# Documentation of My Capstone Project on Predicting Default in Credit Card Payment for the Next Month

**Ibiyengha Tobin**

**301256083**

**TABLE OF CONTENTS**

# INTRODUCTION

## 1.0 Background

The goal of this project is to apply machine learning algorithms to identify the primary factors that influence the likelihood of credit card default while demonstrating the statistical foundations of the approaches used.

Based on the data of the customer and previous transactions, the goal is to build an automated model that can both identify the essential variables and predict a credit card default.

Credit card default has a huge influence on the financial organisations that provide these credit cards. Credit card failures cause financial losses for financial institutions since they are unable to reclaim outstanding balances and interest costs from defaulting customers, affecting their balance sheets and profitability. Credit card defaults can have a significant impact on the overall performance of a financial institution's loan portfolio, influencing crucial performance measures such as delinquency rates, charge-off rates, and loan loss reserves. Consistently high default rates can harm a financial institution's reputation and undermine consumer trust, potentially leading to customer loss and making it difficult to attract new clients.

## 2.0 Problem Statement

This project stems from the fact that credit card default is a major source of concern for banks and financial organisations.

The main objective of this project is to develop a model that can be used to both identify the important variables and predict a credit card default based on client information and previous transactions.

Financial institutions will also have early warning indicators of probable default, enabling them to manage and reduce credit risk by acting pro-actively.

## 3.0 Goals and Metrics

Goal 1: To create a model that can accurately predict credit card default for the next month
Metric 1: Assess all the models accuracy by using metrics, like accuracy score, confusion matrix, K-Fold Cross Validation and Stratified K-Fold Cross Validation.

Goal 2: To determine the identify the important factors that influence credit card default
Metric 2: Identify the important variables that have an impact on credit card default.

Goal 3: To positively impact the performance of a financial institution's loan portfolio and help them to manage and reduce credit risk.
Metric 3: Measure the decline in credit card default after implementing the prediction model.

## 4.0 Assumptions

- It is possible that the features picked for credit card default prediction will remain stable and predictive throughout time.
- There may be a failure to consider that people who use credit cards may face unforeseen financial problems that tax their resources and make it difficult for them to meet their credit card payment obligations, such as medical expenses, home repairs, or automobile accidents.

**Limitations:**

- The features (variables) selected as input can have a substantial impact on the ability of a prediction model to make correct predictions. When irrelevant or redundant features are used, the model's performance may degrade.
- The type and volume of data available for training have a significant impact on prediction model performance. If the data is untrustworthy, erroneous, or biased, the model's predictions may be incorrect.
- The model does not account for future changes in consumer behaviour, the status of the economy, or other factors that may influence credit card defaults.

## DATA SOURCES

## 5.0 Dataset Introduction

Default of Credit Card Clients in Taiwan. 30000 Customers who defaulted in credit card payments in Taiwan. Information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.
Data Source: https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients

## 6.0 Data Dictionary

1. ID: ID of each client
2. LIMIT_BAL: Amount of given credit in New Taiwan(NT) dollars
3. GENDER: (1 = Male, 2 = Female)
4. EDUCATION (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
5. MARRIAGE : Marital status (1=married, 2=single, 3=others)
6. AGE: Age in years
7. REPAY_SEPT : Repayment status in September, 2005
8. REPAY_AUG: Repayment status in August, 2005
9. REPAY_JULY: Repayment status in July, 2005
10. REPAY_JUNE: Repayment status in June, 2005
11. REPAY_MAY: Repayment status in May, 2005
12. REPAY_APR Repayment status in April, 2005
13. BILL_AMT_SEPT: Amount of bill statement in September, 2005 (NT dollar)
14. BILL_AMT_AUG: Amount of bill statement in August, 2005 (NT dollar)
15. BILL_AMT_JULY: Amount of bill statement in July, 2005 (NT dollar)
16. BILL_AMT_JUNE: Amount of bill statement in June, 2005 (NT dollar)
17. BILL_AMT_MAY: Amount of bill statement in May, 2005 (NT dollar)
18. BILL_AMT_APR: Amount of bill statement in April, 2005 (NT dollar)
19. PREV_AMT_SEPT: Amount of previous payment in September, 2005 (NT dollar)
20. PREV_AMT_AUG: Amount of previous payment in August, 2005 (NT dollar)
21. PREV_AMT_JULY: Amount of previous payment in July, 2005 (NT dollar)
22. PREV_AMT_JUNE: Amount of previous payment in June, 2005 (NT dollar)
23. PREV_AMT_MAY: Amount of previous payment in May, 2005 (NT dollar)

24. PREV_AMT_APR: Amount of previous payment in April, 2005 (NT dollar)
25. DEFAULT_PAY_NEXTMNTH: (1=yes, 0=no)

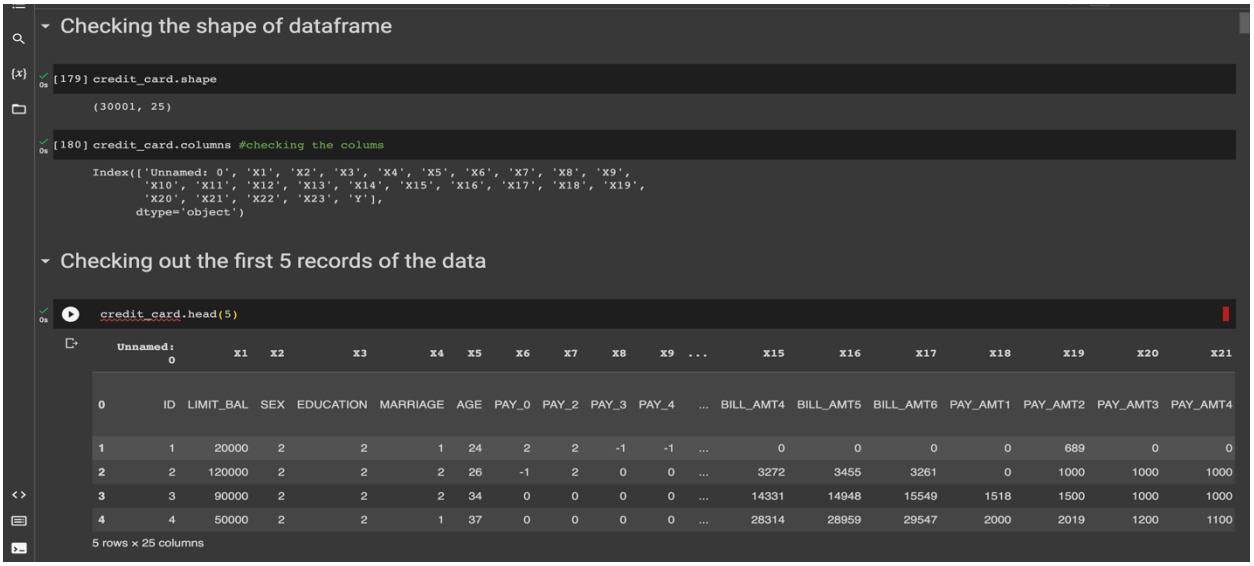
I used classification analysis to examine details about the consumers, such as their age, marital status, education, and payment history in the past, in order to determine whether or not they will miss their credit card payment for the subsequent month. I created an automated model that can both identify the important variables and predict a credit card default based on the client's information and previous transactions.

## DATA EXPLORATION

### 7.0 Data Exploration Techniques

**Importing and reading in the dataset:**


**Checking the Shape and first and last records of Data:**



Checking out the last 5 records of the data

```
[182] credit_card.tail(5)
```

| | Unnamed: 0 | X1 | X2 | X3 | X4 | X5 | X6 | X7 | X8 | X9 | ... | X15 | X16 | X17 | X18 | X19 | X20 | X21 | X22 | X23 | Y |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **29996** | 29996 | 220000 | 1 | 3 | 1 | 39 | 0 | 0 | 0 | 0 | ... | 88004 | 31237 | 15980 | 8500 | 20000 | 5003 | 3047 | 5000 | 1000 | 0 |
| **29997** | 29997 | 150000 | 1 | 3 | 2 | 43 | -1 | -1 | -1 | -1 | ... | 8979 | 5190 | 0 | 1837 | 3526 | 8998 | 129 | 0 | 0 | 0 |
| **29998** | 29998 | 30000 | 1 | 2 | 2 | 37 | 4 | 3 | 2 | -1 | ... | 20878 | 20582 | 19357 | 0 | 0 | 22000 | 4200 | 2000 | 3100 | 1 |
| **29999** | 29999 | 80000 | 1 | 3 | 1 | 41 | 1 | -1 | 0 | 0 | ... | 52774 | 11855 | 48944 | 85900 | 3409 | 1178 | 1926 | 52964 | 1804 | 1 |
| **30000** | 30000 | 50000 | 1 | 2 | 1 | 46 | 0 | 0 | 0 | 0 | ... | 36535 | 32428 | 15313 | 2078 | 1800 | 1430 | 1000 | 1000 | 1000 | 1 |

5 rows × 25 columns

## Arranging the Columns:

I made Index 0 the column names because the columns were not named. Then I dropped index 0 because it has the same information as the columns.

Arranging the columns

```
[183] credit_card.columns = credit_card.iloc[0] #making index 0 the column names
      credit_card = credit_card.drop(credit_card.index[0]) #droping index 0 because it has the same information as the columns
```

```
credit_card.head(5)
```

| | ID | LIMIT_BAL | SEX | EDUCATION | MARRIAGE | AGE | PAY_0 | PAY_2 | PAY_3 | PAY_4 | ... | BILL_AMT4 | BILL_AMT5 | BILL_AMT6 | PAY_AMT1 | PAY_AMT2 | PAY_AMT3 | PAY_AMT4 | PAY_AMT5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 20000 | 2 | 2 | 1 | 24 | 2 | 2 | -1 | -1 | ... | 0 | 0 | 0 | 0 | 689 | 0 | 0 | 0 |
| 2 | 2 | 120000 | 2 | 2 | 2 | 26 | -1 | 2 | 0 | 0 | ... | 3272 | 3455 | 3261 | 0 | 1000 | 1000 | 1000 | 0 |
| 3 | 3 | 90000 | 2 | 2 | 2 | 34 | 0 | 0 | 0 | 0 | ... | 14331 | 14948 | 15549 | 1518 | 1500 | 1000 | 1000 | 1000 |
| 4 | 4 | 50000 | 2 | 2 | 1 | 37 | 0 | 0 | 0 | 0 | ... | 28314 | 28959 | 29547 | 2000 | 2019 | 1200 | 1100 | 1069 |
| 5 | 5 | 50000 | 1 | 2 | 1 | 57 | -1 | 0 | -1 | 0 | ... | 20940 | 19146 | 19131 | 2000 | 36681 | 10000 | 9000 | 689 |

5 rows × 25 columns

## Renaming the Columns:

I renamed the column names to give better insight into each column

Renaming the columns

```
[185] credit_card.columns = ['ID','LIMIT_BAL','GENDER','EDUCATION','MARRIAGE','AGE','REPAY_SEPT','REPAY_AUG','REPAY_JULY','REPAY_JUNE','REPAY_MAY','REPAY_APR',
                             'BILL_AMT_SEPT','BILL_AMT_AUG','BILL_AMT_JULY','BILL_AMT_JUNE','BILL_AMT_MAY','BILL_AMT_APR','PREV_AMT_SEPT','PREV_AMT_AUG',
                             'PREV_AMT_JULY','PREV_AMT_JUNE','PREV_AMT_MAY','PREV_AMT_APR','DEFAULT_PAY_NEXTMNTH']
```

```
credit_card.head(5)
```

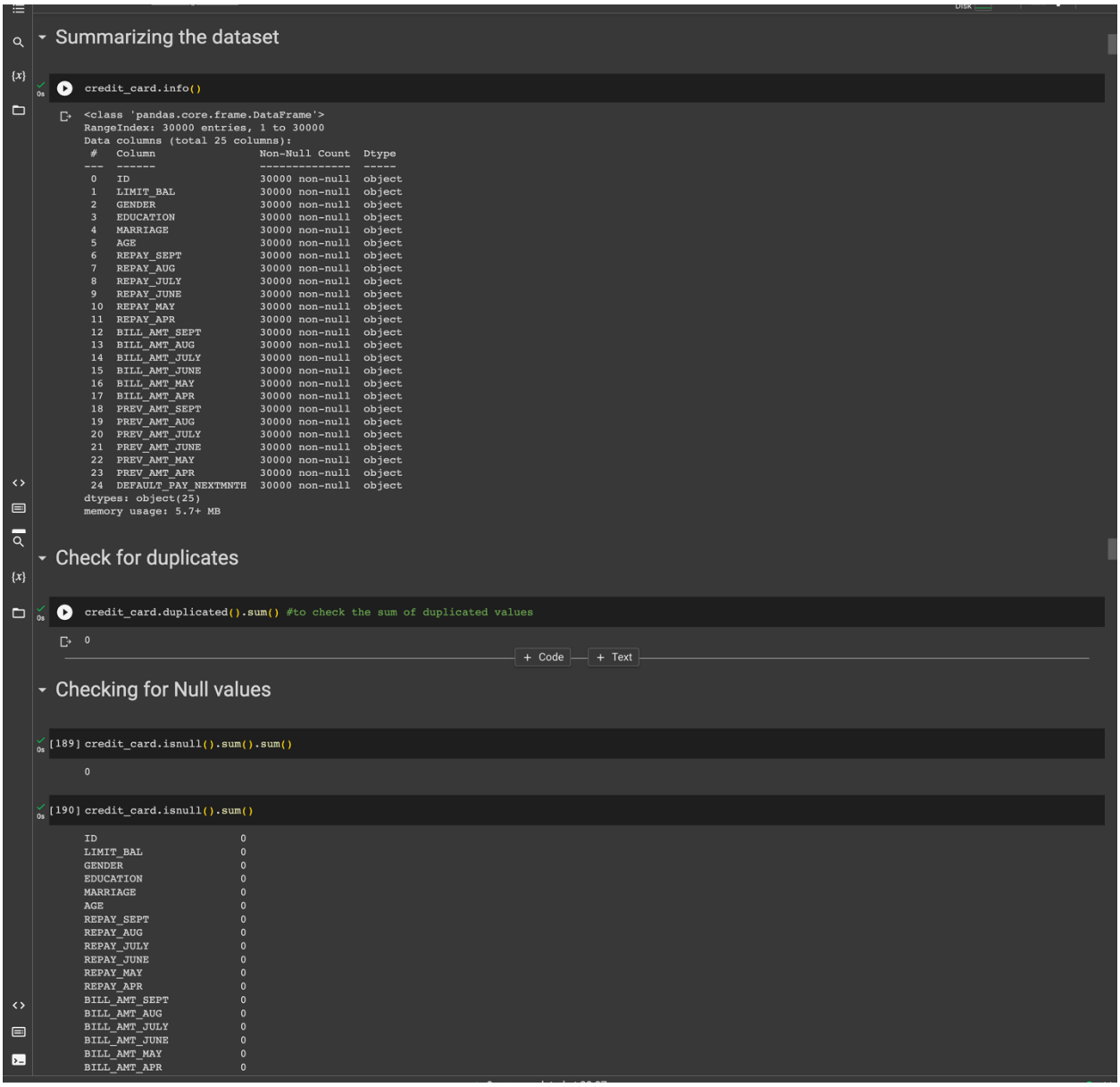
| | ID | LIMIT_BAL | GENDER | EDUCATION | MARRIAGE | AGE | REPAY_SEPT | REPAY_AUG | REPAY_JULY | REPAY_JUNE | ... | BILL_AMT_JUNE | BILL_AMT_MAY | BILL_AMT_APR | PREV_AMT_SEPT |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 20000 | 2 | 2 | 1 | 24 | 2 | 2 | -1 | -1 | ... | 0 | 0 | 0 | 0 |
| 2 | 2 | 120000 | 2 | 2 | 2 | 26 | -1 | 2 | 0 | 0 | ... | 3272 | 3455 | 3261 | 0 |
| 3 | 3 | 90000 | 2 | 2 | 2 | 34 | 0 | 0 | 0 | 0 | ... | 14331 | 14948 | 15549 | 1518 |
| 4 | 4 | 50000 | 2 | 2 | 1 | 37 | 0 | 0 | 0 | 0 | ... | 28314 | 28959 | 29547 | 2000 |
| 5 | 5 | 50000 | 1 | 2 | 1 | 57 | -1 | 0 | -1 | 0 | ... | 20940 | 19146 | 19131 | 2000 |

5 rows × 25 columns

.

**Summarizing the dataset:**

I checked on the information of the data, checked for null and duplicates and the datatypes. These preliminary processes in data preprocessing establish the groundwork for efficient and accurate data analysis, resulting in more trustworthy insights and better decision-making.:



Checking for datatypes

```
credit_card.dtypes
```

```
ID                    object
LIMIT_BAL             object
GENDER                object
EDUCATION             object
MARRIAGE              object
AGE                   object
REPAY_SEPT            object
REPAY_AUG             object
REPAY_JULY            object
REPAY_JUNE            object
REPAY_MAY             object
REPAY_APR             object
BILL_AMT_SEPT         object
BILL_AMT_AUG          object
BILL_AMT_JULY         object
BILL_AMT_JUNE         object
BILL_AMT_MAY          object
BILL_AMT_APR          object
PREV_AMT_SEPT         object
PREV_AMT_AUG          object
PREV_AMT_JULY         object
PREV_AMT_JUNE         object
PREV_AMT_MAY          object
PREV_AMT_APR          object
DEFAULT_PAY_NEXTMNTH  object
dtype: object
```

9s completed at 00:07

After analyzing the dataset, I discovered that there were no null values or duplicates and all the variables data types are objects. This led me to the next step.

## Changing Datatypes of attributes:

It is critical for data preparation, model compatibility, and overall model performance to transform variables from the "object" data type to more appropriate data types, such as numerical or categorical data types. It enables one to maximize the value of the data by extracting useful insights that can improve the accuracy and dependability of your predictions.

changing datatypes of attributes from object to category

```
[192] credit_card['ID'] = credit_card['ID'].astype('int')
      credit_card['GENDER'] = credit_card['GENDER'].astype('category') #Gender is a categorical variable
      credit_card['EDUCATION'] = credit_card['EDUCATION'].astype('category') #Education is a categorical variable
      credit_card['MARRIAGE'] = credit_card['MARRIAGE'].astype('category') #Marriage is a categorical variable
      credit_card['AGE'] = credit_card['AGE'].astype('int')
      credit_card['REPAY_SEPT'] = credit_card['REPAY_SEPT'].astype('int')
      credit_card['REPAY_AUG'] = credit_card['REPAY_AUG'].astype('int')
      credit_card['REPAY_JULY'] = credit_card['REPAY_JULY'].astype('int')
      credit_card['REPAY_JUNE'] = credit_card['REPAY_JUNE'].astype('int')
      credit_card['REPAY_MAY'] = credit_card['REPAY_MAY'].astype('int')
      credit_card['REPAY_APR'] = credit_card['REPAY_APR'].astype('int')
      credit_card['DEFAULT_PAY_NEXTMNTH'] = credit_card['DEFAULT_PAY_NEXTMNTH'].astype('int')
```

changing datatypes of attributes from object to int

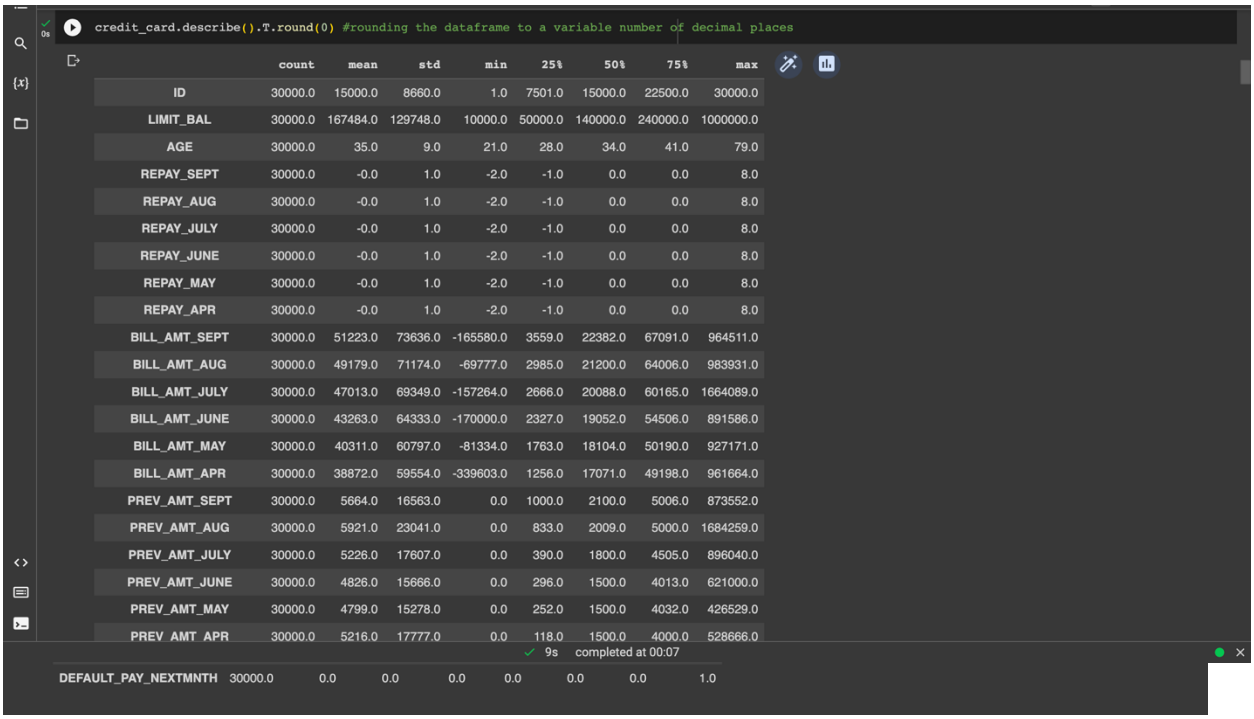
+ Code + Text

```
[193] credit_card['LIMIT_BAL'] = credit_card['LIMIT_BAL'].astype('int')
      credit_card['BILL_AMT_SEPT'] = credit_card['BILL_AMT_SEPT'].astype('int')
      credit_card['BILL_AMT_AUG'] = credit_card['BILL_AMT_AUG'].astype('int')
      credit_card['BILL_AMT_JULY'] = credit_card['BILL_AMT_JULY'].astype('int')
      credit_card['BILL_AMT_JUNE'] = credit_card['BILL_AMT_JUNE'].astype('int')
      credit_card['BILL_AMT_MAY'] = credit_card['BILL_AMT_MAY'].astype('int')
      credit_card['BILL_AMT_APR'] = credit_card['BILL_AMT_APR'].astype('int')
      credit_card['PREV_AMT_SEPT'] = credit_card['PREV_AMT_SEPT'].astype('int')
      credit_card['PREV_AMT_AUG'] = credit_card['PREV_AMT_AUG'].astype('int')
      credit_card['PREV_AMT_JULY'] = credit_card['PREV_AMT_JULY'].astype('int')
      credit_card['PREV_AMT_JUNE'] = credit_card['PREV_AMT_JUNE'].astype('int')
      credit_card['PREV_AMT_MAY'] = credit_card['PREV_AMT_MAY'].astype('int')
      credit_card['PREV_AMT_APR'] = credit_card['PREV_AMT_APR'].astype('int')
```

9s completed at 00:07

## Descriptive Statistics Summary:

The describe method provides rapid insights into the data's central patterns and dispersion. It gives statistics for numerical variables including mean, median, standard deviation, minimum, and maximum. You can better comprehend the data's distribution and dispersion due to these statistics.



```
credit_card.describe().T.round(0) #rounding the dataframe to a variable number of decimal places
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| ID | 30000.0 | 15000.0 | 8660.0 | 1.0 | 7501.0 | 15000.0 | 22500.0 | 30000.0 |
| LIMIT_BAL | 30000.0 | 167484.0 | 129748.0 | 10000.0 | 50000.0 | 140000.0 | 240000.0 | 1000000.0 |
| AGE | 30000.0 | 35.0 | 9.0 | 21.0 | 28.0 | 34.0 | 41.0 | 79.0 |
| REPAY_SEPT | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_AUG | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_JULY | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_JUNE | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_MAY | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_APR | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| BILL_AMT_SEPT | 30000.0 | 51223.0 | 73636.0 | -165580.0 | 3559.0 | 22382.0 | 67091.0 | 964511.0 |
| BILL_AMT_AUG | 30000.0 | 49179.0 | 71174.0 | -69777.0 | 2985.0 | 21200.0 | 64006.0 | 983931.0 |
| BILL_AMT_JULY | 30000.0 | 47013.0 | 69349.0 | -157264.0 | 2666.0 | 20088.0 | 60165.0 | 1664089.0 |
| BILL_AMT_JUNE | 30000.0 | 43263.0 | 64333.0 | -170000.0 | 2327.0 | 19052.0 | 54506.0 | 891586.0 |
| BILL_AMT_MAY | 30000.0 | 40311.0 | 60797.0 | -81334.0 | 1763.0 | 18104.0 | 50190.0 | 927171.0 |
| BILL_AMT_APR | 30000.0 | 38872.0 | 59554.0 | -339603.0 | 1256.0 | 17071.0 | 49198.0 | 961664.0 |
| PREV_AMT_SEPT | 30000.0 | 5664.0 | 16563.0 | 0.0 | 1000.0 | 2100.0 | 5006.0 | 873552.0 |
| PREV_AMT_AUG | 30000.0 | 5921.0 | 23041.0 | 0.0 | 833.0 | 2009.0 | 5000.0 | 1684259.0 |
| PREV_AMT_JULY | 30000.0 | 5226.0 | 17607.0 | 0.0 | 390.0 | 1800.0 | 4505.0 | 896040.0 |
| PREV_AMT_JUNE | 30000.0 | 4826.0 | 15666.0 | 0.0 | 296.0 | 1500.0 | 4013.0 | 621000.0 |
| PREV_AMT_MAY | 30000.0 | 4799.0 | 15278.0 | 0.0 | 252.0 | 1500.0 | 4032.0 | 426529.0 |
| PREV_AMT_APR | 30000.0 | 5216.0 | 17777.0 | 0.0 | 118.0 | 1500.0 | 4000.0 | 528666.0 |
| DEFAULT_PAY_NEXTMNTH | 30000.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 |

9s completed at 00:07

The table summarizes the data distribution for each attribute, which helped me comprehend the data's range, spread, and central tendency.

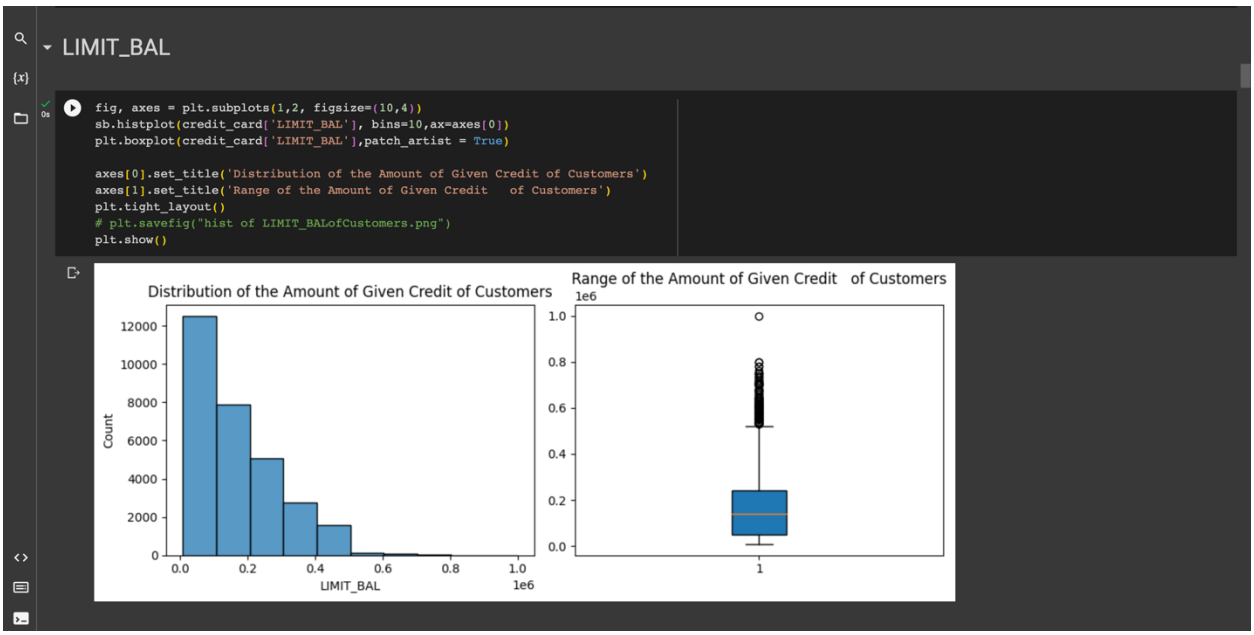
## 8.0 Data Cleansing

Data cleansing is the act of discovering and removing errors, inconsistencies, inaccuracies, and anomalies in a dataset in order to improve its quality, reliability, and accuracy. Data cleansing is essential for gaining trustworthy insights from the data and making sound decisions. Poor data quality can lead to inaccurate findings, incorrect forecasts, and biased analyses.

There are no missing values and duplicates so I did not have to handle them. The only thing I had to clean were outliers. Outliers are data points that differ dramatically from the rest of the data. Outliers, while not usually incorrect, can have an effect on the distribution and analysis. I decided to remove the outliers.

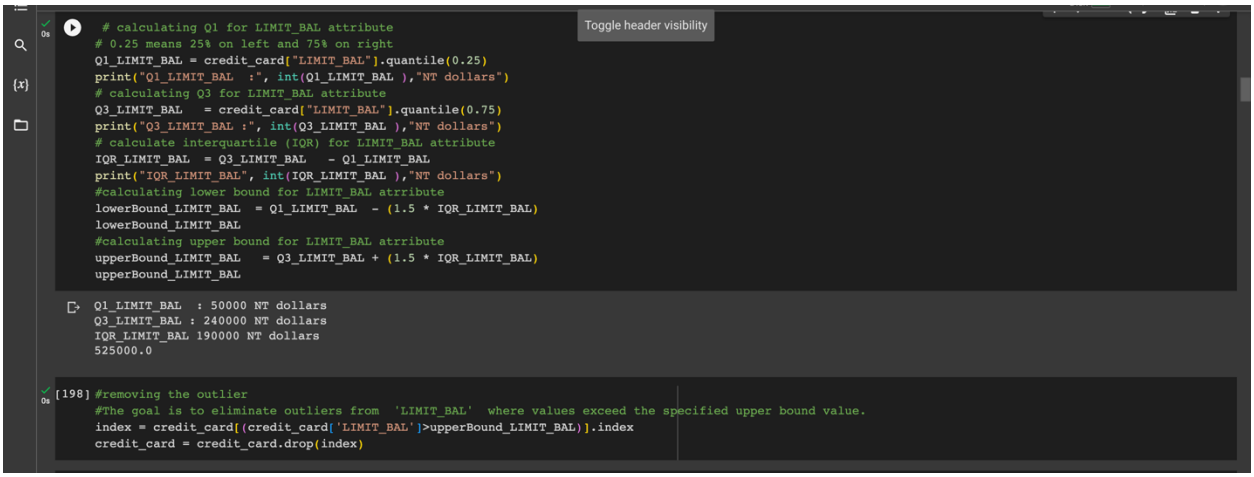
**Visualization and Detection of the Outliers:**


- **LIMIT_BAL Variable:**

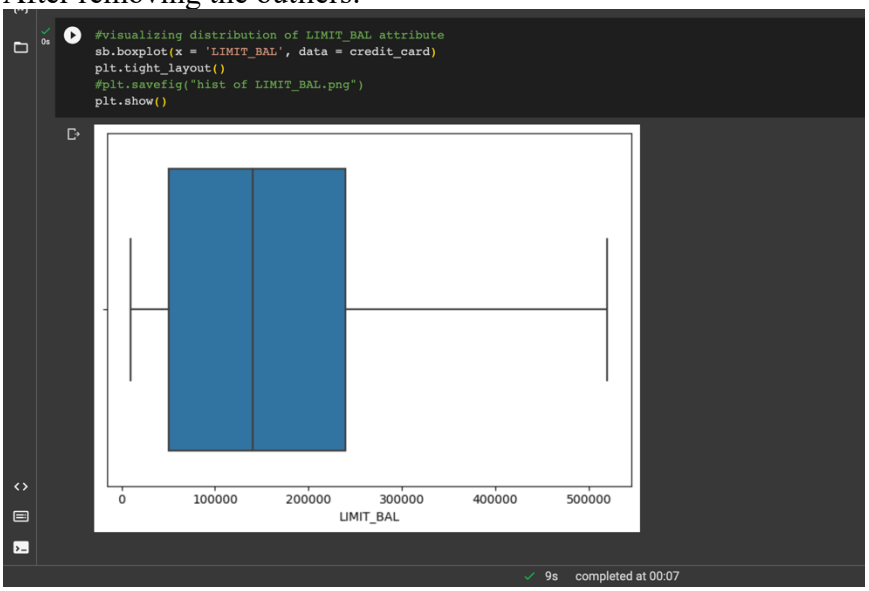


Looking at the boxplot, Outliers were detected in LIMIT_BAL attribute, so I removed these outliers by calculating the limits on the variable and taking outliers as observations that fall outside of the range.

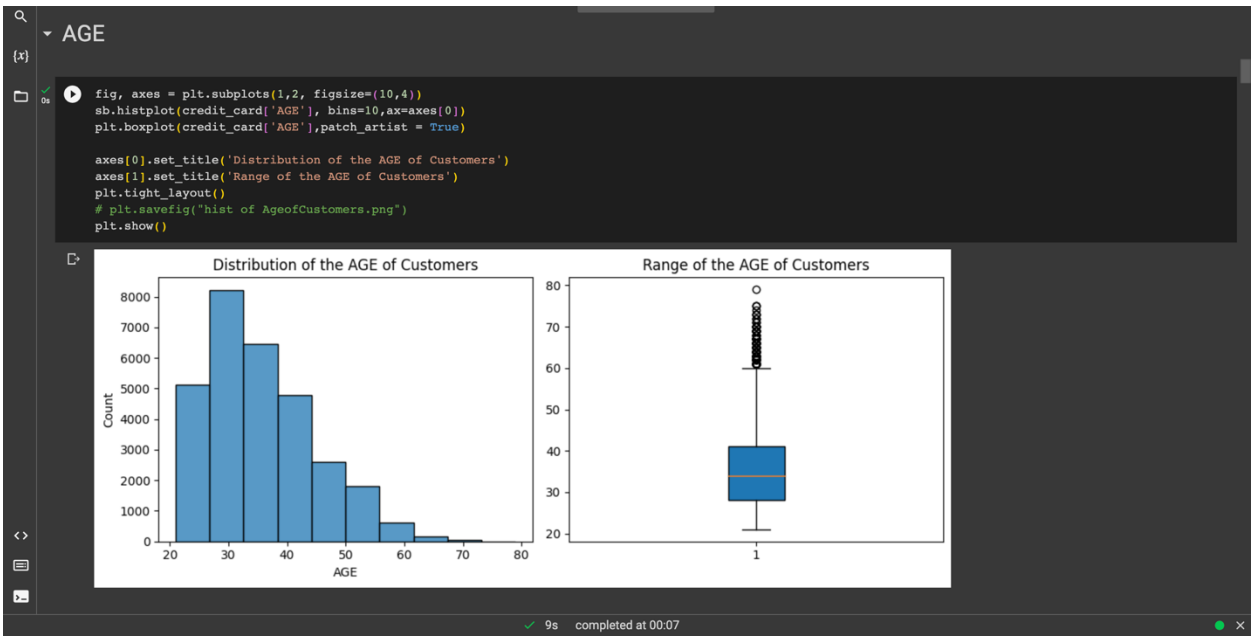
Dealing with these outliers, I used the Quantile method, a strong statistical technique that focuses on particular percentiles of the data distribution to identify and mitigate outliers, to address this. Calculating the lower and upper quantiles—specifically, the lower quantile, or 25th percentile, and the upper quantile, or 75th percentile—allows to establish acceptable ranges for the data. These quantiles mark the boundaries at which data points are no longer regarded as outliers. Potential outliers include any data points that are below Q1 or above Q3. For this variable I removed all the data points that exceeded the upper bound value.




After removing the outliers:



- **AGE Variable:**




Looking at the boxplot, Outliers were detected in Age attribute, so I removed these outliers by calculating the limits on the variable and taking outliers as observations that fall outside of the range.

Dealing with these outliers, I used the Quantile method, a strong statistical technique that focuses on particular percentiles of the data distribution to identify and mitigate outliers, to address this. Calculating the lower and upper quantiles—specifically, the lower quantile, or 25th percentile, and the upper quantile, or 75th percentile—allows to establish acceptable ranges for the data. These quantiles mark the boundaries at which data points are no longer regarded as outliers. Potential outliers include any data points that are below Q1 or above Q3. For this variable I removed all the data points that exceeded the upper bound value.

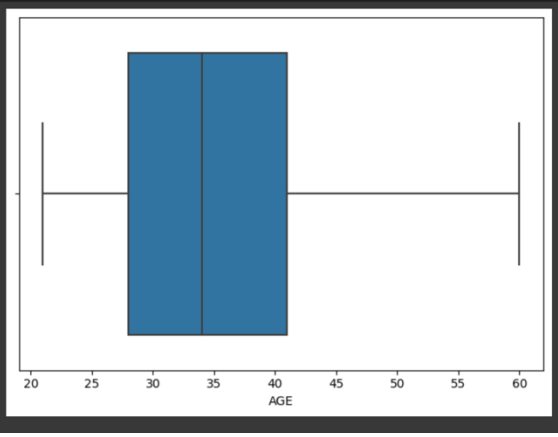
```
import numpy as np
# calculating Q1 for AGE attribute
# 0.25 means 25% on left and 75% on right
Q1_AGE = credit_card["AGE"].quantile(0.25)
print("Q1_AGE:", int(Q1_AGE),"years")
# calculating Q3 for AGE attribute
Q3_AGE = credit_card["AGE"].quantile(0.75)
print("Q3_AGE", int(Q3_AGE),"years")
# calculate interquartile (IQR) for AGE attribute
IQR_AGE = Q3_AGE - Q1_AGE
print("IQR_AGE", int(IQR_AGE),"years")
#calculating lower bound for AGE atrribute
lowerBound_AGE = Q1_AGE - (1.5 * IQR_AGE)
lowerBound_AGE
#calculating upper bound for AGE atrribute
upperBound_AGE = Q3_AGE + (1.5 * IQR_AGE)
upperBound_AGE
```

```
Q1_AGE: 28 years
Q3_AGE 41 years
IQR_AGE 13 years
60.5
```

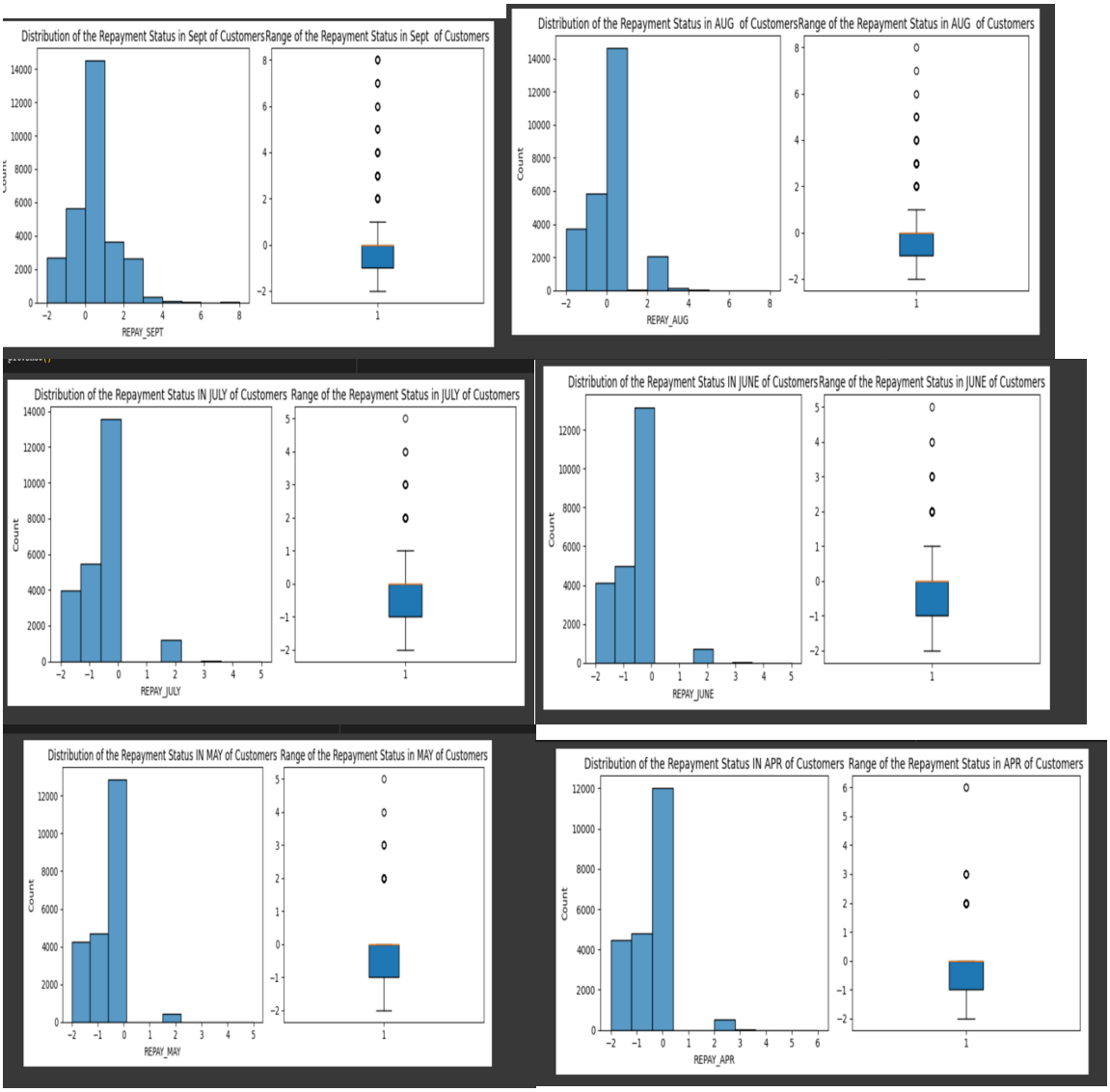
```
[202] #removing the outlier
#The goal is to eliminate outliers from  'AGE'  where values exceed the specified upper bound value.
index = credit_card[(credit_card['AGE']>upperBound_AGE)].index
credit_card = credit_card.drop(index)
```

After removing the outliers:

```
#visualizing distribution of age attribute
sb.boxplot(x = 'AGE', data = credit_card)
plt.tight_layout()
#plt.savefig("hist of AGE.png")
plt.show()
```



- **REPAY_SEPT, REPAY_AUG, REPAY_JULY, REPAY_JUNE, REPAY_MAY, REPAY_APR Variables:**

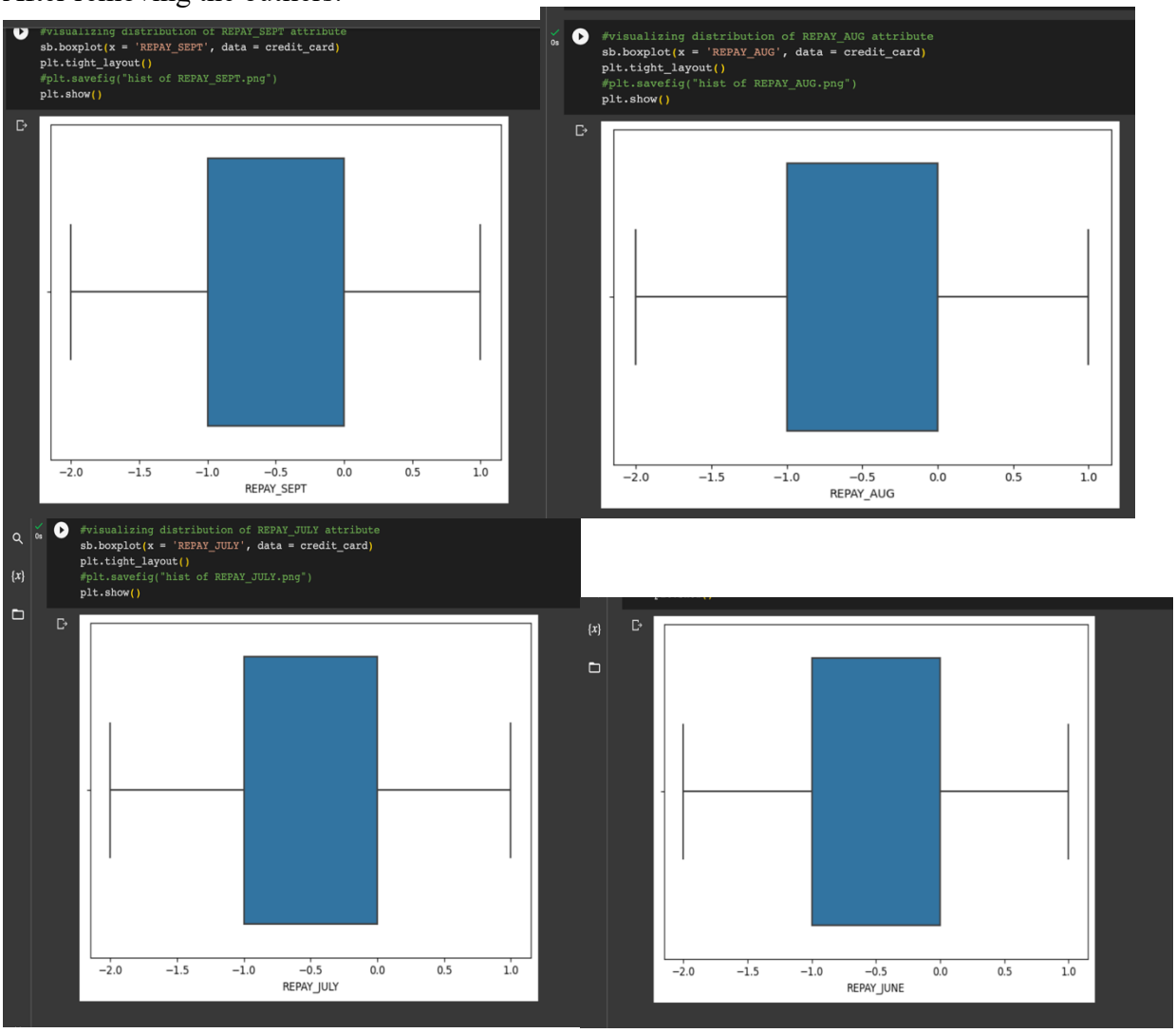



Looking at the boxplots, Outliers were detected in all these attributes, so I removed these outliers by calculating the limits on the variable and taking outliers as observations that fall outside of the range.

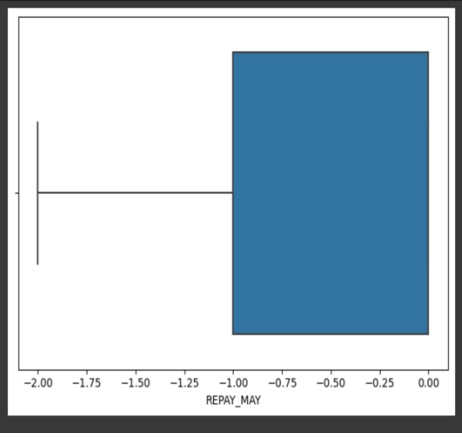
Dealing with these outliers, I used the Quantile method, a strong statistical technique that focuses on particular percentiles of the data distribution to identify and mitigate outliers, to address this. Calculating the lower and upper quantiles—specifically, the lower quantile, or 25th percentile, and the upper quantile, or 75th percentile—allows to establish acceptable ranges for the data. These

quantiles mark the boundaries at which data points are no longer regarded as outliers. Potential outliers include any data points that are below Q1 or above Q3. For these variables I removed all the data points that exceeded the upper bound value.

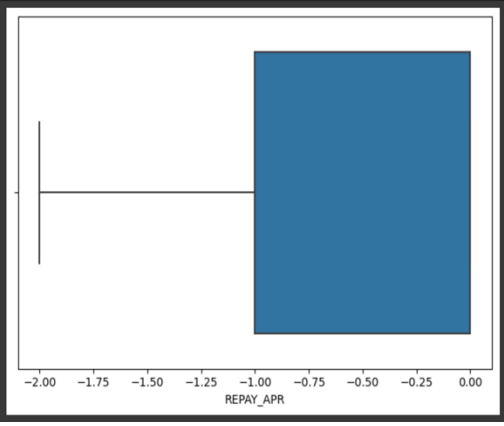
After removing the outliers:



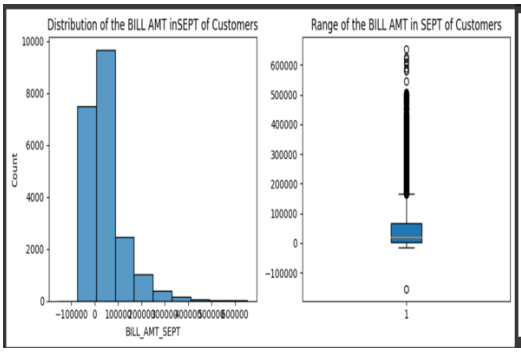
```
#visualizing distribution of REPAY_MAY attribute
sb.boxplot(x = 'REPAY_MAY', data = credit_card)
plt.tight_layout()
#plt.savefig("hist of REPAY_MAY .png")
plt.show()
```

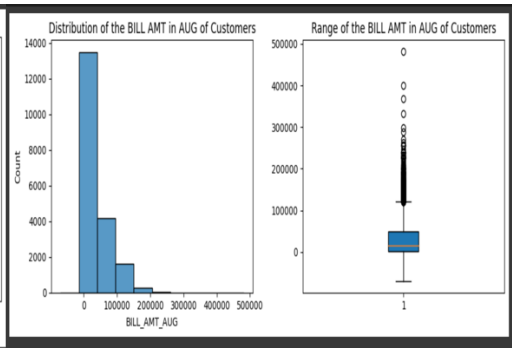


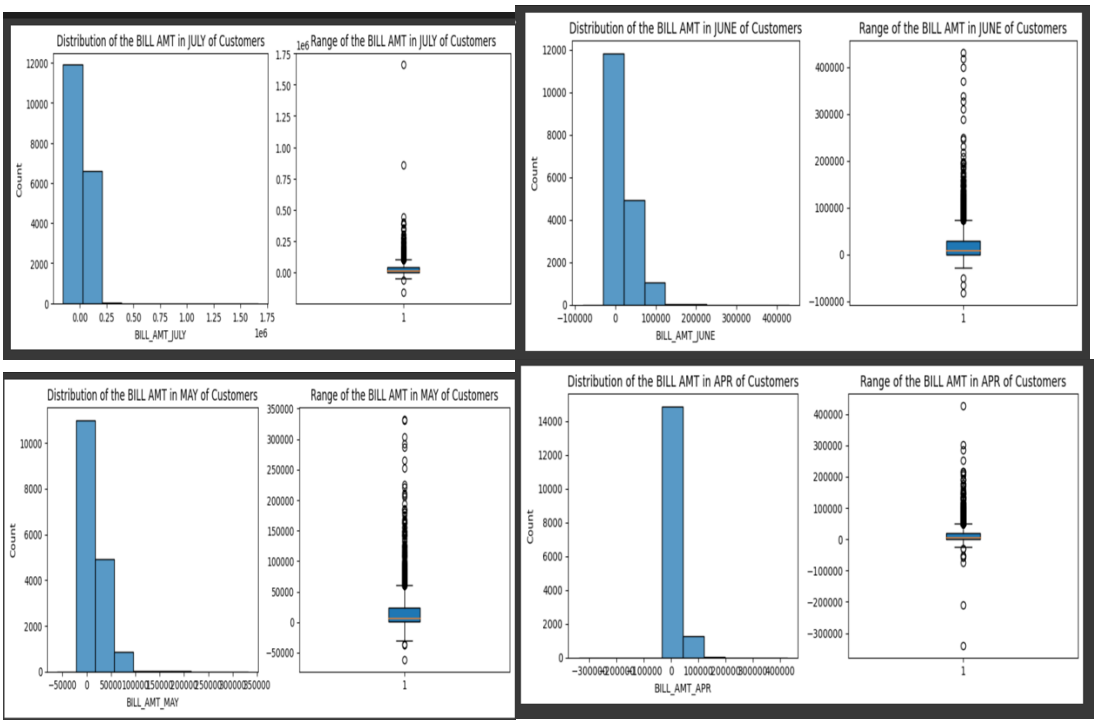

```
#visualizing distribution of REPAY_APR attribute
sb.boxplot(x = 'REPAY_APR', data = credit_card)
plt.tight_layout()
#plt.savefig("hist of REPAY_APR.png")
plt.show()
```




- **BILL_AMT_SEPT, BILL_AMT_AUG, BILL_AMT_JULY, BILL_AMT_JUNE, BILL_AMT_MAY, BILL_AMT_ Variables:**



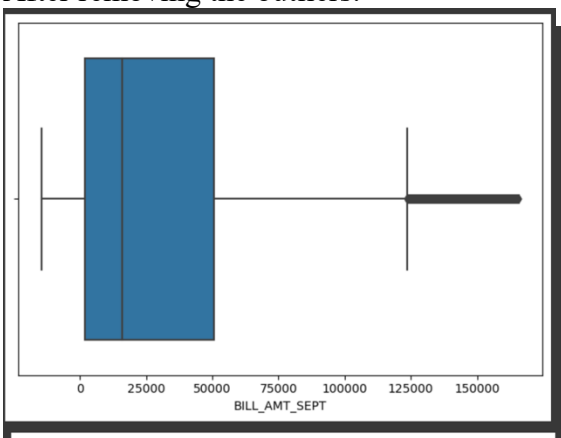



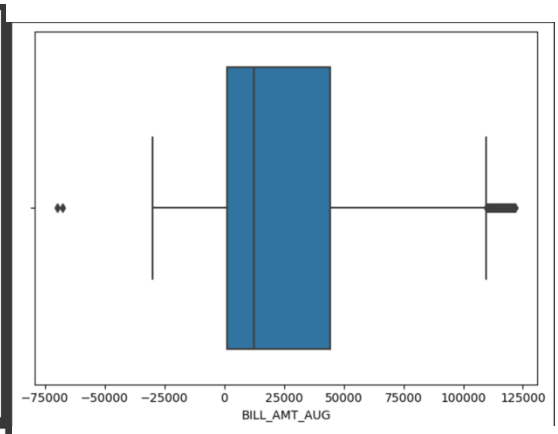


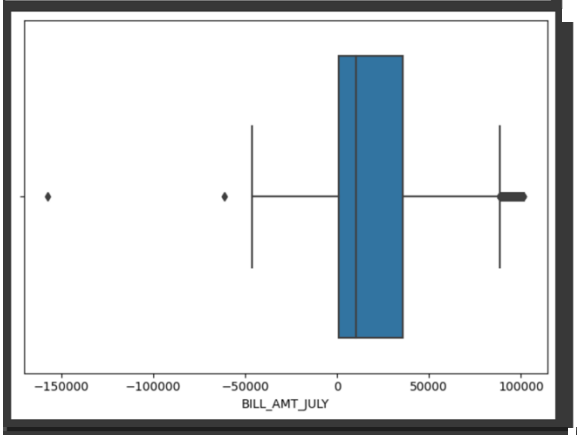

Looking at the boxplots, Outliers were detected in all these attributes, so I removed these outliers by calculating the limits on the variable and taking outliers as observations that fall outside of the range.

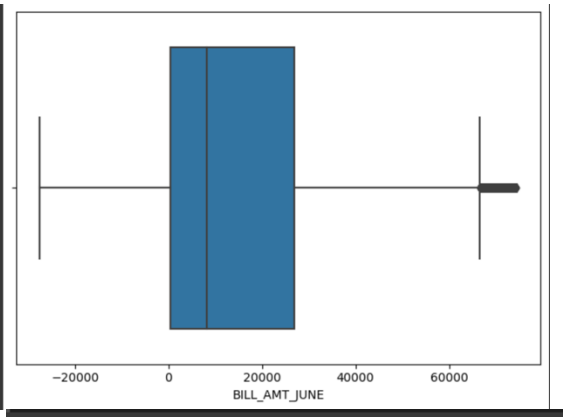
Dealing with these outliers, I used the Quantile method, a strong statistical technique that focuses on particular percentiles of the data distribution to identify and mitigate outliers, to address this. Calculating the lower and upper quantiles—specifically, the lower quantile, or 25th percentile, and the upper quantile, or 75th percentile—allows to establish acceptable ranges for the data. These quantiles mark the boundaries at which data points are no longer regarded as outliers. Potential outliers include any data points that are below Q1 or above Q3. For these variables I removed all the data points that are either more than the provided upper bound or less than the specified lower bound.

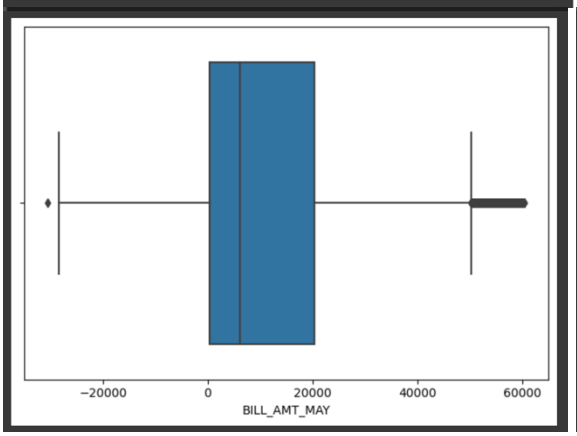
After removing the outliers:

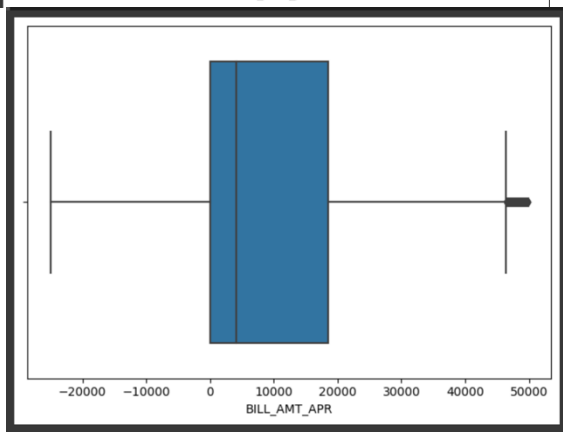


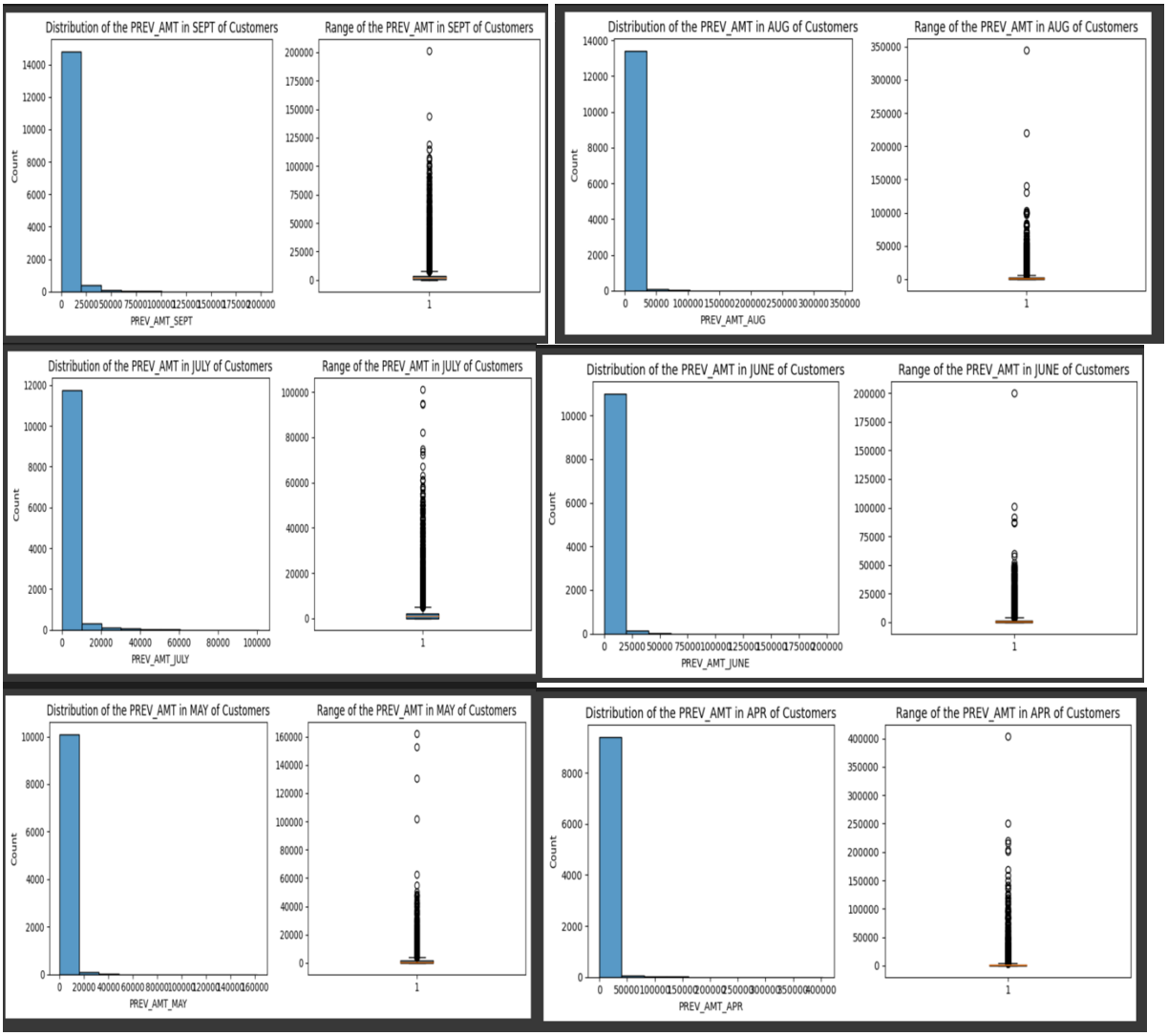



- **PREV_AMT_SEPT, PREV_AMT_AUG, PREV_AMT_JULY, PREV_AMT_JUNE, PREV_AMT_MAY, PREV_AMT_APR Variables**

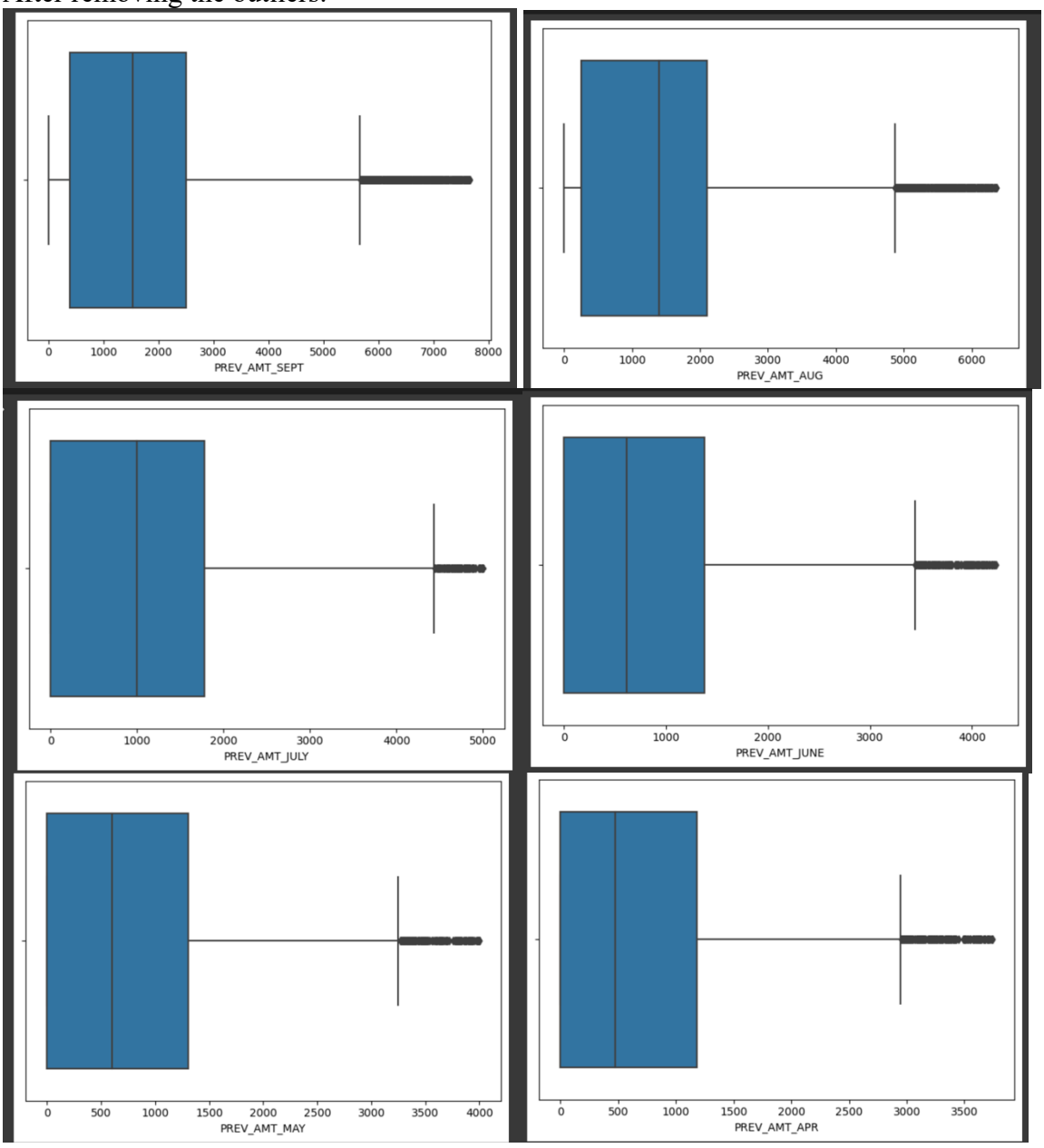



Looking at the boxplots, Outliers were detected in all these attributes, so I removed these outliers by calculating the limits on the variable and taking outliers as observations that fall outside of the range.

Dealing with these outliers, I used the Quantile method, a strong statistical technique that focuses on particular percentiles of the data distribution to identify and mitigate outliers, to address this. Calculating the lower and upper quantiles—specifically, the lower quantile, or 25th percentile, and the upper quantile, or 75th percentile—allows to establish acceptable ranges for the data. These quantiles mark the boundaries at which data points are no longer regarded as outliers. Potential

outliers include any data points that are below Q1 or above Q3. For these variables I removed all the data points that exceeded the upper bound value.

After removing the outliers:



## Correlation and Heat Map:

I used heatmap to depict the relationship and correlation between variables in the dataset.

Correlation between numeric attributes

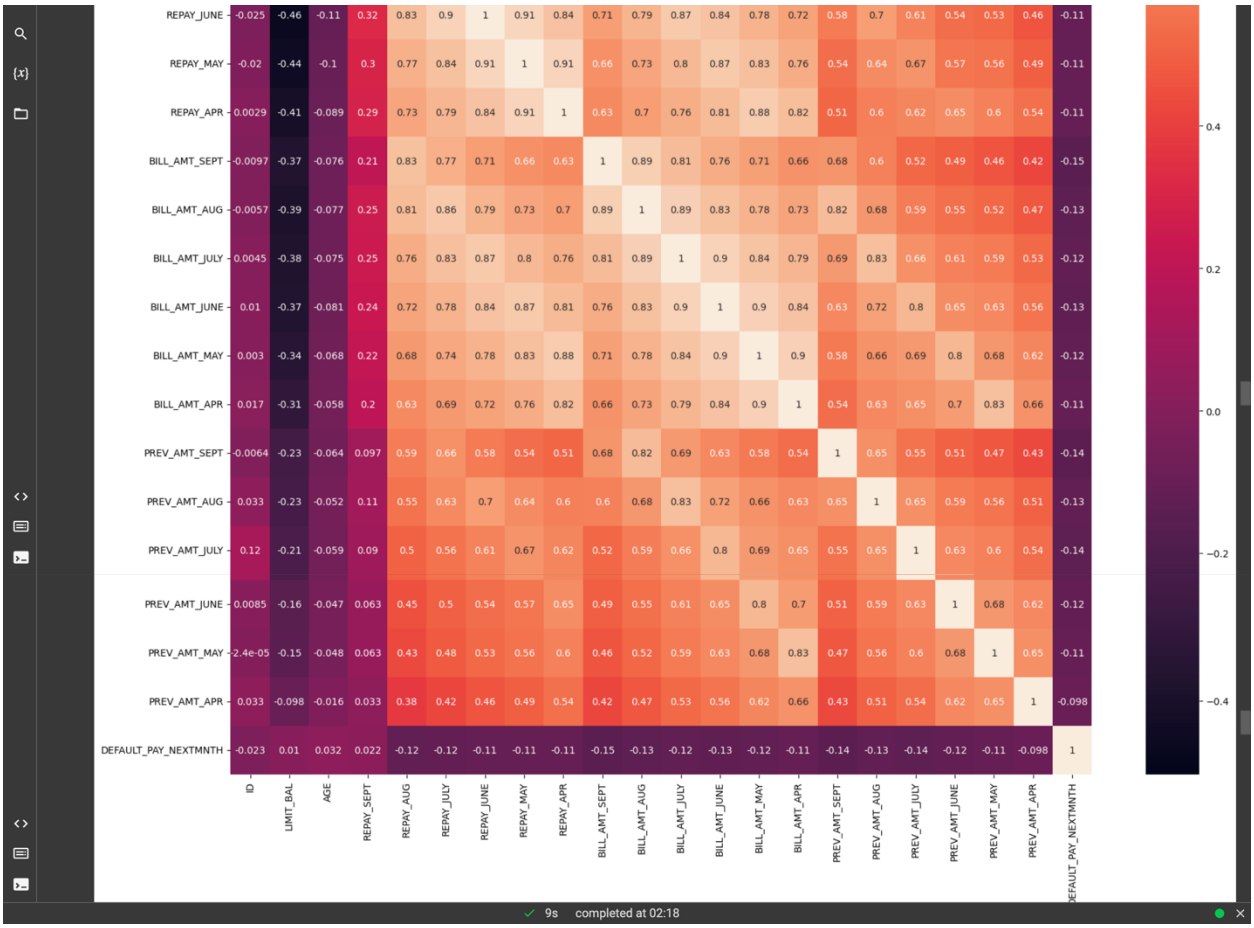
```
credit_card.corr()
```

| | ID | LIMIT_BAL | AGE | REPAY_SEPT | REPAY_AUG | REPAY_JULY | REPAY_JUNE | REPAY_MAY | REPAY_APR | BILL_AMT_SEPT | ... | BILL_AMT_JUNE | BII |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ID | 1.000000 | 0.024292 | 0.012260 | -0.060411 | -0.019008 | -0.020245 | -0.026602 | -0.022702 | -0.000203 | -0.009100 | ... | 0.038920 | |
| LIMIT_BAL | 0.024292 | 1.000000 | 0.124031 | -0.260146 | -0.470329 | -0.455917 | -0.432633 | -0.412859 | -0.387084 | -0.253799 | ... | -0.301429 | |
| AGE | 0.012260 | 0.124031 | 1.000000 | -0.052736 | -0.084830 | -0.080425 | -0.075299 | -0.072806 | -0.058891 | -0.023971 | ... | -0.036697 | |
| REPAY_SEPT | -0.060411 | -0.260146 | -0.052736 | 1.000000 | 0.371708 | 0.376586 | 0.349579 | 0.331710 | 0.318031 | 0.259447 | ... | 0.274310 | |
| REPAY_AUG | -0.019008 | -0.470329 | -0.084830 | 0.371708 | 1.000000 | 0.896775 | 0.814786 | 0.763346 | 0.723617 | 0.605831 | ... | 0.609559 | |
| REPAY_JULY | -0.020245 | -0.455917 | -0.080425 | 0.376586 | 0.896775 | 1.000000 | 0.899875 | 0.829497 | 0.785077 | 0.564244 | ... | 0.646022 | |
| REPAY_JUNE | -0.026602 | -0.432633 | -0.075299 | 0.349579 | 0.814786 | 0.899875 | 1.000000 | 0.912103 | 0.836518 | 0.518386 | ... | 0.667616 | |
| REPAY_MAY | -0.022702 | -0.412859 | -0.072806 | 0.331710 | 0.763346 | 0.829497 | 0.912103 | 1.000000 | 0.905108 | 0.483544 | ... | 0.679896 | |
| REPAY_APR | -0.000203 | -0.387084 | -0.058891 | 0.318031 | 0.723617 | 0.785077 | 0.836518 | 0.905108 | 1.000000 | 0.475390 | ... | 0.665776 | |
| BILL_AMT_SEPT | -0.009100 | -0.253799 | -0.023971 | 0.259447 | 0.605831 | 0.564244 | 0.518386 | 0.483544 | 0.475390 | 1.000000 | ... | 0.744200 | |
| BILL_AMT_AUG | -0.007679 | -0.294366 | -0.020022 | 0.274607 | 0.618802 | 0.647986 | 0.607270 | 0.572572 | 0.559245 | 0.906142 | ... | 0.832475 | |
| BILL_AMT_JULY | 0.010436 | -0.304103 | -0.023063 | 0.274168 | 0.610720 | 0.647932 | 0.661780 | 0.631560 | 0.617699 | 0.821988 | ... | 0.908976 | |
| BILL_AMT_JUNE | 0.038920 | -0.301429 | -0.036697 | 0.274310 | 0.609559 | 0.646022 | 0.667616 | 0.679896 | 0.665776 | 0.744200 | ... | 1.000000 | |
| BILL_AMT_MAY | 0.002614 | -0.290394 | -0.034026 | 0.269454 | 0.598304 | 0.633650 | 0.653137 | 0.673194 | 0.705445 | 0.687881 | ... | 0.918270 | |
| BILL_AMT_APR | 0.009880 | -0.282633 | -0.022872 | 0.256702 | 0.570632 | 0.604207 | 0.623092 | 0.640878 | 0.677038 | 0.649372 | ... | 0.874625 | |
| PREV_AMT_SEPT | -0.010822 | -0.168163 | -0.044045 | 0.080549 | 0.464124 | 0.523236 | 0.440176 | 0.401344 | 0.377306 | 0.488756 | ... | 0.445186 | |
| PREV_AMT_AUG | 0.027431 | -0.175103 | -0.029059 | 0.101259 | 0.448167 | 0.525356 | 0.585655 | 0.519347 | 0.479038 | 0.427915 | ... | 0.517887 | |
| PREV_AMT_JULY | 0.123720 | -0.141771 | -0.037674 | 0.083785 | 0.388758 | 0.436568 | 0.484902 | 0.536495 | 0.487096 | 0.350754 | ... | 0.533949 | |
| PREV_AMT_JUNE | 0.008395 | -0.095740 | -0.026275 | 0.056549 | 0.336552 | 0.375120 | 0.402246 | 0.437967 | 0.497339 | 0.328687 | ... | 0.463045 | |
| PREV_AMT_MAY | -0.002425 | -0.103564 | -0.030410 | 0.060853 | 0.341018 | 0.384329 | 0.421588 | 0.441832 | 0.480710 | 0.318164 | ... | 0.464876 | |
| PREV_AMT_APR | 0.035194 | -0.078563 | -0.008027 | 0.040887 | 0.307031 | 0.341316 | 0.367362 | 0.392777 | 0.433268 | 0.295281 | ... | 0.426282 | |
| DEFAULT_PAY_NEXTMNTH | -0.022861 | 0.028365 | 0.030654 | 0.021095 | -0.122337 | -0.116389 | -0.107522 | -0.107952 | -0.104455 | -0.110819 | ... | -0.105134 | |

22 rows × 22 columns

```
#corelation matrix using pearson method(used for values base), sperman method used for the rank used values
plt.figure(figsize=(20,20))
sb.heatmap(credit_card.corr(method='spearman'), annot=True)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```

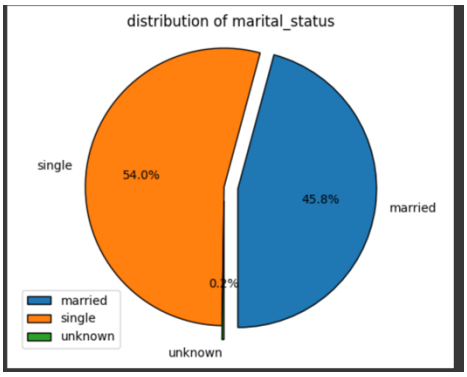



## Visualization of the Categorical Attributes:

- **MARRIAGE**

```
countmarried = credit_card.loc[credit_card["MARRIAGE"]== "1",].shape[0]
countsingle = credit_card.loc[credit_card["MARRIAGE"]== "2",].shape[0]
countunknown = credit_card.loc[credit_card["MARRIAGE"]== "3",].shape[0]
countunknown = credit_card.loc[credit_card["MARRIAGE"]== "0",].shape[0]
print("count of married:", countmarried)
print("count of single:", countsingle)
print("count of unknown:", countunknown)

labels = ['married', 'single', 'unknown']
slices  = [countmarried, countsingle, countunknown]
explode = [0.1,0,0.1]
plt.pie(slices, labels = labels, explode = explode, startangle = -90, autopct='%1.1f%%', wedgeprops={'edgecolor':'black'})
plt.axis("equal")
plt.title("distribution of marital_status")
plt.legend()
plt.show()
```

```
count of married: 3977
count of single: 4684
count of unknown: 21
```

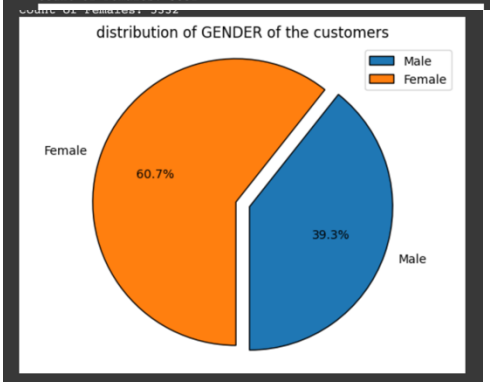



Here, the marital attribute in my dataset provides information about customers marital status. It has a three-category single, married and unknown. This pie chart shows that the customers in Taiwan is having 45.8% customers are married marked as a blue color, 54.0% are single and 0.2% are unknown customers.

- **GENDER**

```
# to make pie plots
countMale = credit_card.loc[credit_card["GENDER"]== "1",].shape[0]
countFemale = credit_card.loc[credit_card["GENDER"]== "2",].shape[0]
print("count of Males:", countMale)
print("count of Females:", countFemale)

labels = ['Male', 'Female']
slices  = [countMale, countFemale]
explode = [0.1,0]
plt.pie(slices, labels = labels, explode = explode, startangle = -90, autopct='%1.1f%%', wedgeprops={'edgecolor':'black'})
plt.axis("equal")
plt.title("distribution of GENDER of the customers")
plt.legend()
plt.show()
```

```
count of Males: 3457
count of Females: 5332
```

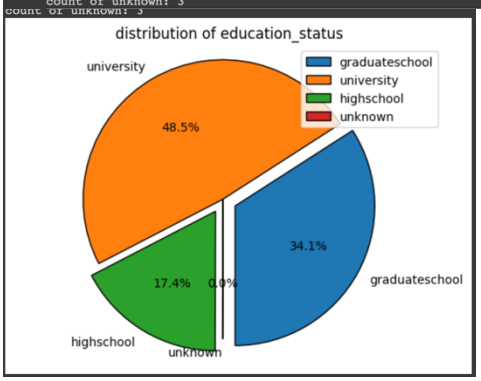



Here, the Gender attribute in my dataset provides information about customers Genders. It has a two-category male and female. This pie chart shows that the customers in Taiwan is having 39.3% customers are males marked as a blue color, 60.7%% are females marked as orange color

- **EDUCATION**

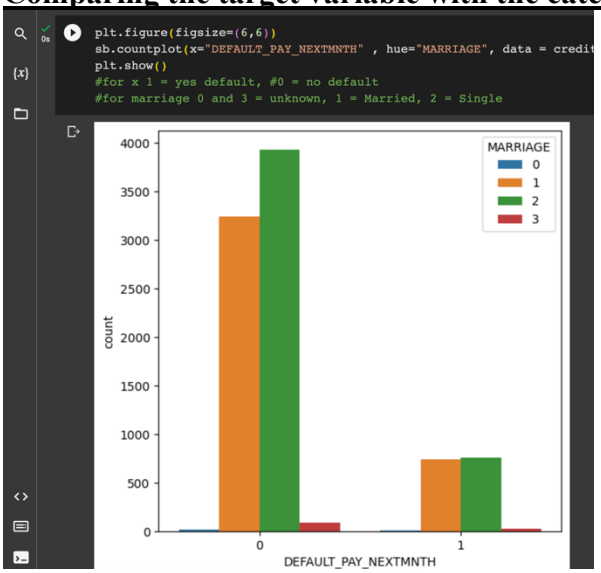
```
countgraduateschool = credit_card.loc[credit_card["EDUCATION"]== "1",].shape[0]
countuniversity = credit_card.loc[credit_card["EDUCATION"]== "2",].shape[0]
counthighschool = credit_card.loc[credit_card["EDUCATION"]== "3",].shape[0]
countunknown = credit_card.loc[credit_card["EDUCATION"]== "4",].shape[0]
countunknown = credit_card.loc[credit_card["EDUCATION"]== "5",].shape[0]
countunknown = credit_card.loc[credit_card["EDUCATION"]== "6",].shape[0]
countunknown = credit_card.loc[credit_card["EDUCATION"]== "0",].shape[0]
print("count of graduateschool:", countgraduateschool)
print("count of university:", countuniversity)
print("count of highschool:", counthighschool)
print("count of unknown:", countunknown)

labels = ['graduateschool', 'university', 'highschool', 'unknown']
slices  = [countgraduateschool, countuniversity, counthighschool, countunknown]
explode = [0.1,0,0.1,0]
plt.pie(slices, labels = labels, explode = explode, startangle = -90, autopct='%1.1f%%', wedgeprops={'edgecolor':'black'})
plt.axis("equal")
plt.title("distribution of education_status")
plt.legend()
plt.show()
```

```
count of graduateschool: 2946
count of university: 4191
count of highschool: 1499
count of unknown: 3
```



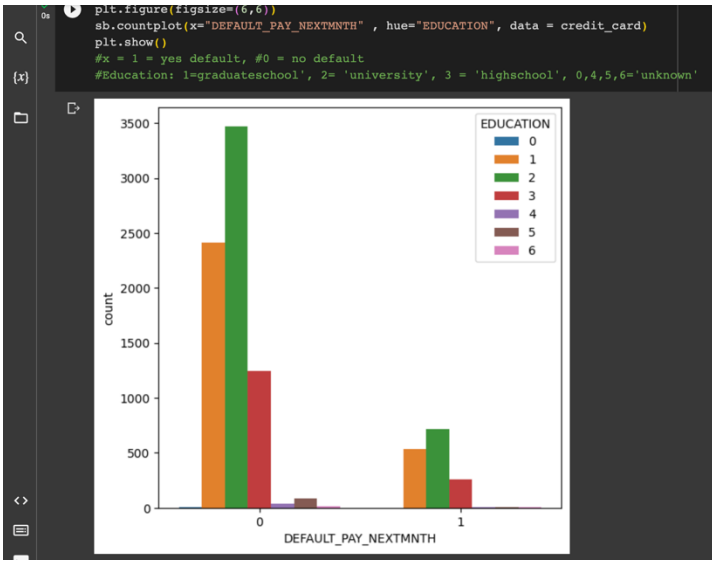

This attribute is on the education status of the customers. It divides into graduate school, university, high school and unknown. 48.5% of customers are having university education and rest of them are having graduate school and high school, while the remaining are marked as unknown.

## Comparing the target variable with the categorical variables




Here, it shows that of the customers who are married, single or their marital status is unknown are likely to default in payment for the next mont .


Here, it differentiate between the genders of the customers who are likely to default in their payment and who are likely not to.




Here, it differentiate between education status of the customers who are likely to default in their payment and who are likely not to.

## 9.0 Summary

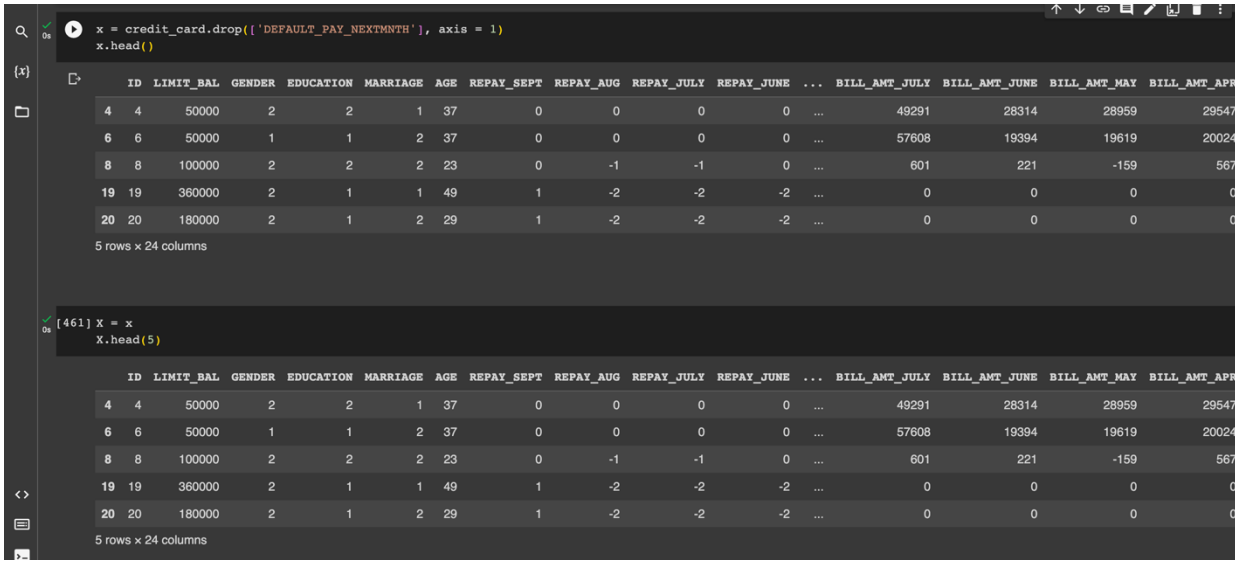
This data investigation has given me a thorough understanding of the data's properties as well as prospective difficulties. I've learned about the dataset's structure, including the amount of rows and columns, data types, and so on. For numerical data, I calculated and examined summary statistics such as mean, median, standard deviation, and quartiles. I examined the data distribution and looked for probable outliers or skewed distributions. Using correlation matrices or heatmaps, I investigated correlations between numerical variables. To obtain insights into data distribution, I produced visualizations including pie charts, bar charts, and box plots.

It's an important stage that led my subsequent actions, allowing me to effectively preprocess and analyze the data to derive significant insights and make educated judgements.

## DATA PREPARATION

### 10.0 Data Preparation Needs

The next step before building the models is to prepare the data for the modeling, so I want to normalize the dataset but before normalizing I first removed the 'DEFAULT_PAY_NEXTMNTH' from the dataset, which is the target variable. In general, we do not want to normalise the target variable, so I separated the target variable from the other feature variables.



```
x = credit_card.drop(['DEFAULT_PAY_NEXTMNTH'], axis = 1)
x.head()
```

| | ID | LIMIT_BAL | GENDER | EDUCATION | MARRIAGE | AGE | REPAY_SEPT | REPAY_AUG | REPAY_JULY | REPAY_JUNE | ... | BILL_AMT_JULY | BILL_AMT_JUNE | BILL_AMT_MAY | BILL_AMT_APR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 4 | 50000 | 2 | 2 | 1 | 37 | 0 | 0 | 0 | 0 | ... | 49291 | 28314 | 28959 | 29547 |
| 6 | 6 | 50000 | 1 | 1 | 2 | 37 | 0 | 0 | 0 | 0 | ... | 57608 | 19394 | 19619 | 20024 |
| 8 | 8 | 100000 | 2 | 2 | 2 | 23 | 0 | -1 | -1 | 0 | ... | 601 | 221 | -159 | 567 |
| 19 | 19 | 360000 | 2 | 1 | 1 | 49 | 1 | -2 | -2 | -2 | ... | 0 | 0 | 0 | 0 |
| 20 | 20 | 180000 | 2 | 1 | 2 | 29 | 1 | -2 | -2 | -2 | ... | 0 | 0 | 0 | 0 |

5 rows × 24 columns

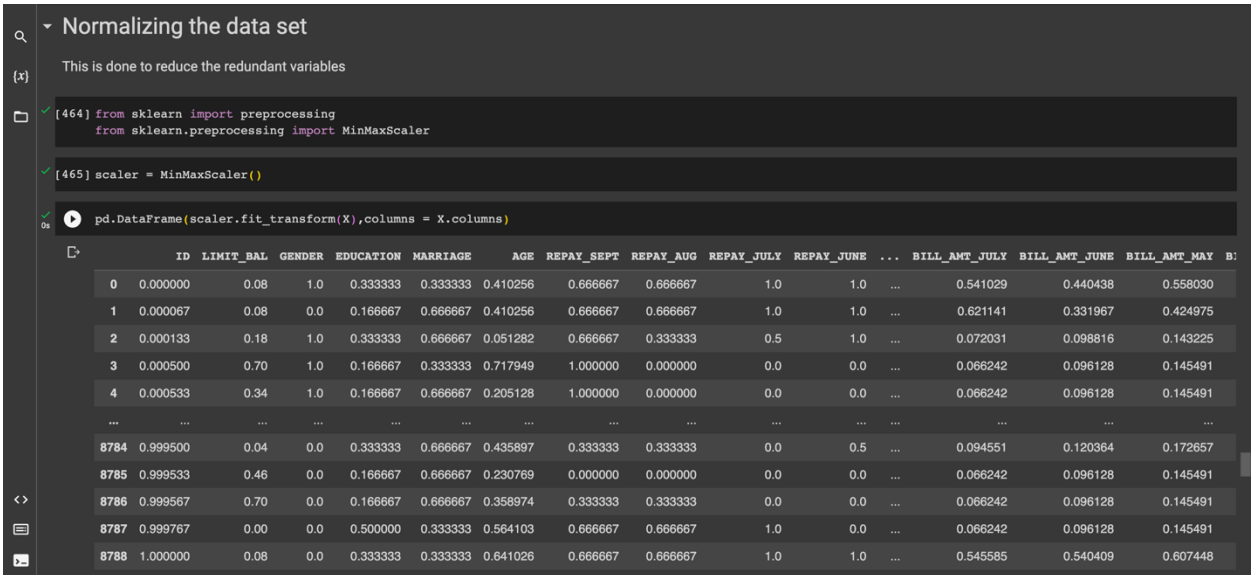
```
[461] X = x
      X.head(5)
```

| | ID | LIMIT_BAL | GENDER | EDUCATION | MARRIAGE | AGE | REPAY_SEPT | REPAY_AUG | REPAY_JULY | REPAY_JUNE | ... | BILL_AMT_JULY | BILL_AMT_JUNE | BILL_AMT_MAY | BILL_AMT_APR |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 4 | 50000 | 2 | 2 | 1 | 37 | 0 | 0 | 0 | 0 | ... | 49291 | 28314 | 28959 | 29547 |
| 6 | 6 | 50000 | 1 | 1 | 2 | 37 | 0 | 0 | 0 | 0 | ... | 57608 | 19394 | 19619 | 20024 |
| 8 | 8 | 100000 | 2 | 2 | 2 | 23 | 0 | -1 | -1 | 0 | ... | 601 | 221 | -159 | 567 |
| 19 | 19 | 360000 | 2 | 1 | 1 | 49 | 1 | -2 | -2 | -2 | ... | 0 | 0 | 0 | 0 |
| 20 | 20 | 180000 | 2 | 1 | 2 | 29 | 1 | -2 | -2 | -2 | ... | 0 | 0 | 0 | 0 |

5 rows × 24 columns

**<u>Normalizing the dataset:</u>**

**This is done to reduce redundant variables.**



Normalizing the data set

This is done to reduce the redundant variables

```
[464] from sklearn import preprocessing
      from sklearn.preprocessing import MinMaxScaler
```

```
[465] scaler = MinMaxScaler()
```

```
pd.DataFrame(scaler.fit_transform(X),columns = X.columns)
```

| | ID | LIMIT_BAL | GENDER | EDUCATION | MARRIAGE | AGE | REPAY_SEPT | REPAY_AUG | REPAY_JULY | REPAY_JUNE | ... | BILL_AMT_JULY | BILL_AMT_JUNE | BILL_AMT_MAY | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.000000 | 0.08 | 1.0 | 0.333333 | 0.333333 | 0.410256 | 0.666667 | 0.666667 | 1.0 | 1.0 | ... | 0.541029 | 0.440438 | 0.558030 | |
| 1 | 0.000067 | 0.08 | 0.0 | 0.166667 | 0.666667 | 0.410256 | 0.666667 | 0.666667 | 1.0 | 1.0 | ... | 0.621141 | 0.331967 | 0.424975 | |
| 2 | 0.000133 | 0.18 | 1.0 | 0.333333 | 0.666667 | 0.051282 | 0.666667 | 0.333333 | 0.5 | 1.0 | ... | 0.072031 | 0.098816 | 0.143225 | |
| 3 | 0.000500 | 0.70 | 1.0 | 0.166667 | 0.333333 | 0.717949 | 1.000000 | 0.000000 | 0.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| 4 | 0.000533 | 0.34 | 1.0 | 0.166667 | 0.666667 | 0.205128 | 1.000000 | 0.000000 | 0.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 8784 | 0.999500 | 0.04 | 0.0 | 0.333333 | 0.666667 | 0.435897 | 0.333333 | 0.333333 | 0.0 | 0.5 | ... | 0.094551 | 0.120364 | 0.172657 | |
| 8785 | 0.999533 | 0.46 | 0.0 | 0.166667 | 0.666667 | 0.230769 | 0.000000 | 0.000000 | 0.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| 8786 | 0.999567 | 0.70 | 0.0 | 0.166667 | 0.666667 | 0.358974 | 0.333333 | 0.333333 | 0.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| 8787 | 0.999767 | 0.00 | 0.0 | 0.500000 | 0.333333 | 0.564103 | 0.666667 | 0.666667 | 1.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| 8788 | 1.000000 | 0.08 | 0.0 | 0.333333 | 0.333333 | 0.641026 | 0.666667 | 0.666667 | 1.0 | 1.0 | ... | 0.545585 | 0.540409 | 0.607448 | |

```
scaled_df = pd.DataFrame(scaler.fit_transform(X),columns = X.columns)
scaled_df
#fit_transform() is used on the training data so that we can scale the training data and also learn the
#scaling parameters of that data.
```

| | ID | LIMIT_BAL | GENDER | EDUCATION | MARRIAGE | AGE | REPAY_SEPT | REPAY_AUG | REPAY_JULY | REPAY_JUNE | ... | BILL_AMT_JULY | BILL_AMT_JUNE | BILL_AMT_MAY | B |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.000000 | 0.08 | 1.0 | 0.333333 | 0.333333 | 0.410256 | 0.666667 | 0.666667 | 1.0 | 1.0 | ... | 0.541029 | 0.440438 | 0.558030 | |
| 1 | 0.000067 | 0.08 | 0.0 | 0.166667 | 0.666667 | 0.410256 | 0.666667 | 0.666667 | 1.0 | 1.0 | ... | 0.621141 | 0.331967 | 0.424975 | |
| 2 | 0.000133 | 0.18 | 1.0 | 0.333333 | 0.666667 | 0.051282 | 0.666667 | 0.333333 | 0.5 | 1.0 | ... | 0.072031 | 0.098816 | 0.143225 | |
| 3 | 0.000500 | 0.70 | 1.0 | 0.166667 | 0.333333 | 0.717949 | 1.000000 | 0.000000 | 0.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| 4 | 0.000533 | 0.34 | 1.0 | 0.166667 | 0.666667 | 0.205128 | 1.000000 | 0.000000 | 0.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| 8784 | 0.999500 | 0.04 | 0.0 | 0.333333 | 0.666667 | 0.435897 | 0.333333 | 0.333333 | 0.0 | 0.5 | ... | 0.094551 | 0.120364 | 0.172657 | |
| 8785 | 0.999533 | 0.46 | 0.0 | 0.166667 | 0.666667 | 0.230769 | 0.000000 | 0.000000 | 0.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| 8786 | 0.999567 | 0.70 | 0.0 | 0.166667 | 0.666667 | 0.358974 | 0.333333 | 0.333333 | 0.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| 8787 | 0.999767 | 0.00 | 0.0 | 0.500000 | 0.333333 | 0.564103 | 0.666667 | 0.666667 | 1.0 | 0.0 | ... | 0.066242 | 0.096128 | 0.145491 | |
| 8788 | 1.000000 | 0.08 | 0.0 | 0.333333 | 0.333333 | 0.641026 | 0.666667 | 0.666667 | 1.0 | 1.0 | ... | 0.545585 | 0.540409 | 0.607448 | |

8789 rows × 24 columns

## Dividing the dataset into train test split:

A key practice in machine learning and data analysis is dividing a dataset into training and testing subsets. The dataset is typically separated into two parts: a training set for training the model's parameters and a test (or validation) set for evaluating the model's performance.

```
Dividing the dataset into train test split

[470] from sklearn.model_selection import train_test_split
      train_input,test_input,train_output,test_output = train_test_split(scaled_df, y ,test_size=0.20,random_state = 3)

[471] train_input.shape

      (7031, 24)

      train_output.shape

      (7031,)

[473] scaled_df.shape[0] *0.80

      7031.200000000001

Checking if the target variable is balanced or not in the training set

[474] train_output.value_counts()

      0    5808
      1    1223
      Name: DEFAULT_PAY_NEXTMNTH, dtype: int64
```

I noticed the target variable is not balanced so I used SMOTE for balancing the target variable. When the target variable is not balanced this might result in skewed model training and poor predictive performance. Balancing the target variable can help address these concerns and improve machine learning model performance overall.

```
from imblearn.over_sampling import SMOTE

[476] SMOTE = SMOTE()

[477] train_input_SMOTE,train_output_SMOTE = SMOTE.fit_resample(train_input,train_output)

[478] print("train_input_SMOTE",train_input_SMOTE.shape,"\n",
          "train_output_SMOTE",train_output_SMOTE.shape,"\n")

      train_input_SMOTE (11616, 24)
       train_output_SMOTE (11616,)


[479] train_output_SMOTE.value_counts()

      0    5808
      1    5808
      Name: DEFAULT_PAY_NEXTMNTH, dtype: int64
```

Now that I have prepared the data, I now moved to the building of the models.

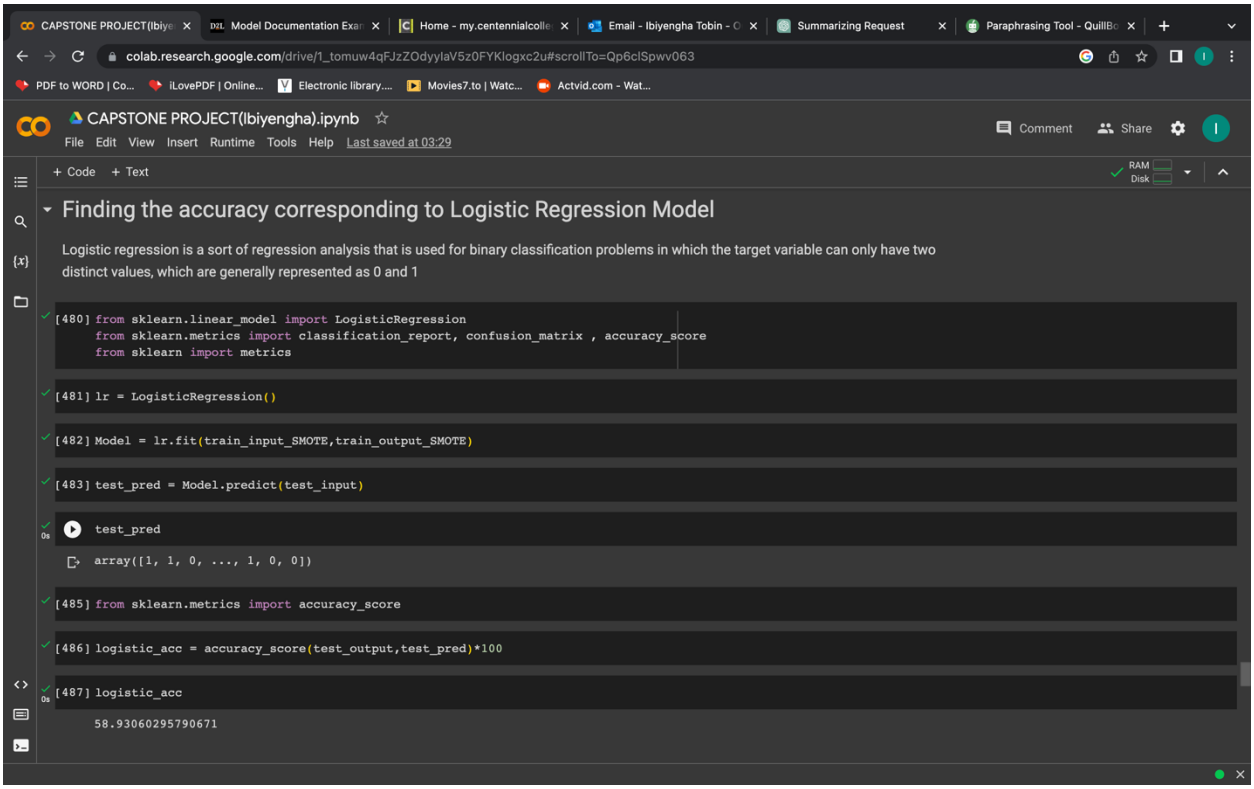
## MODEL EXPLORATION

### 11.0 Modeling Approach/ Introduction

The modelling Approach for this Classification analysis is creating predictive models that allocate input data points to predetermined groupings or categories. Classification is a supervised machine learning problem in which the goal is to learn a mapping from input features to a target class label.

## Logistic Regression Model

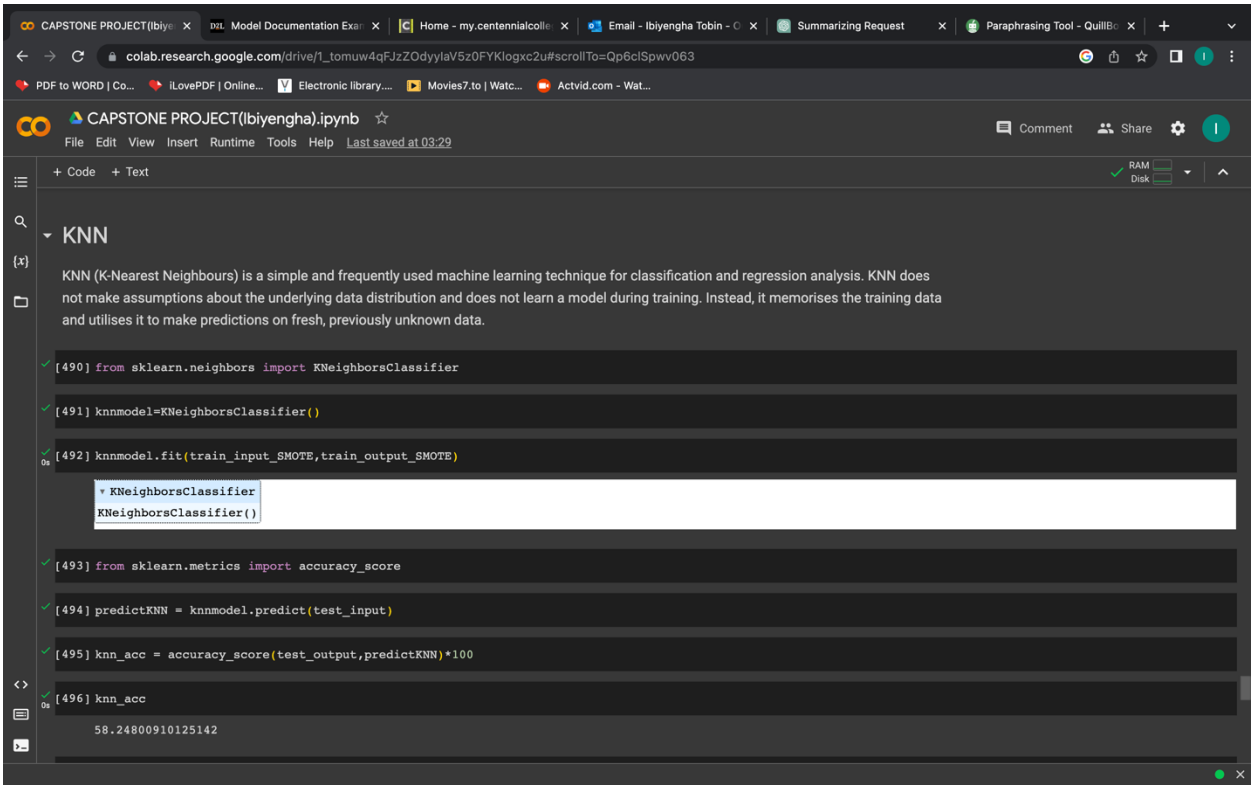
I built a logistic regression model**.** Logistic regression is a sort of regression analysis that is used for binary classification problems in which the target variable can only have two distinct values, which are generally represented as 0 and 1.




**The logistic accuracy was 58.93.**

## KNN Model

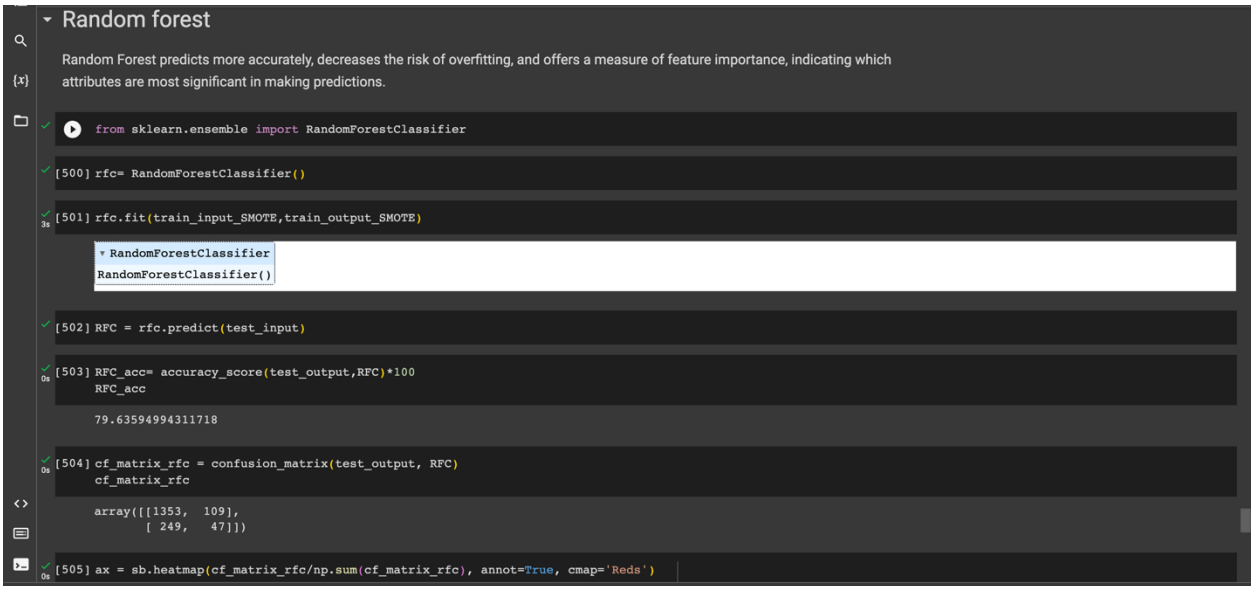
The second model I built was KNN. KNN (K-Nearest Neighbours) is a simple and frequently used machine learning technique for classification and regression analysis. KNN does not make assumptions about the underlying data distribution and does not learn a model during training. Instead, it memorises the training data and utilises it to make predictions on fresh, previously unknown data.



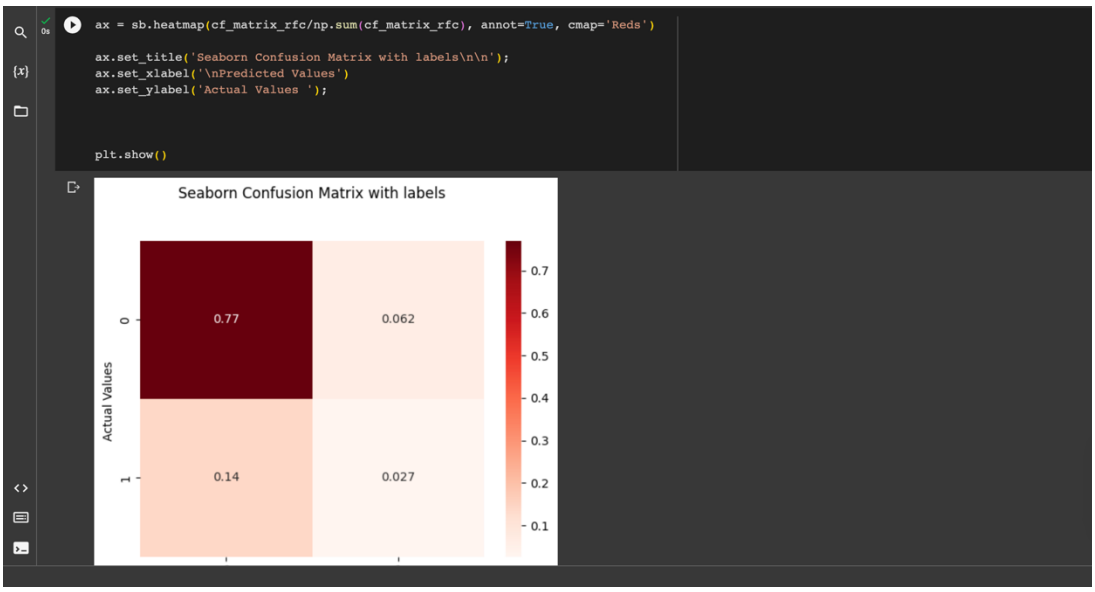

The KNN accuracy was 58.25.

## Random Forest Model

My third model was random forest. Random Forest predicts more accurately, decreases the risk of overfitting, and offers a measure of feature importance, indicating which attributes are most significant in making predictions.




The accuracy was 79.64.

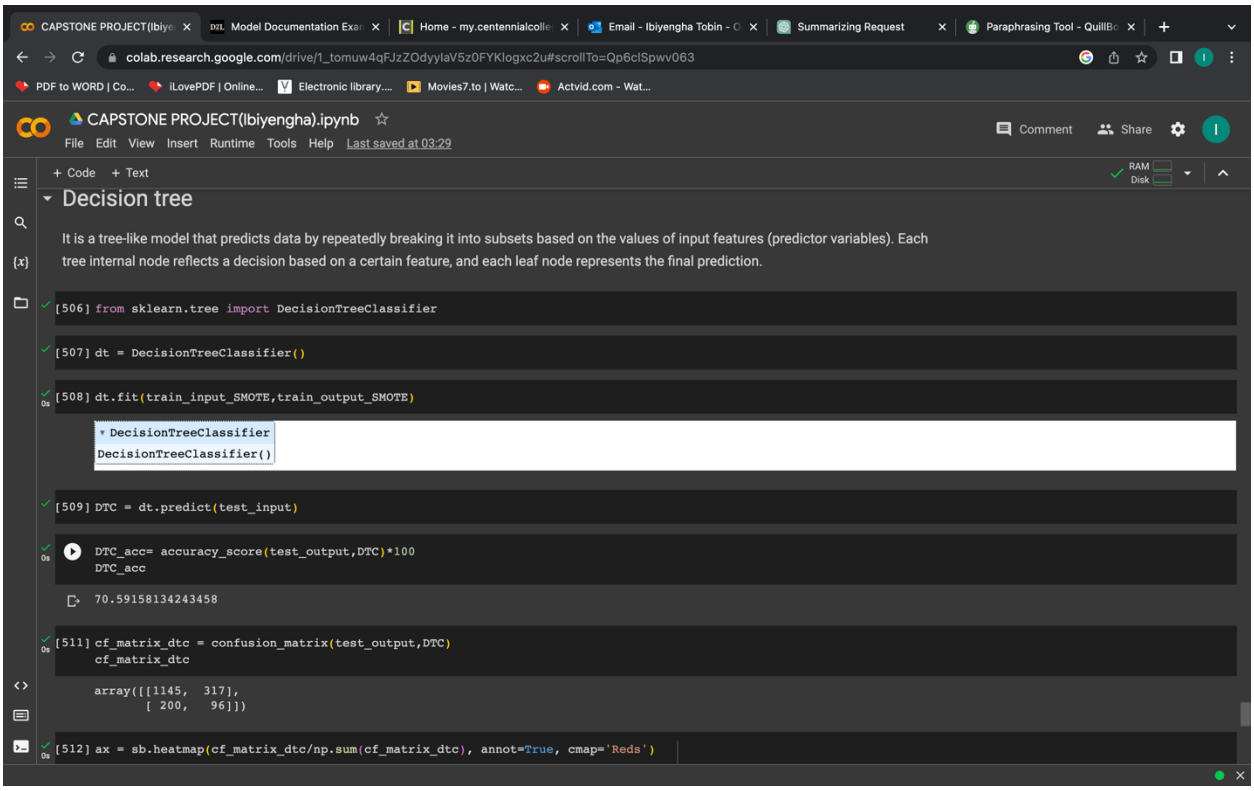



**Confusion Matrix for the Random Forest Model:**

The Random Forest Model correctly predicted 77% No Credit Card default (True Negatives) and correctly predicted 2.7% there would be a default (True Positive). The model predicted 6.2% that there would be a default where there was actually no default (False Positive), and predicted 14% there would not be a default but there was a default (False Negative).

## Decision Tree Model

The last model I built was the Decision Tree. It is a tree-like model that predicts data by repeatedly breaking it into subsets based on the values of input features (predictor variables). Each tree internal node reflects a decision based on a certain feature, and each leaf node represents the final prediction.




The accuracy was 70.6

## 12.0 Model Comparison


Looking at the accuracy table, the best model is the Random Forest. I believe using SMOTE earlier made the accuracy for the random forest better.

I also used K-Fold Cross Validation and Stratified K-Fold Cross Validation to evaluate the effectiveness of this prediction model on credit card default.

## K-FOLD CROSS VALIDATION

K-fold cross-validation is a frequently used technique in machine learning for measuring model performance and minimising overfitting concerns.

```
K_Fold_accuracy = [Acc_knn_kFold, Lr_knn_kFold, RFC_knn_kFold,DTC_knn_kFold]
eval["K_Fold"] = K_Fold_accuracy
eval
```

| | Model | train_test_split | K_Fold |
|---|---|---|---|
| 0 | knn | 58.248009 | 0.803163 |
| 1 | LogisticRegression | 58.930603 | 0.827175 |
| 2 | Random Forest | 79.635950 | 0.825809 |
| 3 | Decision Tree | 70.591581 | 0.692007 |

Looking at this, the logistic regression is the best model(0.827) with the Random forest coming second(0.823).

## STRATIFIED K- FOLD CROSS VALIDATION

An improvement on K-fold that takes into account the class distribution within the dataset is called stratified K-fold cross-validation. Each fold in a stratified K-fold preserves a distribution of classes that is identical to the distribution in the original dataset since the class proportions are retained in each fold.

```
from sklearn.model_selection import cross_val_score, StratifiedKFold

StratifiedKfold = StratifiedKFold(n_splits = 5)

classifiers2 = []
classifiers2.append(KNeighborsClassifier())
classifiers2.append(LogisticRegression())
classifiers2.append(RandomForestClassifier())
classifiers2.append(DecisionTreeClassifier())
from sklearn.model_selection import cross_val_score

accuracy_results2 = []
for a in classifiers2:
    accuracy_results2.append(cross_val_score(a, train_input, train_output, scoring = "accuracy",cv = kFold))

strati_k_fold_Acc = [accuracy_results2[0].mean(), accuracy_results2[1].mean(), accuracy_results2[2].mean(),accuracy_results2[3].mean()]
eval['strati_k_fold'] = strati_k_fold_Acc
eval
```

| | Model | train_test_split | K_Fold | strati_k_fold |
|---|---|---|---|---|
| 0 | knn | 58.248009 | 0.803163 | 0.800312 |
| 1 | LogisticRegression | 58.930603 | 0.827175 | 0.826055 |
| 2 | Random Forest | 79.635950 | 0.825809 | 0.823352 |
| 3 | Decision Tree | 70.591581 | 0.692007 | 0.721802 |

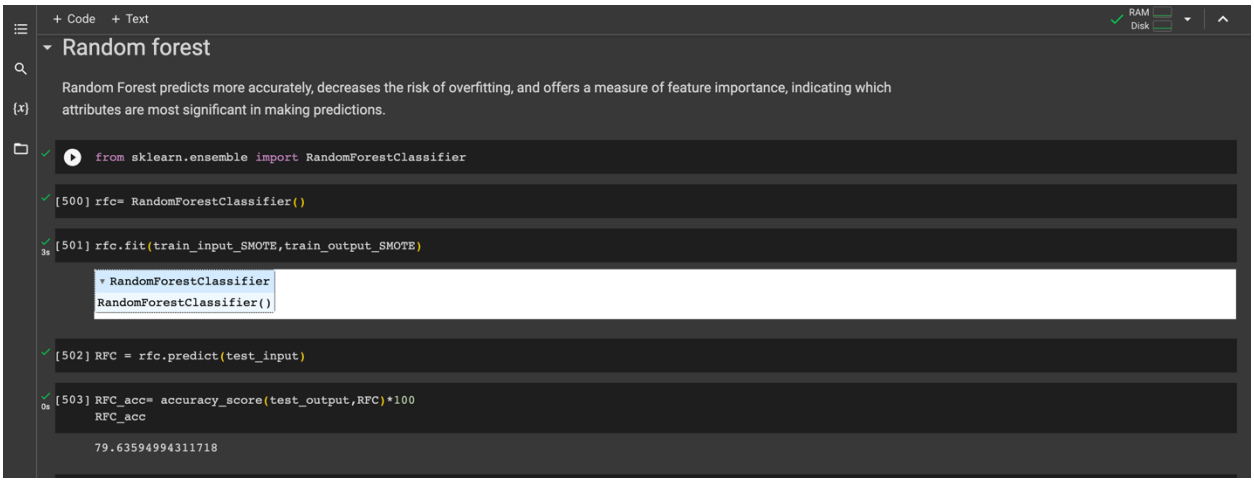
For this the best model is the Logistic Regression(0.826) with the random forest coming in second(0.823).

## MODEL RECOMMENDATION

### 13.0 Model Selection

The model I have selected is the *Random Forest Model*. It has the best accuracy score(79.64) and looking at the K-Fold cross-validation and the stratified K-fold cross-validation it was the second best model, the difference between it and the Logistic Regression was just little.

Random Forest has a reputation for being reliable and consistent between runs and datasets. The ensemble nature of the algorithm is responsible for its stability.




Random Forest is a versatile and powerful algorithm that excels at prediction tasks because to its ensemble nature, robustness, feature importance insights, and capacity to handle a variety of data formats.

Random forest provides insights into feature importance, which will help in knowing the important variables that play a major role in credit card default.

### 14.0 Model Theory

The core notion behind Random Forest is that by combining the predictions of numerous decision trees, the strengths of individual trees are emphasized while their shortcomings (such as overfitting) are reduced. This leads in a more accurate and robust model for this credit card default prediction for the next month.

## 14.1 Model Assumptions and Limitations

It's worth noting that Random Forest's strength resides in its capacity to manage a wide range of data properties without making significant assumptions. However, as with any modelling, it is critical to ensure that the data is appropriately preprocessed, features are picked, and the model's performance is rigorously assessed using techniques such as cross-validation.

The limitation of this model is that after removing the outliers the records dropped to Eight thousand seven hundred and eighty nine records from the thirty thousand records before, therefore this model is only good for data that is Eight thousand seven hundred and eighty nine records or lesser than that.

Before removing outliers:

```
credit_card.describe().T.round(0) #rounding the dataframe to a variable number of decimal places
```

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| ID | 30000.0 | 15000.0 | 8660.0 | 1.0 | 7501.0 | 15000.0 | 22500.0 | 30000.0 |
| LIMIT_BAL | 30000.0 | 167484.0 | 129748.0 | 10000.0 | 50000.0 | 140000.0 | 240000.0 | 1000000.0 |
| AGE | 30000.0 | 35.0 | 9.0 | 21.0 | 28.0 | 34.0 | 41.0 | 79.0 |
| REPAY_SEPT | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_AUG | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_JULY | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_JUNE | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_MAY | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| REPAY_APR | 30000.0 | -0.0 | 1.0 | -2.0 | -1.0 | 0.0 | 0.0 | 8.0 |
| BILL_AMT_SEPT | 30000.0 | 51223.0 | 73636.0 | -165580.0 | 3559.0 | 22382.0 | 67091.0 | 964511.0 |
| BILL_AMT_AUG | 30000.0 | 49179.0 | 71174.0 | -69777.0 | 2985.0 | 21200.0 | 64006.0 | 983931.0 |
| BILL_AMT_JULY | 30000.0 | 47013.0 | 69349.0 | -157264.0 | 2666.0 | 20088.0 | 60165.0 | 1664089.0 |
| BILL_AMT_JUNE | 30000.0 | 43263.0 | 64333.0 | -170000.0 | 2327.0 | 19052.0 | 54506.0 | 891586.0 |
| BILL_AMT_MAY | 30000.0 | 40311.0 | 60797.0 | -81334.0 | 1763.0 | 18104.0 | 50190.0 | 927171.0 |
| BILL_AMT_APR | 30000.0 | 38872.0 | 59554.0 | -339603.0 | 1256.0 | 17071.0 | 49198.0 | 961664.0 |
| PREV_AMT_SEPT | 30000.0 | 5664.0 | 16563.0 | 0.0 | 1000.0 | 2100.0 | 5006.0 | 873552.0 |
| PREV_AMT_AUG | 30000.0 | 5921.0 | 23041.0 | 0.0 | 833.0 | 2009.0 | 5000.0 | 1684259.0 |
| PREV_AMT_JULY | 30000.0 | 5226.0 | 17607.0 | 0.0 | 390.0 | 1800.0 | 4505.0 | 896040.0 |
| PREV_AMT_JUNE | 30000.0 | 4826.0 | 15666.0 | 0.0 | 296.0 | 1500.0 | 4013.0 | 621000.0 |

After removing outliers:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| ID | 8789.0 | 14761.783138 | 8734.481749 | 4.0 | 7225.0 | 14548.0 | 22329.0 | 30000.0 |
| LIMIT_BAL | 8789.0 | 138302.423484 | 112128.754345 | 10000.0 | 50000.0 | 100000.0 | 200000.0 | 510000.0 |
| AGE | 8789.0 | 35.160086 | 9.148149 | 21.0 | 28.0 | 34.0 | 42.0 | 60.0 |
| REPAY_SEPT | 8789.0 | -0.358744 | 0.869231 | -2.0 | -1.0 | 0.0 | 0.0 | 1.0 |
| REPAY_AUG | 8789.0 | -0.686881 | 0.825618 | -2.0 | -1.0 | 0.0 | 0.0 | 1.0 |
| REPAY_JULY | 8789.0 | -0.744112 | 0.849878 | -2.0 | -2.0 | 0.0 | 0.0 | 0.0 |
| REPAY_JUNE | 8789.0 | -0.779952 | 0.868350 | -2.0 | -2.0 | 0.0 | 0.0 | 0.0 |
| REPAY_MAY | 8789.0 | -0.808966 | 0.878630 | -2.0 | -2.0 | -1.0 | 0.0 | 0.0 |
| REPAY_APR | 8789.0 | -0.858118 | 0.885970 | -2.0 | -2.0 | -1.0 | 0.0 | 0.0 |
| BILL_AMT_SEPT | 8789.0 | 19885.815337 | 24916.239166 | -4894.0 | 390.0 | 8710.0 | 33634.0 | 156786.0 |
| BILL_AMT_AUG | 8789.0 | 17386.557401 | 22228.452918 | -5174.0 | 181.0 | 5889.0 | 29122.0 | 120651.0 |
| BILL_AMT_JULY | 8789.0 | 15166.401070 | 19648.076560 | -6877.0 | 0.0 | 4160.0 | 26602.0 | 96940.0 |
| BILL_AMT_JUNE | 8789.0 | 12562.975424 | 16294.868563 | -7905.0 | 0.0 | 2799.0 | 21090.0 | 74329.0 |
| BILL_AMT_MAY | 8789.0 | 10762.910115 | 14222.446558 | -10213.0 | 0.0 | 1848.0 | 19324.0 | 59984.0 |
| BILL_AMT_APR | 8789.0 | 9899.770395 | 13710.677169 | -11610.0 | 0.0 | 1261.0 | 18694.0 | 49756.0 |
| PREV_AMT_SEPT | 8789.0 | 1498.433952 | 1370.400117 | 0.0 | 208.0 | 1443.0 | 2007.0 | 7662.0 |
| PREV_AMT_AUG | 8789.0 | 1337.986574 | 1228.751277 | 0.0 | 0.0 | 1300.0 | 2000.0 | 6336.0 |
| PREV_AMT_JULY | 8789.0 | 1007.410172 | 1023.246556 | 0.0 | 0.0 | 933.0 | 1583.0 | 5000.0 |
| PREV_AMT_JUNE | 8789.0 | 806.743429 | 887.869381 | 0.0 | 0.0 | 595.0 | 1210.0 | 4186.0 |

## 15.0 Model Sensitivity to Key Drivers

When compared to individual decision trees or other algorithms, Random Forest models are often more robust and less susceptible to the selection of key drivers.

Sensitivity varies depending on data qualities and modelling settings, however the ensemble aspect of Random Forest often helps to limit the influence of highly influential features.

## CONCLUSION AND RECOMMENDATIONS

### 16.0 Impacts on Business Problem(Scope of the recommended Model)

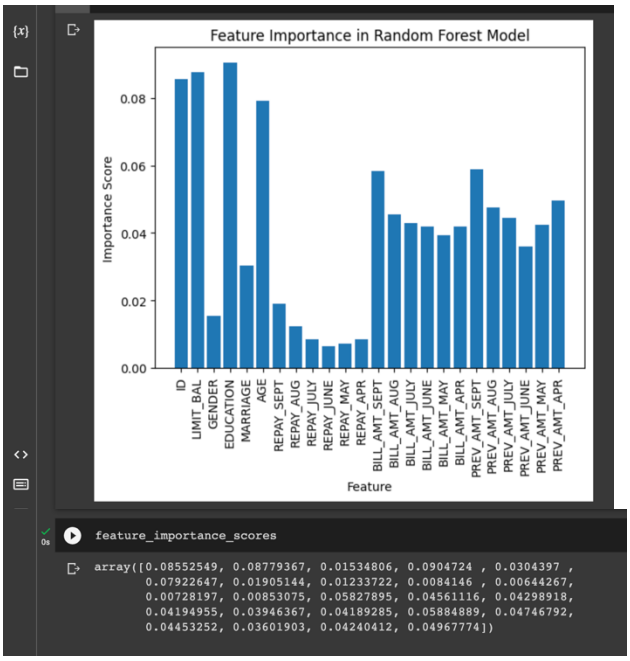
To discuss this, I will analyze the feature importance of the selected model (Random Forest)

Feature Importance for the selected model(Random Forest)

```
feature_importance_scores = rfc.feature_importances_
```

```
[542] feature_names = list(train_input_SMOTE.columns)
```

```
import matplotlib.pyplot as plt

# Create a bar chart of feature importance scores
plt.bar(range(len(feature_importance_scores)), feature_importance_scores)
plt.xticks(range(len(feature_importance_scores)), feature_names, rotation='vertical')
plt.xlabel('Feature')
plt.ylabel('Importance Score')
plt.title('Feature Importance in Random Forest Model')
plt.show()
```




```
feature_importance_scores
```

```
array([0.08552549, 0.08779367, 0.01534806, 0.0904724 , 0.0304397 ,
       0.07922647, 0.01905144, 0.01233722, 0.0084146 , 0.00644267,
       0.00728197, 0.00853075, 0.05827895, 0.04561116, 0.04298918,
       0.04194955, 0.03946367, 0.04189285, 0.05884889, 0.04746792,
       0.04453252, 0.03601903, 0.04240412, 0.04967774])
```

The most important variable that influences the likelihood of a customer defaulting in credit card payment for the next month is Education (0.0905). So the education status is an important factor that should be looked at in predicting credit card default. Amount of given credit(0.0855)and the Age(0.0792) of the customers are also important factors that predict the credit card default of a customer.

## 17.0 Recommended Next Steps

1. Deployment and surveillance:

The trained Random Forest model should be deployed in a production setting, making that the financial institutions can handle batch processing or real-time predictions as needed. They shoul put in place monitoring procedures to keep tabs on the model's operation and identify any potential changes in the data's distribution or accuracy over time.

2. Managing Risk and Making Decisions:

They should consider incorporating the model's projections into the financial institution's decision-making procedures for processing applications, determining credit limits, and controlling risk exposure.
Achieve a compromise between minimising false positives (approving excellent customers) and false negatives (missing true defaulters) when defining thresholds for forecasting credit card defaults.

3. Typical Model Upkeep:

To make sure the model is correct and up to date, update it frequently when new data becomes available. Re-evaluate the model's performance and, if necessary, think about retraining, particularly if the distribution of the underlying data changes.

4. Communication and Cooperation:

The financial institution should Encourage cooperation among data scientists, subject-matter specialists, and business stakeholders to guarantee that the model complies with regulatory standards and commercial objectives. Inform the appropriate stakeholders of the model's advantages, drawbacks, and ramifications.

**REFERENCES**


⇒ UC Irvine Machine Learning Repository (2016, January 1) Default of credit card clients

- https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients

⇒ Xiang Gao et al 2021 J. Phys.: Conf. Ser. 1828 012122

- https://iopscience.iop.org/article/10.1088/1742-6596/1828/1/012122/pdf

⇒ Equifax Canada Market Pulse Quarterly Credit Trends Report (2022, Sept 6) *Financial Stress Mounts, Credit Card Demand and Debt Rise*

- https://www.consumer.equifax.ca/about-equifax/press-releases/-/blogs/financial-stress-mounts-credit-card-demand-and-debt-rise/