Capstone Project 2 : Credit Score Classification

2023-12-06

Overview

Credit Score is one of the most important personal score to maintain at it's best case, It represents the creditworthiness of that person, which is the likelihood that people will pay their bills [1].

In this project I will use the Credit Score dataset from Kaggle [2], the dataset contains one target which is the credit score index (Poor, Standard, and Good) and 27 different features, the goal of this project is to predict the credit score index of the customer based on his/her input features. The dataset contains 100,000 data entries where every customer has multiple entries over many months.

The dataset contains two files, *train* and *test*, however the *test* file is not labeled which will leave us with the 100000 entries in the *train* file, the *train* file will be loaded into the *data* variable.

The first note about the dataset is that it contains some missing values and the data need to be cleaned, that's why the first section will be about cleaning the dataset and features engineering, then some exploratory data analysis and features selection will be done the dataset, then the models will be built to predict the credit score based on the selected features.

I built 4 models using *gam*, *knn*, *rpart* and *rf* algorithms using the caret package, I used the default parameters for the first time, then based on their training plot I extended the search space for more values. The best performing model was the *rf* with accuracy of 79.2% using the tuning parameter *mtry* value of 30.

Contents

Dataset Cleaning and Features Engineering

Unfortunately, The dataset is not very clean and many preprocessing need to be done, some columns has missing values, the code below showing the columns has NA values and the count of those missing values:

```
na_vals <- colSums(is.na(org_data))
as_tibble_row(na_vals[na_vals>0])%>% kbl(caption = 'Columns with missing values')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 1: Columns with missing values

Monthly_Inhand_Salary	Num_Credit_Inquiries	Credit_History_Age
15002	1965	9030

I also noted that while exploring the dataset some missing values are denoted by other values like empty strings or _ or other values.

Table-2 summarize the dataset and the processing required to clean it.

```
dset_cols %>% kbl(caption = 'The dataset summary')%>%
  kable_styling(latex_options=c("striped")) %>%
  column_spec(1,width = "3cm") %>% column_spec(2:3,width = "6.5cm")
```

Explicit NA Handeling

In this section, I will handle the NA values in the our dataset, I will start first with Monthly_Inhand_Salary column, the following code showing a sample of original values of the column:

```
head(org_data%>%select(Customer_ID,Monthly_Inhand_Salary))%>%
  kbl(caption = 'Monthly Inhand Salary Sample')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 3: Monthly Inhand Salary Sample

Customer_ID	Monthly_Inhand_Salary
CUS_0xd40	1824.843
CUS_0xd40	NA
CUS_0xd40	NA
CUS_0xd40	NA
CUS_0xd40	1824.843
CUS_0xd40	NA

For the same customer, usually this value will change a lot from month to month, I will take the average value as the mode value for each customer as in the following code:

```
mis_mod <- data %>% filter(!is.na(Monthly_Inhand_Salary)) %>%
  group_by(Customer_ID) %>% summarise(mode = mean(Monthly_Inhand_Salary))
```

NA values for each customer will be replaced with the average value as follows:

```
data <- data%>%left_join(mis_mod,by='Customer_ID')%>%
  mutate(Monthly_Inhand_Salary=ifelse(is.na(Monthly_Inhand_Salary),mode,Monthly_Inhand_Salary))
```

The following code is used to validate the results:

Table 2: The dataset summary

Name	Description	Processing_Required
ID	Represents a unique identification of an entry	None
Customer ID	Represents a unique identification of a person	None
Month	Represents the month of the year	Strings to integers encoding
Name	Represents the name of a person	None, since this column will not be used in the models
Age	Represents the age of the person	Removing the "_" characters, convertting to integers and outliers handling
SSN	Represents the social security number of a person	None, since this column will not be used in the models
Occupation	Represents the occupation of the person	Handling the "——" values, then encode strings to integers
Annual Income	Represents the annual income of the person	Removing the "_" characters, convertting to numbers and outliers handling
Monthly Inhand Salary	Represents the monthly base salary of a person	NA Handling
Num Bank Accounts	Represents the number of bank accounts a person holds	Outliers handling
Num Credit Card	Represents the number of other credit cards held by a person	Outliers handling
Interest Rate	Represents the interest rate on credit card	Outliers handling
Num of Loan	Represents the number of loans taken from the bank	Removing the "_" characters, convertting to integers and outliers handling
Type of Loan	Represents the types of loan taken by a person	Split over "," then convert to one-hot-coding columns
Delay from due date	Represents the average number of days delayed from the payment date	None
Num of Delayed Payment	Represents the average number of payments delayed by a person	Remove the implicit missing values ("_","", and negatives), then impute them
Changed Credit Limit	Represents the percentage change in credit card limit	Replace the "_" with zeros
Num Credit Inquiries	Represents the number of credit card inquiries	NA Handling
Credit Mix	Represents the classification of the mix of credits	Strings to integers encoding
Outstanding Debt	Represents the remaining debt to be paid (in USD)	Removing the "_" characters, convertting to numbers
Credit Utilization Ratio	Represents the utilization ratio of credit card	None
Credit History Age	Represents the age of credit history of the person	Split into year and month (one-hot-coding), then NA handling
Payment of Min Amount	Represents whether only the minimum amount was paid by the person	Strings to integers encoding
Total EMI per month	Represents the monthly EMI payments (in USD)	Outliers handling
Amount invested monthly	Represents the monthly amount invested by the customer (in USD)	Remove the implicit missing values ("", and "___10000___"), then impute them
Payment Behaviour	Represents the payment behavior of the customer (in USD)	Strings to integers encoding
Monthly Balance	Represents the monthly balance amount of the customer (in USD)	Remove the implicit missing values (empty values), then impute them
Credit_Score	Represents the bracket of credit score (Poor, Standard, Good)	Strings to integers encoding

```
Monthly_Inhand_Salary_validation_sample %>%
  kbl(caption = 'Monthly Inhand Salary Fixed Sample')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 4: Monthly Inhand Salary Fixed Sample

Customer_ID	Monthly_Inhand_Salary	mode
CUS_0xd40	1824.843	1824.843
CUS_0xd40	1824.843	1824.843
CUS_0xd40	1824.843	1824.843
CUS_0xd40	1824.843	1824.843
CUS_0xd40	1824.843	1824.843
CUS_0xd40	1824.843	1824.843

The mode will be removed by the following code:

```
data <- data %>% select(-mode)
```

The Num_Credit_Inquiries column has the same issue as the Monthly_Inhand_Salary column and it will be handled in the same way. The following code showing a sample of the NA issue:

```
head(org_data%>%select(Customer_ID,Num_Credit_Inquiries),n=20)%>%
  kbl(caption = 'Num Credit Inquiries Sample')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 5: Num Credit Inquiries Sample

Customer_ID	Num_Credit_Inquiries
CUS_0xd40	4
CUS_0xd40	4
CUS_0xd40	4
CUS_0xd40	4
CUS_0xd40	4
CUS_0xd40	4
CUS_0xd40	4
CUS_0xd40	4
CUS_0x21b1	2
CUS_0x21b1	2
CUS_0x21b1	2
CUS_0x21b1	2
CUS_0x21b1	2
CUS_0x21b1	2
CUS_0x21b1	2
CUS_0x21b1	2
CUS_0x21b1	2
CUS_0x2dbc	3
CUS_0x2dbc	3
CUS_0x2dbc	NA
CUS_0x2dbc	3

For the same customer, usually this value will change a lot from month to month, I will take the average value as the mode value for each customer as in the following code:

```
mis_mod <- data %>% filter(!is.na(Num_Credit_Inquiries)) %>%
  group_by(Customer_ID) %>% summarise(mode = mean(Num_Credit_Inquiries))
```

NA values for each customer will be replaced with the average value as follows:

```
data <- data%>%left_join(mis_mod,by='Customer_ID')%>%
  mutate(Num_Credit_Inquiries=ifelse(is.na(Num_Credit_Inquiries),mode,Num_Credit_Inquiries))
```

The following code is used to validate the results:

```
Num_Credit_Inquiries_validation_sample %>%
  kbl(caption = 'Monthly Inhand Salary Fixed Sample')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 6: Monthly Inhand Salary Fixed Sample

Customer_ID	Num_Credit_Inquiries	mode
CUS_0xd40	4	4
CUS_0xd40	4	4
CUS_0xd40	4	4
CUS_0xd40	4	4
CUS_0xd40	4	4
CUS_0xd40	4	4
CUS_0xd40	4	4
CUS_0xd40	4	4
CUS_0x21b1	2	2
CUS_0x21b1	2	2
CUS_0x21b1	2	2
CUS_0x21b1	2	2
CUS_0x21b1	2	2
CUS_0x21b1	2	2
CUS_0x21b1	2	2
CUS_0x21b1	2	2
CUS_0x21b1	2	2
CUS_0x2dbc	3	3
CUS_0x2dbc	3	3
CUS_0x2dbc	3	3
CUS_0x2dbc	3	3

The mode will be removed by the following code:

```
data <- data %>% select(-mode)
```

Dealing with Credit_History_Age column is a little more tricky, a sample from it telling us is is a complex datatype:

```
head(org_data%>%select(Customer_ID,Credit_History_Age),n=10)%>%
  kbl(caption = 'Credit History Age Sample')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 7: Credit History Age Sample

Customer_ID	Credit_History_Age
CUS_0xd40	22 Years and 1 Months
CUS_0xd40	NA
CUS_0xd40	22 Years and 3 Months
CUS_0xd40	22 Years and 4 Months
CUS_0xd40	22 Years and 5 Months
CUS_0xd40	22 Years and 6 Months
CUS_0xd40	22 Years and 7 Months
CUS_0xd40	NA
CUS_0x21b1	26 Years and 7 Months
CUS_0x21b1	26 Years and 8 Months

Values are going in order with 1 month step, the first thing need to be do with this column is to separate the year and the month using the following code:

```
data$Credit_History_Age_Year <- sapply(data$Credit_History_Age,function(string){
  # Split over the space, then take the first value for the year
  parts <- str_split(string," ")[[1]]
  as.integer(parts[1])
})
data$Credit_History_Age_Month <- sapply(data$Credit_History_Age,function(string){
  # Split over the space, then take the forth value for the year
  parts <- str_split(string," ")[[1]]
  as.integer(parts[4])
})
```

After separating the year and month, they will look as the following:

```
credit_history_age_aftersplit_sample%>%
  kbl(caption = 'Credit History Age After Separating')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 8: Credit History Age After Separating

Customer_ID	Credit_History_Age	Credit_History_Age_Year	Credit_History_Age_Month
CUS_0xd40	22 Years and 1 Months	22	1
CUS_0xd40	NA	NA	NA
CUS_0xd40	22 Years and 3 Months	22	3
CUS_0xd40	22 Years and 4 Months	22	4
CUS_0xd40	22 Years and 5 Months	22	5
CUS_0xd40	22 Years and 6 Months	22	6
CUS_0xd40	22 Years and 7 Months	22	7
CUS_0xd40	NA	NA	NA
CUS_0x21b1	26 Years and 7 Months	26	7
CUS_0x21b1	26 Years and 8 Months	26	8
CUS_0x21b1	26 Years and 9 Months	26	9
CUS_0x21b1	26 Years and 10 Months	26	10
CUS_0x21b1	26 Years and 11 Months	26	11
CUS_0x21b1	27 Years and 0 Months	27	0
CUS_0x21b1	27 Years and 1 Months	27	1
CUS_0x21b1	27 Years and 2 Months	27	2
CUS_0x2dbc	17 Years and 9 Months	17	9
CUS_0x2dbc	17 Years and 10 Months	17	10
CUS_0x2dbc	17 Years and 11 Months	17	11
CUS_0x2dbc	NA	NA	NA

The Year and Month values we will be handled, then the original column Credit_History_Age will be dropped, the following function will be used to impute the values of the NA's:

```
#This function to handle one pass through all missing values, it checks the ones before and
#after every missing value,Since we may have 3 NA or more in a row, we will call this
#function many time until everything is handled.
#In case of 3 NAs in a row as example, the first call of the function will handle the first
#and the last NA rows, the next call will handle the one at the middle
fix_credit_history_age_1pass <- function(dta)
{
  # find the indices of the entries which has the year as NA, whenever year is NA, month
  # will be NA as well
  nas <- which(is.na(dta$Credit_History_Age_Year))
  for(nai in unique(nas))
  {
    current <- dta[nai,] # the entry which has the NA
    before <- dta[nai-1,] # th entry before it
    after <- dta[nai+1,] # the entry after it
    #check if the one before belongs to same customer and not NA
    if(current$Customer_ID==before$Customer_ID&&!is.na(before$Credit_History_Age_Year))
    {
      # if so, make the year and month of the current entry equal to the one before plus
      # add 1 to the month
      current$Credit_History_Age_Year = before$Credit_History_Age_Year
      current$Credit_History_Age_Month = before$Credit_History_Age_Month+1  # 1 month after
      if(current$Credit_History_Age_Month>11) # fix the month year issue
      {
        current$Credit_History_Age_Year <- current$Credit_History_Age_Year+1
        current$Credit_History_Age_Month <-0
```

```
      }
    }
    #check if the one after belongs to same customer and not NA
    else if(current$Customer_ID==after$Customer_ID&&!is.na(after$Credit_History_Age_Year))
    {
      current$Credit_History_Age_Year = after$Credit_History_Age_Year
      current$Credit_History_Age_Month = after$Credit_History_Age_Month-1 # 1 month before
      if(current$Credit_History_Age_Month>11)# fix the month year issue
      {
        current$Credit_History_Age_Year <- current$Credit_History_Age_Year+1
        current$Credit_History_Age_Month <-0
      }
    }
    dta[nai,] = current;
  }
  dta # return the dataset again
}
```

The following code will use the developed function to handle NA values

```
## we will call the function many times until all NA go away,
data%>%group_by(Customer_ID)%>%filter( is.na(Credit_History_Age_Year)) %>% summarise(missing=n())
%>% filter(missing>0) %>%summarise(total_count=n())
# We have 6650 rows with NAs
## first of the function
data<- fix_credit_history_age_1pass(data)
#Check How many NA rows left?
data%>%group_by(Customer_ID)%>%filter( is.na(Credit_History_Age_Year)) %>% summarise(missing=n())
%>% filter(missing>0) %>%summarise(total_count=n())
# Now only 107 missing, another pass
data<- fix_credit_history_age_1pass(data)
data%>%group_by(Customer_ID)%>%filter( is.na(Credit_History_Age_Year)) %>% summarise(missing=n())
%>% filter(missing>0) %>%summarise(total_count=n())
# Now only 9 missing, another pass
data<- fix_credit_history_age_1pass(data)
data%>%group_by(Customer_ID)%>%filter( is.na(Credit_History_Age_Year)) %>% summarise(missing=n())
%>% filter(missing>0) %>%summarise(total_count=n())
# Now only 1 missing, another pass
data<- fix_credit_history_age_1pass(data)
data%>%group_by(Customer_ID)%>%filter( is.na(Credit_History_Age_Year)) %>% summarise(missing=n())
%>% filter(missing>0) %>%summarise(total_count=n())
# Now, 0 missing, all done
```

The following code is used to validate the results:

```
credit_history_age_validation_sample %>%
  kbl(caption = 'Credit History Age Fixed Sample')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 9: Credit History Age Fixed Sample

Customer_ID	Credit_History_Age	Credit_History_Age_Year	Credit_History_Age_Month
CUS_0xd40	22 Years and 1 Months	22	1
CUS_0xd40	NA	22	2
CUS_0xd40	22 Years and 3 Months	22	3
CUS_0xd40	22 Years and 4 Months	22	4
CUS_0xd40	22 Years and 5 Months	22	5
CUS_0xd40	22 Years and 6 Months	22	6
CUS_0xd40	22 Years and 7 Months	22	7
CUS_0xd40	NA	22	8
CUS_0x21b1	26 Years and 7 Months	26	7
CUS_0x21b1	26 Years and 8 Months	26	8
CUS_0x21b1	26 Years and 9 Months	26	9
CUS_0x21b1	26 Years and 10 Months	26	10
CUS_0x21b1	26 Years and 11 Months	26	11
CUS_0x21b1	27 Years and 0 Months	27	0
CUS_0x21b1	27 Years and 1 Months	27	1
CUS_0x21b1	27 Years and 2 Months	27	2
CUS_0x2dbc	17 Years and 9 Months	17	9
CUS_0x2dbc	17 Years and 10 Months	17	10
CUS_0x2dbc	17 Years and 11 Months	17	11
CUS_0x2dbc	NA	18	0

The original Credit_History_Age will be removed by the following code:

```
data <- data %>% select(-Credit_History_Age)
```

Implicit NA Handeling

Some entries have values that indicate a missing value, like - or —-, etc. In this section we will handle those values, in our dataset, those values appear for the columns that supposed to be either integers or numeric but they are strings, I will also examine the string columns, the following code will show a snapshot of our dataset:

```
str(data_after_removing_explcit_na)
```

```
## 'data.frame':    100000 obs. of  29 variables:
##  $ ID                      : num  5634 5635 5636 5637 5638 ...
##  $ Customer_ID             : chr  "CUS_0xd40" "CUS_0xd40" "CUS_0xd40" "CUS_0xd40" ...
##  $ Month                   : chr  "January" "February" "March" "April" ...
##  $ Name                    : chr  "Aaron Maashoh" "Aaron Maashoh" "Aaron Maashoh" "Aaron Maashoh" ..
##  $ Age                     : chr  "23" "23" "-500" "23" ...
##  $ SSN                     : chr  "821-00-0265" "821-00-0265" "821-00-0265" "821-00-0265" ...
##  $ Occupation              : chr  "Scientist" "Scientist" "Scientist" "Scientist" ...
##  $ Annual_Income           : chr  "19114.12" "19114.12" "19114.12" "19114.12" ...
##  $ Monthly_Inhand_Salary   : num  1825 1825 1825 1825 1825 ...
##  $ Num_Bank_Accounts       : int  3 3 3 3 3 3 3 3 2 2 ...
##  $ Num_Credit_Card         : int  4 4 4 4 4 4 4 4 4 4 ...
##  $ Interest_Rate           : int  3 3 3 3 3 3 3 3 6 6 ...
##  $ Num_of_Loan             : chr  "4" "4" "4" "4" ...
##  $ Type_of_Loan            : chr  "Auto Loan, Credit-Builder Loan, Personal Loan, and Home Equity Loa
##  $ Delay_from_due_date     : int  3 -1 3 5 6 8 3 3 3 7 ...
##  $ Num_of_Delayed_Payment  : chr  "7" "" "7" "4" ...
```

```
##  $ Changed_Credit_Limit    : chr  "11.27" "11.27" "_" "6.27" ...
##  $ Num_Credit_Inquiries    : num  4 4 4 4 4 4 4 4 2 2 ...
##  $ Credit_Mix              : chr  "_" "Good" "Good" "Good" ...
##  $ Outstanding_Debt        : chr  "809.98" "809.98" "809.98" "809.98" ...
##  $ Credit_Utilization_Ratio: num  26.8 31.9 28.6 31.4 24.8 ...
##  $ Payment_of_Min_Amount   : chr  "No" "No" "No" "No" ...
##  $ Total_EMI_per_month     : num  49.6 49.6 49.6 49.6 49.6 ...
##  $ Amount_invested_monthly : chr  "80.41529543900253" "118.28022162236736" "81.699521264648" "199.458
##  $ Payment_Behaviour       : chr  "High_spent_Small_value_payments" "Low_spent_Large_value_payments"
##  $ Monthly_Balance         : chr  "312.49408867943663" "284.62916249607184" "331.2098628537912" "223
##  $ Credit_Score            : chr  "Good" "Good" "Good" "Good" ...
##  $ Credit_History_Age_Year : num  22 22 22 22 22 22 22 22 26 26 ...
##  $ Credit_History_Age_Month: num  1 2 3 4 5 6 7 8 7 8 ...
```

Monthly_Inhand_Salary, Num_Bank_Accounts, Num_Credit_Card, Interest_Rate, Delay_from_due_date, Num_Credit_Inquiries, Credit_Utilization_Ratio, Total_EMI_per_month, Credit_History_Age_Year, and Credit_History_Age_Month are already fine and they don't have any implicit missing values, I will process the others one by one.

Age, Annual_Income, Num_of_Loan, and Outstanding_Debt

Displaying some non-numeric values for the Age:

```
org_data %>% filter(str_detect(Age, "^[0-9,.,-]+$")==FALSE | Age=='') %>% select(Age) %>% unique() %>% t
```

```
## Selecting by Age
```

```
## [1] Age
## <0 rows> (or 0-length row.names)
```

It seems the "_" is added to the age as human error, let's remove them and try to convert the column to integer.

```
org_data$Age<- str_replace_all(data$Age, "_", "")
# try to convert it to integer again
tmp_age <- as.integer(data$Age)
```

It works!, then make it persistent

```
data$Age <- as.integer(data$Age)
```

Some values are very big and can't be considered as valid customers age, we will handle them in the outliers section.

Same process is applied to the Annual_Income,Num_of_Loan and Outstanding_Debt to remove the extra "_" characters and convert the columns to numeric,integers and numeric respectively.

Num_of_Delayed_Payment

Displaying some non-numeric values for the Num_of_Delayed_Payment:

```
org_data %>% filter(str_detect(Num_of_Delayed_Payment, "^[0-9,.,-]+$") == FALSE | Num_of_Delayed_Payment
  select(Num_of_Delayed_Payment) %>% unique() %>% top_n(-10)
```

```
## Selecting by Num_of_Delayed_Payment
```

```
##    Num_of_Delayed_Payment
## 1
## 2                     12_
## 3                     11_
```

```
## 4                     10_
## 5                      0_
## 6                     -2_
## 7                     -1_
## 8                     -3_
## 9                   1087_
## 10                  1295_
```

The difference between this case and the columns discussed in the previous section is that this time we have empty an negative values which means they should be dealt with as NA values.

The following code will handle this column:

```
# Their is extra _ that should be removed, also there are empty strings
# and negative numbers
data$Num_of_Delayed_Payment<- str_replace_all(data$Num_of_Delayed_Payment, "_", "")
#Will assume the empty strings and the negative values to be NA, then we will impute
#them with the average number per customer
data$Num_of_Delayed_Payment<- ifelse(data$Num_of_Delayed_Payment=="",NA
                                     ,data$Num_of_Delayed_Payment)
data$Num_of_Delayed_Payment <- as.integer(data$Num_of_Delayed_Payment)
data$Num_of_Delayed_Payment<- ifelse(data$Num_of_Delayed_Payment<0 ,NA,
                                     data$Num_of_Delayed_Payment)

#now find the average per customer
mis_mod <- data %>% filter(!is.na(Num_of_Delayed_Payment)) %>% group_by(Customer_ID) %>%
  summarise(mode = mean(Num_of_Delayed_Payment))
data <- data%>%left_join(mis_mod,by='Customer_ID')%>%mutate(
  Num_of_Delayed_Payment=ifelse(is.na(Num_of_Delayed_Payment),mode,
                                Num_of_Delayed_Payment))
head(data%>%select(Customer_ID,Num_of_Delayed_Payment,mode),n=20)
#remove the mode
data <- data %>% select(-mode)
#convert numeric to integers
data$Num_of_Delayed_Payment <- as.integer(data$Num_of_Delayed_Payment)
```

Changed_Credit_Limit

Displaying some non-numeric values for the Num_of_Delayed_Payment:

```
org_data %>% filter(str_detect(Changed_Credit_Limit, "^[0-9,.,-]+$")==FALSE |
                      Changed_Credit_Limit=='') %>%
  select(Changed_Credit_Limit) %>% unique()  %>% top_n(10)
```

```
## Selecting by Changed_Credit_Limit
```

```
##   Changed_Credit_Limit
## 1                    _
```

The only special thing here are the "_" values that will be replaced by zeros as in the following code:

```
data <- data %>% mutate(Changed_Credit_Limit = ifelse(Changed_Credit_Limit=='_','0',
                                                      Changed_Credit_Limit))
data$Changed_Credit_Limit <- as.numeric(data$Changed_Credit_Limit)
```

Amount_invested_monthly

The non-numeric values are:

```
org_data %>% filter(str_detect(Amount_invested_monthly, "^[0-9,.,-]+$")==FALSE |
                      Amount_invested_monthly=='') %>%
  select(Amount_invested_monthly) %>% unique()  %>% top_n(10)
```

```
## Selecting by Amount_invested_monthly
```

```
##   Amount_invested_monthly
## 1               __10000__
## 2
```

There are two special values, the empty string and "*__10000__*" I assumed both of them to be missing and handled them accordingly using the following code:

```
## the value __10000__ seems to have a special meaning, also the empty values.
## I will change them both to the average value per customer.
data$Amount_invested_monthly <- as.numeric(data$Amount_invested_monthly)
#Now those values are NAs
mis_mod <- data %>% filter(!is.na(Amount_invested_monthly)) %>% group_by(Customer_ID) %>%
  summarise(mode = mean(Amount_invested_monthly))
data <- data%>%left_join(mis_mod,by='Customer_ID')%>%
  mutate(Amount_invested_monthly=ifelse(is.na(Amount_invested_monthly),mode,
                                        Amount_invested_monthly))
data = data%>%select(-(mode))
```

Monthly_Balance

The non-numeric values are:

```
org_data %>% filter(str_detect(Monthly_Balance, "^[0-9,.,-]+$")==FALSE |
                      Monthly_Balance=='') %>%
  select(Monthly_Balance) %>% unique()  %>% top_n(10)
```

```
## Selecting by Monthly_Balance
```

```
##                     Monthly_Balance
## 1
## 2 __-333333333333333333333333333__
```

There are two special values, the empty string and "*__-333333333333333333333333333__*" I assumed both of them to be missing and handled them accordingly using the following code:

```
# some empty values, and weird __-333333333333333333333333333__ value both will
# be considered as NA and will be imputed per Customer
data$Monthly_Balance <- as.numeric(data$Monthly_Balance)
#Now those values are NAs
mis_mod <- data %>% filter(!is.na(Monthly_Balance)) %>% group_by(Customer_ID)
%>% summarise(mode = mean(Monthly_Balance))
data <- data%>%left_join(mis_mod,by='Customer_ID')%>%
  mutate(Monthly_Balance=ifelse(is.na(Monthly_Balance),mode,Monthly_Balance))
data = data%>%select(-(mode))
```

Outliers handling

Box plots are good tools to visualize the data distribution and the outliers, I will start with the Age, the box plot for it looks like:

There is no box visible, this is an indicator of on extreme outlier, we need to reduce this outlier, the following function is used to reduce those exterme outliers:

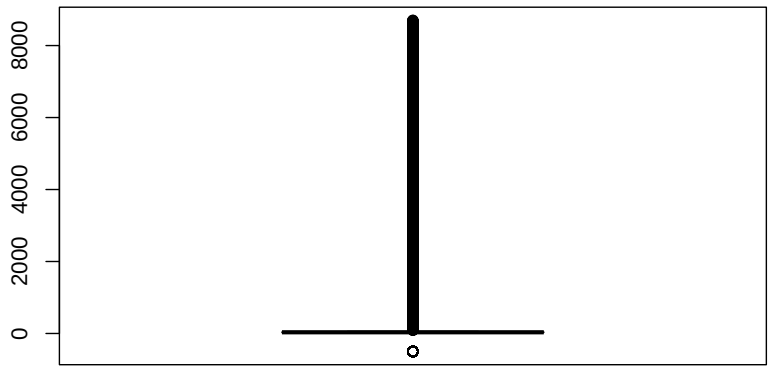


Figure 1: Original boxplot for the Age

```
#I will use an iterative approach, the value suspected to be an outlier is greater
#than X*before_value or grater than X*after_value where X is our threshold multiplier
#then this value will be the before_value or after_value
#throughout the processing I will use 2 for the value of X, if the value suspected
#to be an outlier is > 2*value before or after, then it will be replaced with that value.
#this function will be used in an iterative approch since we may have 3 or more successive
#values that are suspected to be outliers, only the ones are the corners will be handled,
#the one at the middle will be handled in the next call to the function.

#dta is the dataset, column_index is the index of the column to be fixed,
#for example Age has index of 5, X is the multiplication factor to determine
#if the value is in the exterme outlier
fix_outlier_1pass <- function(dta,column_index,X)
{
  # get all the suspected outlier values for the column at column_index using
  # the boxplot function.
  bp <- boxplot(dta[,column_index])
  # loop through all suspected values, unique is used here in order not to process
  # the same value twice
  for (suspected_value in unique(bp$out))
  {
    #find the location of the suspected outlier value for the column given at column_index
    locations = which(dta[,column_index] == suspected_value)
    for (location in locations) #Loop through all location where the suspected value found
    {
      current <- dta[location, ] # the row for value of interest
      before <- dta[location - 1, ] # the row before it
      after <- dta[location + 1, ] # the row after
      # if the previous record for the same customer and the value for this record
      #is more than X time the previous, then this is an extreme outlier
      if (abs(current[,column_index]) > abs(before[,column_index]) * X &&
```

```
          current$Customer_ID == before$Customer_ID) # replaced with the one before
      {
        current[,column_index] = before[,column_index]
      }
      # if the next record for the same customer and the value for this record is
      # more than X time the next, then this is an extreme outlier
      else if (abs(current[,column_index]) > abs(after[,column_index]) * X &&
               current$Customer_ID == after$Customer_ID)
      {
        current[,column_index] = after[,column_index]# replaced with the one after
      }
      dta[location, ] = current # update the row
    }
  }
  dta # return the updated dataset
}
```

To handle the outliers for any column, the function need to be called iteratively, after each call the box plot will be ploted to see the effect of the function call, once an acceptable box plot achieved then we are done, sometime outliers will not go out since mostly they belongs to the same customer as our function to handle the outliers is per customer basis.

The Age column is located at index 5, the following code showing the process of handling it's outliers:

```
boxplot(data$Age)
age_bp1 <- recordPlot()

data <- fix_outlier_1pass(data,5,2)
boxplot(data$Age)
age_bp2 <- recordPlot()

data <- fix_outlier_1pass(data,5,2)
boxplot(data$Age)
age_bp3 <- recordPlot()

data <- fix_outlier_1pass(data,5,2)
boxplot(data$Age)
age_bp4 <- recordPlot()

data <- fix_outlier_1pass(data,5,2)
boxplot(data$Age)
age_bp5 <- recordPlot()
```

The Box plots are shown below:

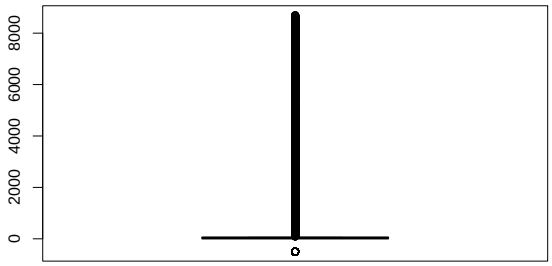
The outliers for all continuous features are handled in the same way, the full code is available in the accompanied R file, the original and fixed BoxPlots for all of them are given in Appendix-A.

Strings to integers encoding

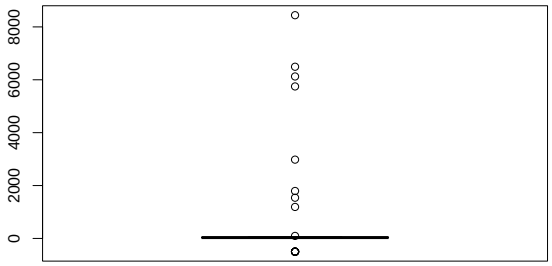
In this section the string-based feautres will be cleaned and encoded to integers, they will be handled one by one as the following:

Month

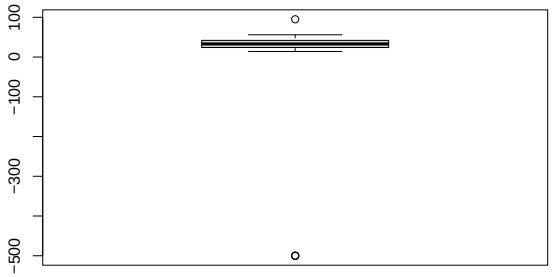
The unique values for the month are given below:



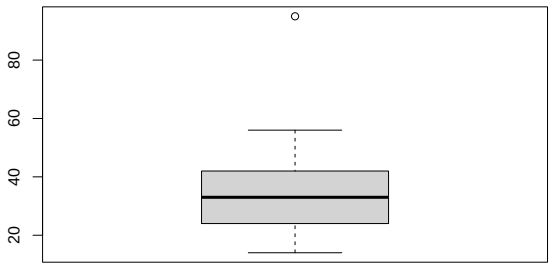
(a) Original Box Plot



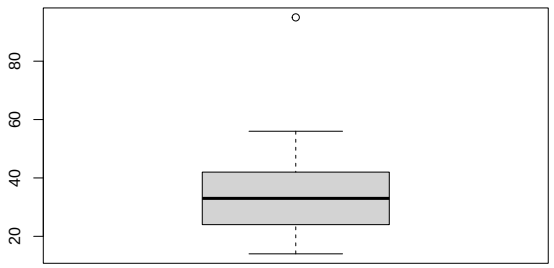
(b) Box Plot after the first call



(c) Box Plot after the second call



(d) Box Plot after the third call



(e) Box Plot after the last call

Figure 2: Handling outliers for the Age

```
unique(data_after_outlier_na$Month)
```

```
## [1] "January"  "February" "March"    "April"    "May"      "June"     "July"     "August"
```

All good, the following code with encode the strings to integers:

```
data <- data%>%mutate(Month=case_when(
  Month == "January"  ~1,
  Month == "February" ~2,
  Month == "March"    ~3,
  Month == "April"    ~4,
  Month == "May"      ~5,
  Month == "June"     ~6,
  Month == "July"     ~7,
  Month == "August"   ~8
))
```

Occupation

The unique values for the month are given below:

```
unique(data_after_outlier_na$Occupation)
```

```
##  [1] "Scientist"     "_______"       "Teacher"       "Engineer"      "Entrepreneur"  "Developer"
##  [7] "Lawyer"        "Media_Manager" "Doctor"        "Journalist"    "Manager"       "Accountant"
## [13] "Musician"      "Mechanic"      "Writer"        "Architect"
```

Let's see how the "_____" looks like in the wild:

```
head(data_after_outlier_na%>%select(Customer_ID,Occupation),n=12)%>%
  kbl(caption = 'Occupation samples')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 10: Occupation samples

Customer_ID	Occupation
CUS_0xd40	Scientist
CUS_0xd40	Scientist
CUS_0xd40	Scientist
CUS_0xd40	Scientist
CUS_0xd40	Scientist
CUS_0xd40	Scientist
CUS_0xd40	Scientist
CUS_0xd40	Scientist
CUS_0x21b1	_____
CUS_0x21b1	Teacher
CUS_0x21b1	Teacher
CUS_0x21b1	Teacher

Occupation will not change frequently for the same customer from month to month, the -—— value indicates a not entered value, for the same customer it will be the same occupation as previous entry.

The following code will handle this misisng value in the Occupation:

```
# I will start by converting it to NA to make the further processing easier
data<- data%>% mutate(Occupation=ifelse(Occupation=="_______",NA,Occupation))
```

```r
fixOcuupation_1pass <- function(dta)
{
  nas <- unique(which(is.na(dta$Occupation)))
  for(nai in nas)
  {
    current <- dta[nai,]
    before <- dta[nai-1,]
    after <- dta[nai+1,]
    #check if the one before belongs to same customer and not na
    if(current$Customer_ID==before$Customer_ID&&!is.na(before$Occupation))
    {
      current$Occupation = before$Occupation
    }
    else if(current$Customer_ID==after$Customer_ID&&!is.na(after$Occupation))
    {
      current$Occupation = after$Occupation
    }
    dta[nai,] = current;
  }
  dta
}
data %>% filter(is.na(Occupation))%>%summarise(missing=n())
#7062 cases
data <- fixOcuupation_1pass(data)
data %>% filter(is.na(Occupation))%>%summarise(missing=n())
#Now 70 only, one more pass
data <- fixOcuupation_1pass(data)
data %>% filter(is.na(Occupation))%>%summarise(missing=n())
#Now 5 only, one more pass
data <- fixOcuupation_1pass(data)
data %>% filter(is.na(Occupation))%>%summarise(missing=n())
#Now 0, all done


#Changing the occupation to integers
data <- data%>%mutate(Occupation=case_when(
  Occupation == "Scientist" ~1,
  Occupation == "Teacher" ~2,
  Occupation == "Engineer" ~3,
  Occupation == "Entrepreneur" ~4,
  Occupation == "Developer" ~5,
  Occupation == "Lawyer" ~6,
  Occupation == "Media_Manager" ~7,
  Occupation == "Doctor" ~8,
  Occupation == "Journalist" ~9,
  Occupation == "Manager" ~10,
  Occupation == "Accountant" ~11,
  Occupation == "Musician" ~12,
  Occupation == "Mechanic" ~13,
  Occupation == "Writer" ~14,
  Occupation == "Architect" ~15
))
```

Type_of_Loan

The following code is used to display 10 unique values for the Type_of_Loan.

```
data_after_outlier_na %>% select(Type_of_Loan) %>% unique() %>% top_n(10)%>%
  kbl(caption = 'Type of Loan Samples')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))%>%
  column_spec(1,width = "16cm")
```

```
## Selecting by Type_of_Loan
```

Table 11: Type of Loan Samples

Type_of_Loan
Student Loan, Student Loan, Student Loan, Debt Consolidation Loan, Home Equity Loan, Debt Consolidation Loan, and Payday Loan
Student Loan, Student Loan, Personal Loan, Personal Loan, Mortgage Loan, and Not Specified
Student Loan, Student Loan, Personal Loan, Student Loan, Debt Consolidation Loan, Home Equity Loan, and Debt Consolidation Loan
Student Loan, Student Loan, Student Loan, and Auto Loan
Student Loan, Student Loan, Student Loan, Home Equity Loan, Debt Consolidation Loan, Not Specified, Auto Loan, and Home Equity Loan
Student Loan, Student Loan, Student Loan, Personal Loan, Mortgage Loan, and Student Loan
Student Loan, Student Loan, Student Loan, and Home Equity Loan
Student Loan, Student Loan, Personal Loan, Mortgage Loan, Auto Loan, Debt Consolidation Loan, Not Specified, Student Loan, and Debt Consolidation Loan
Student Loan, Student Loan, Personal Loan, Payday Loan, Personal Loan, Payday Loan, and Student Loan
Student Loan, Student Loan, Student Loan, Personal Loan, Personal Loan, Mortgage Loan, Mortgage Loan, and Credit-Builder Loan

Looking at the loans type, the following are the individual loan types: Credit-Builder Loan,Auto Loan,Not Specified,Mortgage Loan,Student Loan,Personal Loan,Debt Consolidation Loan,Payday Loan, and Home Equity Loan

The following code is used to convert this column into one-hot-coding columns for the individual loan types:

```
data <- data %>% mutate(
  Type_of_Loan.Credit_Builder = str_detect(Type_of_Loan,"Credit-Builder"),
  Type_of_Loan.Auto = str_detect(Type_of_Loan,"Auto Loan"),
  Type_of_Loan.Not_Specified = str_detect(Type_of_Loan,"Not Specified"),
  Type_of_Loan.Mortgage = str_detect(Type_of_Loan,"Mortgage Loan"),
  Type_of_Loan.Student = str_detect(Type_of_Loan,"Student Loan"),
  Type_of_Loan.Personal = str_detect(Type_of_Loan,"Personal Loan"),
  Type_of_Loan.Debt_Consolidation = str_detect(Type_of_Loan,"Debt Consolidation Loan"),
  Type_of_Loan.Payday = str_detect(Type_of_Loan,"Payday Loan"),
  Type_of_Loan.Home_Equity = str_detect(Type_of_Loan,"Home Equity Loan"),
)
data <- data %>% select(-Type_of_Loan)
```

Now, they will look like:

```
data %>% select(Type_of_Loan.Credit_Builder,Type_of_Loan.Auto,Type_of_Loan.Not_Specified,
                Type_of_Loan.Mortgage,Type_of_Loan.Student,Type_of_Loan.Personal,
                Type_of_Loan.Debt_Consolidation,Type_of_Loan.Payday,
                Type_of_Loan.Home_Equity) %>% sample_n(10)%>%
```

```
  rename('Credit Builder' = Type_of_Loan.Credit_Builder,
         'Auto' = Type_of_Loan.Auto,
         'Not Specified' = Type_of_Loan.Not_Specified,
         'Mortgage' = Type_of_Loan.Mortgage,
         'Student' = Type_of_Loan.Student,
         'Personal' = Type_of_Loan.Personal,
         'Debt Consolidation' = Type_of_Loan.Debt_Consolidation,
         'Payday' = Type_of_Loan.Payday,
         'Home Equity' = Type_of_Loan.Home_Equity,
        ) %>%
  kbl(caption = 'Type of Loan Modified')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))%>%
  column_spec(1:9,width = "1.5cm")
```

Table 12: Type of Loan Modified

Credit Builder	Auto	Not Specified	Mortgage	Student	Personal	Debt Consolidation	Payday	Home Equity
FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	FALSE	FALSE
FALSE	TRUE	TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE
FALSE	FALSE	TRUE	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
FALSE	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE	TRUE	FALSE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

Credit_Mix

The following code will display the unique values for this column and their percentages:

```
data_after_outlier_na %>% group_by(Credit_Mix) %>%
  summarise(Percentage = n()/length(data$Credit_Mix)*100)%>%
  kbl(caption = 'Credit Mix values percentages')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 13: Credit Mix values percentages

Credit_Mix	Percentage
Bad	18.989
Good	24.337
Standard	36.479
_	20.195

The extra - value taking about 20% of our dataset, it is too big to be imputed as other value, It seems to have a spacial meaning which is “Unknown” so it will be kept, the following code will encode those strings as integers:

```
#Now change the classes into numbers: -: 0, Bad: 1, Standard: 2: Good 3:
data <- data %>% mutate(Credit_Mix = case_when(
```

```
  Credit_Mix == '_' ~ 0,
  Credit_Mix == 'Bad' ~ 1,
  Credit_Mix == 'Standard' ~ 2,
  Credit_Mix == 'Good' ~ 3
))
```

Payment_Behaviour

Same as the Credit_Mix, the unique values and their percentages are displayed below:

```
data_after_outlier_na %>% group_by(Payment_Behaviour) %>%
  summarise(Percentage = n()/length(data$Payment_Behaviour)*100)%>%
  kbl(caption = 'Payment Behaviour values percentages')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 14: Payment Behaviour values percentages

Payment_Behaviour	Percentage
!@9#%8	7.600
High_spent_Large_value_payments	13.721
High_spent_Medium_value_payments	17.540
High_spent_Small_value_payments	11.340
Low_spent_Large_value_payments	10.425
Low_spent_Medium_value_payments	13.861
Low_spent_Small_value_payments	25.513

The weird value "!@9#%8" compromising around 8% of the dataset, it will be kept as it is since it can't be imputed to other value, the following code will encode those strings as integers:

```
data <- data %>% mutate(Payment_Behaviour = case_when(
  Payment_Behaviour == '!@9#%8' ~ 0,
  Payment_Behaviour == 'High_spent_Large_value_payments' ~ 1,
  Payment_Behaviour == 'High_spent_Medium_value_payments' ~ 2,
  Payment_Behaviour == 'High_spent_Small_value_payments' ~ 3,
  Payment_Behaviour == 'Low_spent_Large_value_payments' ~ 4,
  Payment_Behaviour == 'Low_spent_Medium_value_payments' ~ 5,
  Payment_Behaviour == 'Low_spent_Small_value_payments' ~ 6,
))
```

Credit_Score

Credit_Score is the target outcome, the unique values for it are as follows:

```
data_after_outlier_na %>% group_by(Credit_Score) %>%
  summarise(Percentage = n()/length(data$Credit_Score)*100)%>%
  kbl(caption = 'Credit Score values percentages')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 15: Credit Score values percentages

Credit_Score	Percentage
Good	17.828
Poor	28.998
Standard	53.174

The following code will encode those strings values into integers:

```
#Replace with integer: 0:Poor, 1:Standard, 2:Good
data <- data %>% mutate(Credit_Score = case_when(
  Credit_Score == 'Poor' ~ 0,
  Credit_Score == 'Standard' ~ 1,
  Credit_Score == 'Good' ~ 2
))
```

Dataset final touches and splitting

Some features we have are categorical and for out machine learning models to work with they must be converted into factors, the following will take care of this:

```
# The original dataset is kept as it is, but we took a copy.
data_copy <- data%>%mutate(
  Month = as.factor(Month),
  Occupation = as.factor(Occupation),
  Credit_Mix = as.factor(Credit_Mix),
  Payment_of_Min_Amount = as.factor(Payment_of_Min_Amount),
  Payment_Behaviour = as.factor(Payment_Behaviour),
  Credit_History_Age_Year = as.factor(Credit_History_Age_Year),
  Credit_History_Age_Month = as.factor(Credit_History_Age_Month),
  Type_of_Loan.Credit_Builder = as.factor(Type_of_Loan.Credit_Builder),
  Type_of_Loan.Auto = as.factor(Type_of_Loan.Credit_Builder),
  Type_of_Loan.Not_Specified = as.factor(Type_of_Loan.Not_Specified),
  Type_of_Loan.Mortgage = as.factor(Type_of_Loan.Mortgage),
  Type_of_Loan.Student = as.factor(Type_of_Loan.Student),
  Type_of_Loan.Personal = as.factor(Type_of_Loan.Personal),
  Type_of_Loan.Debt_Consolidation = as.factor(Type_of_Loan.Debt_Consolidation),
  Type_of_Loan.Payday = as.factor(Type_of_Loan.Payday),
  Type_of_Loan.Home_Equity = as.factor(Type_of_Loan.Home_Equity),
  Credit_Score = as.factor(data_copy$Credit_Score)
)
```

Also some features shouldn't be used in any model since they clearly identify the customers or the data entry, those will be dropped by the following code:

```
data_copy = data_copy%>% select(-Customer_ID)
data_copy = data_copy%>% select(-Name)
data_copy = data_copy%>% select(-SSN)
data_copy = data_copy%>% select(-ID)
```

The dataset will then be randomly spitted into 80% training set and 20% testing set as in the following code:

```
set.seed(1, sample.kind="Rounding") # for backward compatability with old versions of R
test_index <- createDataPartition(y = data_copy$Credit_Score, times = 1, p = 0.2, list = FALSE)
dataset_train <- data_copy[-test_index,]
dataset_test <- data_copy[test_index,]
```

Features Selection and Exploratory Data Analysis

When the number of features is big, features selection will be an important step in building machine learning models, there are generally three ways for features selection [3]:

- Filter method: in which the selection will be based on the features information like the correlation

between the features, this method is fast but it is not able to detect the possible interactions between the feature.
- Wrapper method: in which subsets of the variables are evaluated to find the best one, unlike the Filter method this method is able to detect the possible interactions between the features and it is slow.
- Embedded method: in which it combine the benefits of the previously mentioned two methods by combining the selections with the training at the same time, it will require the use of more advance models.

In this project I will use the Wrapper method to select the features, the *rfe* [4] function from the *caret* package will be used as the features selection algorithm.

RFE stands for recursive feature elimination that implements backwards selection of predictors based on predictor importance ranking. The predictors are ranked and the less important ones are sequentially eliminated prior to modeling [4], the main goal is to get the subset of features that produce the best accuracy.

The following code is used to select the best features:

```
rfe.start_time <- Sys.time() # record the starting time
# I will used parallel processing, choosing the number of cores is dependent on
# the number of actual CPU core and the amount of RAM, this process will require
# around 4GB of ram for every core, I have 16 cores CPU, 64GB RAM but I choosed the value 14
# to have the memory consumption around 56GB  since I don't want my computer to freeze.
# since I am using 10-folds cross-validations, only 10 cores were utilized in this function
# which resulted in 44GB (10+1management=11*4) memory utilization.
cl <- makePSOCKcluster(14) # to define the parallel computing cluster
registerDoParallel(cl)# start the cluster
# I used rfFunctions to be able to capture the non-linear relatations, 10 folds cross-valdiation
control <- rfeControl(functions=rfFuncs, method="cv", number=10,verbose = TRUE)
# 22 is the index of the target (Credit_Score), I choosed the sizes 4,8,16,24,and 32. the default
# values are 4,8,16 but I added 24 and 32 to them
ref_result<- rfe(dataset_train[,-22], dataset_train[,22], sizes=c(4,8,16,24,32), rfeControl=control)
rfe.end_time <- Sys.time() # record the ending time
stopCluster(cl) # stopping the cluster, very important, if not done, computer reboot will be required.
```

The process took:

```
rfe.end_time-rfe.start_time
```

```
## Time difference of 1.295917 hours
```

The output of the selection process is:

```
ref_result
```

```
##
## Recursive feature selection
##
## Outer resampling method: Cross-Validated (10 fold)
##
## Resampling performance over subset size:
##
##  Variables Accuracy  Kappa AccuracySD  KappaSD Selected
##          4   0.7205 0.5261   0.005114 0.008250
##          8   0.7827 0.6362   0.007364 0.013507
##         16   0.8169 0.6964   0.003635 0.006077        *
##         24   0.8131 0.6893   0.004179 0.006963
##         32   0.8033 0.6735   0.004170 0.006713
##
```

```
## The top 5 variables (out of 16):
##    Occupation, Credit_History_Age_Year, Delay_from_due_date, Num_Credit_Card, Changed_Credit_Limit
```

The best combination of features produced the best accuracy was 16, and those 16 features are:

```
ref_result[['optVariables']]
```

```
##  [1] "Occupation"              "Credit_History_Age_Year" "Delay_from_due_date"
##  [4] "Num_Credit_Card"         "Changed_Credit_Limit"    "Total_EMI_per_month"
##  [7] "Age"                     "Monthly_Inhand_Salary"   "Num_of_Delayed_Payment"
## [10] "Outstanding_Debt"        "Interest_Rate"           "Credit_Mix"
## [13] "Annual_Income"           "Month"                   "Num_Bank_Accounts"
## [16] "Monthly_Balance"
```

The correlation between the selected variables is shown below:

```
corrplot(cor(data[,ref_result[['optVariables']]]),tl.cex = 0.7,type = 'lower',method = 'square')
```

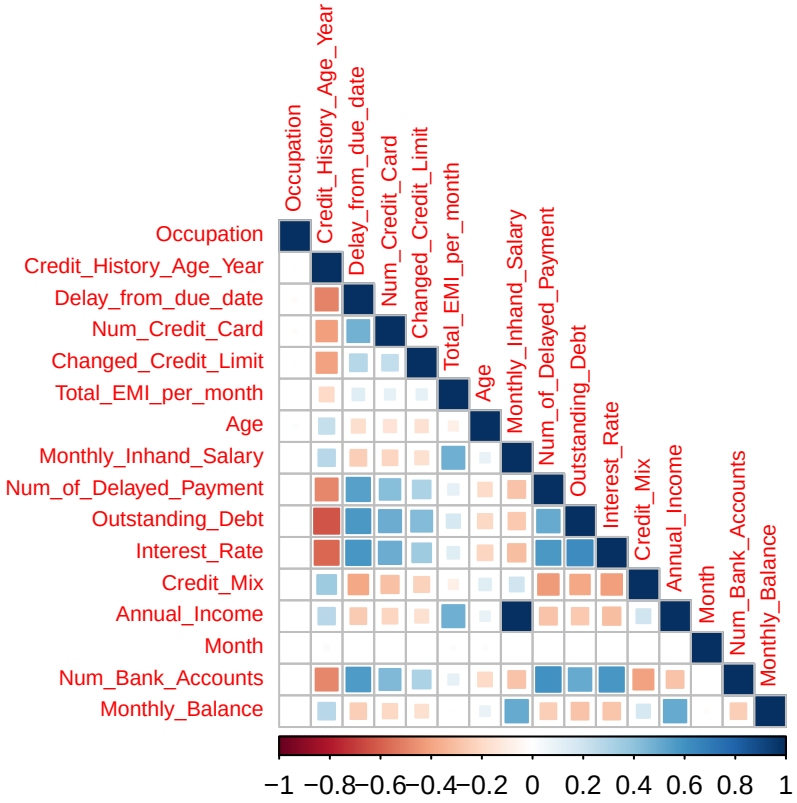


Figure 3: Correlation between the selected features

Based on this correlation plot, I have the following notes:

- Annual_Income and Monthly_inhand_Salary are highly positively correlated, I will remove one if I am going to build linear model only, but since it was chosen by the *rfe* algorithm I will keep it.
- For people who has long credit history (high value for Credit_History_Age_Year) the have the following common among them:
  - They have very low Outstandind_Debt, means they are paying their bills on time.
  - They have very low Num_of_Delayed_Payment, which supports the first previous note.

  - They have very low Interset_Rate, low number of delayed payment means low interest rate in most banks.
  - They have low number of bank account and credit cards, it is interesting.
- The notes for the people who has long credit history are completed reversed for the people who keep delaying the payment or who has large number of credit cards.
- It seems there is a relation between the number of bank account and the financial status, the more bank account the person has, the more number of delayed payment who has and the higher the interest rate and outstanding debit, is the number of bank accounts resulting on those consequences or the people try to resolve those consequences by having more bank accounts which will drag them more into those problems!

Models Buidling

In this section models will be built to predict the Credit_Score based on the selected 16 features, the *dataset_train* will be used in building the models while the *dataset_test* will only be used in testing them. I will use *caret* pacakge to build and test the machine learning models, the following models will be used:

- Generalized Additive Model *gam*
- k-Nearest Neighbors *knn*
- Classification trees *rpart*
- Random Forests *rf*

To make building the models easier I created a copy of the dataset that only inlcuded the selected features, this dataset is the one to be used in all models, the code to create it is given below:

```
rfe_dataset_train = dataset_train[,ref_result[['optVariables']]]
rfe_dataset_test = dataset_test[,ref_result[['optVariables']]]
rfe_dataset_train$Credit_Score = dataset_train$Credit_Score
rfe_dataset_test$Credit_Score = dataset_test$Credit_Score
```

To control the models training process and to increase the robustness of the models, I used 5-folds cross validation, the hyperparameters will be searched using *grid* search that will try them all, the following code will create the training control object that will be used in all models:

```
control <- trainControl(method='cv',
                        number=5,
                        search='grid')
```

I also developed the following function to calculate the testing accuracy for any model:

```
test_accuracy <- function(model,dst)
{
  pred <- predict(model,dst);
  mean(pred==dst$Credit_Score)
}
```

GAM model

I started the model training usign the default tuning parameters in the *caret* package as in the following code:

```
# Create the parallel cluster definition
cl <- makePSOCKcluster(14)
# Start the parallel cluster
registerDoParallel(cl)
# Record the starting time
gam_rfe16.start.time <- Sys.time()
gam_rfe16 <- train(pred_exp16,
```

```
                  data=dataset_train,
                  method='gam',
                  metric='Accuracy',
                  trControl = control
)
# Record the end time
gam_rfe16_.end.time <- Sys.time()
print(gam_rfe16)
# Stopping the parallel cluster
stopCluster(cl)
# Find the accuracy on the testing dataset
gam_rfe16_test_acc <- test_accuracy(gam_rfe16,rfe_dataset_test)
```

The training time for this model was:

```
gam_rfe16_.end.time-gam_rfe16.start.time
```

```
## Time difference of 54.71091 mins
```

The hyper parameter tuning plot is show below:

```
plot(gam_rfe16)
```

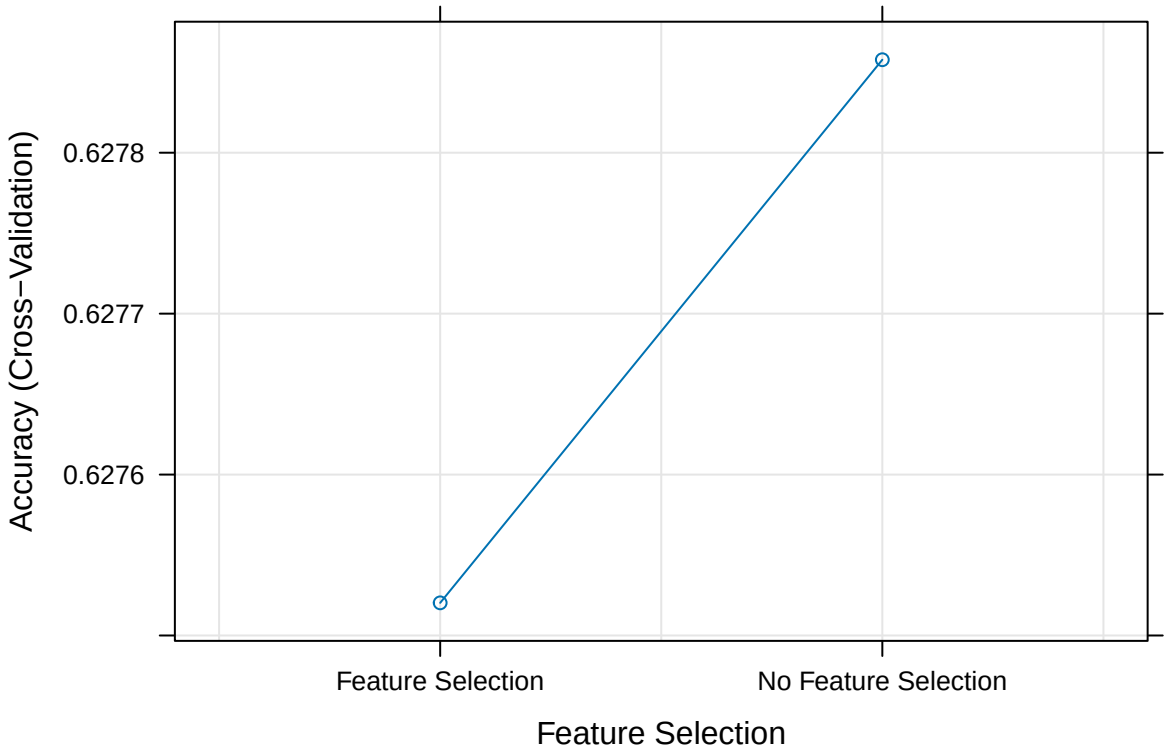


Figure 4: Default hyper parameter tuning plot for the GAM model

The following code showing the best cross-validation and testing accuracy achieved:

```r
# cat is used here to print the new lines, otherwise, they will be printed as raw \n
cat(sprintf("The best cross-validation accuracy is %1.3f and the testing accuracy is %1.3f",
        gam_rfe16[["results"]][["Accuracy"]][1],gam_rfe16_test_acc))
```

```
## The best cross-validation accuracy is 0.628 and the testing accuracy is 0.629
```

KNN model

I started the model training using the default tuning parameters in the *caret* package as in the following code:

```r
# Create the parallel cluster definition
cl <- makePSOCKcluster(14)
# Start the parallel cluster
registerDoParallel(cl)
# Record the starting time
knn_rfe16.start.time <- Sys.time()

knn_rfe16 <- train(Credit_Score~.,
                           data=rfe_dataset_train,
                           method='knn',
                           metric='Accuracy',
                           trControl = control
)
# Record the end time
knn_rfe16.end.time <- Sys.time()
# Stopping the parallel cluster
stopCluster(cl)
# Find the accuracy on the testing dataset
knn_rfe16_test_acc <- test_accuracy(knn_rfe16,rfe_dataset_test)
```

The training time for this model was:

```r
knn_rfe16.end.time-knn_rfe16.start.time
```

```
## Time difference of 7.860205 mins
```

The hyper parameter tuning plot is show below:

```r
plot(knn_rfe16)
```

The highest cross-validation accuracy achieved using the first value of k which has the value of 5, let's expand the search by adding few more k that are less than 4 as in the following code:

```r
# let's try k= 3,4 as well
cl <- makePSOCKcluster(14)
registerDoParallel(cl)
knnv2_rfe16.start.time <- Sys.time()

knnv2_rfe16 <- train(Credit_Score~.,
                     data=rfe_dataset_train,
                     method='knn',
                     metric='Accuracy',
                     trControl = control,
                     tuneGrid = data.frame(k = c(3,4,5,7,9))
)
knnv2_rfe16.end.time <- Sys.time()
knnv2_rfe16_test_acc <- test_accuracy(knnv2_rfe16,rfe_dataset_test)
```

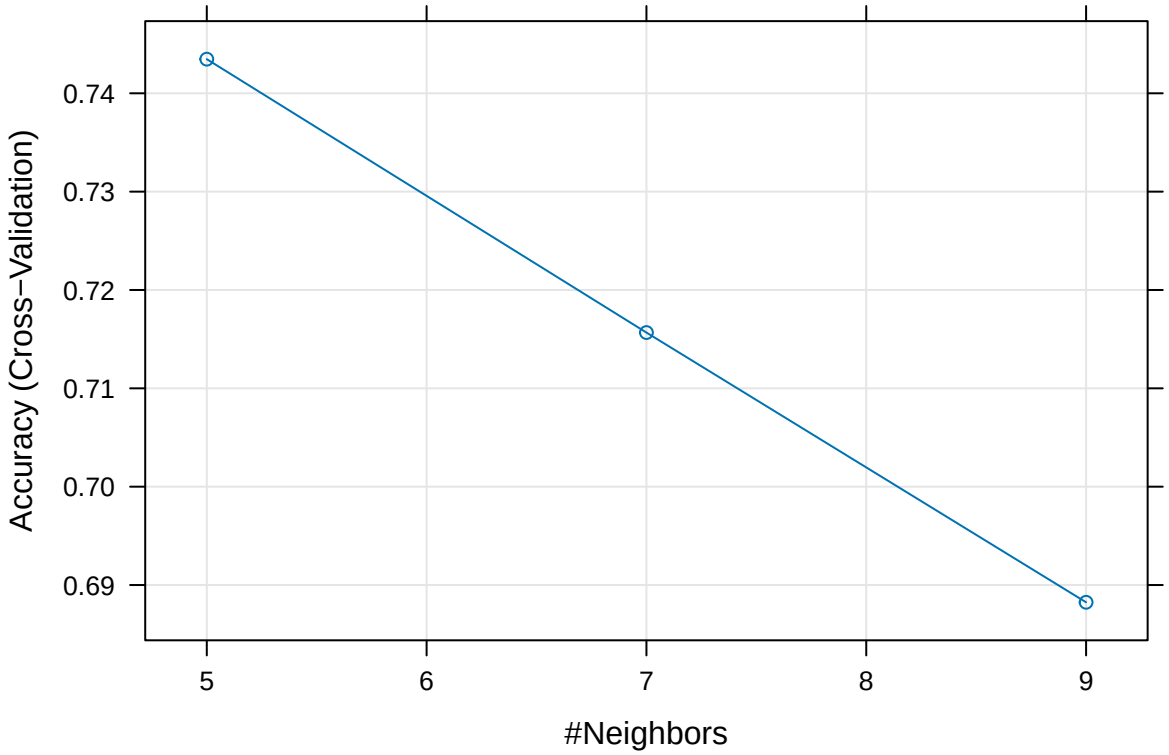


Figure 5: Default hyper parameter tuning plot for the KNN model

The training time for this model was:

```
knnv2_rfe16.end.time-knnv2_rfe16.start.time
```

```
## Time difference of 11.62915 mins
```

The hyper parameter tuning plot is show below:

```
plot(knnv2_rfe16)
```

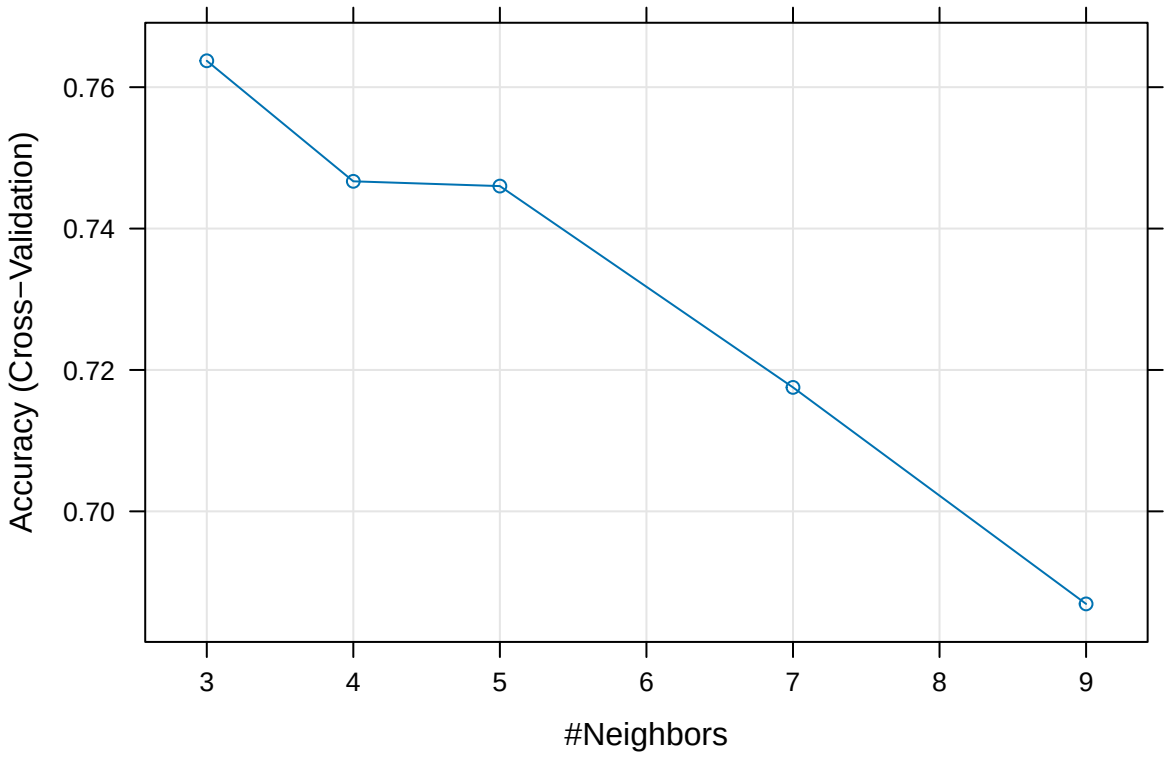


Figure 6: Modified hyper parameter tuning plot for the KNN model

The following code showing the best cross-validation and testing accuracies achieved and the k used to get it:

```
# cat is used here to print the new lines, otherwise, they will be printed as raw \n
cat(sprintf("The best cross-validation accuracy is %1.3f which was obtained using the
k value of %d the testing accuracy is %1.3f",
        knnv2_rfe16[["results"]][["Accuracy"]][1],knnv2_rfe16[["results"]][["k"]][1],
        knnv2_rfe16_test_acc))
```

```
## The best cross-validation accuracy is 0.764 which was obtained using the
## k value of 3 the testing accuracy is 0.767
```

RPART model

Same as previous models, I started the model training using the default tuning parameters in the *caret* package as in the following code:

```
# Create the parallel cluster definition
cl <- makePSOCKcluster(14)
# Start the parallel cluster
registerDoParallel(cl)
# Record the starting time
rpart_rfe16.start.time <- Sys.time()
rpart_rfe16 <- train(pred_exp16,
                            data=dataset_train,
                            method='rpart',
                            metric='Accuracy',
                            trControl = control
)
# Record the end time
rpart_rfe16_.end.time <- Sys.time()
# Stopping the parallel cluster
stopCluster(cl)
# Find the accuracy on the testing dataset
rpart_rfe16_test_acc <- test_accuracy(rpart_rfe16,rfe_dataset_test)
```

The training time for this model was:

```
rpart_rfe16_.end.time-rpart_rfe16.start.time
```

```
## Time difference of 28.46967 secs
```

The hyper parameter tuning plot is show below:

```
plot(rpart_rfe16)
```

The highest cross-validation accuracy achieved using the first value of *cp* , let's expand the search by adding few more k that are less than this value as in the following code:

```
cl <- makePSOCKcluster(14)
registerDoParallel(cl)
rpartv2_rfe16.start.time <- Sys.time()
rpartv2_rfe16 <- train(Credit_Score~.,
                       data=rfe_dataset_train,
                       method='rpart',
                       metric='Accuracy',
                       trControl = control,
                       tuneGrid = data.frame(cp = c(0.0005,0.001,0.005,0.01,0.012,0.015,
                                                    0.016,0.017,0.01748532,0.018,0.02,0.03,0.05,
                                                    0.07,0.1,0.15696743,0.17,0.2))
)
rpartv2_rfe16.end.time <- Sys.time()
stopCluster(cl)
rpartv2_rfe16_test_acc <- test_accuracy(rpartv2_rfe16,rfe_dataset_test)
```

The training time for this model was:

```
rpartv2_rfe16.end.time-rpartv2_rfe16.start.time
```

```
## Time difference of 11.36798 secs
```

The hyper parameter tuning plot is show below:

```
plot(rpartv2_rfe16)
```

The following code showing the best cross-validation and testing accuracies achieved and the *cp* used to get it:

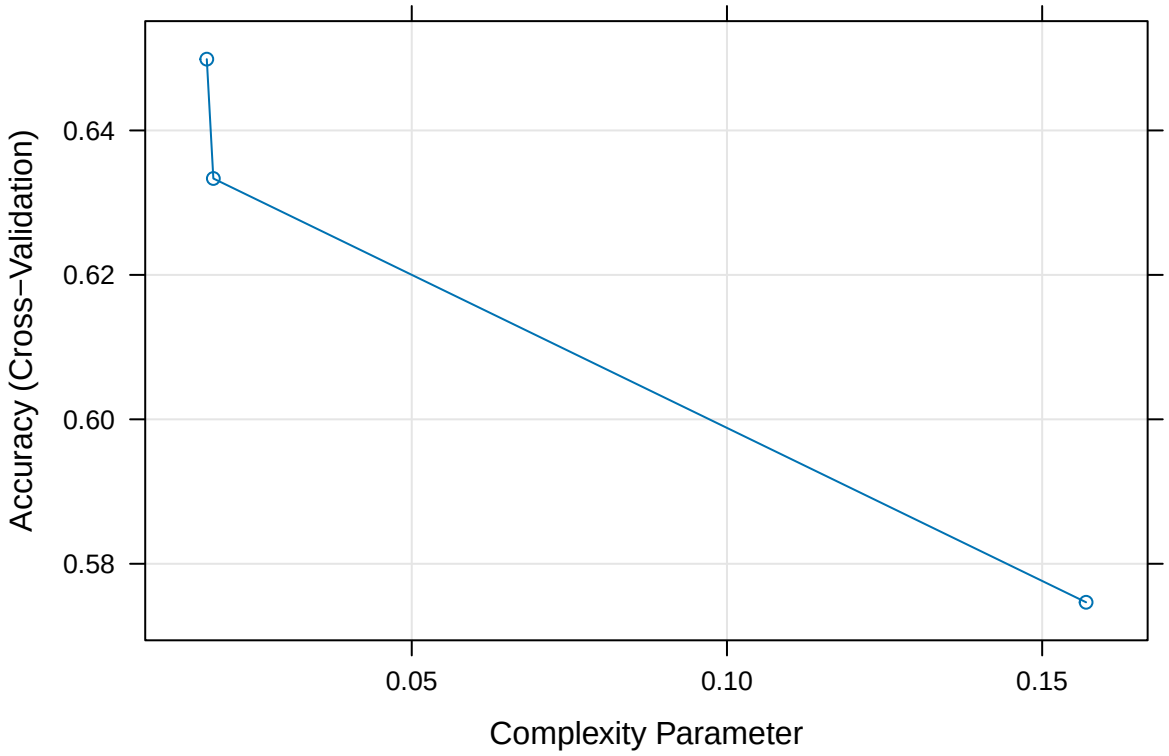


Figure 7: Default hyper parameter tuning plot for the RPART model

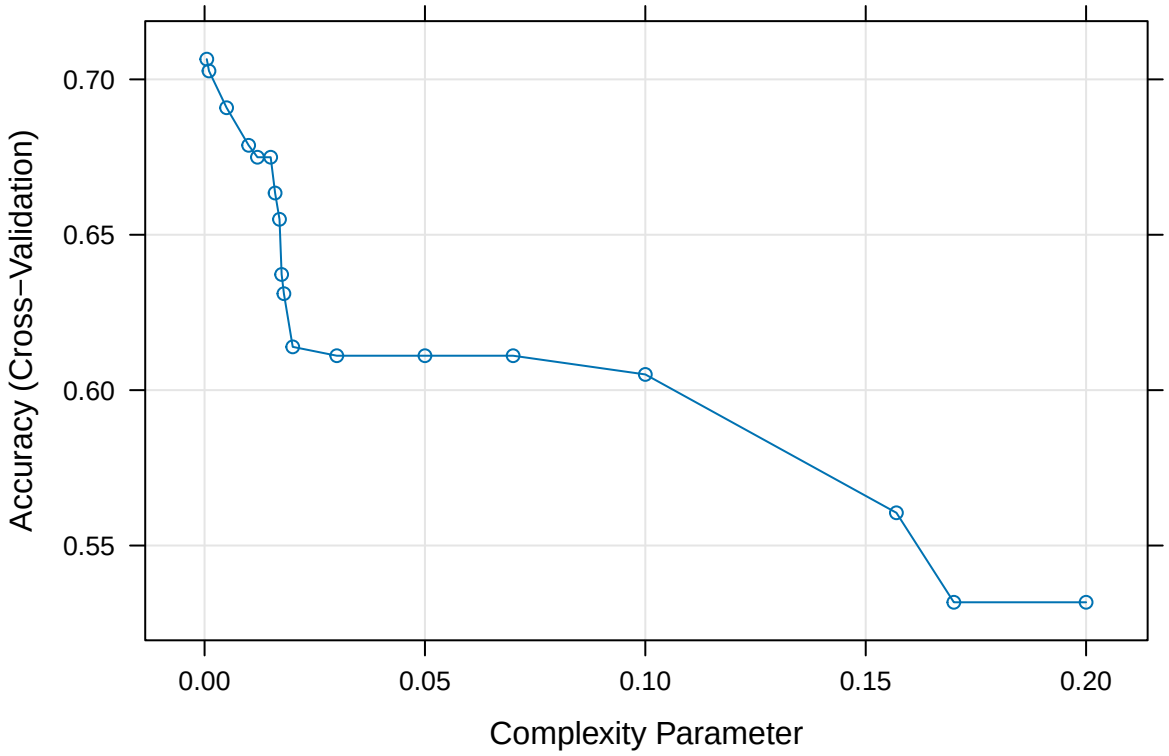


Figure 8: Modified hyper parameter tuning plot for the RPART model

```
# cat is used here to print the new lines, otherwise, they will be printed as raw \n
cat(sprintf("The best cross-validation accuracy is %1.3f which was obtained using the
cp value of %1.5f the testing accuracy is %1.3f",
        rpartv2_rfe16[["results"]][["Accuracy"]][1],rpartv2_rfe16[["results"]][["cp"]][1],
        rpartv2_rfe16_test_acc))
```

```
## The best cross-validation accuracy is 0.707 which was obtained using the
## cp value of 0.00050 the testing accuracy is 0.703
```

RF model

I will use the same approch for the RF model, I will start with the default values for the model hyper parameters, then I will try to expand my search, the basic model is shown below:

```
# Create the parallel cluster definition
cl <- makePSOCKcluster(14)
# Start the parallel cluster
registerDoParallel(cl)
# Record the starting time
rf_rfe16.start.time <- Sys.time()
rf_rfe16 <- train(pred_exp16,
                    data=dataset_train,
                    method='rf',
                    metric='Accuracy',
                    trControl = control
)
# Record the end time
rf_rfe16_.end.time <- Sys.time()
print(rf_rfe16)
# Stopping the parallel cluster
stopCluster(cl)
# Find the accuracy on the testing dataset
rf_rfe16_test_acc <- test_accuracy(rf_rfe16,rfe_dataset_test)
```

The training time for this model was:

```
rf_rfe16_.end.time-rf_rfe16.start.time
```

```
## Time difference of 19.23752 mins
```

The hyper parameter tuning plot is show below:

```
plot(rf_rfe16)
```

The highest cross-validation accuracy achieved using the value of 35 for *mtry* , let's expand the search by adding few more and few after as in the following code:

```
# I used a cluster size of 10 cores, since I faced memory issues for 14 cores.
cl <- makePSOCKcluster(10)
registerDoParallel(cl)
rfv2_rfe16.start.time <- Sys.time()
rfv2_rfe16 <- train(pred_exp16,
                  data=dataset_train,
                  method='rf',
                  metric='Accuracy',
                  trControl = control,
                  tuneGrid = data.frame(mtry = seq(30,40,2)) # from 30 to 40 step 2
```

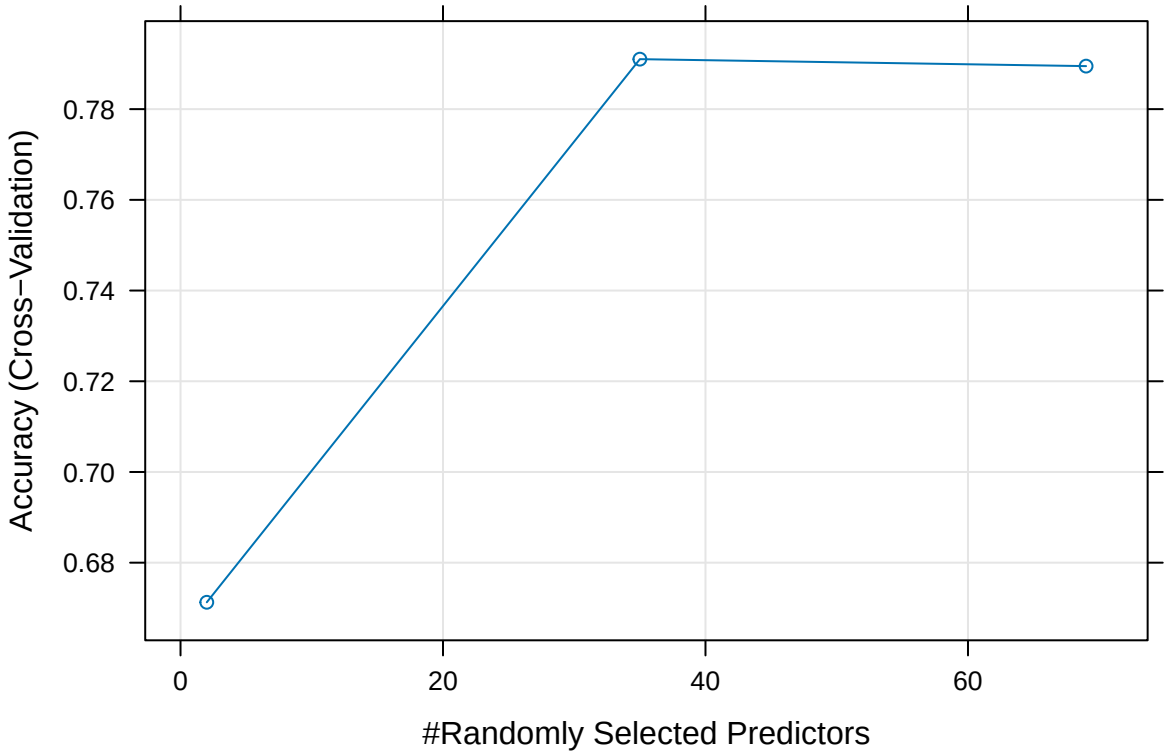


Figure 9: Default hyper parameter tuning plot for the RF model

```
)
rfv2_rfe16_.end.time <- Sys.time()
stopCluster(cl)
rfv2_rfe16_rfe16_test_acc <- test_accuracy(rfv2_rfe16,rfe_dataset_test)
```

The training time for this model was:

```
rfv2_rfe16_.end.time-rfv2_rfe16.start.time
```

```
## Time difference of 31.58674 mins
```

The hyper parameter tuning plot is show below:

```
plot(rfv2_rfe16)
```

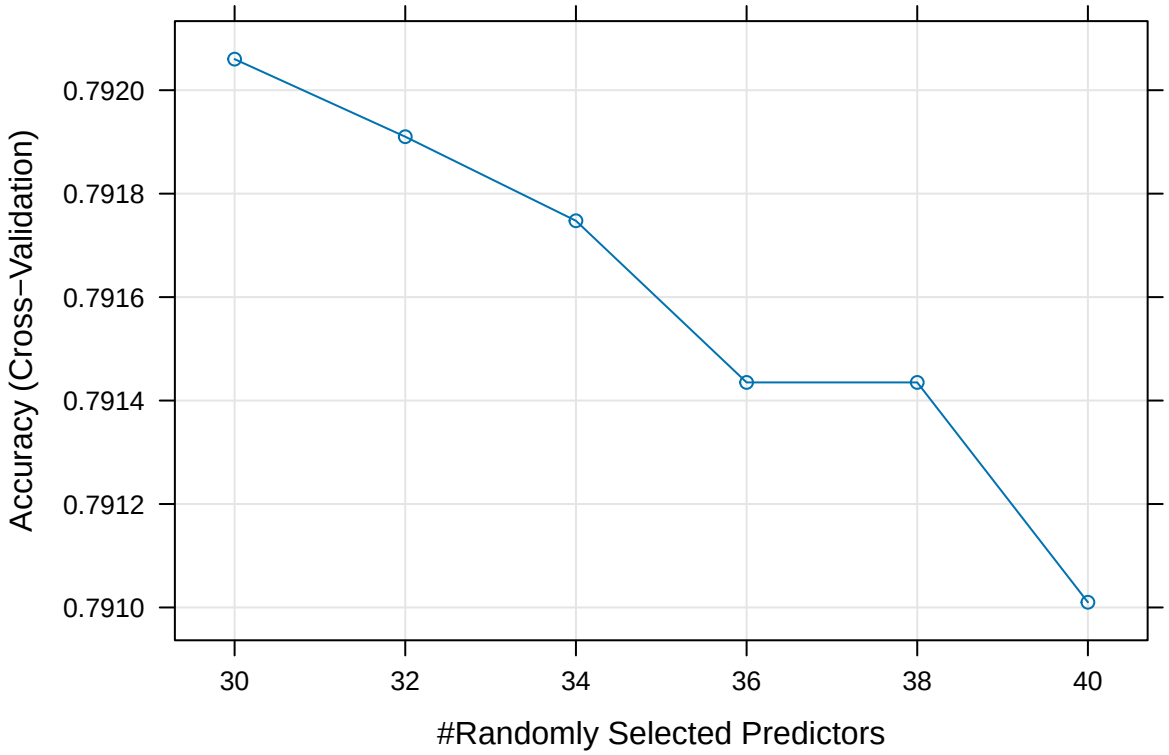


Figure 10: Modified hyper parameter tuning plot for the RF model

The following code showing the best cross-validation and testing accuracies achieved and the *mtry* used to get it:

```
# cat is used here to print the new lines, otherwise, they will be printed as raw \n
cat(sprintf("The best cross-validation accuracy is %1.3f which was obtained using the
mtry value of %d the testing accuracy is %1.3f",
        rfv2_rfe16[["results"]][["Accuracy"]][1],rfv2_rfe16[["results"]][["mtry"]][1],
        rfv2_rfe16_rfe16_test_acc))
```

```
## The best cross-validation accuracy is 0.792 which was obtained using the
## mtry value of 30 the testing accuracy is 0.792
```

Results

In this project I built 4 different models to predict the Credit Score, the following table showing thier names along with their accuracy during the training (Cross Validation) and their accuracy on the never seen testing dataset.

```
models_info %>% kbl(caption = 'Predciting Models Summary')%>%
  kable_styling(latex_options=c("striped","HOLD_position"))
```

Table 16: Predciting Models Summary

Model Name	CV Accuracy	Testing Accuracy	Tunning Parameter
GAM	0.62786	0.62927	select = 0, method = GCV.Cp
KNN	0.76373	0.76701	k = 3
RPART	0.70651	0.70281	cp = 0.00050
RF	0.79206	0.79176	mtry = 30

It is clear that the worst performing model is the GAM model, followed by RPART, this is expected since those are the least advance models I am using in this project, the best performing one is the RF model followed by the KNN model, the best accuracy achieved is 79.206% by the RF model.

Thanks to the 5-folds cross validations during the training process, it is clear that we don't have over fitting since the training and testing accuracies are very close in all models used in the project.

Back to the result of the features selection, the most 16 important features are given below:

```
ref_result[["optVariables"]]
```

```
##  [1] "Occupation"              "Credit_History_Age_Year" "Delay_from_due_date"
##  [4] "Num_Credit_Card"         "Changed_Credit_Limit"    "Total_EMI_per_month"
##  [7] "Age"                     "Monthly_Inhand_Salary"   "Num_of_Delayed_Payment"
## [10] "Outstanding_Debt"        "Interest_Rate"           "Credit_Mix"
## [13] "Annual_Income"           "Month"                   "Num_Bank_Accounts"
## [16] "Monthly_Balance"
```

The most important feature is the Occupation which is expected, the more paying job you have the financial problems you will face and hence the best credit score you will get. Credit History Age is important as well, as shown previously in the EDA section, people with long history dealing with banks and longer age they are the less the chance to get financial problems and hence they can maintain better Credit Score

Conclusion

In this project I studied the Credit Score Classification [2] dataset that has 27 features and 1 outcome variable which is the Credit Score, then I did an intensive cleaning to the features including NA handling, removing textual data from the numeric columns and outliers handling.

After that I did recursive feature elimination *rfe* to decide on the of number and name of the features to give us the best accuracy, finally I built 4 models to predict the Credit Score based on the selected features, the best performing model was RF Model which achieve a cross-validation accuracy of 0.792 and testing accuracy of 0.792 using the tuning parameter *mtry* value of 30.

The accuracy of the models I built can be improved by doing extensive hyper parameter tuning, but this will take long time (days or weeks) and I feel it is beyond the scope of the project, also other models can be used for prediction, lastly combining multiple models into one big one can also increase the accuracy.

References

[1] https://en.wikipedia.org/wiki/Credit_score

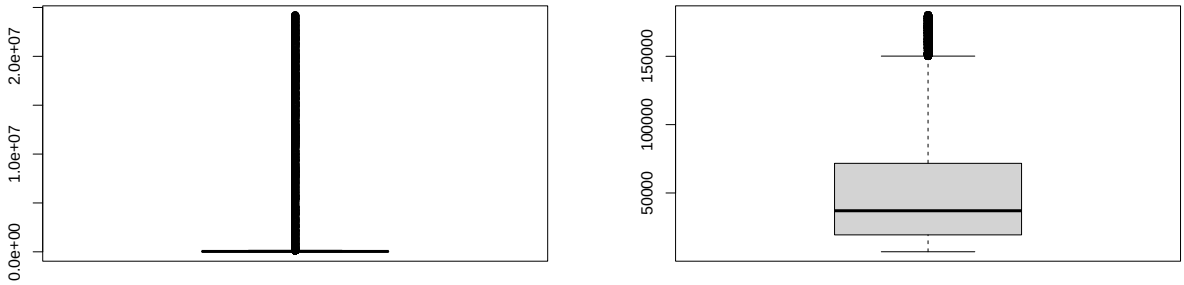
[2] https://www.kaggle.com/datasets/parisrohan/credit-score-classification

[3] https://en.wikipedia.org/wiki/Feature_selection

[4] https://www.rdocumentation.org/packages/caret/versions/6.0-92/topics/rfe

Appendices

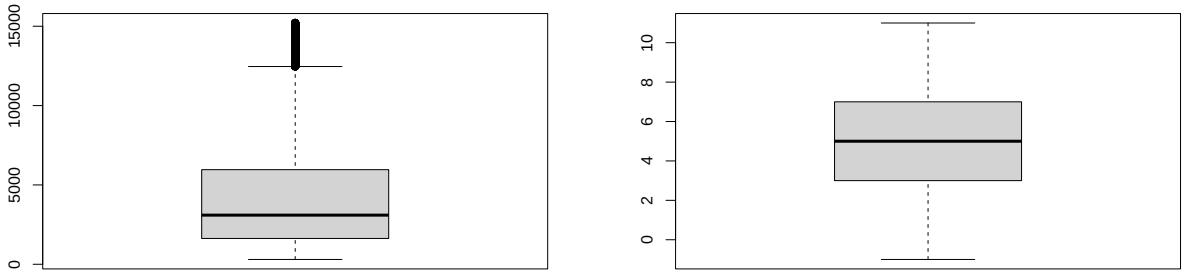
Appendix A: The Box Plots for the continous features before and after outliers handling



(a) Original Box Plot

(b) Box Plot after outliers handling

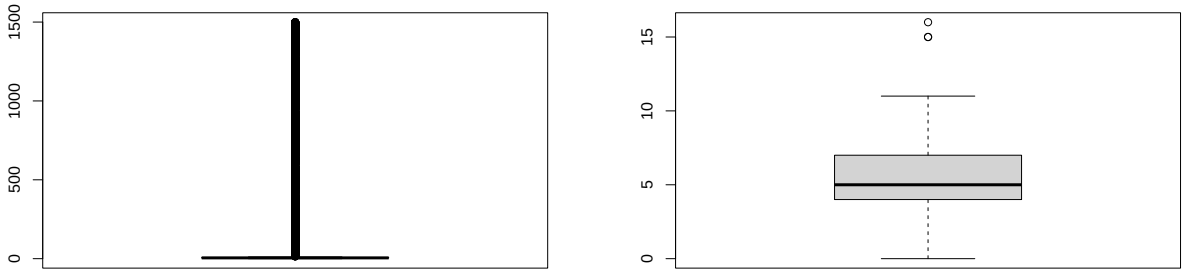
Figure 11: Handling outliers for the Annual Income



(a) Original Box Plot

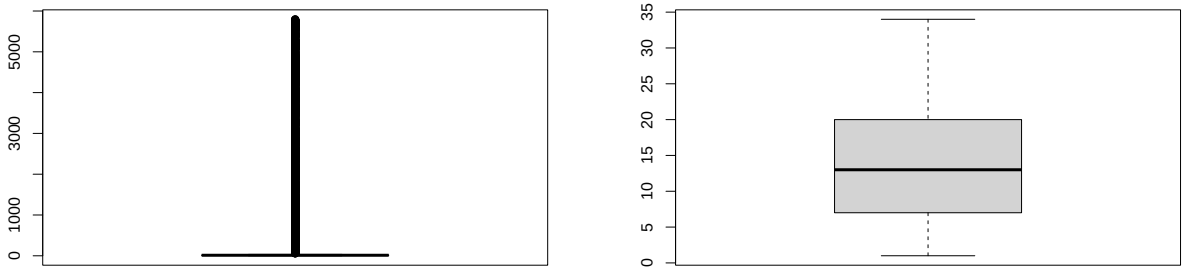
(b) Box Plot after outliers handling

Figure 12: Handling outliers for the Monthly Inhand Salary



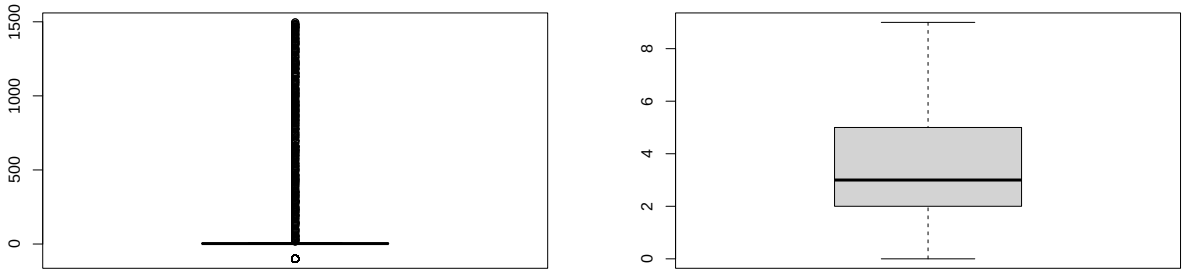
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 13: Handling outliers for the Num Credit Card



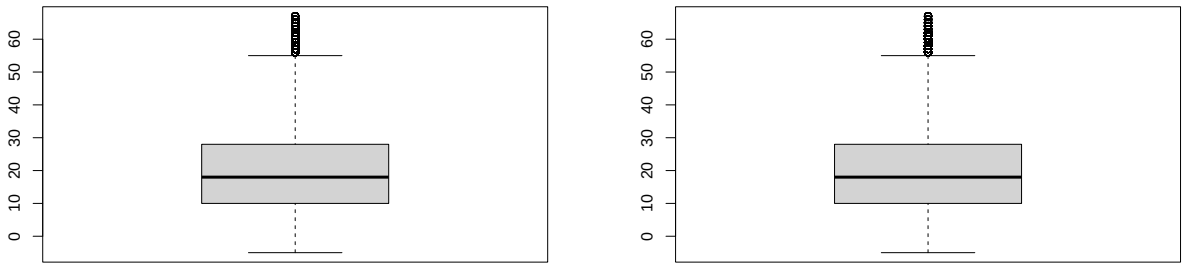
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 14: Handling outliers for the Interest Rate



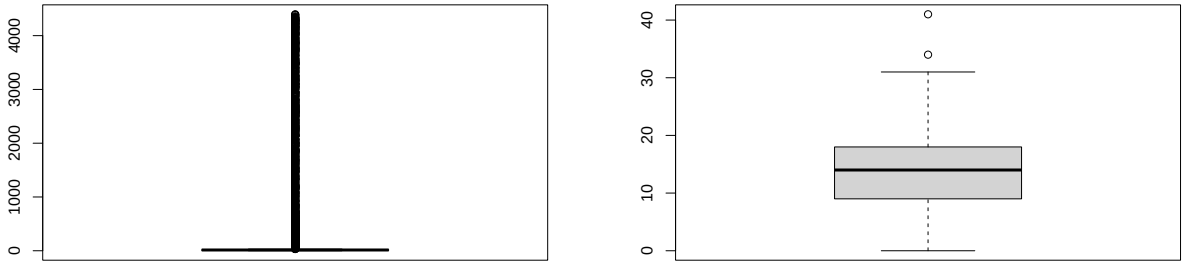
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 15: Handling outliers for the Num of Loan



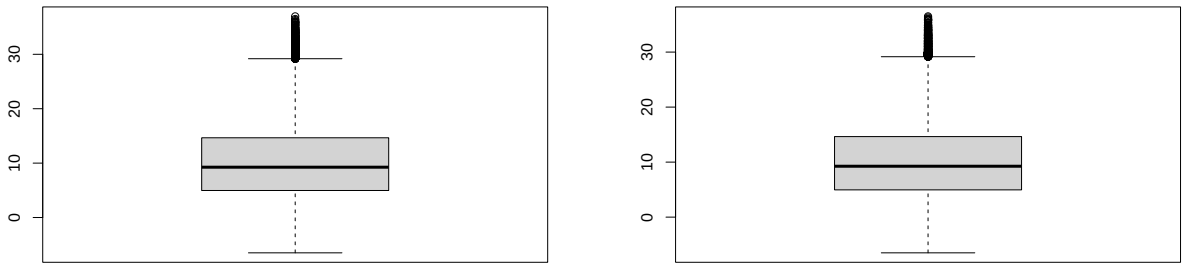
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 16: Handling outliers for the Delay from due date



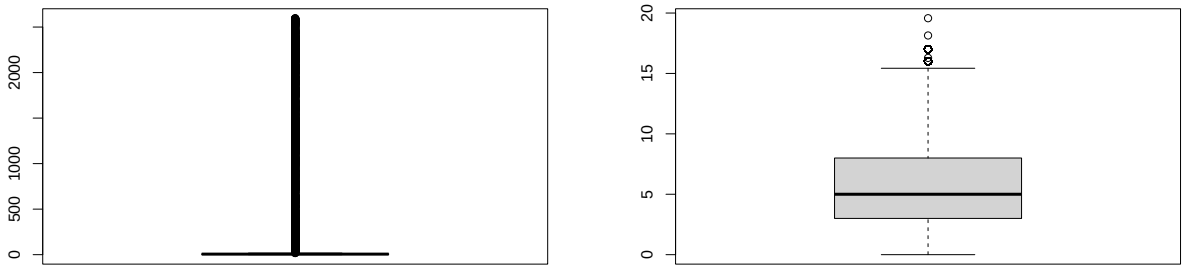
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 17: Handling outliers for the Num of Delayed Payment



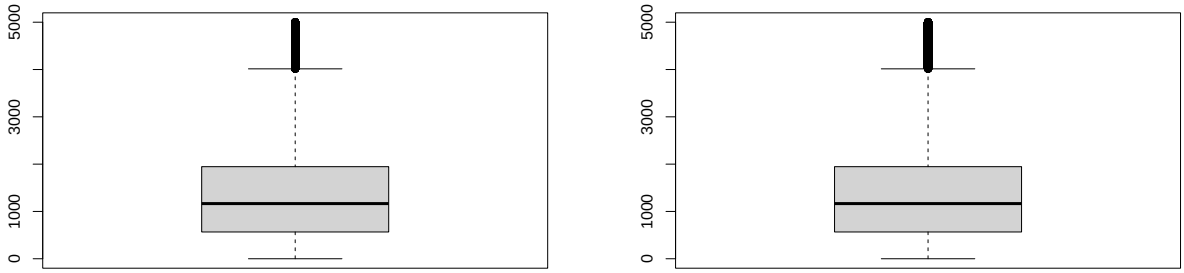
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 18: Handling outliers for the Changed Credit Limit



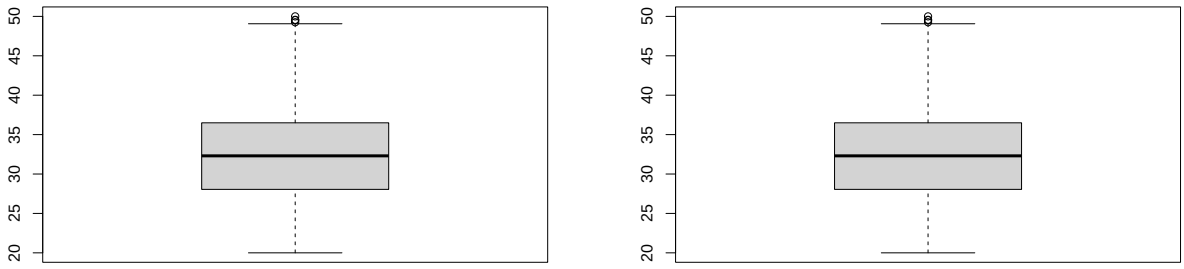
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 19: Handling outliers for the Num Credit Inquiries



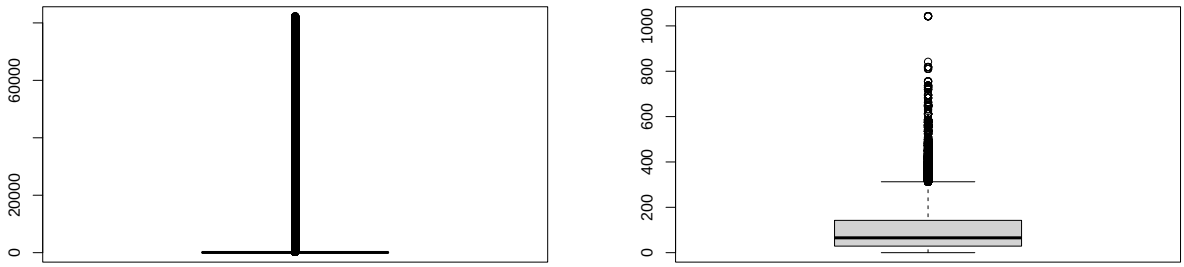
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 20: Handling outliers for the Outstanding Debt



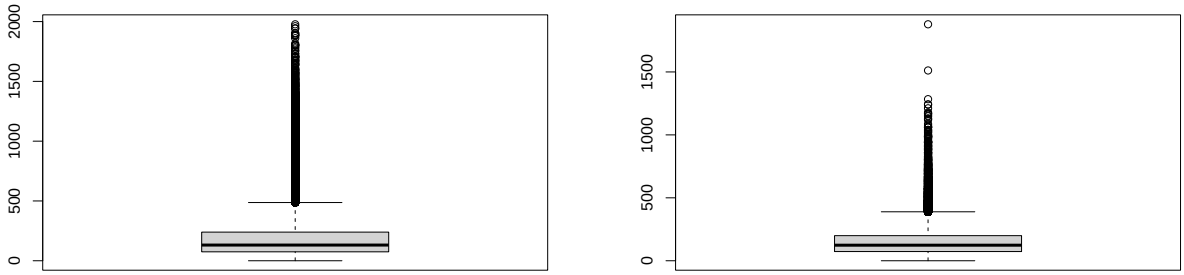
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 21: Handling outliers for the Credit Utilization Ratio



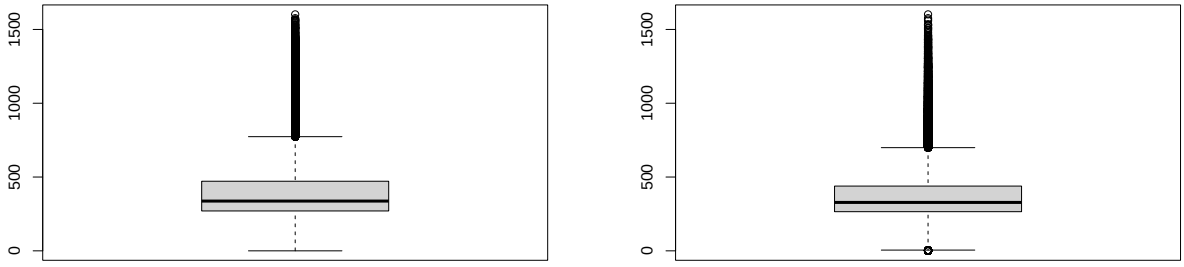
(a) Original Box Plot (b) Box Plot after outliers handling

Figure 22: Handling outliers for the Total EMI per month



(a) Original Box Plot (b) Box Plot after outliers handling

Figure 23: Handling outliers for the Amount invested monthly



(a) Original Box Plot (b) Box Plot after outliers handling

Figure 24: Handling outliers for the Monthly Balance