Credit Card Default Detection by analysing various features using Mathematical and Statistical tools in Python.

Divyanshu
21103104

List of Figures

Contents

Abstract

Before lending the Credit card or loans to individuals, banks and non-banking financial institutions have to make sure that these clients will be able the pay back the loans within the timeline. To increase payback ratio, these institutions make use of various methods including data mining concepts which can analyze the data of clients and can predict the chance of the client to default.
Here the dataset from kaggle.com, which contains information of different individuals. First data mining concepts is used to convert this raw dataset in complete dataset. After analyzing the data set statistically logistic regression technique is applied to train and test our model.
This project will provide step by step guide from the mathematics of logistic regression algorithm to the analysis of statistical results of dataset and machine learning model.

Problem Statement

Credit Card Default Detection by analysing various features using Mathematical and Statistical tools in Python.

Chapter 1

Introduction to Data mining

1.1 Introduction

Modern science and engineering use first-principle models to describe biological beings, physical systems and social systems around us, such as Maxwell Equations and Newton's laws of motions. This is useful, but they have its limitations. These principles have rigid constraints, but sometimes it is hard to find the first principle in the real world, or satisfying those constraints is impossible. We can use probability theory and statistics to tackle this problem to understand the relationship between different variables. To use these mathematical principles, we need data.

From academic scholars to business enterprises, everyone understands the value of data. Academic scholars collect data to predict the experimental results, and business enterprises collect data to understand customer behaviour and increase the bottom line. These collected datasets are raw so not of any use directly. First, this data is converted into the form a computer can understand. In the advancement in computer hardware and software, we can run many models with just one click and extract information from the data. This process is known as data mining. The definition is given below.
"**Data mining is the process of applying computational methods to large amounts of data in order to reveal new non-trivial and relevant information.**" (11)

In practice, we use data mining to achieve two goals: prediction and description. In predictions, we use data mining to make models from known data that can predict the unknown data or future data with mathematics and machine learning concepts. In the description, we try to analyze the patterns in the dataset to understand the dataset and draw conclusions which are helpful in the particular domain.

1. **Predictive Data-mining**
   Used to make a model from the given dataset.
2. **Descriptive Data-mining**
   To gauge non-trivial information from the given dataset.

Data mining can be used in a variety of fields. The primary tasks of data mining are given below.

1. **Regression**
   Discovery of predictive learning function, which is used to map data with a real valued predictive variable.
2. **Classification**
   The Discovery of predictive learning function, which is used to classify the data into multiple classes.
3. **Clustering**
   A standard descriptive task which can be done to identify finite clusters to describe the data.
4. **Summarization**
   Techniques that summarise the whole data into a compact form.
5. **Dependency modelling**
   A model which can find the dependency of the variables and can perform statistical analysis.
6. **Change and deviation detection**
   Finding outliers and serious changes in data.

1.2 Fundamental of modelling

Data mining is a combination of mathematics, computer science and control theory engineering. So to understand data mining, the interaction of these fields is necessary to understand. As we have seen, data mining does not deal with the first-order principles, so dependent and independent variables are used to understand this unknown mathematical system. This process is called system identification which is a top-down approach and involves two steps.

1. **Structure identification**
   To understand this step, we need some prior knowledge about the target systems and class of models within which we can find a suitable model.

   In this step, domain knowledge about the field is required to determine the class of models so we can get the most suitable model for our system. This class of models is represented by a parameterized function $y = f(u, t)$, where y is the model's output, u is the input vector, and t is parameter vector. To determine the function f we depend on mathematical boundaries, first principles and the person's experience with the particular system.
2. **Parameter identification**
   After defining the structure model, we need optimization to determine parameters vector t such that $y^* = f(u, t^*)$ can describe the system properly.

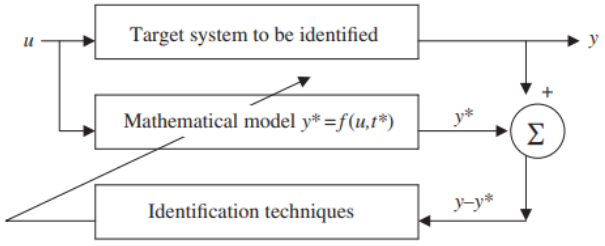
The algorithm and flowchart of system identification is given below:

1. **Algorithm** (12)
   (a) Specify a class of formalized models and parameterize it, $y^* = f(u, t)$.

(b) Identify the parameter and choose the parameters that will best fit the data set (the difference $y - y^*$ is minimal).

(c) To see if the model identified response correctly to an unseen data set or not that check through validation tests.

(d) If the results of the validation test are as expected, stop the process.



flow chart

1.3 Steps of Data mining

1. **Clear problem statement and hypothesis**
   Having a clear problem statement is essential in data mining because, without a clear problem statement, data mining will give general results which are not applied to any particular phenomena. Domain knowledge is required to extract information through a dataset, e.g. working on credit card default detection, and background knowledge of the financial system's functioning is required. From a clear problem statement and domain knowledge, we can formulate a concrete hypothesis which can be checked using mathematics and machine learning algorithms.

2. **Collect the data**
   This step includes the generation and collection of datasets. We can generate data in two ways. One is by doing a control experiment or by observing. Many free datasets are available on Kaggle and other platforms that can be used in data mining concepts to make models. We also should consider the quality of the data because if there are so many null or the dataset is random, we cannot conclude out of them.

3. **Data pre-processing**
   While gathering the dataset, it is never the case that we get the data incomplete form in which we can directly apply mathematical calculations. We have to do operations in the dataset to make it complete. This process commonly has two steps

   (a) **Detecting outliers and removing them.**
   Outliers are the unusual data points which do not match with the overall dataset. This data point can skew the model and results. The following two strategies are applied to dealing with the outliers.

      i. First detect and then remove the outliers from the dataset.

      ii. Use a model which has not very sensitive to the outliers.

(b) **Encoding, scaling and selecting features**

In scaling, if one feature has a range [0,1] and another has a range [1,200] will not have the same weight on applied techniques. Their impact is not the same on the results, so we normalize all the features to the range between [0,1].

Suppose we have variable values, such as M and F or other object classification. We cannot run the model directly on this, so we first assign each value with a different numerical, e.g. M as 0 and F as 1.

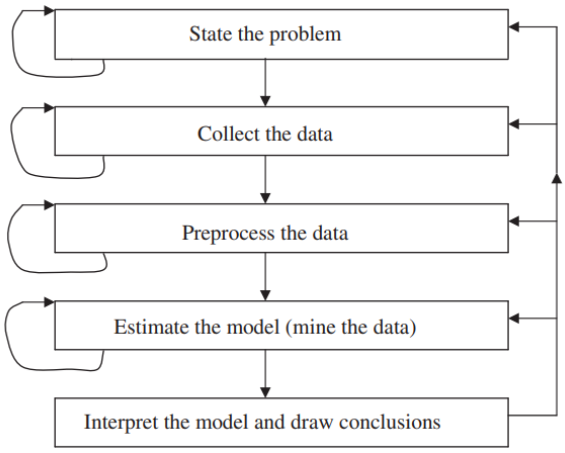
Each feature has not had the same impact on the datasets in feature selection. So we have to select those feature only which has a greater impact on the target feature and remove those feature from other features. We can use correlation for this process.

(c) **Estimate the model**

This is the most important step in data mining process. We use several machine learning models to get the results in these processes. Selecting models depends upon the dataset and problem statement. Multiple models can be applied to one problem to select the best model through the performance.

(d) **Interpret the model and make conclusions**

Data mining is used for decision-making purposes. So after selecting the estimated model, different performance matrices are used to understand the model's logic and dependability, e.g. confusion matrix, accuracy score, and f1 score.



Data mining steps

Chapter 2

Logistic Regression

2.1 Introduction

Logistic Regression is one of the most popular and widely used supervised machine learning algorithms used in Classification problems such as this one, where we need to predict discrete values (0 and 1), unlike the case in Linear Regression, where we deal with continuous values. E.g., To check whether a tumour is malignant or benign (Binary Classification), hand-written digit recognition (Multi-Class Classification), etc.

2.2 Logistic Regression and Linear Regression

Linear Regression works best when the dependent and the independent variables have a linear relationship. This linear function, also called the 'hypothesis function,' is represented as $h_\theta(x) = \theta_0 + \theta_1 x = \theta^T x$, where θ_0 is the intercept along the y-axis and θ_1 is the slope. This algorithm uses the concept of drawing a 'best-fitting line' passing through all the data points such that the cost function is minimum. The cost function, also known as the mean squared error function, is represented as

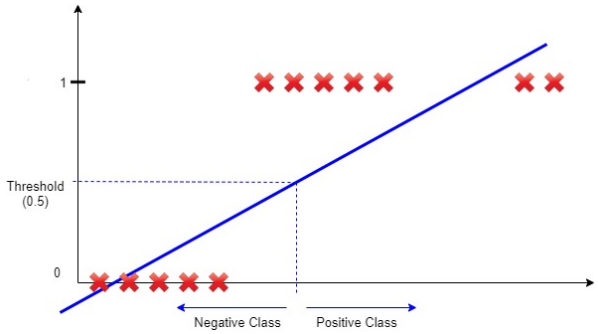
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2 \tag{2.1}$$

where m is the total number of training examples, $h_\theta(xi)$ is the predicted value, and y_i is the actual value. This function is minimized using optimizing techniques such as the Gradient Descent. In the following paragraph, we will see why this algorithm might fail to predict accurate discrete values when dealing with classification problems and how Logistic Regression uses an S-shaped curve which works better than the linear function.

One of the foremost problems with using Linear Regression with classification problems is that this algorithm predicts continuous values, while we want to have discrete values as the final answer. This can still be modified to get discrete values from Linear Regression by taking a threshold value, say $y = 0.5$, to classify output. For any value of x, if $y(x)$ is greater than 0.5 we classify it as positive class (1) and

below 0.5 as negative class (0) (See Fig. 2.1). At first glance, this method seems reasonable and has the potential to work, but we start witnessing problems in the presence of outliers. The best fitting line gets severely deviated, and the aforementioned strategy starts giving anomalous results (See Fig. 2.2). Another problem is that since Linear Regression gives continuous values as results, it can very possibly give answers for $h_\theta(x)$ which are less than 0 and greater than 1, the above strategy failing yet again. Hence, Linear Regression is not usually preferred.

Linear Regression 1



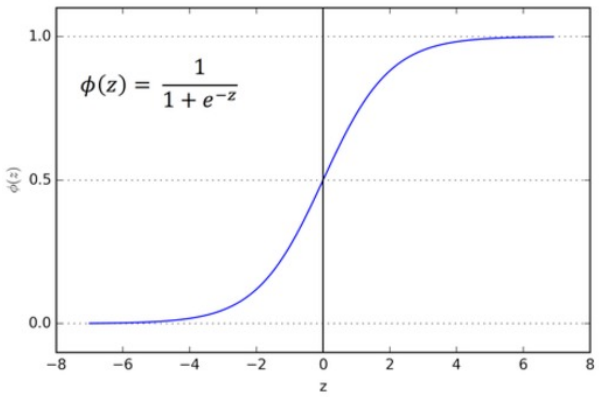
Linear Regression 2

2.3 Logistic Regression

We aim for a function which gives $0 \leq h_\theta(x) \leq 1$ as the result. The hypothesis function $h_\theta(x) = \theta^T x$ in Linear Regression is modified by making the right side a function of g.

Hence, the hypothesis function for Logistic Regression $h_\theta(x) = g(\theta^T x)$, where $g(z) = 1/(1 + e^{-z})$ is called the sigmoid function or the logistic function. Therefore, $h_\theta(x) = 1/(1 + e^{-\theta^T x})$.

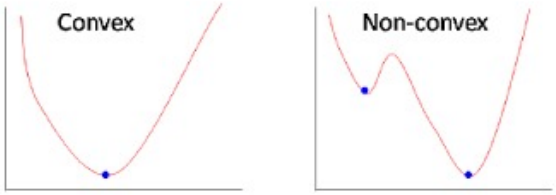
This sigmoid function is an S-shaped curve and has the range $(0, 1)$ (having asymptotes at 0 and 1), which now is convenient for our classification problem where $y = \{0, 1\}$. The value of $h_\theta(x)$ gives the estimated probability of $y = 1$ for any input x. E.g., for any particular value of x, $h_\theta(x) = 0.7$ means there is 70% chance of y being of positive class. We now say that for values of $h_\theta(x) \geq 0.5, y = 1$ is predicted, and for $h_\theta(x) < 0.5, y = 0$. (See Fig. 2.4) Now, we rewrite the cost function for the Linear Regression as



Logistic Regression

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}[h_\theta(x^{(i)}) - y^{(i)}]^2 \tag{2.2}$$

$$Cost(h_\theta(x), y) = \frac{1}{2}[h_\theta(x^{(i)}) - y^{(i)}]^2 \tag{2.3}$$



Convex and Non-convex

This, however, results in a non-convex cost function (See Fig. 4). The function has various local minima, and hence applying Gradient Descend will be a huge problem as it won't be guaranteed if it will converge to the global minima. In contrast, we would want a convex cost function, a single bow shaped function, where applying Gradient Descent would guarantee a global minima. Hence we define another cost function for Logistic Regression which is convex and will be used from hereon forward.

$$Cost(h_\theta(x), y) = \left\{ \begin{array}{ll} -\log(h_\theta(x)), & \text{if } y = 1 \\ -\log(1 - h_\theta(x)), & \text{if } y = 0 \end{array} \right\} \tag{2.4}$$

This can be written together in a compact way as

$$Cost(h_\theta(x), y) = -y\log(h_\theta(x)) - (1-y)\log(1 - h_\theta(x)) \tag{2.5}$$

Now, putting this value of $Cost(h_\theta(x), y)$ in 2.2, we have

$$
\begin{aligned}
J(\theta) &= \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) \\
&= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] \\
&= -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} Cost(h_\theta(x), y)
\end{aligned}
\tag{2.6}
$$

Now, to fit parameter theta, we apply Gradient Descent on this $J(\theta)$ to get the global minima at that particular theta and minimize the function. Henceforth, for any new value of x, the predicted value will be $h_\theta(x) = 1/(1 + e^{-\theta^T x})$, where $h_\theta(x)$ is the probability of $y = 1$.

Chapter 3

Solution to the problem using Python

The steps to solution of problem include:

1. Loading Python libraries and data set.
2. Cleaning of data.
3. Optimization.
4. Visualization.
5. Application of Statistical model.

3.1 Pre-processing of dataset

Converting raw data into clean data is called Pre-processing of data.

3.1.1 Importing libraries and loading data

1. **Libraries**
   First import Pandas and **NumPy** libraries.
   Use Pandas to load the data from its location address.
2. **Memory reduction**
   The memory usage of the Data frame is **286.2+ MB** which is too big and can create a hindrance in execution.
   For all the datatypes present in this dataset, i.e., object, int and float, all the values in a particular column were taken into consideration and according to the minimum and maximum value of that column, the minimum bit signed integer or float was assigned to it. Repeating this for all the columns significantly reduces the memory usage.

   As a result, memory gets reduced to 92.38 MB (Decreased by 67.7%).
3. **df.shape()**
   gives the number of rows and columns in the data set which is **(307511, 122)**

4. **df.info()**
   Provides the name of each column and their respective data types.
5. **df.describe()**
   Gives statistical details of each column such as total count, mean value and other statistical information.

3.1.2 Dealing with duplicate and missing data

Duplicate values

1. **df.duplicated().sum()**
   will give the total number of duplicate data in the data set. In this particular data set, there is no duplicate data hence returns 0.

Steps for working with missing data

1. Identify missing data
   (a) **df.isnull().sum()**
   Gives total null entries in each column.There are **9152465** null values in all the columns.
2. Deal with missing data
   (a) Drop the whole column if the number of null entries is too high
   (b) Replace the null entries with the mean value by taking the mean of the column.
3. Correct data format

3.1.3 About Target variable

- 1 - client with payment difficulties.
- 0 - all the other cases.

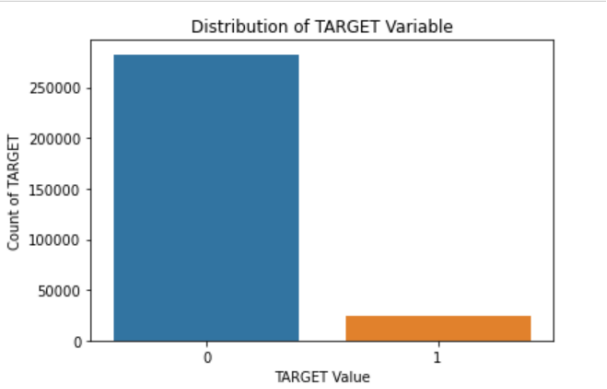
df['TARGET'].value_counts() gives the total counts of 1's and 0's which is

- $0 - 282686$
- $1 - 24825$

The Percentage of clients who have paid their loan on time is: 91.93 %
The Percentage of clients who have NOT paid their loan is: 8.07 %
The Ratio of Data Imbalance is 11.39.

This suggests a large number of applicants have paid the loan back on time.
This can also be seen in graphical form using : **sns.countplot(df['TARGET'])**

Conclusion There were way more loans repaid on time compared to the loans not paid on time. More than 250000 loans were repaid, and less than 50000 loans were not repaid or Or we can say that this data is **highly imbalanced**.



Target Variable Distribution

3.1.4 Working on handling missing data

Working on each data type separately

- Objects
- Integers
- Floats

Objects

1. **df.select dtypes('object').columns**
   Separating Object data type.
2. **df[object cols].isnull().sum()**
   Searching for null values in these columns.(only 2 columns have null values in this case.)
3. **print(col, df[col].nunique())**
   To find number of unique entries in every column using for col in object cols. So later label encoding can be done (converting string entries to numbers i.e 1,2,3..etc).

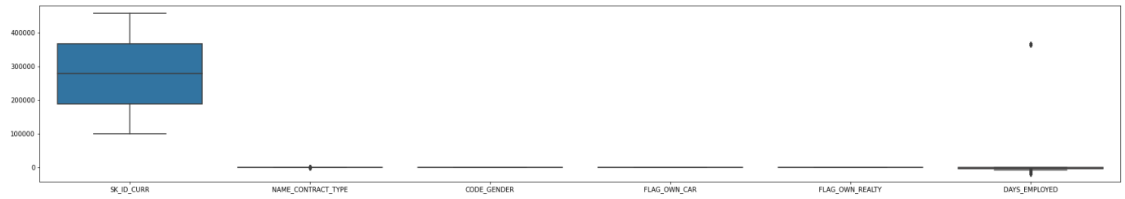
- **Label encoding**
  First the null values in this column is replaced by the mode of the column.
  df['CODE GENDER'] = le.fit transform(df['CODE GENDER']) will convert genders i.e Male, Female into numbers 1,2 respectively.

  Now after this process there is no null entry left in any of the object type columns.

Integer data type

1. **df.select dtypes(include=['int8', 'int16', 'int32', 'int64']).columns**
   This includes all the object data type columns which are converted into integers.
   This dataset contains total 45 columns.
2. **To detect outliers**
   plt.figure(figsize=(30,5))

sns.boxplot(data=df.select dtypes('int'))
plt.show()



Outliers detection

In this set of columns, the column 'DAYS EMPLOYED' is found to contain an outlier. It is found that a total of 55374 rows contain the entry 365243.

Column description

How many days before applying for loan the person started their current employment? We can see

the outlier here with the value 365243. These were probably NaN values (since all other values are negative and this is the only one positive) and got replaced by 365243, so we convert these to NaN and then fill with the mean of other values in the column.

The entry 365243 days is irrelevant as it counts to more than 1000 years.

We replace this value with NaN using
df['DAYS EMPLOYED'] = df['DAYS EMPLOYED'].replace(365243, np.nan)
And then replace the NaN values with calculated mean value of the column using
df['DAYS EMPLOYED'].fillna(df['DAYS EMPLOYED'].mean(), inplace=True)

3. **HEATMAP**
   plt.figure(figsize=(20,10))
   sns.heatmap(df[int cols].corr(), cmap='Blues', vmin=-1, vmax=1)

4. **Inferences**

   (a) FLAG_OWN_CAR and CODE_GENDER are somewhat correlated.

   (b) DAYS_BIRTH is highly correlated with CNT_CHILDREN, DAYS_EMPLOYED, DAYS_ID_PUBLISH,and FLAG_EMP_PHONE.

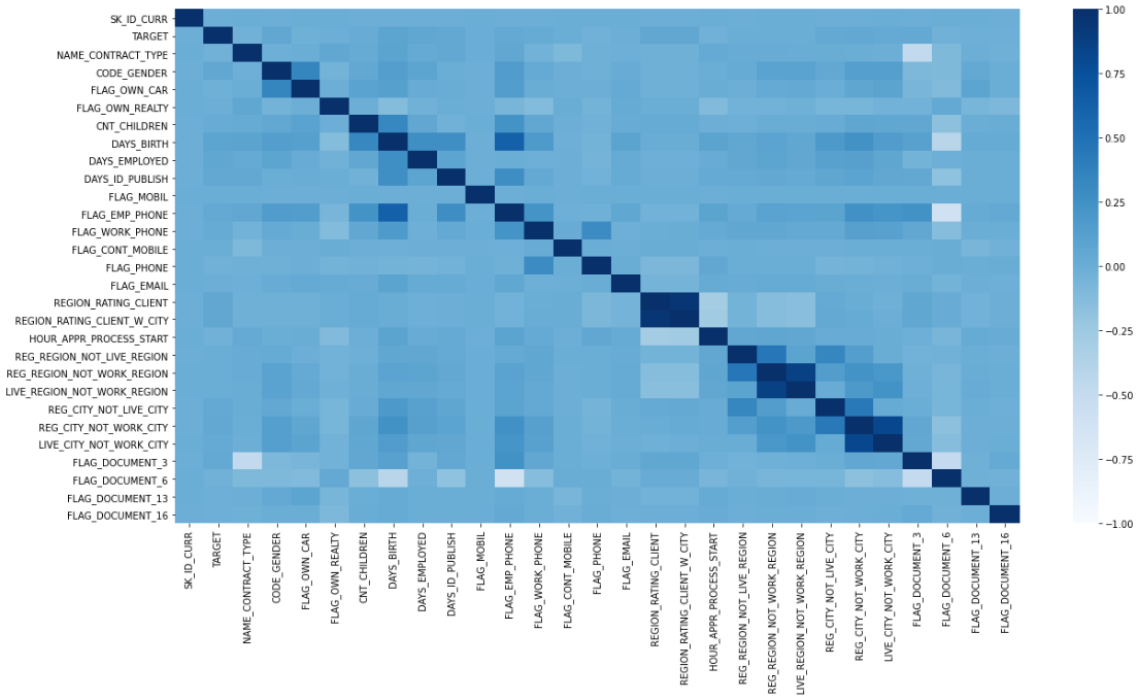
   (c) REGION_RATING_CLIENT and REGION_RATING CLIENT_W_CITY are extremely correlated.

   (d) FLAG_DOCUMENT_6 is very less correlated with DAYS_BIRTH and FLAG_EMP_PHONE.

Float data type

1. **df[float cols].isnull().sum().sort values(ascending= False)**
   Gives 34 columns with null entries and the total count of null entries.

2. **Interpolation**
   The float data type columns contain numerous null entries in every column so to optimize this forward interpolation is used.
   df.interpolate(method ='linear', limit_direction ='forward', inplace=True)



Heatmap 1

3. **HEATMAP**
   Heat map of float value columns using:
   plt.figure(figsize=(20,10))
   sns.heatmap(df[float_cols].corr(), cmap='Greens', vmin=-1, vmax=1)

4. **Inferences**

   (a) AMT_GOODS_PRICE and AMT_CREDIT are very highly correlated.

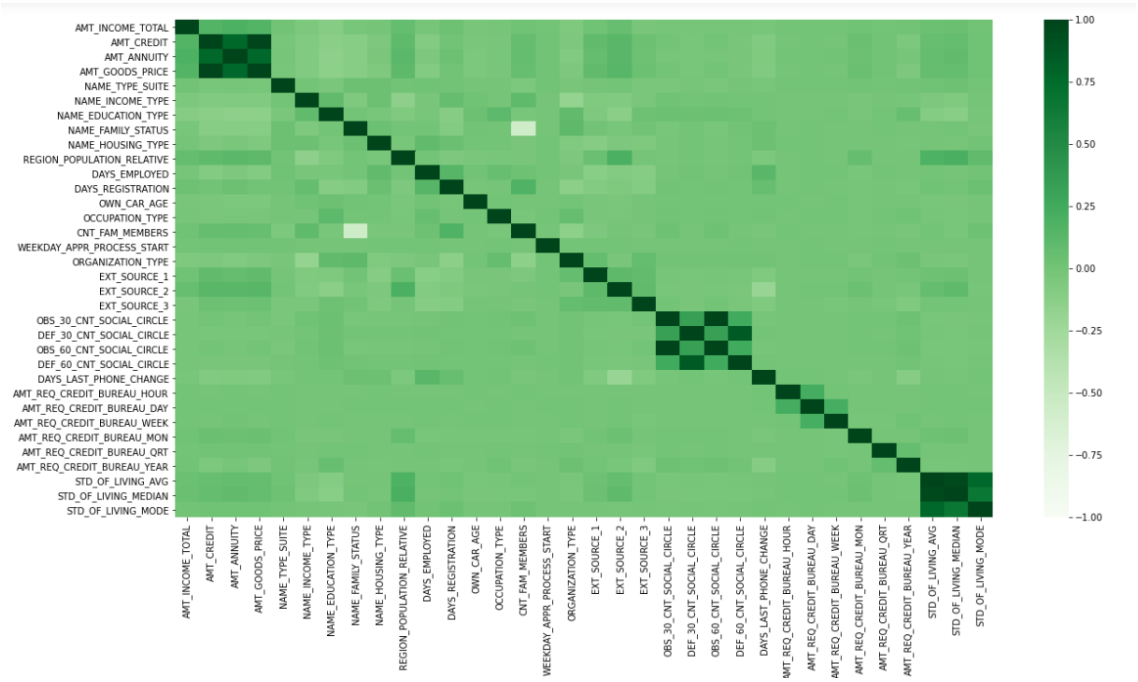
   (b) AMT_ANNUITY is also highly correlated with AMT_CREDIT and AMT_GOODS_PRICE.

   (c) OBS_30_CNT_SOCIAL_CIRCLE, DEF_30_CNT_SOCIAL_CIRCLE, OBS_60_CNT_SOCIAL_CIRCLE, and DEF_30_CNT_SOCIAL_CIRCLE are correlated among each other.

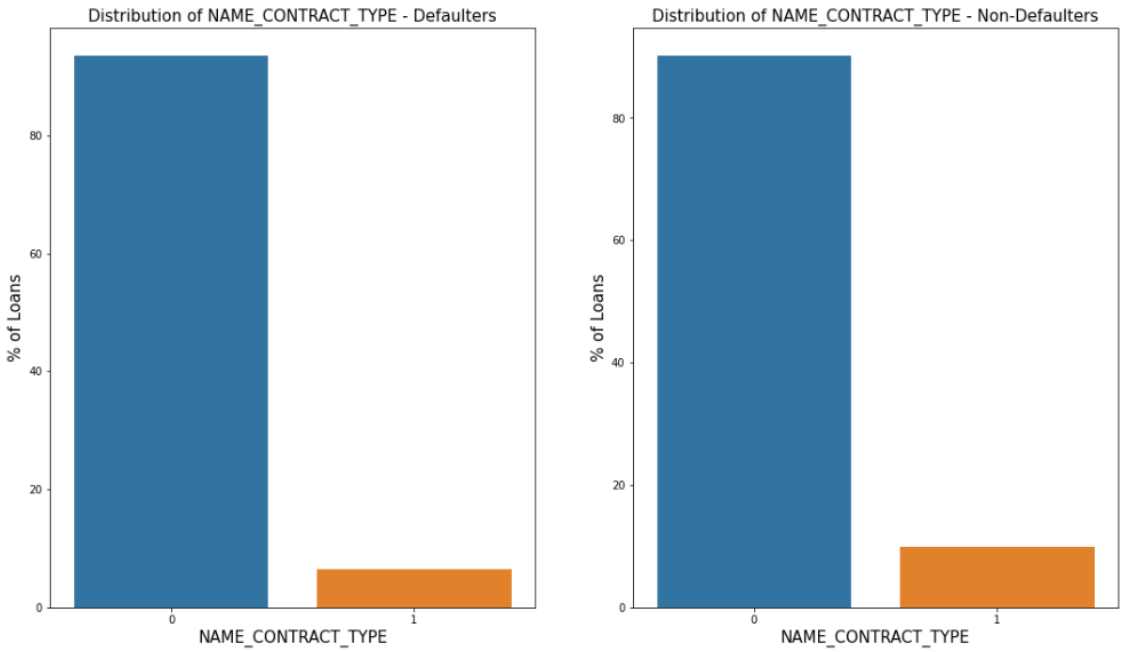
   (d) STD_OF_LIVING_AVG, STD_OF_LIVING_MEDIAN, and STD_OF_LIVING_MODE are extremely correlated with each other.

   (e) CNT_FAM_MEMBERS is less correlated with NAME_FAMILY_STATUS.

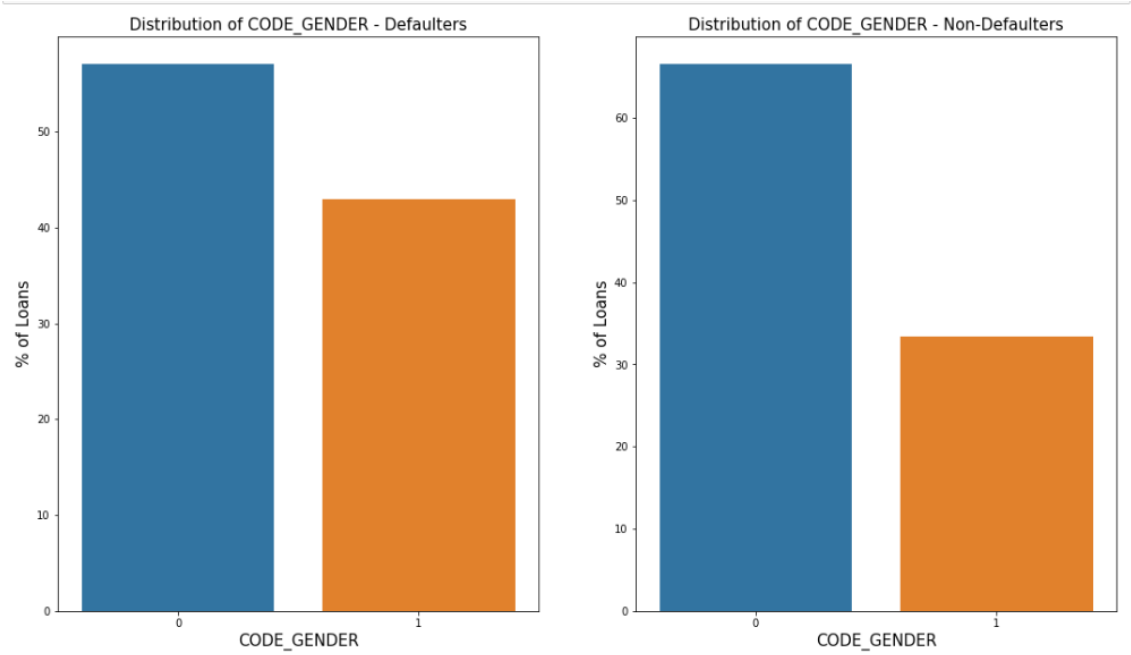
3.2 Visualization



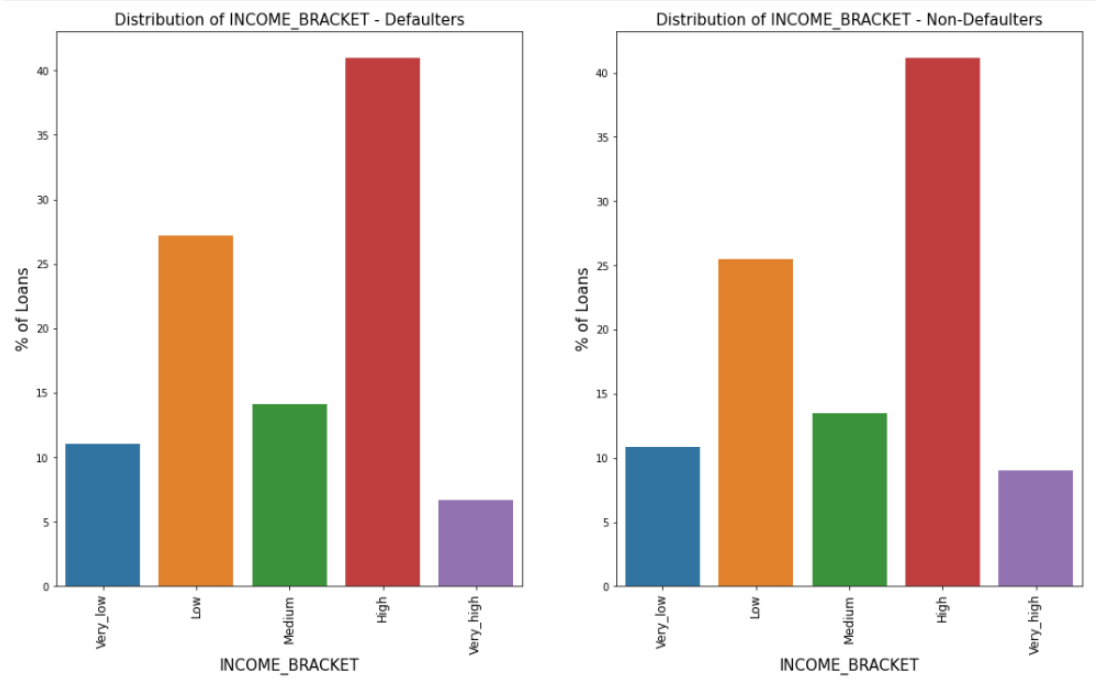
Heatmap 2



NAME_CONTRACT_TYPE



CODE_GENDER



INCOME_BRACKET

Chapter 4

Implementation of Code in Python

4.1 Statistical analysis and Data cleaning

Here after understanding all the data mining steps we have applied those concepts in Python, find below the code in notebook form.

Importing necessary libraries and loading data

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[ ]:
```
pd.set_option('display.max_columns', 122)
pd.set_option('display.max_rows', 150)
```

[ ]:
```
df = pd.read_csv(r'C:\Users\surya\Desktop\Jupyter Python Programs\Mini Project -␣
↪Data Mining\application_data.csv')
df.head()
```

[ ]:
```
df1 = df.copy()
```

Reducing memory usage by dataframes

[ ]:
```
df.info()
```

[ ]:
```
# Refrence - https://www.kaggle.com/gemartin/load-data-reduce-memory-usage

# Other references -
# 1. https://www.analyticsvidhya.com/blog/2021/04/
↪how-to-reduce-memory-usage-in-python-pandas/#:~:
↪text=This%20can%20reduce%20memory%20usage,to%20the%20range%20of%20values.
```

```
# 2. https://www.geeksforgeeks.org/pandas-memory-management/#:~:
 ↪text=Memory_usage()%3A,labels%20present%20in%20the%20Index.
 ↪&text=However%2C%20Info()%20only%20gives,of%20each%20column%20in%20bytes.

def reduce_mem_usage(df):
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).
 ↪max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).
 ↪max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).
 ↪max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).
 ↪max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.
 ↪float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('object')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))
```

```
    return df
df = reduce_mem_usage(df)
```

Data Summary

[ ]:
```
df.shape
```

[ ]:
```
df.info(verbose=True)
```

[ ]:
```
df.describe()
```

[ ]:
```
# get the count, size and Unique value in each column of application data
df.agg(['count','size','nunique'])
```

[ ]:

[ ]:

Checking for duplicate records and null values

[ ]:
```
df[df.duplicated()]
```

[ ]:
```
df.duplicated().sum()
```

[ ]:
```
df.isnull().sum().sum()
```

[ ]:
```
df.isnull().sum().sort_values(ascending=False)
```

[ ]:
```
round(100*(df.isnull().sum()/len(df)),2).sort_values(ascending=False)
```

[ ]:
```
plt.figure(figsize = (30,15))   # For adjusting size of heat map, ref = https://
 ↪stackoverflow.com/questions/38913965/
 ↪make-the-size-of-a-heatmap-bigger-with-seaborn

sns.heatmap(df.isnull(), yticklabels=False, cbar=False)
```

[ ]:

[ ]:

Distribution of TARGET variable

[ ]:
```
df['TARGET'].value_counts()
```

[ ]:
```
defaulters = df[df['TARGET']==1]
nondefaulters = df[df['TARGET']==0]
```

[ ]:
```
percentage_defaulters = (len(defaulters)*100)/len(df)
percentage_nondefaulters = (len(nondefaulters)*100)/len(df)

print("The Percentage of people who have paid their loan is:",␣
 ↪round(percentage_nondefaulters,2),"%")
print("The Percentage of people who have NOT paid their loan is:",␣
 ↪round(percentage_defaulters,2),"%")
print("The Ratio of Data Imbalance is:", round(len(nondefaulters)/len(defaulters),2))
```

[ ]:
```
import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

sns.countplot(df['TARGET'])
plt.xlabel("TARGET Value")
plt.ylabel("Count of TARGET")
plt.title("Distribution of TARGET Variable")
plt.show()
```

Conclusion - There are far more loans that were repaid on time than loans that were not repaid. More than 25000 loans were repaid, less than 5000 loans were not repaid.

[ ]:

[ ]:

[ ]:

[ ]:
```
try:
    for col in [col for col in df1.columns if 'AVG' in col]+[col for col in df1.
 ↪columns if 'MEDI' in col]+[col for col in df1.columns if 'MODE' in col]:
        print(col, ' --------- ', df1['TARGET'].corr(df1[col]))
except TypeError:
    pass
```

[ ]:

4.1.1 Making 1 column out of multiple columns

Mean (avg) columns

[ ]:
```
AVG_cols = [col for col in df.columns if 'AVG' in col]
AVG_cols
```

[ ]:

```
# float16 was giving problems (mean and sum as nan), so converting them to float64
df['YEARS_BEGINEXPLUATATION_AVG'] = df['YEARS_BEGINEXPLUATATION_AVG'].astype(np.
 ↪float64)
df['YEARS_BUILD_AVG'] = df['YEARS_BUILD_AVG'].astype(np.float64)
```

[ ]:
```
avg_summary = pd.DataFrame(columns=['col_name', 'null_sum', 'mean', 'median'])
for col in AVG_cols:
    #print(col, "    ", df[col].isnull().sum(), "    ", df[col].mean(), "    ",␣
 ↪df[col].median())
    avg_summary = avg_summary.append({'col_name': col,
                                      'null_sum': df[col].isnull().sum(),
                                      'mean': df[col].mean(),
                                      'median': df[col].median()}, ignore_index=True)
avg_summary
```

[ ]:
```
for col in AVG_cols:
    df[col].fillna(df[col].mean(), inplace=True)
```

[ ]:
```
avg_df = df[AVG_cols]
avg_df.head()
```

[ ]:
```
df['STD_OF_LIVING_AVG'] = avg_df.mean(axis=1)
```

[ ]:
```
df.drop(columns = AVG_cols, inplace=True)
```

[ ]:
```
df['TARGET'].corr(df['STD_OF_LIVING_AVG'])
```

Median columns

[ ]:
```
MEDI_cols = [col for col in df.columns if 'MEDI' in col]
MEDI_cols
```

[ ]:
```
# float16 was giving problems (mean and sum as nan), so converting them to float64
df['YEARS_BEGINEXPLUATATION_MEDI'] = df['YEARS_BEGINEXPLUATATION_MEDI'].astype(np.
 ↪float64)
df['YEARS_BUILD_MEDI'] = df['YEARS_BUILD_MEDI'].astype(np.float64)
```

[ ]:
```
median_summary = pd.DataFrame(columns=['col_name', 'null_sum', 'mean', 'median'])
for col in MEDI_cols:
    #print(col, "    ", df[col].isnull().sum(), "    ", df[col].mean(), "    ",␣
 ↪df[col].median())
    median_summary = median_summary.append({'col_name': col,
                                      'null_sum': df[col].isnull().sum(),
                                      'mean': df[col].mean(),
```

```
                                    'median': df[col].median()}, ignore_index=True)
median_summary
```

[ ]:
```
for col in MEDI_cols:
    df[col].fillna(df[col].median(), inplace=True)
```

[ ]:
```
median_df = df[MEDI_cols]
median_df.head()
```

[ ]:
```
df['STD_OF_LIVING_MEDIAN'] = median_df.mean(axis=1)
```

[ ]:
```
df.drop(columns = MEDI_cols, inplace=True)
```

[ ]:
```
df['TARGET'].corr(df['STD_OF_LIVING_MEDIAN'])
```

Mode columns

[ ]:
```
MODE_cols = [col for col in df.columns if 'MODE' in col]
MODE_cols
```

[ ]:
```
# float16 was giving problems (mean and sum as nan), so converting them to float64
df['YEARS_BEGINEXPLUATATION_MODE'] = df['YEARS_BEGINEXPLUATATION_MODE'].astype(np.
 ↪float64)
df['YEARS_BUILD_MODE'] = df['YEARS_BUILD_MODE'].astype(np.float64)
```

[ ]:
```
# Some columns in this have Object datatype, so we cannot directly operate on them
for col in MODE_cols:
    print(col, "                    ", df[col].dtypes)
```

[ ]:
```
for col in MODE_cols:
    if df[col].dtypes == 'O':
        print(col)
```

[ ]:
```
obj_mode = df[['FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE',␣
 ↪'EMERGENCYSTATE_MODE']]
obj_mode.head()
```

[ ]:
```
for col in obj_mode:
    print(df[col].value_counts(), '\n')
```

[ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

[ ]:
```
```

```
for col in obj_mode:
    df[col] = le.fit_transform(df[col].astype(str))  # why used .astype(str) - https:
 ↪//stackoverflow.com/questions/58868256/
 ↪scikit-learn-label-encoder-resulting-in-error-argument-must-be-a-string-or-numb
    df[col] = df[col]/df[col].max()
```

[ ]:
```
df[['FONDKAPREMONT_MODE', 'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE',␣
 ↪'EMERGENCYSTATE_MODE']]
```

[ ]:
```
mode_summary = pd.DataFrame(columns=['col_name', 'null_sum', 'mean', 'median'])
for col in MODE_cols:
    #print(col, "    ", df[col].isnull().sum(), "    ", df[col].mean(), "    ",␣
 ↪df[col].median())
    mode_summary = mode_summary.append({'col_name': col,
                                        'null_sum': df[col].isnull().sum(),
                                        'mean': df[col].mean(),
                                        'median': df[col].median(),
                                        'mode': df[col].mode()[0]}, ignore_index=True)
mode_summary
```

[ ]:
```
for col in MODE_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

[ ]:
```
mode_df = df[MODE_cols]
mode_df.head()
```

[ ]:
```
df['STD_OF_LIVING_MODE'] = mode_df.mean(axis=1)
```

[ ]:
```
df.drop(columns = MODE_cols, inplace=True)
```

[ ]:
```
df['TARGET'].corr(df['STD_OF_LIVING_MODE'])
```

Finally, after converting multiple avg, median and mode columns into their individual single column

[ ]:
```
df.head()
```

[ ]:
```
df.shape
```

We reduced 122-78 = 44 columns with this method

[ ]:

[ ]:

4.1.2 Handling Missing Values Datatype wise

1. Object (strings) datatype

```
[ ]: print("Object type values:",np.count_nonzero(df.select_dtypes('object').columns))
```

```
[ ]: object_cols = df.select_dtypes('object').columns
     object_cols
```

```
[ ]: df[object_cols]
```

```
[ ]: # https://stackoverflow.com/questions/36808434/
      ↪label-encoder-encoding-missing-values#:~:
      ↪text=Don't%20use%20LabelEncoder%20with%20missing%20values.
     df[object_cols].isnull().sum()
```

```
[ ]: df['NAME_TYPE_SUITE'] = df['NAME_TYPE_SUITE'].replace(np.nan, df['NAME_TYPE_SUITE'].
      ↪mode()[0])
     df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'].replace(np.nan, df['OCCUPATION_TYPE'].
      ↪mode()[0])
```

```
[ ]: for col in object_cols:
         print(col, df[col].nunique())
```

```
[ ]: # Replacing XNA gender by mode
     df['CODE_GENDER'] = df['CODE_GENDER'].replace('XNA', df['CODE_GENDER'].mode()[0])
```

```
[ ]: # https://www.analyticsvidhya.com/blog/2020/03/
      ↪one-hot-encoding-vs-label-encoding-using-scikit-learn/
     from sklearn.preprocessing import LabelEncoder
     le = LabelEncoder()
```

```
[ ]: df['NAME_CONTRACT_TYPE'] = le.fit_transform(df['NAME_CONTRACT_TYPE'])
     df['CODE_GENDER'] = le.fit_transform(df['CODE_GENDER'])
     df['FLAG_OWN_CAR'] = le.fit_transform(df['FLAG_OWN_CAR'])
     df['FLAG_OWN_REALTY'] = le.fit_transform(df['FLAG_OWN_REALTY'])
     df['NAME_TYPE_SUITE'] = le.fit_transform(df['NAME_TYPE_SUITE'].astype(str))
     df['NAME_INCOME_TYPE'] = le.fit_transform(df['NAME_INCOME_TYPE'])
     df['NAME_EDUCATION_TYPE'] = le.fit_transform(df['NAME_EDUCATION_TYPE'])
     df['NAME_FAMILY_STATUS'] = le.fit_transform(df['NAME_FAMILY_STATUS'])
     df['NAME_HOUSING_TYPE'] = le.fit_transform(df['NAME_HOUSING_TYPE'])
     df['OCCUPATION_TYPE'] = le.fit_transform(df['OCCUPATION_TYPE'].astype(str))
     df['WEEKDAY_APPR_PROCESS_START'] = le.fit_transform(df['WEEKDAY_APPR_PROCESS_START'])
     df['ORGANIZATION_TYPE'] = le.fit_transform(df['ORGANIZATION_TYPE'])
```

The following columns had various categories, so we divide all the values of a particular column with

their maximum value so as to normalize it and make sure our model doesn't give a lot of importance to higher values.

```
[ ]: df['NAME_TYPE_SUITE'] = df['NAME_TYPE_SUITE'] / df['NAME_TYPE_SUITE'].max()
     df['NAME_INCOME_TYPE'] = df['NAME_INCOME_TYPE'] / df['NAME_INCOME_TYPE'].max()
     df['NAME_EDUCATION_TYPE'] = df['NAME_EDUCATION_TYPE'] / df['NAME_EDUCATION_TYPE'].
      ↪max()
     df['NAME_FAMILY_STATUS'] = df['NAME_FAMILY_STATUS'] / df['NAME_FAMILY_STATUS'].max()
     df['NAME_HOUSING_TYPE'] = df['NAME_HOUSING_TYPE'] / df['NAME_HOUSING_TYPE'].max()
     df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'] / df['OCCUPATION_TYPE'].max()
     df['WEEKDAY_APPR_PROCESS_START'] = df['WEEKDAY_APPR_PROCESS_START'] /␣
      ↪df['WEEKDAY_APPR_PROCESS_START'].max()
     df['ORGANIZATION_TYPE'] = df['ORGANIZATION_TYPE'] / df['ORGANIZATION_TYPE'].max()
```

```
[ ]: df[object_cols]
```

```
[ ]: df[object_cols].isnull().sum()
```

```
[ ]: df.shape
```

```
[ ]:
```

```
[ ]:
```

2. Integer datatype

```
[ ]: print("Int type values:",np.count_nonzero(df.select_dtypes(include=['int8', 'int16',␣
      ↪'int32', 'int64']).columns))
```

```
[ ]: int_cols = df.select_dtypes(include=['int8', 'int16', 'int32', 'int64']).columns
     int_cols
```

```
[ ]: df[int_cols]
```

```
[ ]: flag_doc_cols = [col for col in df.columns if 'FLAG_DOCUMENT' in col]
```

```
[ ]: for col in flag_doc_cols:
         print(col, '  ', df['TARGET'].corr(df[col]))
```

```
[ ]: # Dropping columns with value 0 upto 2 decimal places (i.e. 0.00.....)
     flag_drop_col = ['FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',␣
      ↪'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8',
                      'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11',␣
      ↪'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_14',
```

```
                'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18',␣
↪'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
                'FLAG_DOCUMENT_21']
df.drop(flag_drop_col, axis=1, inplace=True)
```

[ ]:

[ ]:

[ ]:
```
int_cols = df.select_dtypes(include=['int8', 'int16', 'int32', 'int64']).columns
int_cols
```

[ ]:
```
df[int_cols].isnull().sum()
```

[ ]:
```
plt.figure(figsize=(30,5))
sns.boxplot(data=df.select_dtypes('int'))
plt.show()
```

[ ]:
```
plt.figure(figsize=(30,5))
sns.boxplot(df['DAYS_EMPLOYED'])
```

[ ]:
```
df['DAYS_EMPLOYED'].value_counts()
```

We can see the outlier here with the value 365243. These were probably NaN values (since all other values are negative and this is the only one positive) and got replaced by 365243, so we convert these to NaN and then fill with the mean of other values in the cooumn.

[ ]:
```
df['DAYS_EMPLOYED'] = df['DAYS_EMPLOYED'].replace(365243, np.nan)
```

[ ]:
```
df['DAYS_EMPLOYED'].fillna(df['DAYS_EMPLOYED'].mean(), inplace=True)
```

[ ]:
```
df['DAYS_EMPLOYED'].value_counts()
```

[ ]:
```
df[int_cols].hist(figsize=(25,25), ec='w')
plt.show()
```

[ ]:
```
def color_(value):
    if value < 0 :
        color = 'red'
    elif value == 1 :
        color = 'blue'
    else:
        color = 'green'
    return 'color: %s' % color
df[int_cols].corr().style.applymap(color_)
```

[ ]:
```
plt.figure(figsize=(20,10))
sns.heatmap(df[int_cols].corr(), cmap='Blues', vmin=-1, vmax=1)
```

Inference -

1. FLAG_OWN_CAR and CODE_GENDER are somewhat correlated.
2. DAYS_BIRTH is highly correlated with CNT_CHILDREN, DAYS_EMPLOYED, DAYS_ID_PUBLISH and FLAG_EMP_PHONE.
3. REGION_RATING_CLIENT and REGION_RATING_CLIENT_W_CITY are extremly correlated.
4. FLAG_DOCUMENT_6 is very less correlated with DAYS_BIRTH and FLAG_EMP_PHONE.

[ ]:
```
df.shape
```

[ ]:
```

```

3. float datatype

[ ]:
```
print("float type values:",np.count_nonzero(df.select_dtypes(include=['float16',␣
↪'float32', 'float64']).columns))
```

[ ]:
```
float_cols = df.select_dtypes(include=['float16', 'float32', 'float64']).columns
float_cols
```

[ ]:
```
df[float_cols].isnull().sum().sort_values(ascending = False)
```

[ ]:
```
df.shape
```

[ ]:
```
df.interpolate(method ='linear', limit_direction ='forward', inplace=True)
```

[ ]:
```
df.shape
```

[ ]:
```
df[float_cols].isnull().sum().sort_values(ascending = False)
```

[ ]:
```
df['OWN_CAR_AGE']
```

[ ]:
```
df.shape
```

[ ]:
```
df.dropna(axis = 'index', how='any', inplace=True)
```

[ ]:
```
df.isnull().sum().sum()
```

[ ]:
```
df.shape
```

[ ]:
```
plt.figure(figsize=(50,15))
sns.boxplot(data=df[float_cols])
plt.show()
```

```
[ ]: plt.figure(figsize=(30,5))
     sns.boxplot(df['AMT_INCOME_TOTAL'])
     plt.show()
```

```
[ ]: df['AMT_INCOME_TOTAL'].describe()
```

```
[ ]: plt.scatter(df['AMT_INCOME_TOTAL'], df['SK_ID_CURR'], label= "stars", color=␣
      ↪"green", marker= "*", s=30)
```

```
[ ]: df[float_cols].hist(figsize=(25,25), ec='w')
     plt.show()
```

```
[ ]: def color_(value):
         if value < 0 :
             color = 'red'
         elif value == 1 :
             color = 'blue'
         else:
             color = 'green'
         return 'color: %s' % color
     df[float_cols].corr().style.applymap(color_)
```

```
[ ]: plt.figure(figsize=(20,10))
     sns.heatmap(df[float_cols].corr(), cmap='Greens', vmin=-1, vmax=1)
```

Inference -

1. AMT_GOODS_PRICE and AMT_CREDIT are very highly correlated.
2. AMT_ANNUITY is also highly correlated with AMT_CREDIT and AMT_GOODS_PRICE.
3. OBS_30_CNT_SOCIAL_CIRCLE, DEF_30_CNT_SOCIAL_CIRCLE, OBS_60_CNT_SOCIAL_CIRCLE and DEF_30_CNT_SOCIAL_CIRCLE are correlated among each other.
4. STD_OF_LIVING_AVG, STD_OF_LIVING_MEDIAN and STD_OF_LIVING_MODE are extremely correlated among each other.
5. CNT_FAM_MEMBERS is less correlated with NAME_FAMILY_STATUS.

```
[ ]:
```

4.1.3 Correlation Matrix and Heatmap

```
[ ]: def color_(value):
         if value < 0 :
             color = 'red'
         elif value == 1 :
             color = 'blue'
```

```
    else:
        color = 'green'
    return 'color: %s' % color
```

[ ]:
```
df.corr().style.applymap(color_)
```

[ ]:
```
plt.figure(figsize=(30,20))
sns.heatmap(df.corr(), cmap='Oranges', vmin=-1, vmax=1)
```

[ ]:
```
df.corr()['TARGET'].sort_values(ascending=False)
```

[ ]:
```
# sk_id_curr on hold
# sus - amt_income_total, FLAG_OWN_REALTY,
low_corr_cols = ['CNT_FAM_MEMBERS', 'OBS_30_CNT_SOCIAL_CIRCLE',␣
 ↪'OBS_60_CNT_SOCIAL_CIRCLE', 'NAME_TYPE_SUITE',
                 'REG_REGION_NOT_WORK_REGION', 'REG_REGION_NOT_LIVE_REGION',␣
 ↪'WEEKDAY_APPR_PROCESS_START',
                 'LIVE_REGION_NOT_WORK_REGION', 'AMT_REQ_CREDIT_BUREAU_DAY',␣
 ↪'FLAG_MOBIL', 'FLAG_CONT_MOBILE',
                 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_HOUR',␣
 ↪'FLAG_EMAIL', 'AMT_REQ_CREDIT_BUREAU_QRT',
                 'SK_ID_CURR', 'NAME_FAMILY_STATUS', 'AMT_REQ_CREDIT_BUREAU_MON']
```

[ ]:
```
df.drop(low_corr_cols, axis=1, inplace=True)
```

[ ]:
```
df.shape
```

[ ]:
```
df.to_csv(r'C:\Users\surya\Desktop\Jupyter Python Programs\Mini Project - Data␣
 ↪Mining\clean_credit_card_fraud_detection.csv', index=False)
```

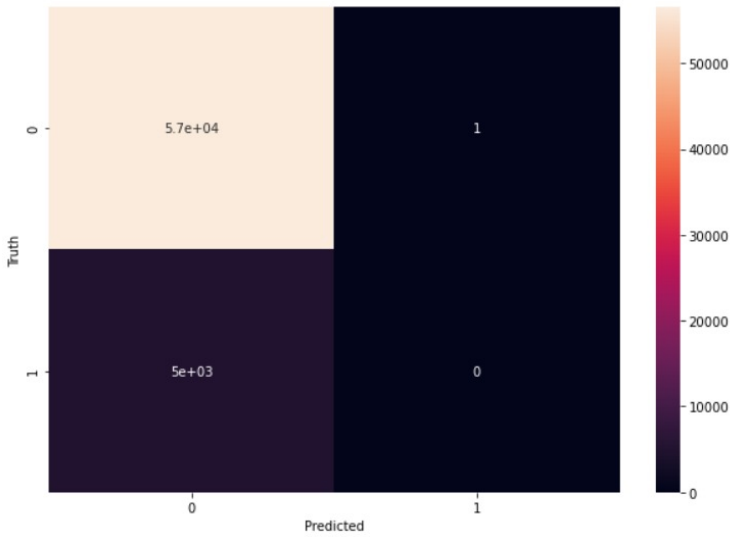
[ ]:
```

```

Chapter 5

Conclusion and Future Projections

5.1 Conclusion

After applying Logistic Regression, the model's accuracy is calculated, which is the number of correct prediction made divided by the total number of predictions made. This model works with an accuracy of 0.9192546583850931, i.e., 91.92 We also construct a Confusion Matrix to get more intuition about our model's performance. A confusion matrix is a table which is used to describe how a classification model has preformed on a test data for which the true values are already known to us (See Fig.). The general idea is to count the number of times instances of class A are classified as class B.



CONFUSION MATRIX

Interpretation of Confusion matrix
For truth values 0, the model gave correct predictions as 0 itself a total of 56536 times (out of 61502 testing data). Therefore, the percentage of correct predictions is (56536/61502)*100 = 91.9254%. For truth value 1, the model predicted 0 4965 times, and the percentage is 8.0729%. Similarly, for truth value 0, it predicted 1 only once, the percentage being 0.00162%, and for truth value 1, it did not predict 1 even once, hence the percentage being 0. Here, the fact that the model predicts over 90% of the test cases correctly is in our

favor. Nevertheless, it can also be noticed that we have a case of over fitting, where the model has trained itself too well according to the training set and is having problems in generalizing new testing sets. This problem can be refined by using under sampling techniques and further looking into other measures, which are discussed in the next section.

5.2 Future Projections

As discussed above, the over-fitting problem can be tackled by undersampling techniques. There are other solutions for over-fitting as well, such as Cross-validation, training with more data, removing more irrelevant features, Regularization, Ensembling, etc. Other measures can also be calculated along with model accuracy, giving a better intuition of the model's performance. These include Confusion Matrix, Precision, Recall, and f1 score.
From Confusion Matrix we get more metrics in order evaluate the model,

1. True Positive (TP)
2. False Positive (FP)
3. True Negative (TN)
4. False Negative (FN)

Precision is the ratio of correct positive predictions to the total predicted positives. precision = (TP) / (TP+FP) Recall is the number of correct positive truths divided by the total number of elements that actually belong to the positive class (i.e., the sum of true positives and false negatives recall = (TP) / (TP+FN) It is convenient to combine recall and precision into a single metric called the F1 score, which is the harmonic mean of recall and precision and thereby gives the model's overall performance.

$$F1\_score = 2 * \frac{(precision * recall)}{(precision + recall)} \tag{5.1}$$

Bibliography

[1] https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/.

[2] https://www.geeksforgeeks.org/confusion-matrix-machine-learning/.

[3] https://www.geeksforgeeks.org/understanding-logistic-regression/.

[4] https://www.geeksforgeeks.org/ml-cost-function-in-logistic-regression/.

[5] https://www.geeksforgeeks.org/understanding-logistic-regression/.

[6] https://www.analyticsvidhya.com/blog/2020/12/beginners-take-how-logistic-regression-is-related-to-linear-regression/.

[7] https://www.analyticsvidhya.com/blog/2021/10/building-an-end-to-end-logistic-regression-model/#:~:text=The%20goal%20of%20Logistic%20Regression,two%20values%3A%20pass%20and%20fail.

[8] https://satishgunjal.com/binary_lr/.

[9] https://medium.com/@KrishnaRaj_Parthasarathy/ml-classification-why-accuracy-is-not-a-best-measure-for-assessing-ceeb964ae47c#:~:text=Even%20when%20model%20fails%20to,evaluation%20for%20our%20classification%20model.

[10] https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/#:~:text=A%20confusion%20matrix%20is%20a,related%20terminology%20can%20be%20confusing.

[11] Aleksi Kallio and Jarno Tuimala. *Data Mining*, pages 525–528. Springer New York, New York, NY, 2013. doi:10.1007/978-1-4419-9863-7_599.

[12] Mehmed Kantardzic. *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons, 2011.