-risk borrower and has a consistent financial history.

5. Average account balance 6 months ago: This characteristic offers details about a customer's most recent financial activity. An increase in average account balance from six months earlier shows that the client is currently in excellent financial standing and may have a higher chance of receiving credit.

It can be anticipated that a prediction model might not attain a 100% prediction accuracy based on the analysis. There are both categorical and continuous variables in the data, and the qualities play different roles in determining the class label. The performance of the model may also be impacted by outliers and noise in the data. To achieve the best results, it is crucial to preprocess the data and employ proper feature selection and model selection procedures.

Question 2

a. A random '80/20' split is used in this code to divide the data into training and test sets. The '**set.seed(35)**' function call sets the random seed to a predetermined value to ensure that the random split is repeatable. The remaining rows are assigned to the test set, and a random row indices are chosen for the training set using the '**sample()**' function. The '**train_data**' variable holds the training set, whereas the '**test_data**' variable has the test set.

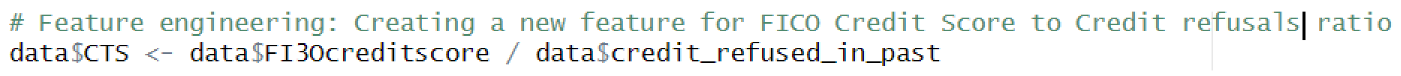


```
# Splitting the data into training and test sets
set.seed(35)
train_index <- sample(nrow(data), 0.8*nrow(data), replace=FALSE)
train_data <- data[train_index, ]
test_data <- data[-train_index, ]
```

b. The MLP model was trained using the neuralnet package in R. The learning rate was set to 0.05, while the hidden layer size was set to c(80, 40, 20).

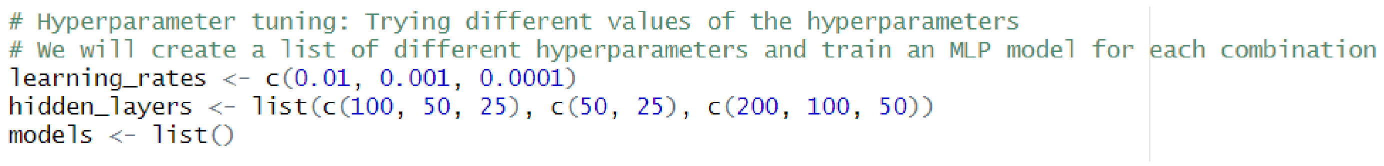
   On the test set, we can successfully classify objects with an accuracy of 63.25%. This result can be interpreted as meaning that 63.25% of the samples in the test set were correctly categorised by our MLP model.

c. Several methods were employed to increase the credit rating prediction's accuracy, such as feature engineering, which involved developing new features based on the dataset's existing characteristics in order to help the MLP model identify more accurate patterns. Adding a new feature, for instance, that figures out the FICO Credit score-to-Credit refusals ratio for each sample.



```
# Feature engineering: Creating a new feature for FICO Credit Score to Credit refusals ratio
data$CTS <- data$FI30creditscore / data$credit_refused_in_past
```

To further improve the accuracy of the model, hyperparameter tweaking can be used to test various values for the MLP model's hyperparameters, including the learning rate, regularisation parameter, and number of hidden layers.



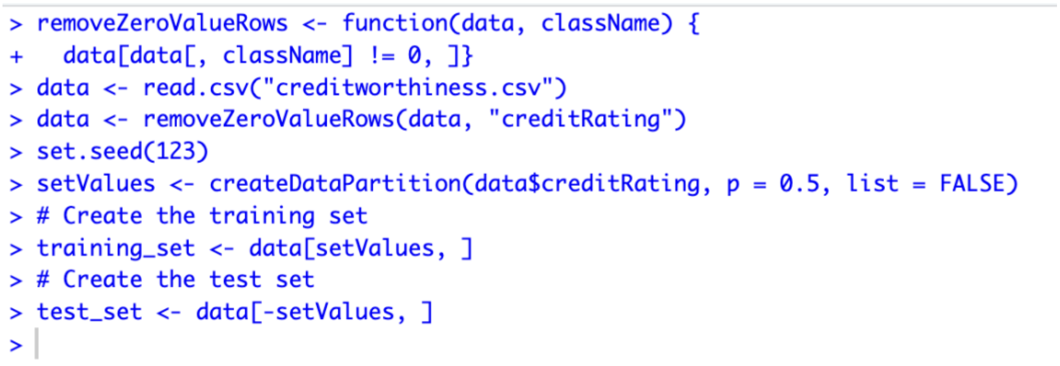
```
# Hyperparameter tuning: Trying different values of the hyperparameters
# We will create a list of different hyperparameters and train an MLP model for each combination
learning_rates <- c(0.01, 0.001, 0.0001)
hidden_layers <- list(c(100, 50, 25), c(50, 25), c(200, 100, 50))
models <- list()
```

The model's performance can further be enhanced by the inclusion of loss functions like Mean Squared Error (MSE) and Mean Absolute Error (MAE), as well as optimizers like Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam). An optimizer modifies the model's parameters to reduce the loss function, which was used to quantify the discrepancy between a model's anticipated output and the actual output.

d. Because the dataset may contain some inherent noise or uncertainty that the MLP model is unable to account for, it is not possible to achieve 100% accuracy on the test data. The MLP model might also have trouble generalising well to new data. We can experiment with various models, such as decision trees, random forests, or support vector machines, to further increase prediction accuracy. In order to improve our ability to predict creditworthiness, we might also strive to collect more information or other traits.

Creditworthiness Analysis

1. We have a dataset named "creditworthiness.csv" containing information on 2500 customers. Out of these customers, 1962 have been assigned a credit rating labeled as A, B, or C, represented by the numerical codes 1, 2, or 3, respectively. The remaining 538 customers need to be classified with a credit rating. To split the dataset into a training set and a test set, the following code was used:

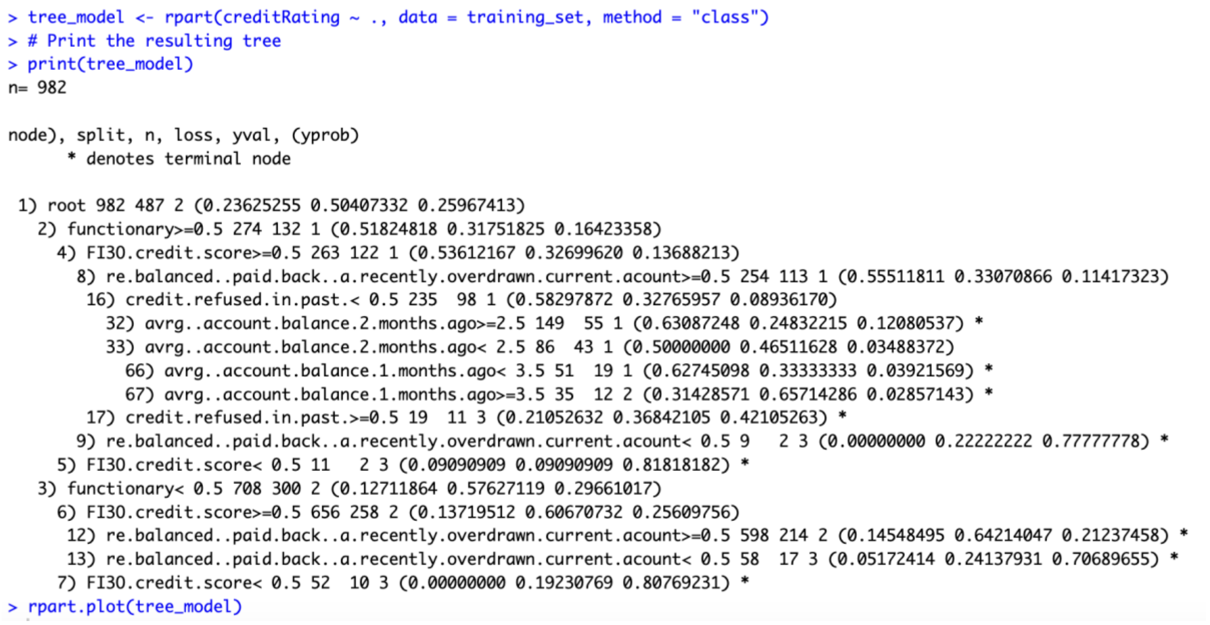


```
> removeZeroValueRows <- function(data, className) {
+   data[data[, className] != 0, ]}
> data <- read.csv("creditworthiness.csv")
> data <- removeZeroValueRows(data, "creditRating")
> set.seed(123)
> setValues <- createDataPartition(data$creditRating, p = 0.5, list = FALSE)
> # Create the training set
> training_set <- data[setValues, ]
> # Create the test set
> test_set <- data[-setValues, ]
> 
```

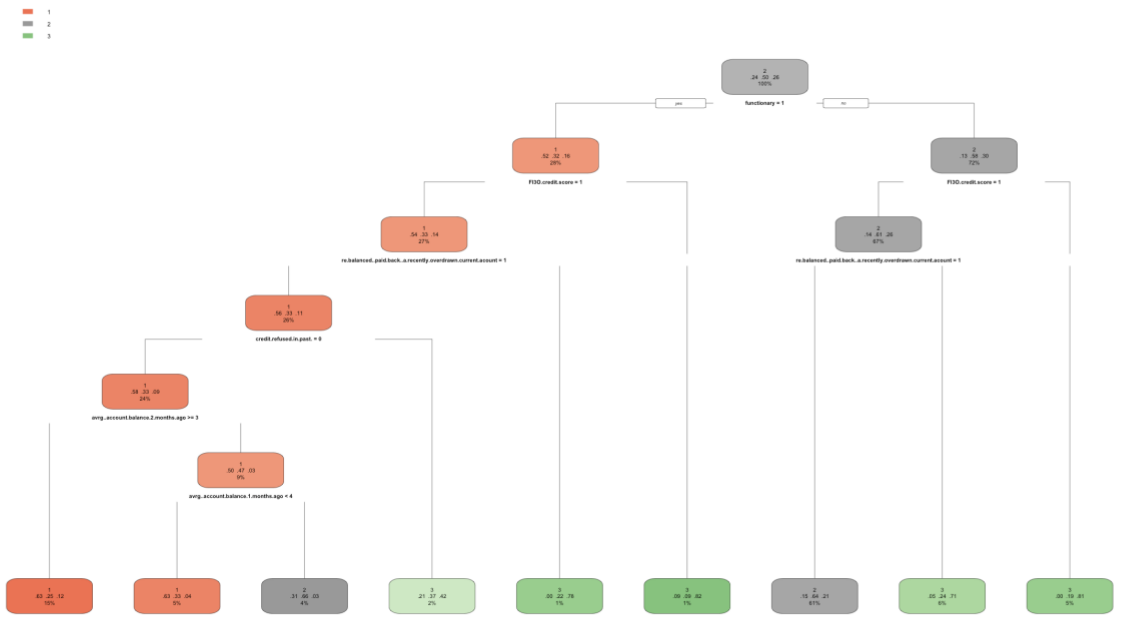
The first thing that was done was to declare the "removeZeroValueRows()" function, which takes arguments to remove rows where the chosen column has a value of 0. reading the dataset into the variable data comes next. The previously declared method is now used to delete any rows that have "creditRating" column values of 0. Additionally, the function "createDataPartition()" is declared to actually divide the data. Using this function, a variable called "setValues" is created once the values in the "creditRating" column have been removed. We choose the data rows that match the setValues produced by "createDataPartition()" to make the training set. In a similar manner, we produce the test set by removing the data rows that match the setValues that were generated.

2. A. DECISION TREE

Using the default settings, the following decision tree was fitted to the training set to predict the credit ratings of customers using all the other variables in the dataset.



```
> tree_model <- rpart(creditRating ~ ., data = training_set, method = "class")
> # Print the resulting tree
> print(tree_model)
n= 982

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

 1) root 982 487 2 (0.23625255 0.50407332 0.25967413)
   2) functionary>=0.5 274 132 1 (0.51824818 0.31751825 0.16423358)
     4) FI30.credit.score>=0.5 263 122 1 (0.53612167 0.32699620 0.13688213)
       8) re.balanced..paid.back..a.recently.overdrawn.current.acount>=0.5 254 113 1 (0.55511811 0.33070866 0.11417323)
        16) credit.refused.in.past.< 0.5 235  98 1 (0.58297872 0.32765957 0.08936170)
          32) avrg..account.balance.2.months.ago>=2.5 149  55 1 (0.63087248 0.24832215 0.12080537) *
          33) avrg..account.balance.2.months.ago< 2.5 86  43 1 (0.50000000 0.46511628 0.03488372)
            66) avrg..account.balance.1.months.ago< 3.5 51  19 1 (0.62745098 0.33333333 0.03921569) *
            67) avrg..account.balance.1.months.ago>=3.5 35  12 2 (0.31428571 0.65714286 0.02857143) *
        17) credit.refused.in.past.>=0.5 19  11 3 (0.21052632 0.36842105 0.42105263) *
       9) re.balanced..paid.back..a.recently.overdrawn.current.acount< 0.5 9   2 3 (0.00000000 0.22222222 0.77777778) *
     5) FI30.credit.score< 0.5 11   2 3 (0.09090909 0.09090909 0.81818182) *
   3) functionary< 0.5 708 300 2 (0.12711864 0.57627119 0.29661017)
     6) FI30.credit.score>=0.5 656 258 2 (0.13719512 0.60670732 0.25609756)
      12) re.balanced..paid.back..a.recently.overdrawn.current.acount>=0.5 598 214 2 (0.14548495 0.64214047 0.21237458) *
      13) re.balanced..paid.back..a.recently.overdrawn.current.acount< 0.5 58  17 3 (0.05172414 0.24137931 0.70689655) *
     7) FI30.credit.score< 0.5 52  10 3 (0.00000000 0.19230769 0.80769231) *
> rpart.plot(tree_model)
```



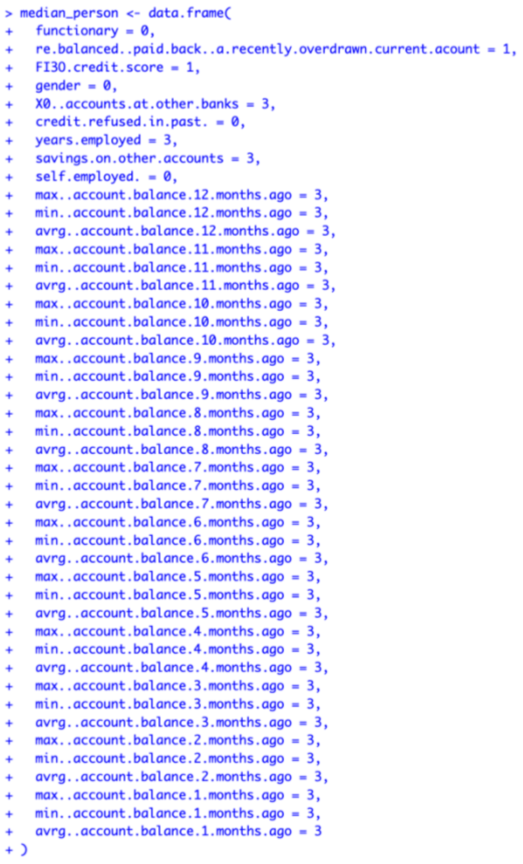
- The rood node in the aforementioned tree indicates that the target variable has 23% credit rating = 1, 51% credit rating = 2 and 26% credit rating = 3.

- The tree shows that the dataset is more biased to credit rating 2. So, the root node is based on whether the customers have credit rating 2 or not. The first split is based on functionary=1 so, 28% has functionary =1 and 72% do not have functionary =1.
- The deviance measure quantifies the overall purity or homogeneity of the instances within each node. A lower deviance indicates a more accurate prediction.
- The length of the longest path from a node to a leaf is referred to as the node's height in a binary tree. A balanced binary tree has a height difference between the left and right subtrees of no more than one. However, we can see from the tree's plot that the left and right subtrees are three inches apart in height. The heights of the two subtrees are noticeably different, which suggests that the tree is unbalanced.

B. Predicting the credit rating of a hypothetical "median" customer

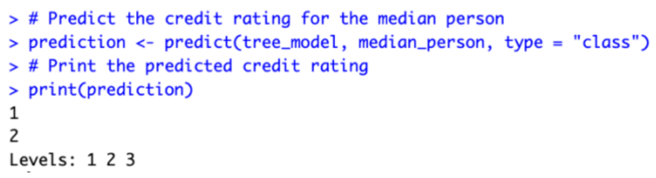
To predict the credit rating of the median customer, the attributes listed in Table 1 were used.

We have therefore, in regard to all the columns in the "**training_set",** created a data frame "**median_person**" with attribute values set to the median values, as demonstrated below.



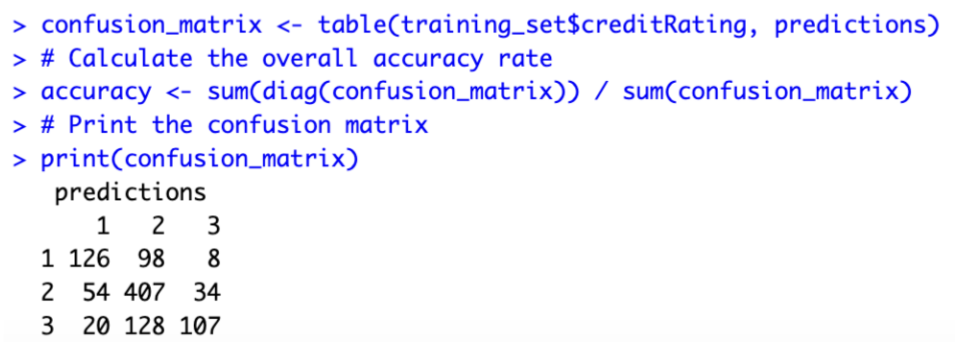
```
> median_person <- data.frame(
+   functionary = 0,
+   re.balanced..paid.back..a.recently.overdrawn.current.acount = 1,
+   FI3O.credit.score = 1,
+   gender = 0,
+   X0..accounts.at.other.banks = 3,
+   credit.refused.in.past. = 0,
+   years.employed = 3,
+   savings.on.other.accounts = 3,
+   self.employed. = 0,
+   max..account.balance.12.months.ago = 3,
+   min..account.balance.12.months.ago = 3,
+   avrg..account.balance.12.months.ago = 3,
+   max..account.balance.11.months.ago = 3,
+   min..account.balance.11.months.ago = 3,
+   avrg..account.balance.11.months.ago = 3,
+   max..account.balance.10.months.ago = 3,
+   min..account.balance.10.months.ago = 3,
+   avrg..account.balance.10.months.ago = 3,
+   max..account.balance.9.months.ago = 3,
+   min..account.balance.9.months.ago = 3,
+   avrg..account.balance.9.months.ago = 3,
+   max..account.balance.8.months.ago = 3,
+   min..account.balance.8.months.ago = 3,
+   avrg..account.balance.8.months.ago = 3,
+   max..account.balance.7.months.ago = 3,
+   min..account.balance.7.months.ago = 3,
+   avrg..account.balance.7.months.ago = 3,
+   max..account.balance.6.months.ago = 3,
+   min..account.balance.6.months.ago = 3,
+   avrg..account.balance.6.months.ago = 3,
+   max..account.balance.5.months.ago = 3,
+   min..account.balance.5.months.ago = 3,
+   avrg..account.balance.5.months.ago = 3,
+   max..account.balance.4.months.ago = 3,
+   min..account.balance.4.months.ago = 3,
+   avrg..account.balance.4.months.ago = 3,
+   max..account.balance.3.months.ago = 3,
+   min..account.balance.3.months.ago = 3,
+   avrg..account.balance.3.months.ago = 3,
+   max..account.balance.2.months.ago = 3,
+   min..account.balance.2.months.ago = 3,
+   avrg..account.balance.2.months.ago = 3,
+   max..account.balance.1.months.ago = 3,
+   min..account.balance.1.months.ago = 3,
+   avrg..account.balance.1.months.ago = 3
+ )
```

The "**predict**" function of the decision tree model was used with the attribute values of the median client to forecast their credit rating. As a result, the median customer's credit score was found to be 2.

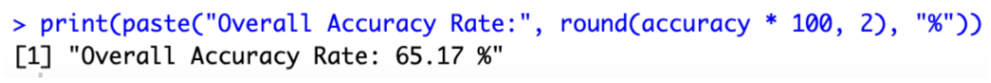


```
> # Predict the credit rating for the median person
> prediction <- predict(tree_model, median_person, type = "class")
> # Print the predicted credit rating
> print(prediction)
1
2
Levels: 1 2 3
```

C. The confusion matrix for extrapolating the credit rating from the tree is displayed in the figure below. The obtained accuracy is 65.17%.

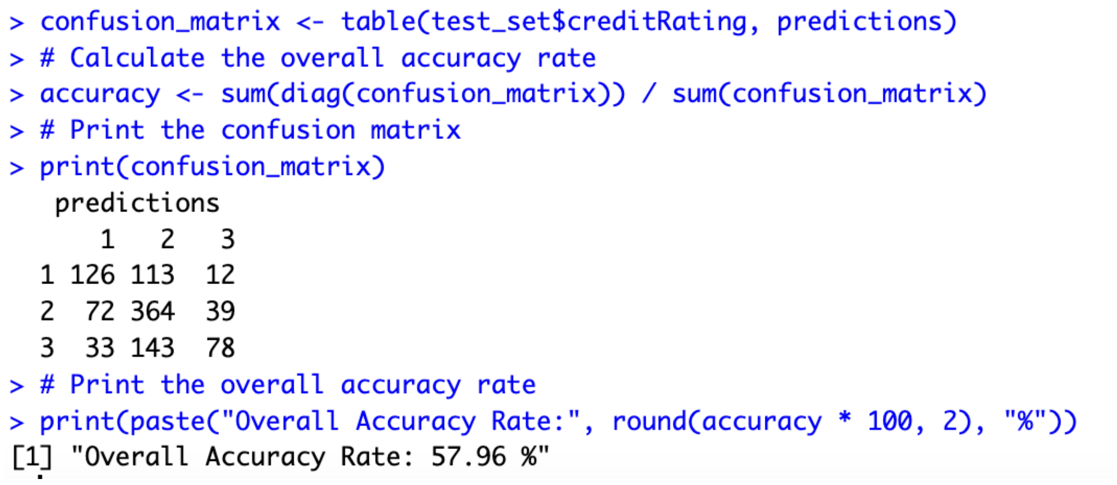


```
> confusion_matrix <- table(training_set$creditRating, predictions)
> # Calculate the overall accuracy rate
> accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
> # Print the confusion matrix
> print(confusion_matrix)
   predictions
      1   2   3
  1 126  98   8
  2  54 407  34
  3  20 128 107
```



```
> print(paste("Overall Accuracy Rate:", round(accuracy * 100, 2), "%"))
[1] "Overall Accuracy Rate: 65.17 %"
```

But in order to forecast the credit rating for test_set using a new tree model, we need the confusion matrix shown below. Expected values are displayed on the y-axis, and actual values are displayed on the x-axis.

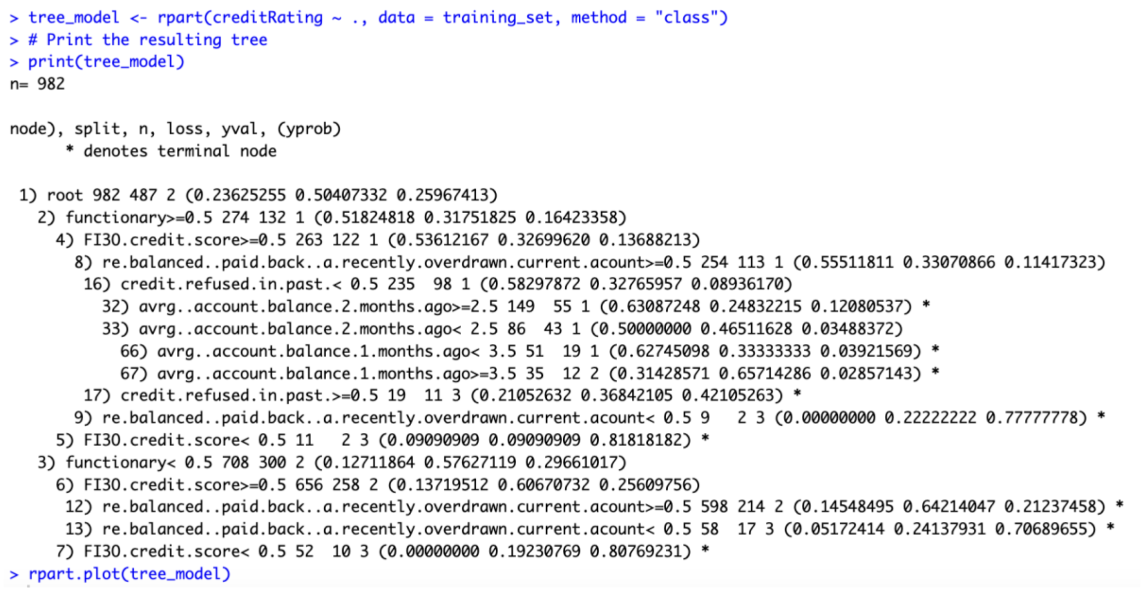


```
> confusion_matrix <- table(test_set$creditRating, predictions)
> # Calculate the overall accuracy rate
> accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
> # Print the confusion matrix
> print(confusion_matrix)
   predictions
      1   2   3
  1 126 113  12
  2  72 364  39
  3  33 143  78
> # Print the overall accuracy rate
> print(paste("Overall Accuracy Rate:", round(accuracy * 100, 2), "%"))
[1] "Overall Accuracy Rate: 57.96 %"
```

Regarding the results interpretation, the model was able to classify 568/980 observations from the test sets correctly. The number of false negatives decreased to 0 in case of the test set, in comparison to the previous matrix. The overall Accuracy Rate is 57.96%. The model had difficulty in classifying for credit rating 3, where only 78 observations have been classified correctly. The model performed better in classifying customers in credit rating 2 where only 111 observations have been classified incorrectly, the rest 364 have been classified correctly.

D. EVALUATION OF NUMERICAL VALUE OF THE GAIN IN ENTROPY CORRESPONDING TO THE FIRST SPLIT AT THE TOP OF THE TREE

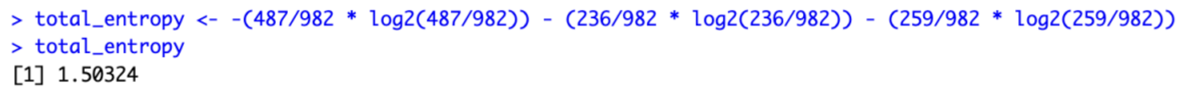
Decision trees have the feature of hierarchically splitting data sets; each consecutive cut is based on a single attribute. Using the supplied dataset, we will forecast the customer's credit score. Entropy is expressed in units of zero for an entirely homogeneous sample and one for an equally distributed sample.



```
> tree_model <- rpart(creditRating ~ ., data = training_set, method = "class")
> # Print the resulting tree
> print(tree_model)
n= 982

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

 1) root 982 487 2 (0.23625255 0.50407332 0.25967413)
   2) functionary>=0.5 274 132 1 (0.51824818 0.31751825 0.16423358)
     4) FI30.credit.score>=0.5 263 122 1 (0.53612167 0.32699620 0.13688213)
       8) re.balanced..paid.back..a.recently.overdrawn.current.acount>=0.5 254 113 1 (0.55511811 0.33070866 0.11417323)
        16) credit.refused.in.past.< 0.5 235  98 1 (0.58297872 0.32765957 0.08936170)
          32) avrg..account.balance.2.months.ago>=2.5 149  55 1 (0.63087248 0.24832215 0.12080537) *
          33) avrg..account.balance.2.months.ago< 2.5 86  43 1 (0.50000000 0.46511628 0.03488372)
            66) avrg..account.balance.1.months.ago< 3.5 51  19 1 (0.62745098 0.33333333 0.03921569) *
            67) avrg..account.balance.1.months.ago>=3.5 35  12 2 (0.31428571 0.65714286 0.02857143) *
        17) credit.refused.in.past.>=0.5 19  11 3 (0.21052632 0.36842105 0.42105263) *
       9) re.balanced..paid.back..a.recently.overdrawn.current.acount< 0.5 9   2 3 (0.00000000 0.22222222 0.77777778) *
     5) FI30.credit.score< 0.5 11   2 3 (0.09090909 0.09090909 0.81818182) *
   3) functionary< 0.5 708 300 2 (0.12711864 0.57627119 0.29661017)
     6) FI30.credit.score>=0.5 656 258 2 (0.13719512 0.60670732 0.25609756)
      12) re.balanced..paid.back..a.recently.overdrawn.current.acount>=0.5 598 214 2 (0.14548495 0.64214047 0.21237458) *
      13) re.balanced..paid.back..a.recently.overdrawn.current.acount< 0.5 58  17 3 (0.05172414 0.24137931 0.70689655) *
     7) FI30.credit.score< 0.5 52  10 3 (0.00000000 0.19230769 0.80769231) *
> rpart.plot(tree_model)
```

Entropy can be used to quantify impurity, disorder, or uncertainty in a variety of situations. Entropy determines which way a Decision Tree divides the data. Information gain (IG) measures how much "information" a feature gives us about the class. The information gain when splitting a dataset along an attribute is determined by the decrease in entropy that results.

And, to consider the information gain, we will base on the information provided in the above “tree_model” output,



```
> total_entropy <- -(487/982 * log2(487/982)) - (236/982 * log2(236/982)) - (259/982 * log2(259/982))
> total_entropy
[1] 1.50324
```

Based on the information from the tree model provided above, the total entropy within the daughter nodes after the first split is 0.99859553 for the left branch and 0.9835077 for the right branch.

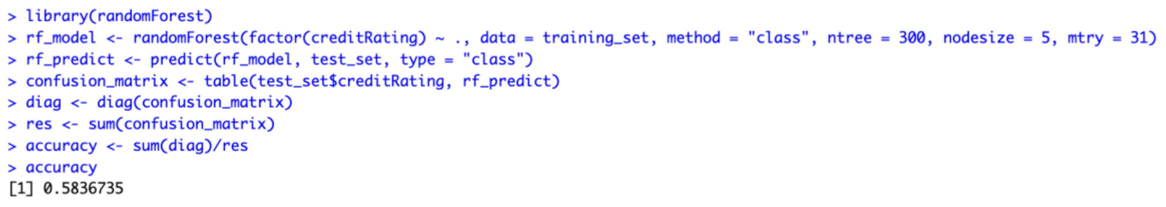
And, to calculate the weighted average entropy, we have considered the number of instances in each branch as the weight factor. In the tree model, there are 274 instances in the left branch and 708 instances in the right branch.

In comparison to the maximum information gain of 1, the first split's information gain of 0.0123 shows only a minor improvement in categorisation accuracy.

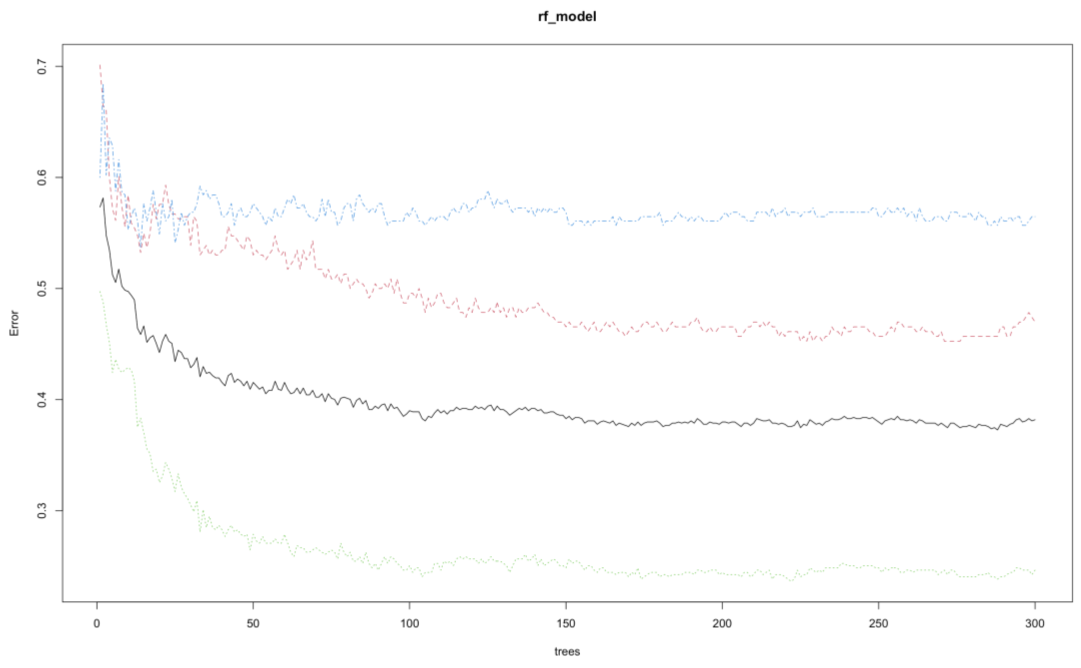
And the total entropy keeps declining as the decision tree grows and more branches are added. This suggests that later breaks in the tree are more effective at lowering uncertainty and raising classification accuracy.

E. RANDOM FOREST

A random forest model was fitted using the training set to improve the prediction. The decision tree's accuracy of 65.17% was exceeded by the best accuracy of 58.37% obtained when the mtry error rate was set to 31.

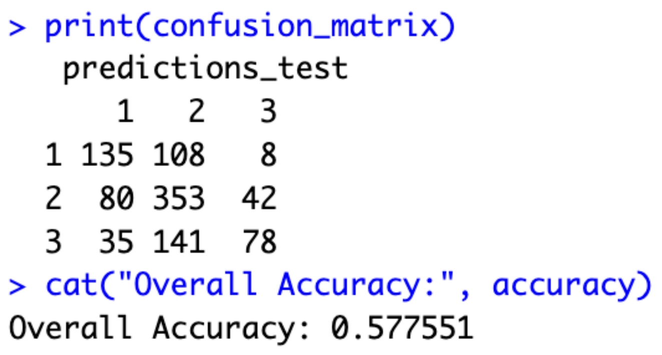


```
> library(randomForest)
> rf_model <- randomForest(factor(creditRating) ~ ., data = training_set, method = "class", ntree = 300, nodesize = 5, mtry = 31)
> rf_predict <- predict(rf_model, test_set, type = "class")
> confusion_matrix <- table(test_set$creditRating, rf_predict)
> diag <- diag(confusion_matrix)
> res <- sum(confusion_matrix)
> accuracy <- sum(diag)/res
> accuracy
[1] 0.5836735
```



F. CONFUSION MATRIX OF RANDOM FOREST FOR CREDIT RATING

The overall accuracy rate is at 57.75% which is just 0.21% lower than the decision tree model on the test data.



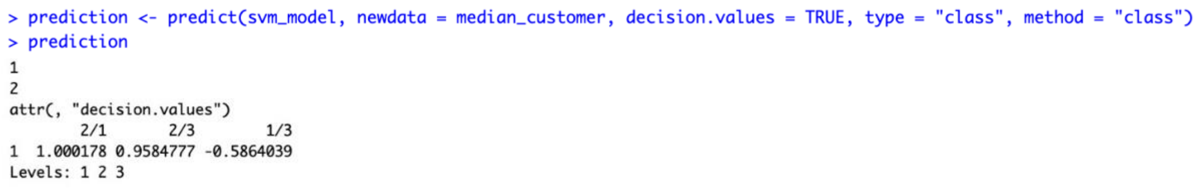
```
> print(confusion_matrix)
   predictions_test
      1   2   3
  1 135 108   8
  2  80 353  42
  3  35 141  78
> cat("Overall Accuracy:", accuracy)
Overall Accuracy: 0.577551
```

3. A. SUPPORT VECTOR MACHINE

We have generated a data frame called "**median_customer**" based on the attributes provided in Table 1 that contains the attribute values set to their corresponding median values. The "**svm_model**" that we had previously trained with the SVM algorithm was then given this data frame as input.

We predicted the credit score of the hypothetical median customer using the trained SVM model on the "**median_customer"** data frame. Decision values are produced by the SVM model and show how confident or certain the predictions are.

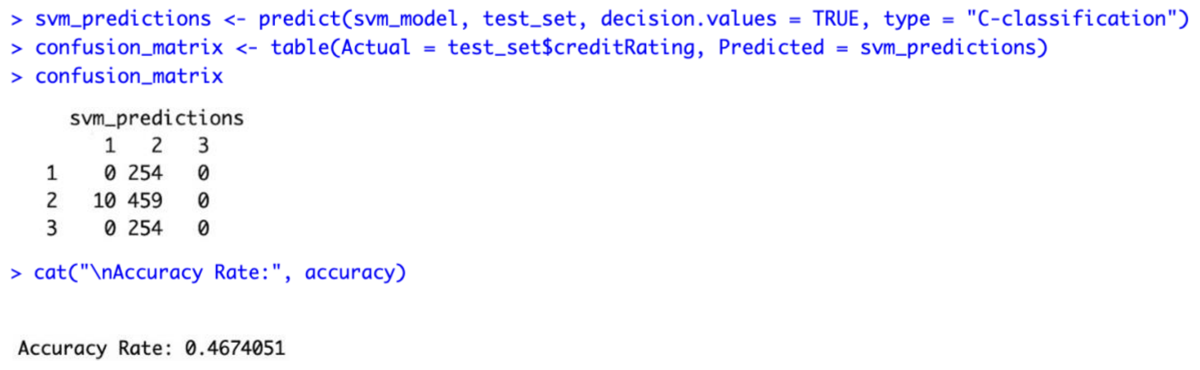
The decision values show how likely each potential credit rating for the average client is according to the model. These numbers can explain how confident the model is in its forecasts.



```
> prediction <- predict(svm_model, newdata = median_customer, decision.values = TRUE, type = "class", method = "class")
> prediction
1
2
attr(, "decision.values")
        2/1       2/3        1/3
1  1.000178 0.9584777 -0.5864039
Levels: 1 2 3
```

The decision values show how far apart our sample is from the hyperplane. Three decision boundaries were produced for this dataset using the svm. Depending on the indication, the sample is either left or right of the hyperplane. The opposite side of the hyperplane is represented in this situation by negative integers.

B. Using the SVM model above a confusion matrix was produced along with the overall accuracy rate.

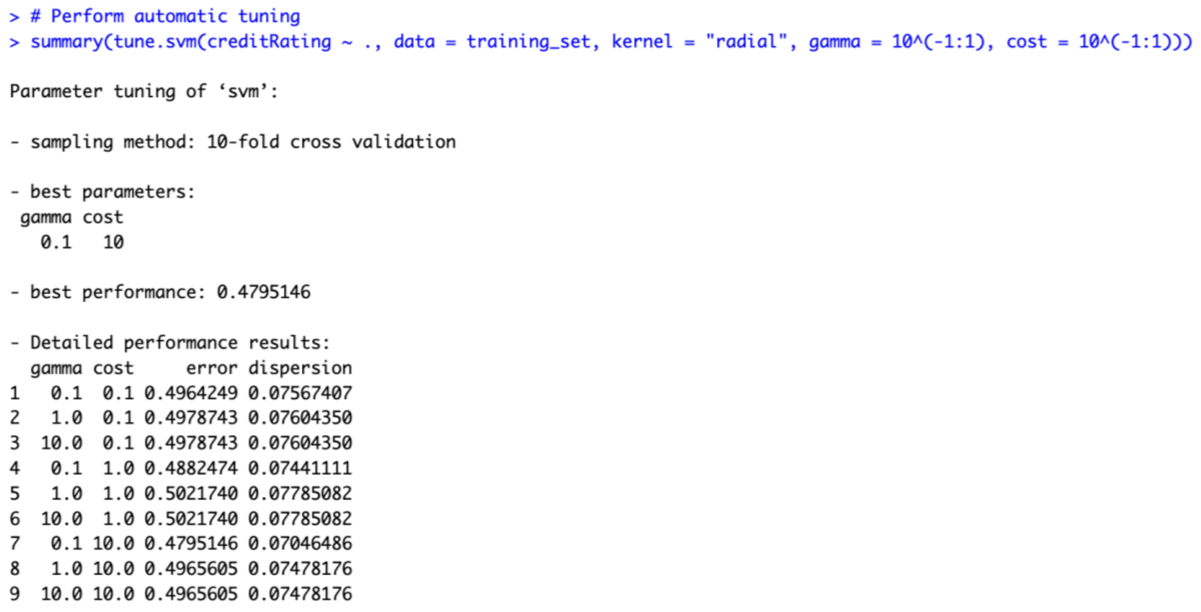


```
> svm_predictions <- predict(svm_model, test_set, decision.values = TRUE, type = "C-classification")
> confusion_matrix <- table(Actual = test_set$creditRating, Predicted = svm_predictions)
> confusion_matrix

      svm_predictions
         1   2   3
   1     0 254   0
   2    10 459   0
   3     0 254   0

> cat("\nAccuracy Rate:", accuracy)


 Accuracy Rate: 0.4674051
```

The model accurately predicted 459 observations as category 2, while misclassifying the remaining observations. The resulting overall accuracy, in the light of the confusion matrix for the test data is 46.74%. And, based on the accuracy achieved, it was a difficult task for SVM to predict credit rating for seen and unseen data, both. Besides, the imbalanced nature of the data can be deemed as one of the reasons for this, especially being more inclined towards category 2.

C. Automatic Tuning of SVM to improve prediction.

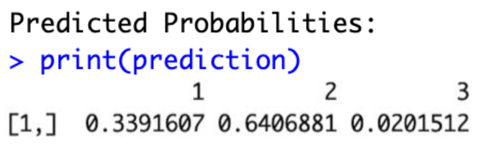
After completing the automatic tuning of SVM using tune.svm(), the best performance/accuracy achieved was 47.95%, which is 1.21% higher than the default svm() function. To perform the tuning, following settings have been used as mentioned below.



```
> # Perform automatic tuning
> summary(tune.svm(creditRating ~ ., data = training_set, kernel = "radial", gamma = 10^(-1:1), cost = 10^(-1:1)))

Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 gamma cost
   0.1   10

- best performance: 0.4795146

- Detailed performance results:
  gamma cost     error dispersion
1   0.1  0.1 0.4964249 0.07567407
2   1.0  0.1 0.4978743 0.07604350
3  10.0  0.1 0.4978743 0.07604350
4   0.1  1.0 0.4882474 0.07441111
5   1.0  1.0 0.5021740 0.07785082
6  10.0  1.0 0.5021740 0.07785082
7   0.1 10.0 0.4795146 0.07046486
8   1.0 10.0 0.4965605 0.07478176
9  10.0 10.0 0.4965605 0.07478176
```

4. NAÏVE BAYES

A. We have predicted the credit rating of the hypothetical median customer based on the attributes listed in Table 1. A similar data frame called **“median_customer”** was created like in decision tree model with median values. The data frame is then used to predict using the **“naïve_model”.**



```
Predicted Probabilities:
> print(prediction)
             1         2         3
[1,]  0.3391607 0.6406881 0.0201512
```

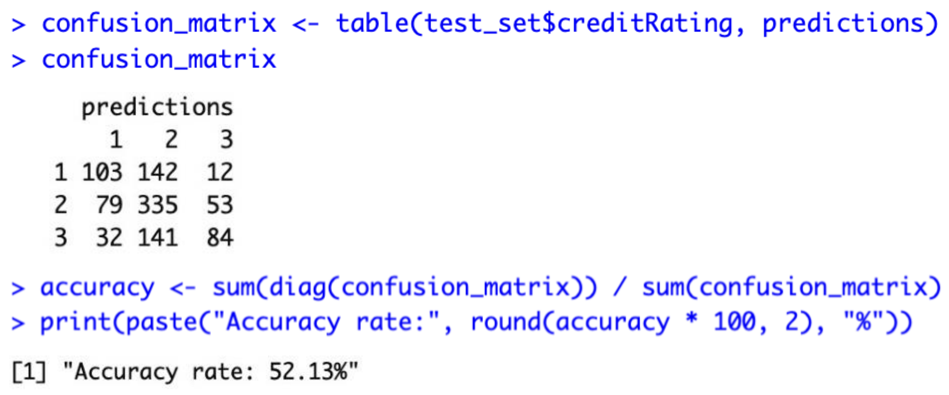
The model predicted that 64.06% probability is for customers with credit rate as 2 and 33.09% probability for customers with credit rating 1.

B. Reproducing and detailing the first 20 lines of the R output for Naïve Bayes fit.

The dataset was prepared by converting categorical variables into numbers and ensuring the right format. The Naive Bayes model learns from the training set, estimating feature probabilities given class labels. It assumes features are independent given the class. Probabilities are calculated by counting occurrences and dividing by total instances. Prior probabilities are computed for each class. To predict a new instance, Bayes' theorem calculates the probability of each class given the features. The class with the highest probability is chosen as the prediction.

C. Confusion Matrix

The confusion matrix for predicting the credit rating using Naive Bayes model on the test set is produced as below and the overall achieved accuracy rate is 52.13 %.



```
> confusion_matrix <- table(test_set$creditRating, predictions)
> confusion_matrix
   predictions
      1   2   3
  1 103 142  12
  2  79 335  53
  3  32 141  84
> accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
> print(paste("Accuracy rate:", round(accuracy * 100, 2), "%"))
[1] "Accuracy rate: 52.13%"
```

5. A. Identifying the best classifiers using the Confusion Matrices

- Decision Tree = 57.96%
- Random Forest = 57.75%
- SVM = 47.95%
- Naïve Bayes = 52.13%

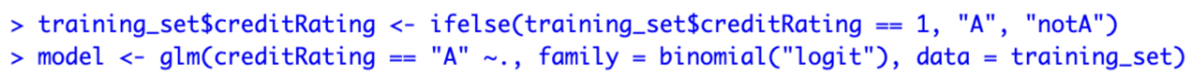
Decision tree and Random Forest seem to be the best classifier for the dataset as seen from the previous classification models.

B. SVM has generated the worst accuracy, ranking as the least perform classifier among all at 47.95% accuracy overall. This could mainly be due to the imbalance in the dataset, an environment in which SVM models usually do not perform well. And, in case multiple classifications, SVM are deemed not to perform and not an ideal option.

C. Precisely, all the classifiers have a bad performance or accuracy with class 1 and 3, especially with class 3, which mostly gets incorrectly predicted by all the classifiers. This information can be interpreted from the confusion matrices generated in each classifier.

6. Predicting the customers based on their credit rating if they fall in category A

A. Fitting logistic regression model

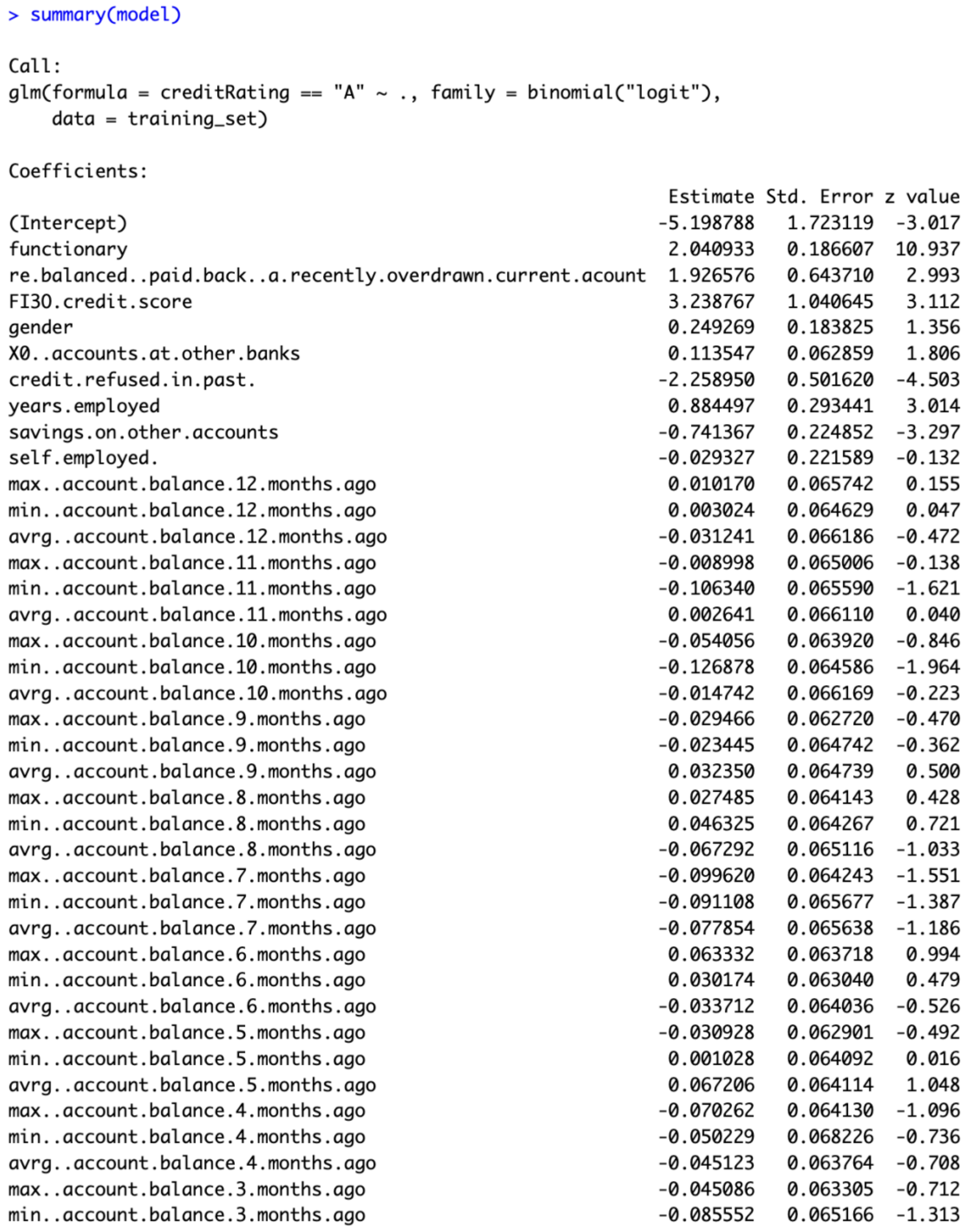


```
> training_set$creditRating <- ifelse(training_set$creditRating == 1, "A", "notA")
> model <- glm(creditRating == "A" ~., family = binomial("logit"), data = training_set)
```

Since, the model was supposed to be fit regarding the prediction based on customers falling in category A. "**ifelse**" function was used for this purpose.

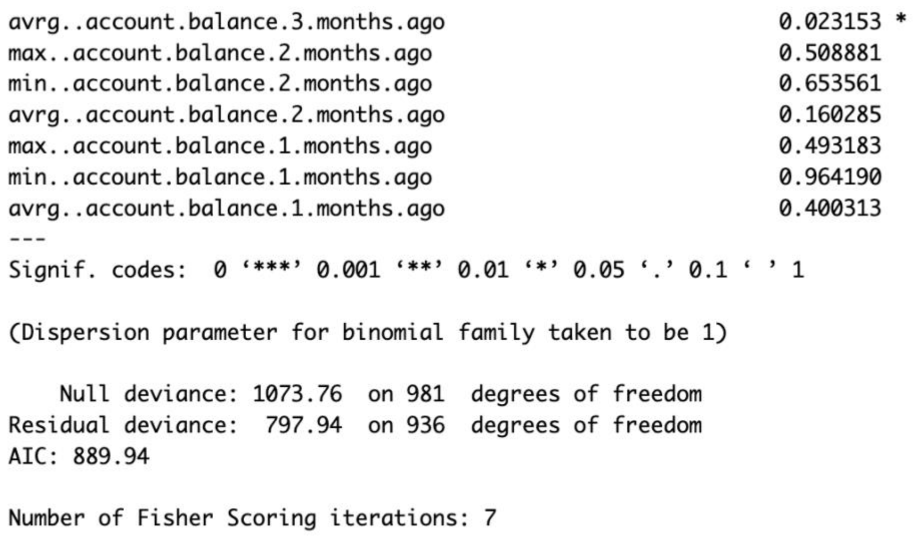
The "**glm()"** method is used in the provided code to build a logistic regression model. Predicting the "**creditRating"** variable from the values of all the other variables in the dataset is the objective. By indicating that we wish to employ every accessible variable for prediction, we apply the formula "**creditRating"**. We are working with a binary classification problem, where the outcome variable can have one of two values, according to the "**family = binomial"** argument. For this kind of issue, the binomial family is appropriate.

B. Logistic Regression Model Fit Summary



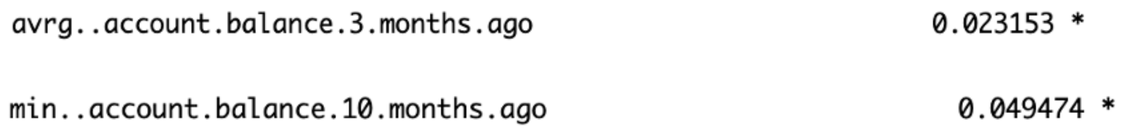
```
> summary(model)

Call:
glm(formula = creditRating == "A" ~ ., family = binomial("logit"),
    data = training_set)

Coefficients:
                                                     Estimate Std. Error z value
(Intercept)                                         -5.198788   1.723119  -3.017
functionary                                          2.040933   0.186607  10.937
re.balanced..paid.back..a.recently.overdrawn.current.acount  1.926576   0.643710   2.993
FI3O.credit.score                                    3.238767   1.040645   3.112
gender                                               0.249269   0.183825   1.356
X0..accounts.at.other.banks                          0.113547   0.062859   1.806
credit.refused.in.past.                             -2.258950   0.501620  -4.503
years.employed                                       0.884497   0.293441   3.014
savings.on.other.accounts                           -0.741367   0.224852  -3.297
self.employed.                                      -0.029327   0.221589  -0.132
max..account.balance.12.months.ago                   0.010170   0.065742   0.155
min..account.balance.12.months.ago                   0.003024   0.064629   0.047
avrg..account.balance.12.months.ago                 -0.031241   0.066186  -0.472
max..account.balance.11.months.ago                  -0.008998   0.065006  -0.138
min..account.balance.11.months.ago                  -0.106340   0.065590  -1.621
avrg..account.balance.11.months.ago                  0.002641   0.066110   0.040
max..account.balance.10.months.ago                  -0.054056   0.063920  -0.846
min..account.balance.10.months.ago                  -0.126878   0.064586  -1.964
avrg..account.balance.10.months.ago                 -0.014742   0.066169  -0.223
max..account.balance.9.months.ago                   -0.029466   0.062720  -0.470
min..account.balance.9.months.ago                   -0.023445   0.064742  -0.362
avrg..account.balance.9.months.ago                   0.032350   0.064739   0.500
max..account.balance.8.months.ago                    0.027485   0.064143   0.428
min..account.balance.8.months.ago                    0.046325   0.064267   0.721
avrg..account.balance.8.months.ago                  -0.067292   0.065116  -1.033
max..account.balance.7.months.ago                   -0.099620   0.064243  -1.551
min..account.balance.7.months.ago                   -0.091108   0.065677  -1.387
avrg..account.balance.7.months.ago                  -0.077854   0.065638  -1.186
max..account.balance.6.months.ago                    0.063332   0.063718   0.994
min..account.balance.6.months.ago                    0.030174   0.063040   0.479
avrg..account.balance.6.months.ago                  -0.033712   0.064036  -0.526
max..account.balance.5.months.ago                   -0.030928   0.062901  -0.492
min..account.balance.5.months.ago                    0.001028   0.064092   0.016
avrg..account.balance.5.months.ago                   0.067206   0.064114   1.048
max..account.balance.4.months.ago                   -0.070262   0.064130  -1.096
min..account.balance.4.months.ago                   -0.050229   0.068226  -0.736
avrg..account.balance.4.months.ago                  -0.045123   0.063764  -0.708
max..account.balance.3.months.ago                   -0.045086   0.063305  -0.712
min..account.balance.3.months.ago                   -0.085552   0.065166  -1.313
```

```
avrg..account.balance.3.months.ago                       0.146242   0.064398   2.271
max..account.balance.2.months.ago                       -0.041900   0.063428  -0.661
min..account.balance.2.months.ago                       -0.029672   0.066112  -0.449
avrg..account.balance.2.months.ago                       0.093472   0.066570   1.404
max..account.balance.1.months.ago                       -0.044698   0.065228  -0.685
min..account.balance.1.months.ago                       -0.002939   0.065473  -0.045
avrg..account.balance.1.months.ago                      -0.054422   0.064706  -0.841
                                                        Pr(>|z|)
(Intercept)                                             0.002552 **
functionary                                              < 2e-16 ***
re.balanced..paid.back..a.recently.overdrawn.current.acount 0.002763 **
FI30.credit.score                                       0.001857 **
gender                                                  0.175095
X0..accounts.at.other.banks                             0.070860 .
credit.refused.in.past.                                 6.69e-06 ***
years.employed                                          0.002576 **
savings.on.other.accounts                               0.000977 ***
self.employed.                                          0.894709
max..account.balance.12.months.ago                      0.877058
min..account.balance.12.months.ago                      0.962683
avrg..account.balance.12.months.ago                     0.636918
max..account.balance.11.months.ago                      0.889904
min..account.balance.11.months.ago                      0.104959
avrg..account.balance.11.months.ago                     0.968133
max..account.balance.10.months.ago                      0.397732
min..account.balance.10.months.ago                      0.049474 *
avrg..account.balance.10.months.ago                     0.823692
max..account.balance.9.months.ago                       0.638491
min..account.balance.9.months.ago                       0.717261
avrg..account.balance.9.months.ago                      0.617288
max..account.balance.8.months.ago                       0.668288
min..account.balance.8.months.ago                       0.471017
avrg..account.balance.8.months.ago                      0.301408
max..account.balance.7.months.ago                       0.120980
min..account.balance.7.months.ago                       0.165379
avrg..account.balance.7.months.ago                      0.235578
max..account.balance.6.months.ago                       0.320252
min..account.balance.6.months.ago                       0.632190
avrg..account.balance.6.months.ago                      0.598568
max..account.balance.5.months.ago                       0.622930
min..account.balance.5.months.ago                       0.987203
avrg..account.balance.5.months.ago                      0.294534
max..account.balance.4.months.ago                       0.273241
min..account.balance.4.months.ago                       0.461602
avrg..account.balance.4.months.ago                      0.479159
max..account.balance.3.months.ago                       0.476341
min..account.balance.3.months.ago                       0.189242
```



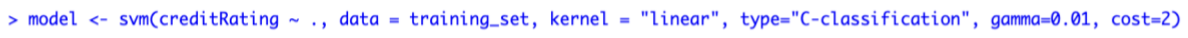
```
avrg..account.balance.3.months.ago                          0.023153 *
max..account.balance.2.months.ago                           0.508881
min..account.balance.2.months.ago                           0.653561
avrg..account.balance.2.months.ago                          0.160285
max..account.balance.1.months.ago                           0.493183
min..account.balance.1.months.ago                           0.964190
avrg..account.balance.1.months.ago                          0.400313
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1073.76  on 981  degrees of freedom
Residual deviance:  797.94  on 936  degrees of freedom
AIC: 889.94

Number of Fisher Scoring iterations: 7
```

C. Based on the summary table provided for the logistic regression model using the “**glm()”** function, it can be noticed that the predictors of credit rating which appear to be significant at 5% significance level are as follows, with the respective Pr values:



```
avrg..account.balance.3.months.ago                 0.023153 *

min..account.balance.10.months.ago                 0.049474 *
```

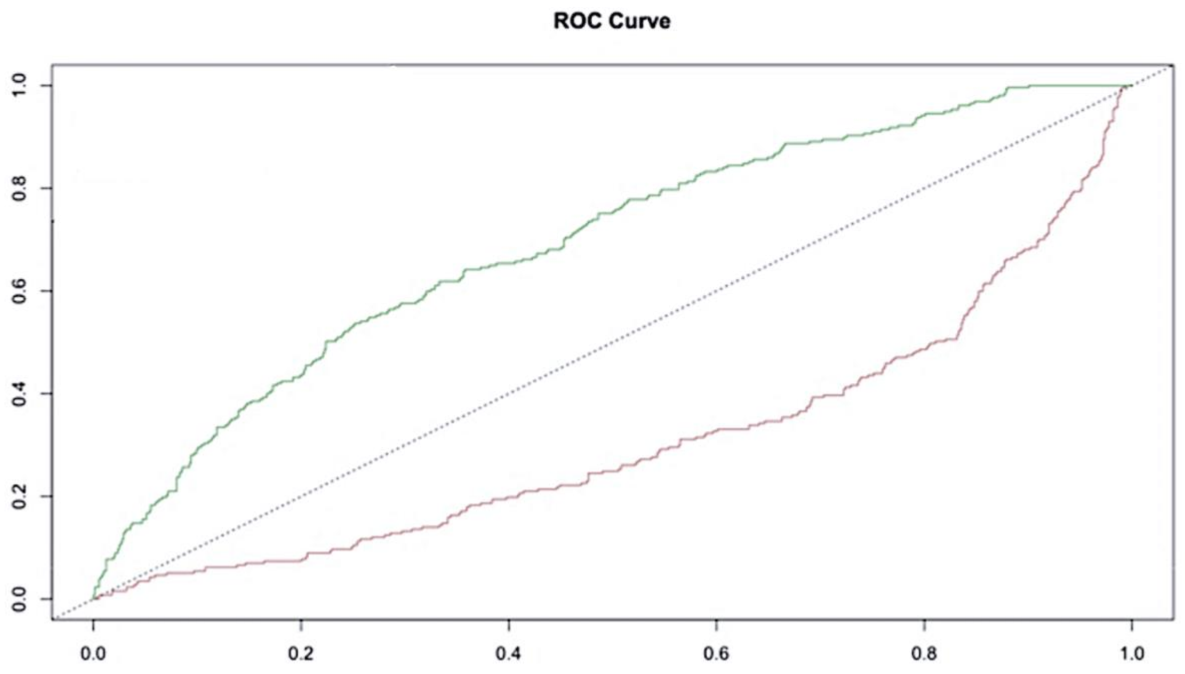
D. Fit SVM model



```
> model <- svm(creditRating ~ ., data = training_set, kernel = "linear", type="C-classification", gamma=0.01, cost=2)
```

Using the “**svm()”** function, an SVM model was fitted. With the help of all the other variables in the dataset, we wish to forecast the credit rating, as shown by the formula "**creditRating~.**". For linear kernel in the SVM model, kernel = "**linear**" option was used.

E. ROC curve for SVM and Logistic regression.



The X-axis of the plot denotes “**False positive rate”** and the Y-axis of the plot denotes “**True positive rate”.**

The ROC curve provides a comparison between the SVM (Red curve) and Logistic Regression (Green curve). It can also be noticed that the Random Classifier line splits the plot exactly in two halves, into true positives and false positives. Logistic regression ROC curve is above the random classifier line, which is better than the ROC curve of the SVM model, clearly stating that the Logistic Regression model is better among the both.