Fundamentals of Data Mining [IT3051]

Mini Project – Final Document

2024

<u>Group Details</u>

Name-Mining Masters

Name with Initials		Registration Number
1.	Deemantha P.H.H.C	IT22560162
2.	Jayarathna J.V.D	IT22577610
3.	Thrimanna A	IT22585530
4.	Wimalarathna B.P.K	IT22059604
5.	Pallewela S. M	IT22594136

Submitted to:

Mr. Prasanna Sumathipala

Date of submission

3/10/2024

Automobile Loan Approval System

GitHub link –https://github.com/chamikadeemantha/FDM_Project

Deployed link –https://loandrive.streamlit.app

Video Links - https://youtu.be/lSvPHMvaDNg

Contents

Declaration

This project report, or any portion of it, was not copied from the internet or any other source, nor was it based on work produced by any organization, institution, another institute, or previous student project team at the Sri Lanka Institute of Information Technology.

Student Number	Name	Signature
IT22560162	**Deemantha P.H.H.C**	[illegible]
IT22577610	**Jayarathna J.V.D**	Diwyanjalee
IT22585530	**Thrimanna A**	[illegible]
IT22059604	**Wimalarathna B.P.K**	[illegible]
IT22594136	**Pallewela S. M**	[illegible]

Abstract

Several technological problems affect our everyday lives and workplaces in the modern world. But many of these problems are constantly being solved by developments in state-of-the-art technology. Our project's goal is to use data mining and machine learning techniques to solve a particular issue that impacts individuals on a daily basis.

Through the use of these technical techniques, we offer a solution that has the potential to either fix the problem or greatly lessen its impact on people and society. This essay outlines how we plan to address the noted issue and illustrates the possible outcomes of our plan.

- **Automobile loan approval system.**

Acknowledgment

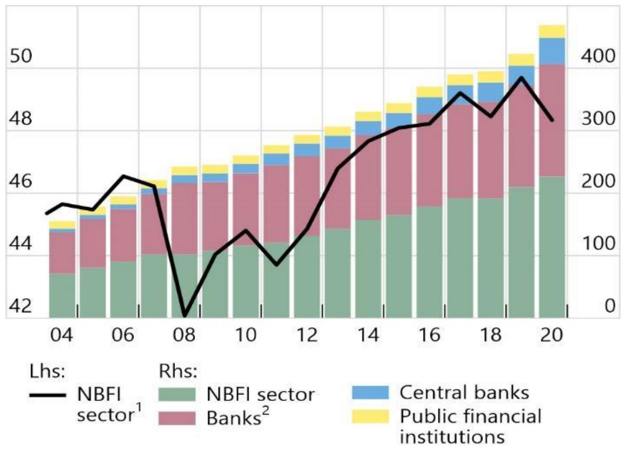
We sincerely thank **Mr. Prasanna Sumathipala** for his crucial support and direction during this module's completion, which allowed us to compile this project report.

Our heartfelt gratitude also goes out to **Mr. Derick Alexander** and **Ms. Malithi Navarathna** for taking the time to consider and approve our project proposal.

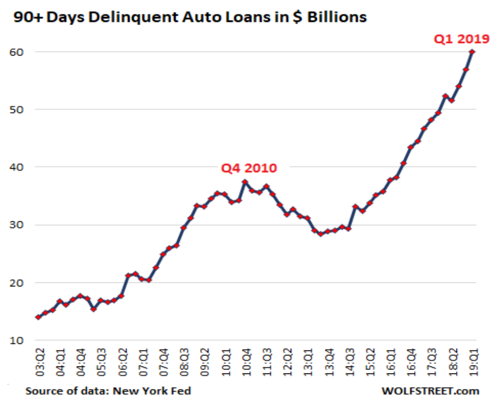
Introduction

In the present moment, consumers' financial stability has been strained as a result of the ongoing financial crisis, rising interest rates, and inflation. The financial system is now facing problems as a result of this.

A nation's economy is supported by its banks and non-banking financial institutions (NBFI). The collapse of these institutions would put the entire economy in jeopardy. Customers defaulting on loans they have secured is a major danger to the current financial system for the reasons indicated above. Due to their lack of the same level of support from central banks as banking institutions, NBFIs have been particularly impacted by this.



A large number of loans fall into the Automobile / Auto loan category because of the issues and shortcomings with public transportation and the necessity for private vehicles for every household. These loans are particularly susceptible to consumer default because the bulk of them offer no financial return. Because NBFI is one of the major vehicle loan lenders, default rates on these loans have climbed dramatically in recent years, which has negatively impacted their profit margins.



Upon thorough examination of the loan defaulting issue, it is evident that determining a method to assess or forecast a customer's capacity to repay a loan would determine whether or not they qualify for loans, particularly auto loans. This would be the most appropriate and workable solution to the problem.

We, the Mining Masters group, have recognized the issue at hand and have made the decision to apply our knowledge to solve it. Since NBFIs are the party most impacted by this issue, it was determined to handle it from their perspective as well as the lender's, or more precisely, from theirs.

The following presents the real-world business problem that we have identified, the method that we have planned to use and solve the problem, and our goal for the solution.

- **Problem** - It is becoming harder for NBFIs to report profits because of the increase in vehicle loan defaults. The organization aims to determine a client's ability to repay a loan alongside the proportionate weighting of each aspect influencing a borrower's ability to do so.

- **Client** – A financial institution that is not a bank (NBFI). A non-banking financial institution (NBFI) is one that is not regulated by a national or international banking regulatory body, or that does not hold a full banking license. NBFI facilitates greater accessibility to financial services, including lending and investment.

- **Solution** – Predict whether a client can pay the requested loan. For each customer loan request, you must predict the default.

- **Goal** - Creating a model that assists the loan approver (the NBFI) assess if the potential borrower will be able to repay the loan. By using the prediction's outcomes, they can decide whether to grant or reject the loan request, avoiding the loss of revenue from a loan default

Thus, **LoanDrive** was produced, **LoanDrive** is an intelligent software solution that addresses the after mentioned business requirement and assists NBFIs in decision-making.
LoanDrive utilizes a predictive model which was custom-built and tuned for the specific task and trained with a live **dataset** that mimics the business requirements.

Data Description

The selected dataset relates to a non-banking financial institution (NBFI) or non-bank financial company (NBFC), a type of financial organization that is not overseen by a national or international banking regulatory body and does not hold a full banking license. In spite of this, NBFCs offer banking-associated financial services such market brokerage, investing, risk pooling, and contractual savings.

Due to a rise in defaults in the auto loan category, the specific NBFI under consideration is presently having trouble turning a profit. The company's goal is to evaluate customers' loan-repayment capacities and comprehend the importance of different factors affecting a borrower's ability to repay a loan.

121,856 rows and 40 different fields of historical data about customers who applied for and were granted auto loans are included in the dataset.

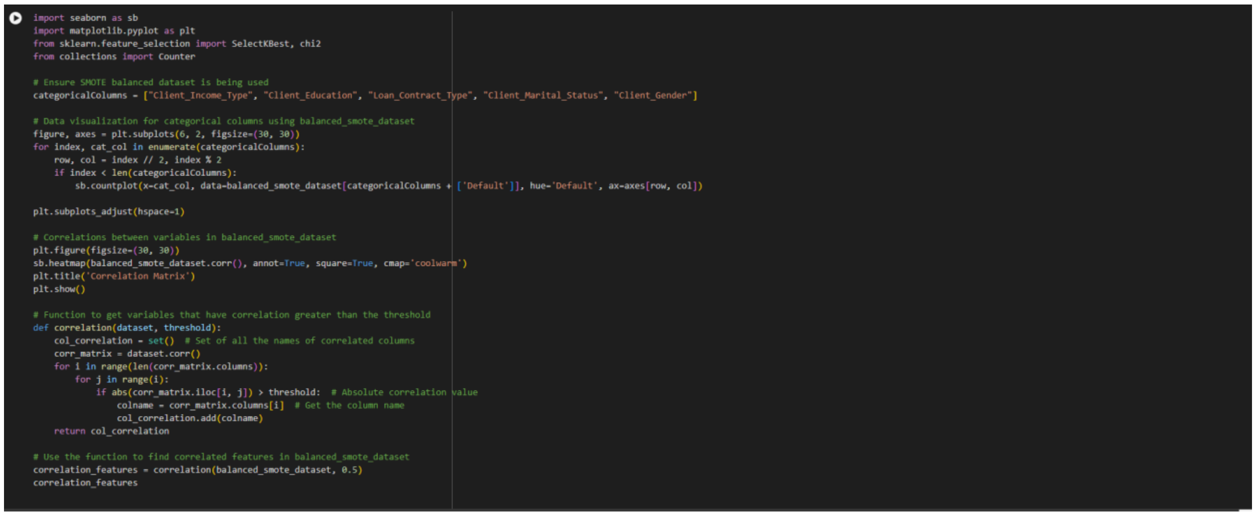
- Data set: [Automobile Loan Default Dataset | Kaggle]

Data Visualization and Preprocessing

Data visualization

The dataset includes both numerical and categorical data. Categorical data has been visualized using bar charts, while numerical data has been represented with box plots. These visualizations were created using the Seaborn Python library (Seaborn, n.d.).

Categorical Data Visualization



```python
import seaborn as sb
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest, chi2
from collections import Counter

# Ensure SMOTE balanced dataset is being used
categoricalColumns = ["Client_Income_Type", "Client_Education", "Loan_Contract_Type", "Client_Marital_Status", "Client_Gender"]

# Data visualization for categorical columns using balanced_smote_dataset
figure, axes = plt.subplots(6, 2, figsize=(30, 30))
for index, cat_col in enumerate(categoricalColumns):
    row, col = index // 2, index % 2
    if index < len(categoricalColumns):
        sb.countplot(x=cat_col, data=balanced_smote_dataset[categoricalColumns + ['Default']], hue='Default', ax=axes[row, col])

plt.subplots_adjust(hspace=1)

# Correlations between variables in balanced_smote_dataset
plt.figure(figsize=(30, 30))
sb.heatmap(balanced_smote_dataset.corr(), annot=True, square=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Function to get variables that have correlation greater than the threshold
def correlation(dataset, threshold):
    col_correlation = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:  # Absolute correlation value
                colname = corr_matrix.columns[i]  # Get the column name
                col_correlation.add(colname)
    return col_correlation

# Use the function to find correlated features in balanced_smote_dataset
correlation_features = correlation(balanced_smote_dataset, 0.5)
correlation_features
```

The distribution of the various categorical variables in the dataset, broken down by the Default status, is shown in the bar charts below. The relationship between these category characteristics and the clients' default state is made clearer by the visualizations.

- Client Income Type vs Default:

  Observation: Income Type 2 has the greatest number of non-default clients, while Income Type 1 has a greater default rate. In the dataset, Income Type 3 is seldom represented.

  The implication is that customers with Income Type 2 are typically more dependable when it comes to loan repayment.

- Client Education vs Default:

  Observation: While Education Type 2 has fewer defaults, clients with Education Type 1 have a greater representation in both default and non-default categories.

  Implication: Clients' degree of education may be a predictor of their likelihood of defaulting, with lower levels of education possibly linked to higher default rates.

- Loan Contract Type vs Default:

Observation: Contract Type 1 loans account for the majority of loans, and default rates are notable. Loans and defaults under Contract Type 2 are lower.

Implication: The kind of loan arrangement may have an impact on the chance of default.

- Client Marital Status vs Default:

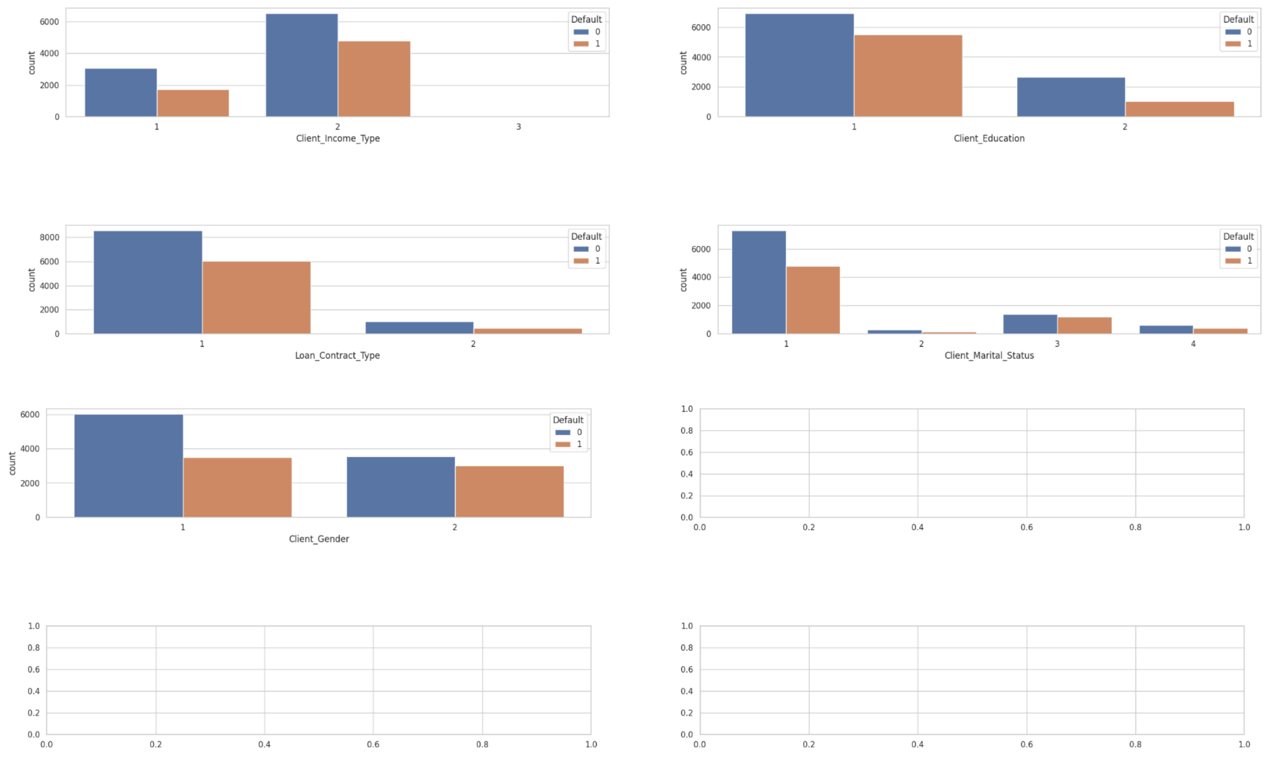
  Observation: Clients with Marital Status 1 (likely single) have the highest default rate. Other statuses have significantly lower counts and defaults.

  Implication: Marital status might affect a client's financial stability and ability to repay loans, with single clients showing higher default rates.

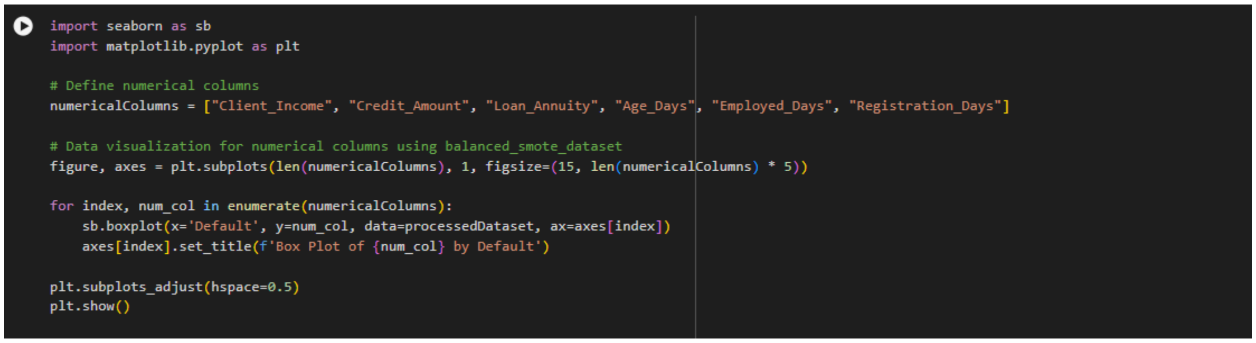
- Client Gender vs Default:

  Observation: Gender 1 (likely male) has a higher count in both default and non-default categories compared to Gender 2 (likely female).

  Implication: Gender distribution shows that males are more represented in the dataset, but the default rate does not show significant differences based on gender.



Numerical Data Visualization


```python
import seaborn as sb
import matplotlib.pyplot as plt

# Define numerical columns
numericalColumns = ["Client_Income", "Credit_Amount", "Loan_Annuity", "Age_Days", "Employed_Days", "Registration_Days"]

# Data visualization for numerical columns using balanced_smote_dataset
figure, axes = plt.subplots(len(numericalColumns), 1, figsize=(15, len(numericalColumns) * 5))

for index, num_col in enumerate(numericalColumns):
    sb.boxplot(x='Default', y=num_col, data=processedDataset, ax=axes[index])
    axes[index].set_title(f'Box Plot of {num_col} by Default')

plt.subplots_adjust(hspace=0.5)
plt.show()
```

The below box plots provide a visual summary of the distribution of various numerical features categorized by 'Default' status (0: No Default, 1: Default)

- Client Income

  Observation: Both the default and non-default groups have a similar customer income distribution, with a significant number of outliers in higher income categories.

  Conclusion: There is no appreciable variation in median income between defaulters and non-defaulters.

- Credit Amount

  The distribution of credit amounts for the two groups has a pattern that is comparable, with a large number of outliers in the upper credit ranges.

  Conclusion: The median credit amounts for defaulters and non-defaulters are similar.

- Loan Annuity

  Although defaulters' loan annuities are somewhat larger, both groups' distribution patterns are similar.

  Conclusion: Higher loan annuities may slightly increase the likelihood of default.

- Age in Days

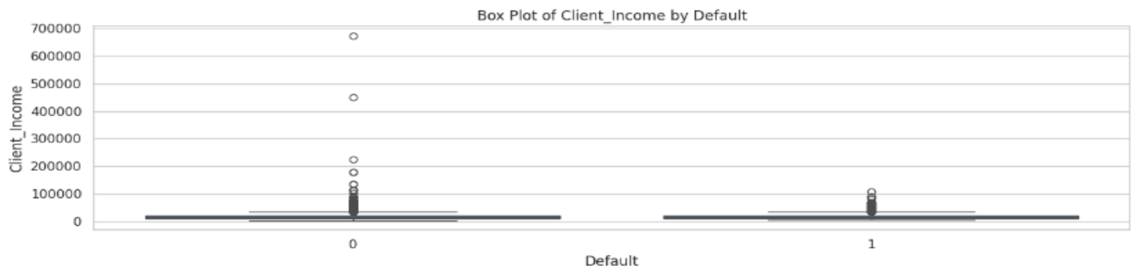
  Age distributions are similar for both groups with no significant difference in medians.

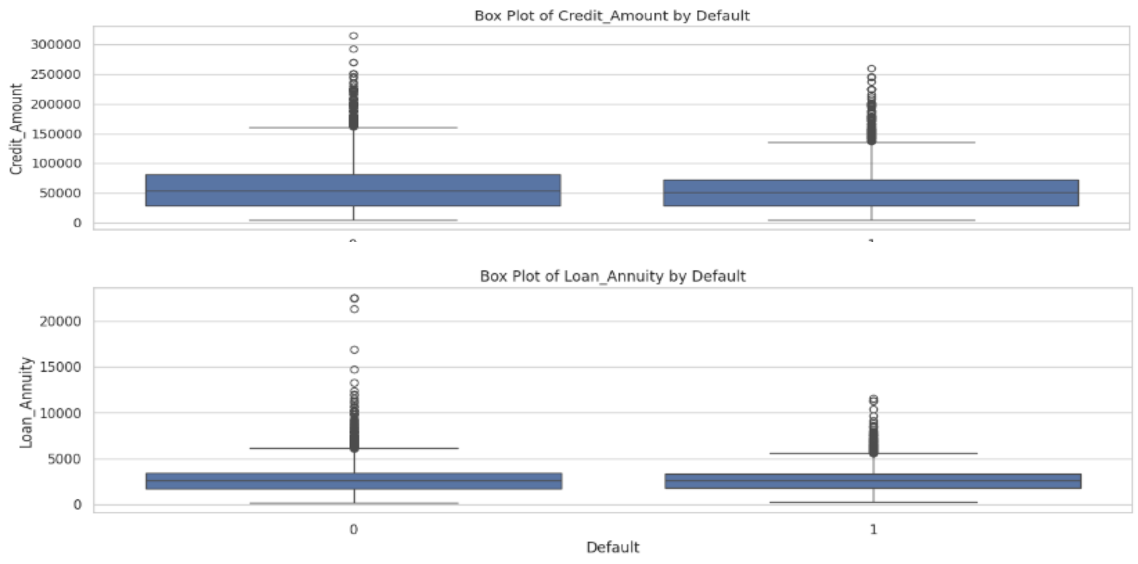
  Conclusion: Age does not significantly impact default status.

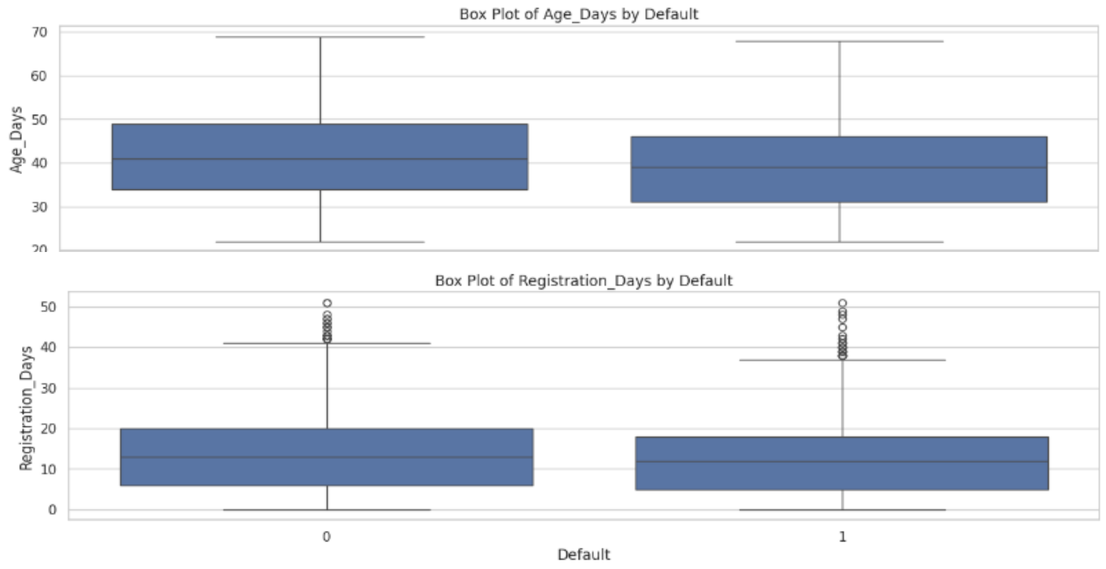
- Registration Days:

Registration days show a similar pattern for both groups, with slight variations in outliers.

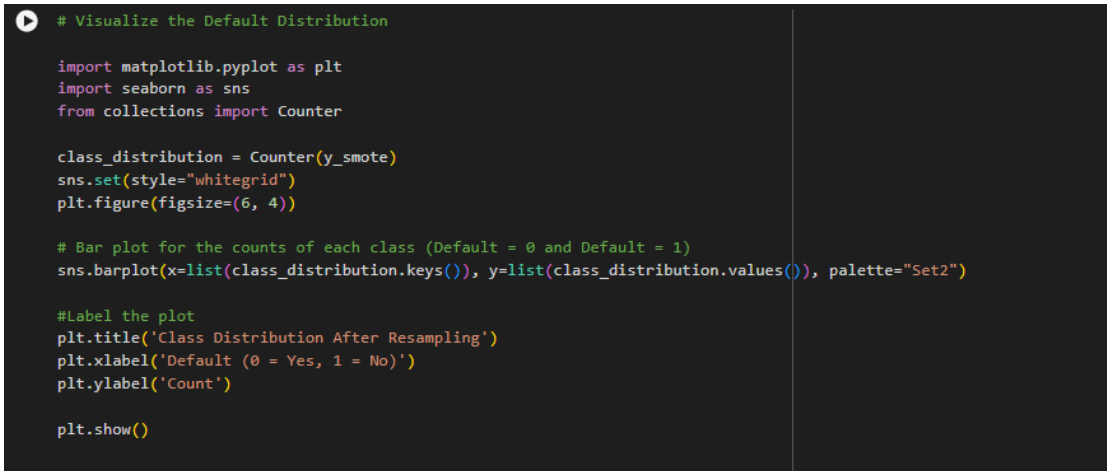
Conclusion: Registration duration is not a strong predictor of default status.



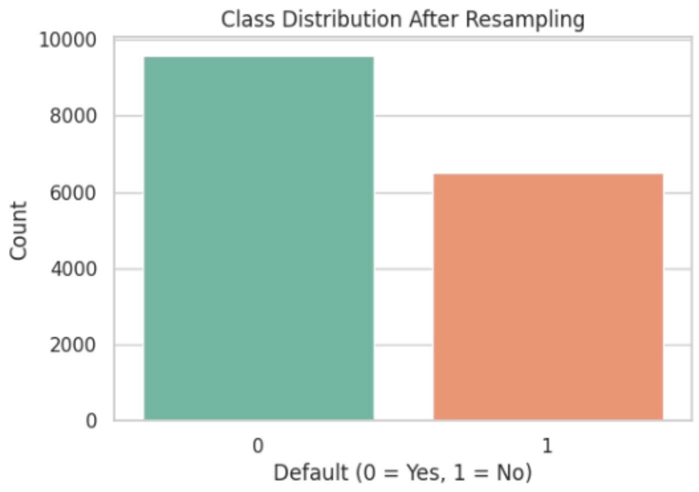




Target variable visualization



```
# Visualize the Default Distribution

import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

class_distribution = Counter(y_smote)
sns.set(style="whitegrid")
plt.figure(figsize=(6, 4))

# Bar plot for the counts of each class (Default = 0 and Default = 1)
sns.barplot(x=list(class_distribution.keys()), y=list(class_distribution.values()), palette="Set2")

#Label the plot
plt.title('Class Distribution After Resampling')
plt.xlabel('Default (0 = Yes, 1 = No)')
plt.ylabel('Count')

plt.show()
```

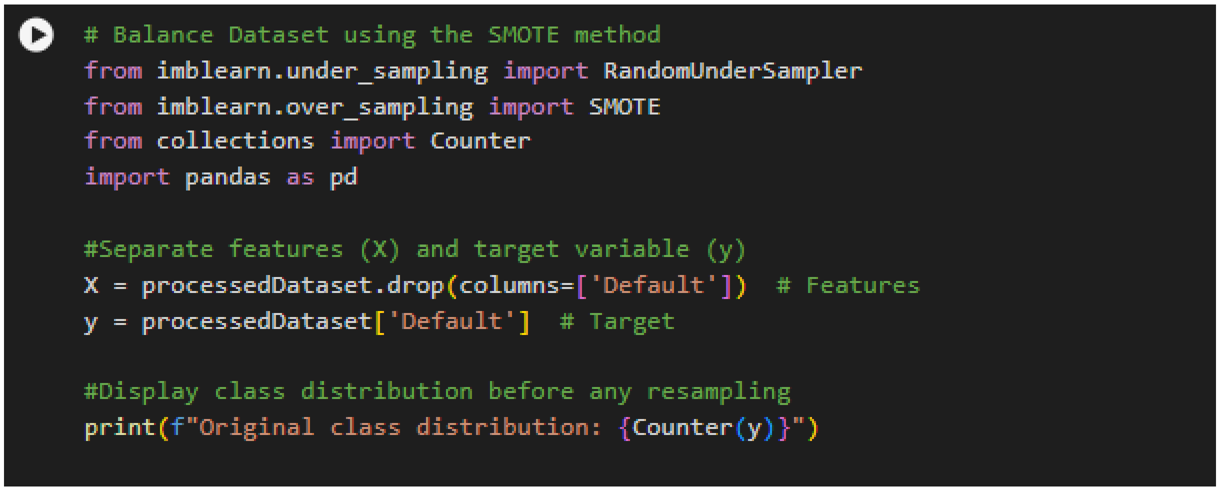


As it is visible the data set is not balanced as the data are biased towards the Not Default class category.

This could be concluded by referring to the above chart where it is visible, the difference between the 2 class values. About 90% of the data are biased towards Not Default.

Overcoming the Issue if Dataset Imbalance and Balancing the Data

The imbalance of the data set can be handled using various resampling techniques like under-sampling, oversampling, SMOTE technique, etc. (Wu, 2022).

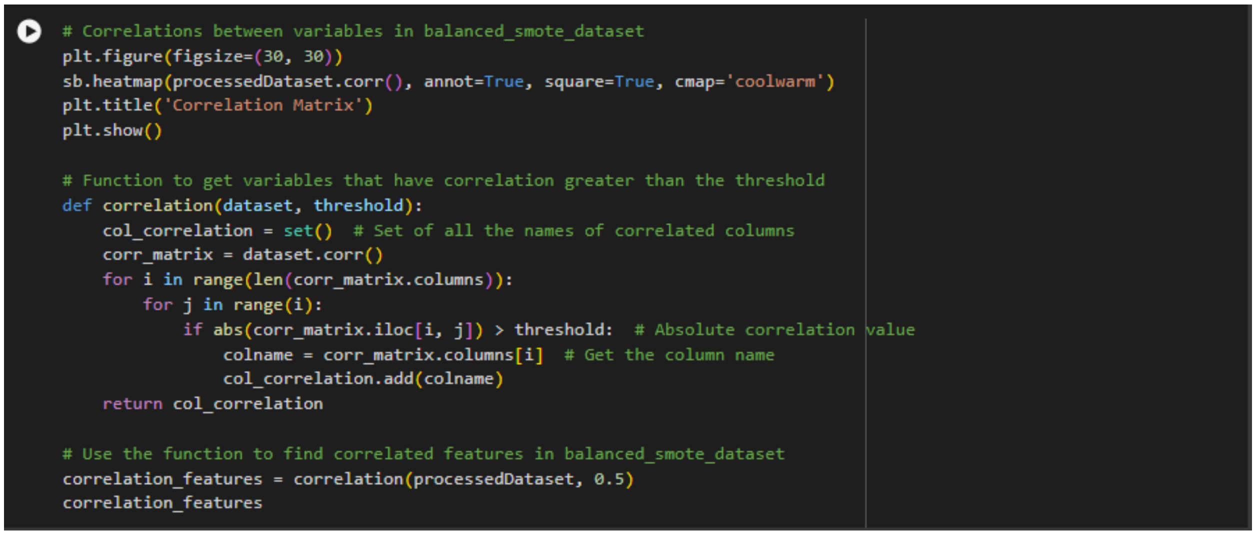


```python
# Balance Dataset using the SMOTE method
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from collections import Counter
import pandas as pd

#Separate features (X) and target variable (y)
X = processedDataset.drop(columns=['Default'])  # Features
y = processedDataset['Default']  # Target

#Display class distribution before any resampling
print(f"Original class distribution: {Counter(y)}")
```

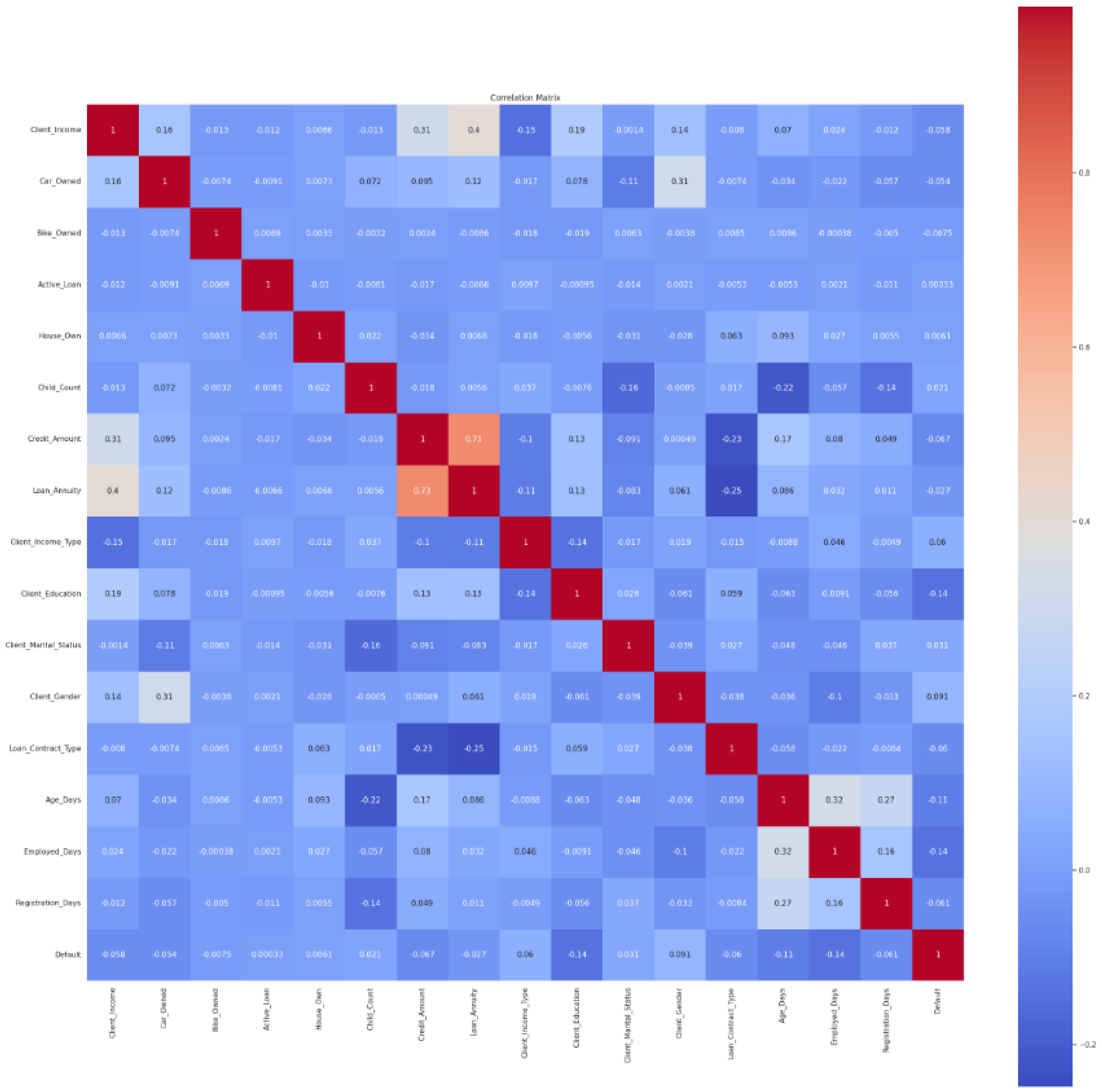
It has been elaborately explained in the latter parts of the report, how the dataset has been balanced and prepared to omit accuracy and other issues that could be present due to the imbalance of the data.

Correlation Heatmap

Correlation heatmaps are a type of plot that visualize the strength of relationships between numerical variables (Kumar, 2022). Correlation plots are used to understand which variables are related to each other and the strength of this relationship.



```python
# Correlations between variables in balanced_smote_dataset
plt.figure(figsize=(30, 30))
sb.heatmap(processedDataset.corr(), annot=True, square=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Function to get variables that have correlation greater than the threshold
def correlation(dataset, threshold):
    col_correlation = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold:  # Absolute correlation value
                colname = corr_matrix.columns[i]  # Get the column name
                col_correlation.add(colname)
    return col_correlation

# Use the function to find correlated features in balanced_smote_dataset
correlation_features = correlation(processedDataset, 0.5)
correlation_features
```


Correlation Matrix
Client_Income
Car_Owned
Bike_Owned
Active_Loan
House_Own
Child_Count
Credit_Amount
Loan_Annuity
Client_Income_Type
Client_Education
Client_Marital_Status
Client_Gender
Loan_Contract_Type
Age_Days
Employed_Days
Registration_Days
Default
0.8
0.6
0.4
0.2
0.0
-0.2

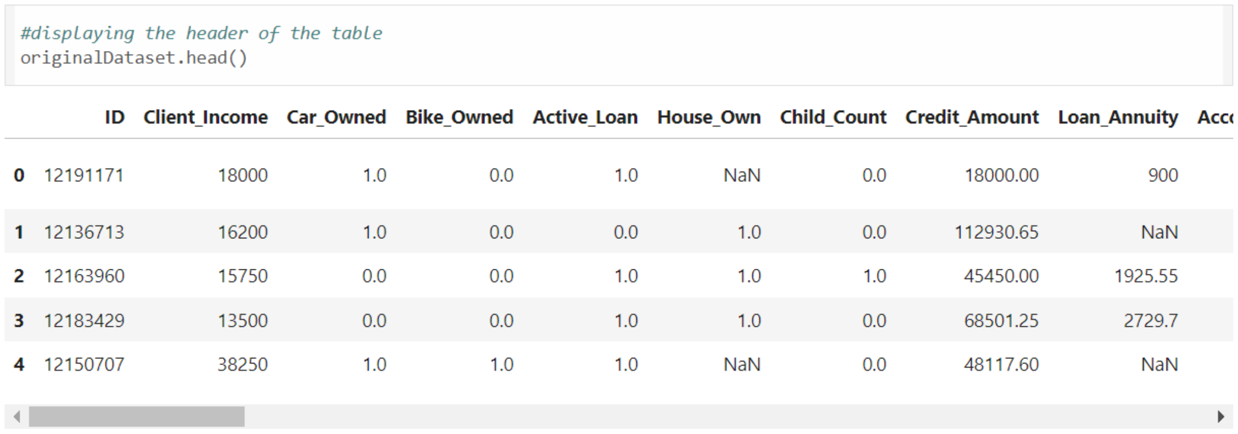
Data Preparation and Preprocessing

The raw data set must go through number of stages such as:

1. Data cleaning
2. Data integration
3. Data reduction
4. Data transformation

before entering into the machine learning model to train the data, to transform into an easier-to-understand format data (Xenonstack, 2018). The following describes the steps and techniques required to prepare data.

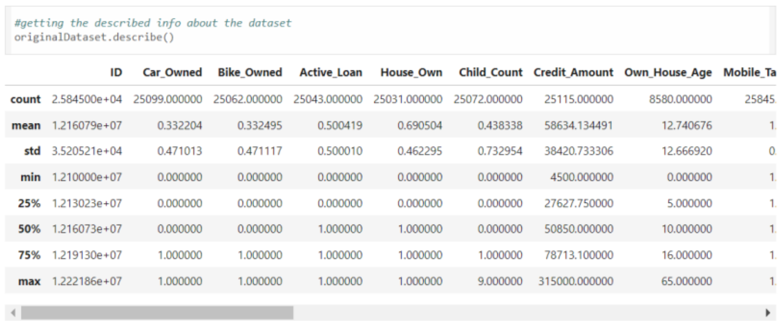
Reading the original dataset



```
#displaying the header of the table
originalDataset.head()
```

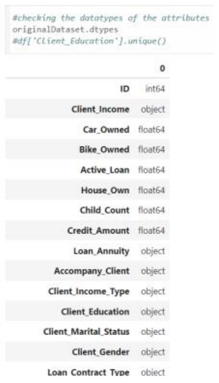
	ID	Client_Income	Car_Owned	Bike_Owned	Active_Loan	House_Own	Child_Count	Credit_Amount	Loan_Annuity	Acc
0	12191171	18000	1.0	0.0	1.0	NaN	0.0	18000.00	900	
1	12136713	16200	1.0	0.0	0.0	1.0	0.0	112930.65	NaN	
2	12163960	15750	0.0	0.0	1.0	1.0	1.0	45450.00	1925.55	
3	12183429	13500	0.0	0.0	1.0	1.0	0.0	68501.25	2729.7	
4	12150707	38250	1.0	1.0	1.0	NaN	0.0	48117.60	NaN	

To obtain an overview of the data and the attributes we will be handling, as well as to comprehend the dataset for the model building, we first read the original dataset. Additionally, this also helps in our understanding of the various data types of the original dataset.



```
#getting the described info about the dataset
originalDataset.describe()
```

	ID	Car_Owned	Bike_Owned	Active_Loan	House_Own	Child_Count	Credit_Amount	Own_House_Age	Mobile_Ta
count	2.584500e+04	25099.000000	25062.000000	25043.000000	25031.000000	25072.000000	25115.000000	8580.000000	25845.
mean	1.216079e+07	0.332204	0.332495	0.500419	0.690504	0.438338	58634.134491	12.740676	1.
std	3.520521e+04	0.471013	0.471117	0.500010	0.462295	0.732954	38420.733306	12.666920	0.
min	1.210000e+07	0.000000	0.000000	0.000000	0.000000	0.000000	4500.000000	0.000000	1.
25%	1.213023e+07	0.000000	0.000000	0.000000	0.000000	0.000000	27627.750000	5.000000	1.
50%	1.216073e+07	0.000000	0.000000	1.000000	1.000000	0.000000	50850.000000	10.000000	1.
75%	1.219130e+07	1.000000	1.000000	1.000000	1.000000	1.000000	78713.100000	16.000000	1.
max	1.222186e+07	1.000000	1.000000	1.000000	1.000000	9.000000	315000.000000	65.000000	1.



```
#checking the datatypes of the attributes
originalDataset.dtypes
#df['Client_Education'].unique()
```

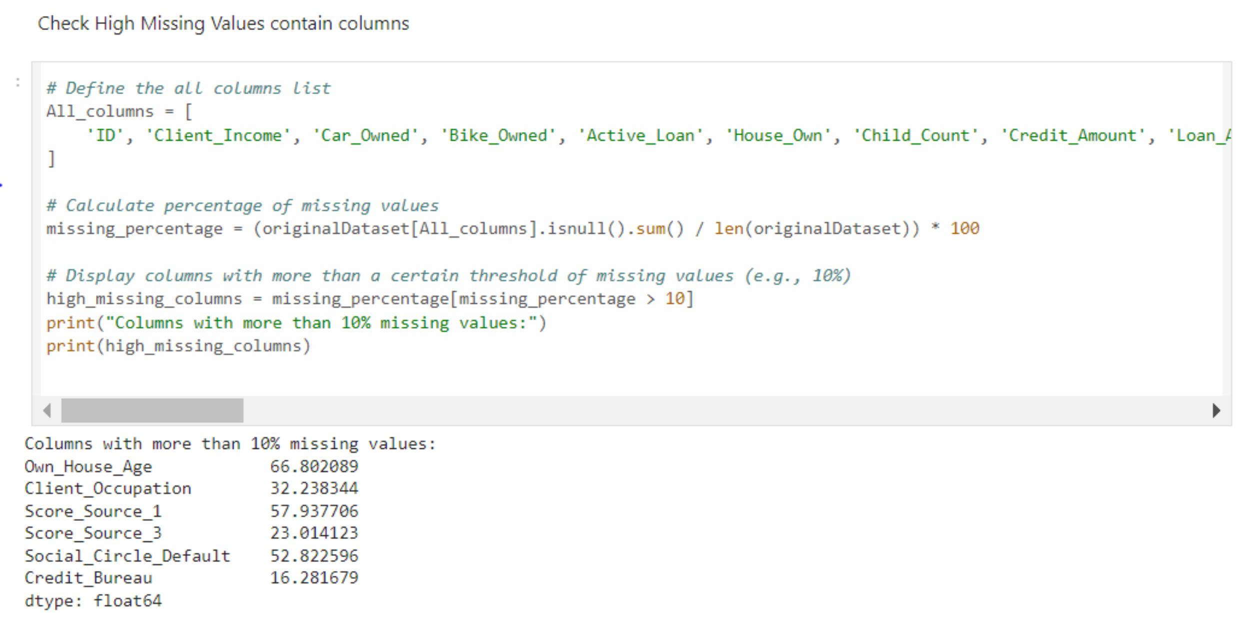
	0
ID	int64
Client_Income	object
Car_Owned	float64
Bike_Owned	float64
Active_Loan	float64
House_Own	float64
Child_Count	float64
Credit_Amount	float64
Loan_Annuity	object
Accompany_Client	object
Client_Income_Type	object
Client_Education	object
Client_Marital_Status	object
Client_Gender	object
Loan_Contract_Type	object

Dimensionality Reduction

Generally, data reduction is done in two ways. i.e., row-wise for data sample reduction and column-wise for data variable reduction (geeksforgeeks, 2022).

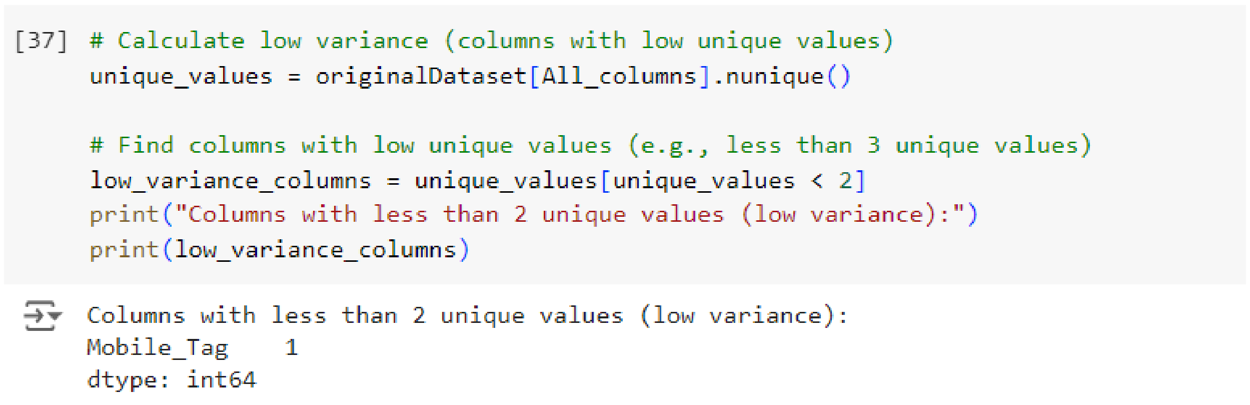
Here, we are column-wise for data variable reduction. The model's irrelevant columns are eliminated, and any columns with more than 10% missing values and the low variance coloumns are eliminated as well.

Then columns like ‘ID’, a unique attribute that does not have predictive power and unambiguous columns like ‘scores1’ are removed from the data set.

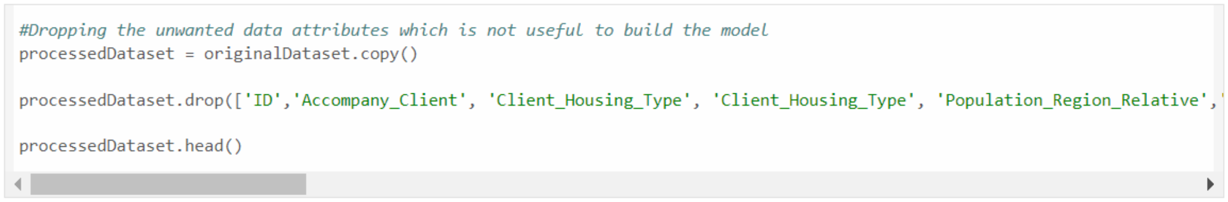

Check High Missing Values contain columns

```
# Define the all columns list
All_columns = [
    'ID', 'Client_Income', 'Car_Owned', 'Bike_Owned', 'Active_Loan', 'House_Own', 'Child_Count', 'Credit_Amount', 'Loan_A
]

# Calculate percentage of missing values
missing_percentage = (originalDataset[All_columns].isnull().sum() / len(originalDataset)) * 100

# Display columns with more than a certain threshold of missing values (e.g., 10%)
high_missing_columns = missing_percentage[missing_percentage > 10]
print("Columns with more than 10% missing values:")
print(high_missing_columns)
```

```
Columns with more than 10% missing values:
Own_House_Age          66.802089
Client_Occupation      32.238344
Score_Source_1         57.937706
Score_Source_3         23.014123
Social_Circle_Default  52.822596
Credit_Bureau          16.281679
dtype: float64
```


```
[37] # Calculate low variance (columns with low unique values)
     unique_values = originalDataset[All_columns].nunique()

     # Find columns with low unique values (e.g., less than 3 unique values)
     low_variance_columns = unique_values[unique_values < 2]
     print("Columns with less than 2 unique values (low variance):")
     print(low_variance_columns)
```

```
Columns with less than 2 unique values (low variance):
Mobile_Tag    1
dtype: int64
```

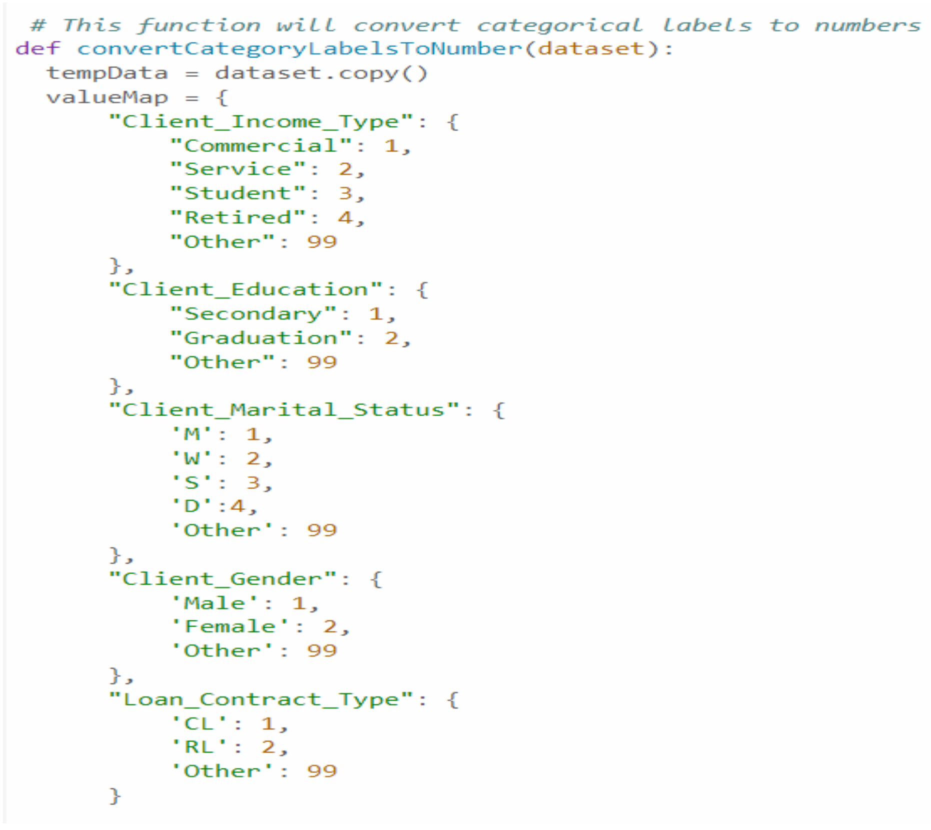

```
#Dropping the unwanted data attributes which is not useful to build the model
processedDataset = originalDataset.copy()

processedDataset.drop(['ID','Accompany_Client', 'Client_Housing_Type', 'Client_Housing_Type', 'Population_Region_Relative','

processedDataset.head()
```

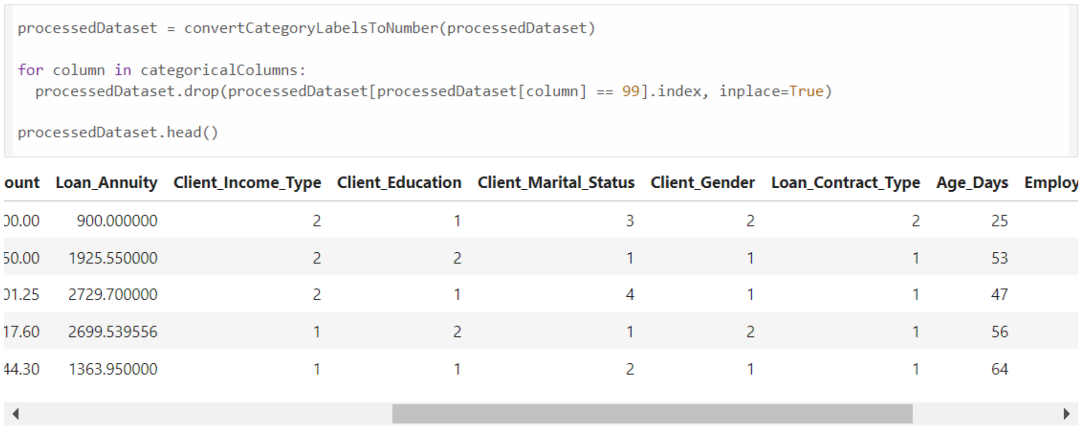
Data Transformation and Normalization

Makes it easier to create prediction models, data transformation can be used to convert categorical variables into numerical ones. (galaktika-soft, n.d.).

Well-known techniques for converting to numerical type include label encoding, one hot encoding, and Dummy coding. Dummy coding and one hot encoding both change one categorical variable into multiple binary columns, as opposed to label encoding, which converts each unique value in a categorical column into a distinct numerical value. Label encoding was used in this situation.

The columns like 'Client_Income', 'Credit_Amount', and 'Loan_Annuity' are converted to numerical types which were originally categorical type columns in the original dataset.


```
 # This function will convert categorical labels to numbers
def convertCategoryLabelsToNumber(dataset):
  tempData = dataset.copy()
  valueMap = {
      "Client_Income_Type": {
          "Commercial": 1,
          "Service": 2,
          "Student": 3,
          "Retired": 4,
          "Other": 99
      },
      "Client_Education": {
          "Secondary": 1,
          "Graduation": 2,
          "Other": 99
      },
      "Client_Marital_Status": {
          'M': 1,
          'W': 2,
          'S': 3,
          'D':4,
          'Other': 99
      },
      "Client_Gender": {
          'Male': 1,
          'Female': 2,
          'Other': 99
      },
      "Loan_Contract_Type": {
          'CL': 1,
          'RL': 2,
          'Other': 99
      }
```



```python
processedDataset = convertCategoryLabelsToNumber(processedDataset)

for column in categoricalColumns:
  processedDataset.drop(processedDataset[processedDataset[column] == 99].index, inplace=True)

processedDataset.head()
```

ount	Loan_Annuity	Client_Income_Type	Client_Education	Client_Marital_Status	Client_Gender	Loan_Contract_Type	Age_Days	Employ
00.00	900.000000	2	1	3	2	2	25	
50.00	1925.550000	2	2	1	1	1	53	
01.25	2729.700000	2	1	4	1	1	47	
17.60	2699.539556	1	2	1	2	1	56	
44.30	1363.950000	1	1	2	1	1	64	

Null Value Handling

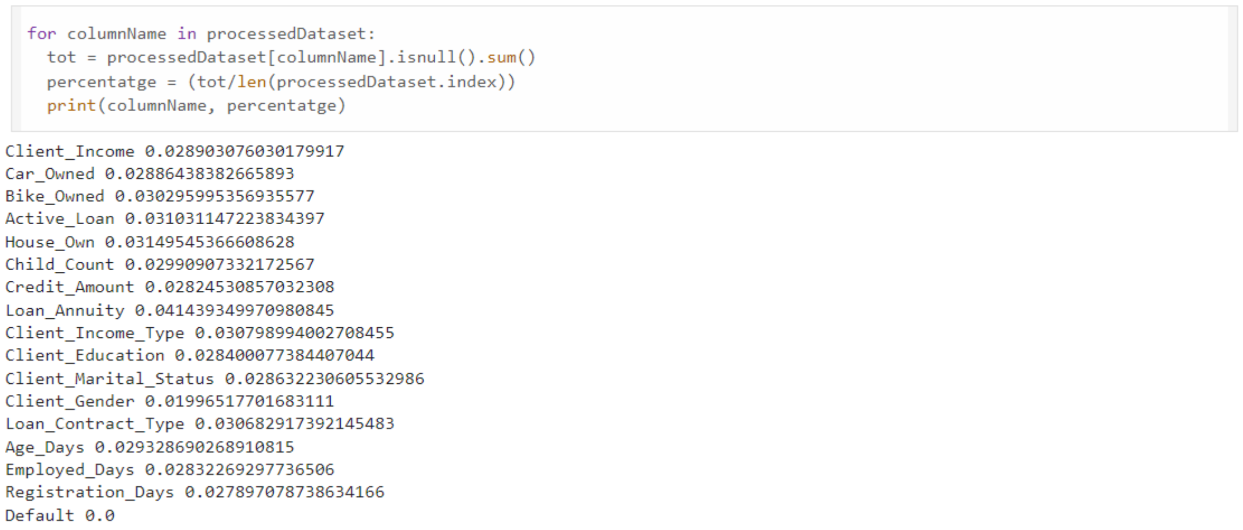
There are two general ways to handle missing values in building operational data. The first is to simply discard data samples with missing values as most data mining algorithms cannot handle data with missing values (Sucky, n.d.). Such a method is only applicable when the proportion of missing values is insignificant. The second is to apply missing value imputation methods to replace missing data with inference values.

Thus, in this step columns containing more than 30% of null values, such as "Own house age" and "Social Circle Default," are not included in the data collection.

To impute missing values from other columns, the mean technique is utilized.

1. Identifying Missing Values

You first explore the dataset using functions like .isnull().sum() to check for columns with missing values. This step helps in identifying which columns contain null values and how many such values exist in each column.



```python
for columnName in processedDataset:
  tot = processedDataset[columnName].isnull().sum()
  percentatge = (tot/len(processedDataset.index))
  print(columnName, percentatge)
```

```
Client_Income 0.028903076030179917
Car_Owned 0.02886438382665893
Bike_Owned 0.030295995356935577
Active_Loan 0.031031147223834397
House_Own 0.03149545366608628
Child_Count 0.02990907332172567
Credit_Amount 0.028245308570323308
Loan_Annuity 0.041439349970980845
Client_Income_Type 0.030798994002708455
Client_Education 0.028400077384407044
Client_Marital_Status 0.028632230605532986
Client_Gender 0.01996517701683111
Loan_Contract_Type 0.030682917392145483
Age_Days 0.029328690268910815
Employed_Days 0.02832269297736506
Registration_Days 0.027897078738634166
Default 0.0
```

2. Handling Missing Values

For numerical columns, fill missing values with the column's mean.

```python
# This function will impute missing numeric values
from sklearn.impute import SimpleImputer
import math

def fillMissingNumericValues(dataset):
  tempData = dataset
  numericColumns = ['Client_Income','Credit_Amount','Loan_Annuity','Age_Days','Employed_Days','Registration_Days']
  wholeNumberColumns = ['House_Own', 'Child_Count', 'Active_Loan', 'Bike_Owned', 'Car_Owned']

  for numericColumn in numericColumns:
    tempData[numericColumn] = pd.to_numeric(tempData[numericColumn],errors = 'coerce')

  imputer = SimpleImputer(strategy='mean', missing_values=np.nan)

  for column in numericColumns:
    data = tempData[[column]]
    imputer = imputer.fit(data)
    tempData[column] = imputer.transform(data)

  for column in wholeNumberColumns:
    imputer = SimpleImputer(strategy='constant',
                        missing_values=np.nan, fill_value=0.0)
    data = tempData[[column]]
    imputer = imputer.fit(data)
    tempData[column] = imputer.transform(data)

  return tempData
```

For categorical columns, fill missing values with the 'Other'.

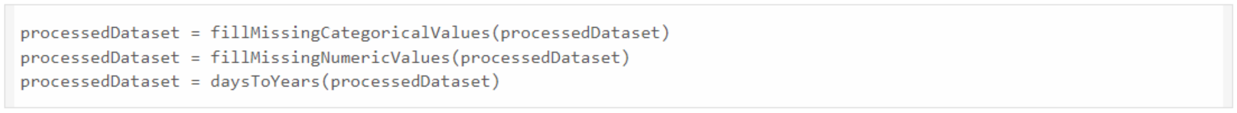
```python
# This function will fill the missing values in categorical data with value 'Other'
def fillMissingCategoricalValues(dataset):
  tempData = dataset
  valueMap = {
    "Client_Income_Type": ['Service','Commercial','Retired' , 'Student' , 'Unemployed'],
    "Client_Education": ['Secondary','Graduation'],
    "Loan_Contract_Type": ['CL', 'RL'],
    "Client_Marital_Status": ['M', 'W', 'S', 'D'],
    "Client_Gender": ['Male', 'Female']
  }

  for column in valueMap.keys():
    tempData[column] = [value if value in valueMap[column] else 'Other' for value in tempData[column]]

  return tempData
```

For days columns, fill missing values with the '0' and convert the days to years.

```python
# This function will convert the days to years
def daysToYears(dataset):
  tempData = dataset
  dayColumns = ['Age_Days','Employed_Days','Registration_Days']

  for column in dayColumns:
    tempData[column] = [math.ceil(days/365) if not math.isnan(days) else 0 for days in tempData[column]]

  return tempData
```

3. Justifying the Chosen Method

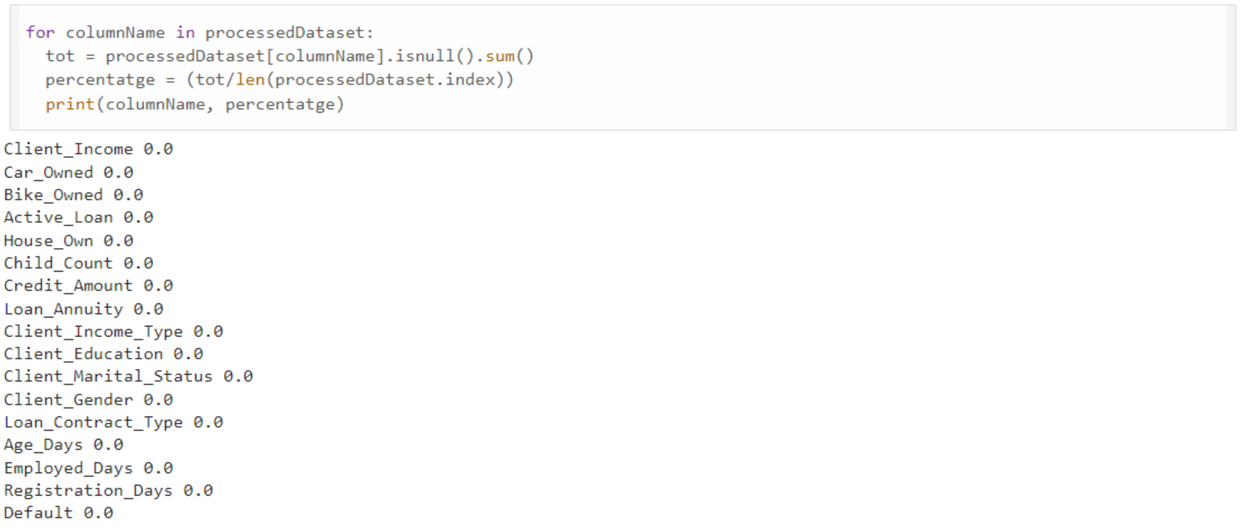
selected a particular method for null value handling.



```
processedDataset = fillMissingCategoricalValues(processedDataset)
processedDataset = fillMissingNumericValues(processedDataset)
processedDataset = daysToYears(processedDataset)
```

4.Verifying the Process

After handling the missing values, recheck the dataset using .isnull().sum() to confirm that all missing values have been dealt with.



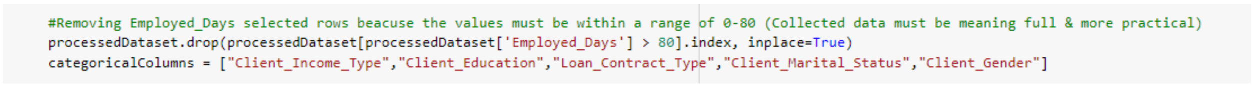
```
for columnName in processedDataset:
  tot = processedDataset[columnName].isnull().sum()
  percentatge = (tot/len(processedDataset.index))
  print(columnName, percentatge)
```

```
Client_Income 0.0
Car_Owned 0.0
Bike_Owned 0.0
Active_Loan 0.0
House_Own 0.0
Child_Count 0.0
Credit_Amount 0.0
Loan_Annuity 0.0
Client_Income_Type 0.0
Client_Education 0.0
Client_Marital_Status 0.0
Client_Gender 0.0
Loan_Contract_Type 0.0
Age_Days 0.0
Employed_Days 0.0
Registration_Days 0.0
Default 0.0
```

After converting The Age from Days to Years and Removing Incorrect Age Rows

It was observed that the dataset records the data in days rather than in years, hence, it was decided to convert this column data into years.

After the conversion of the data, it was observed that there are ages that exceed the maximum human life span. Hence, they were considered incorrect data. An average lifespan was considered as 80 years, and we dropped the rows that record the age greater than 80.



```
#Removing Employed_Days selected rows beacuse the values must be within a range of 0-80 (Collected data must be meaning full & more practical)
processedDataset.drop(processedDataset[processedDataset['Employed_Days'] > 80].index, inplace=True)
categoricalColumns = ["Client_Income_Type","Client_Education","Loan_Contract_Type","Client_Marital_Status","Client_Gender"]
```

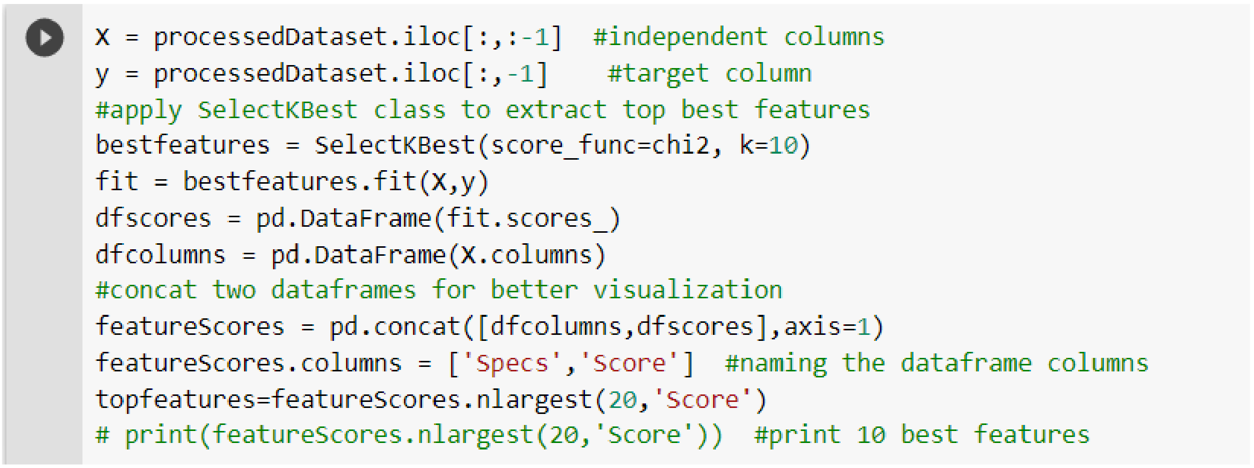
Feature Selection

There are still a lot of columns in the data set after carrying out the aforementioned procedures and preparing them. The following step, feature selection, is the most noticeable one. An alternative term for this is choosing variables or characteristics. Feature selection is the process of selecting the fewest possible useful attribute sets. The following techniques can be used to choose features:

- Information Gain filtering
- Chi-square Test
- Fisher's Score
- Correlation Coefficient score
- K-highest scores

The Correlation Coefficient test's k-highest scores were used to pick the features in this case. The correlation coefficient is a metric used to evaluate a linear relationship between two variables. High correlations with the goal variable and no correlations between the variables themselves make for good model variables. One variable should be utilized for prediction when two are highly correlated because the other doesn't give the model any additional weight. The Pearson correlation has been used for this (Brownlee, 2020).

Even when the cutoff value is set to an absolute 0.5, the correlation coefficient scores can only be used to exclude three attributes. The k-highest scores strategy is used to obtain the best attributes.



```
X = processedDataset.iloc[:,:-1]  #independent columns
y = processedDataset.iloc[:,-1]    #target column
#apply SelectKBest class to extract top best features
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
topfeatures=featureScores.nlargest(20,'Score')
# print(featureScores.nlargest(20,'Score'))  #print 10 best features
```

From the above, the best features were selected to proceed with the model building.

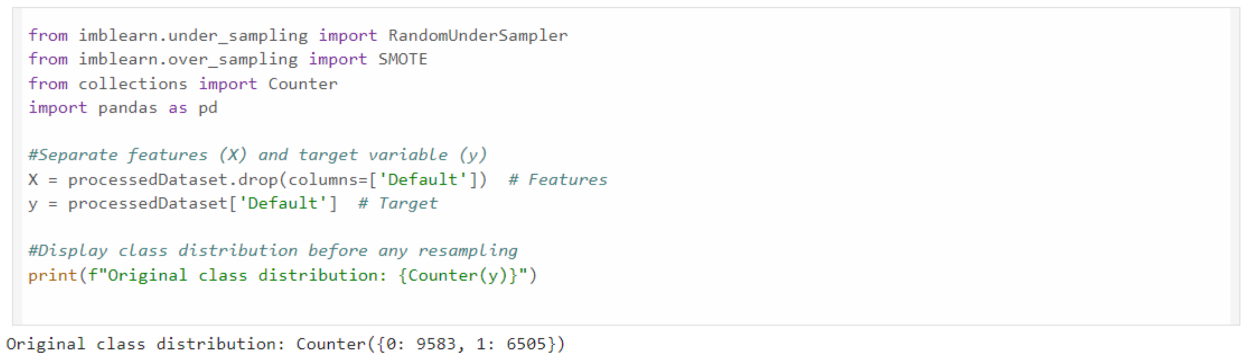
After selecting the top best features, the data set will be divided into the train (x -without target class) and test (y-only target class) vectors.

Solving the Issue of an Imbalanced Data Set

1.Understanding Data Imbalance

As mentioned, and displayed in the data visualization step, the data set is highly imbalanced.

The target was about 90% skewed towards the Not Default type, this was a major issue as this affects the overall accuracy of the model.

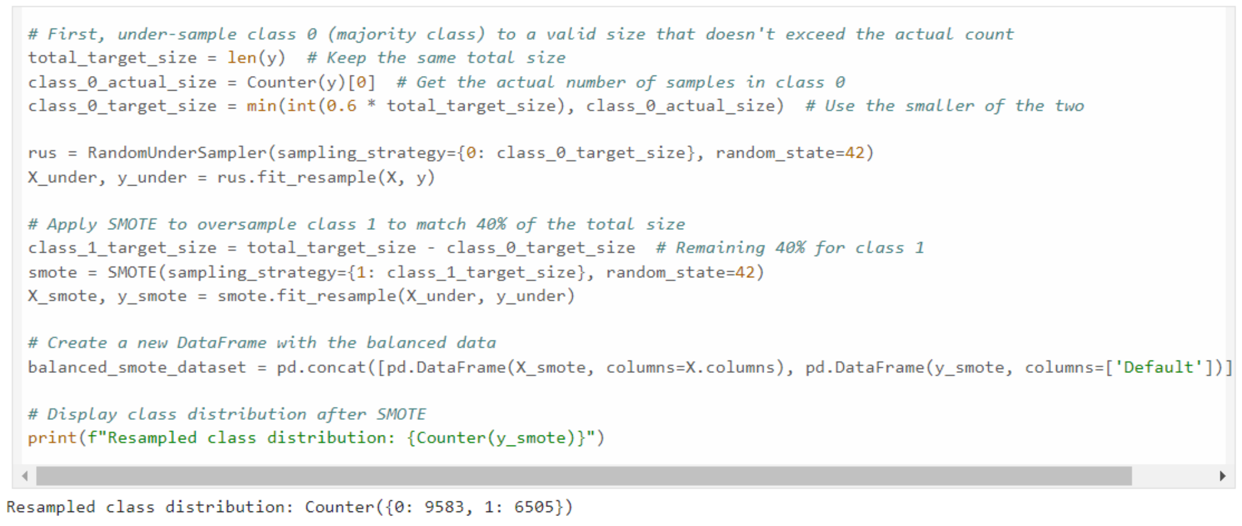


```python
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE
from collections import Counter
import pandas as pd

#Separate features (X) and target variable (y)
X = processedDataset.drop(columns=['Default'])  # Features
y = processedDataset['Default']  # Target

#Display class distribution before any resampling
print(f"Original class distribution: {Counter(y)}")
```

```
Original class distribution: Counter({0: 9583, 1: 6505})
```

2.Use SMOTE Techniques to Handle Imbalanced Data

Therefore, to rebalance the data set SMOTE technique has been used here.



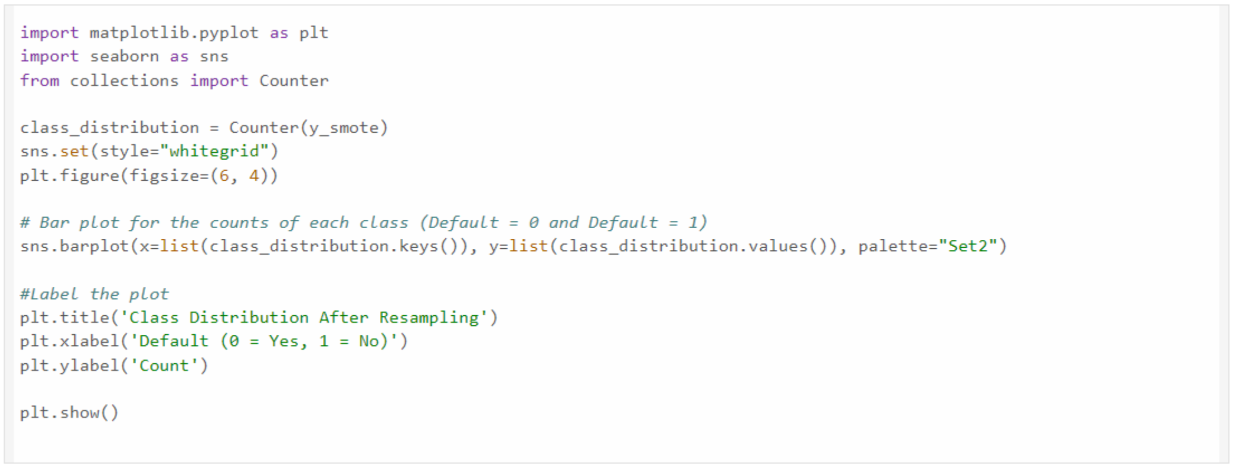
```python
# First, under-sample class 0 (majority class) to a valid size that doesn't exceed the actual count
total_target_size = len(y)  # Keep the same total size
class_0_actual_size = Counter(y)[0]  # Get the actual number of samples in class 0
class_0_target_size = min(int(0.6 * total_target_size), class_0_actual_size)  # Use the smaller of the two

rus = RandomUnderSampler(sampling_strategy={0: class_0_target_size}, random_state=42)
X_under, y_under = rus.fit_resample(X, y)

# Apply SMOTE to oversample class 1 to match 40% of the total size
class_1_target_size = total_target_size - class_0_target_size  # Remaining 40% for class 1
smote = SMOTE(sampling_strategy={1: class_1_target_size}, random_state=42)
X_smote, y_smote = smote.fit_resample(X_under, y_under)

# Create a new DataFrame with the balanced data
balanced_smote_dataset = pd.concat([pd.DataFrame(X_smote, columns=X.columns), pd.DataFrame(y_smote, columns=['Default'])]

# Display class distribution after SMOTE
print(f"Resampled class distribution: {Counter(y_smote)}")
```

```
Resampled class distribution: Counter({0: 9583, 1: 6505})
```

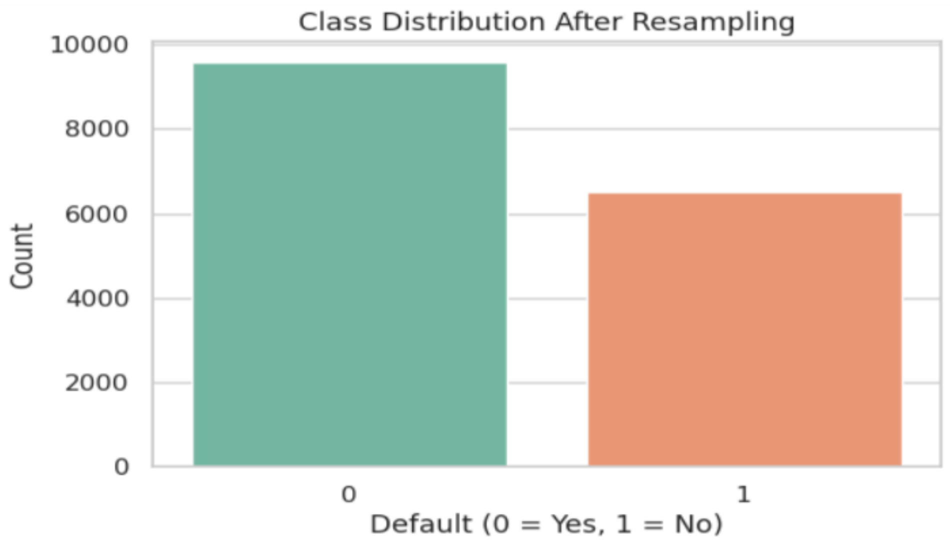
SMOTE allows the model to have more balanced data, improving its ability to learn from both the majority and minority classes. It helps the model generalize better to the minority class, improving performance on important metrics like Recall and F1-Score.

SMOTE helps you generate new, balanced training data that includes both existing and synthetic minority class samples, improving our model's ability to accurately predict both classes in an imbalanced dataset.

After Balance Dataset using the SMOTE method Visualize the Default Distribution



```
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter

class_distribution = Counter(y_smote)
sns.set(style="whitegrid")
plt.figure(figsize=(6, 4))

# Bar plot for the counts of each class (Default = 0 and Default = 1)
sns.barplot(x=list(class_distribution.keys()), y=list(class_distribution.values()), palette="Set2")

#Label the plot
plt.title('Class Distribution After Resampling')
plt.xlabel('Default (0 = Yes, 1 = No)')
plt.ylabel('Count')

plt.show()
```



Proposed Data Mining Solutions

Data mining is considered the process of extracting useful information from a vast amount of data. It’s used to discover new, accurate, and useful patterns in the data, looking for meaning and relevant information for the organization or individual who needs it. It’s a tool used by humans.

On the other hand, machine learning is the process of discovering algorithms that have improved courtesy of experience derived from data. It's the design, study, and development of algorithms that permit machines to learn without human intervention. It's a tool to make machines smarter, eliminating the human element (but not eliminating humans themselves; that would be wrong).

Classification is a technique to categorize data into a given number of classes. This problem needs to be categorized as the 'Default' and 'Not Default' classes. The used classification models are,

1. Support Vector Machine
2. Logistic Regression
3. Decision Tree
4. Random Forest
5. Gaussian Naive Bayes

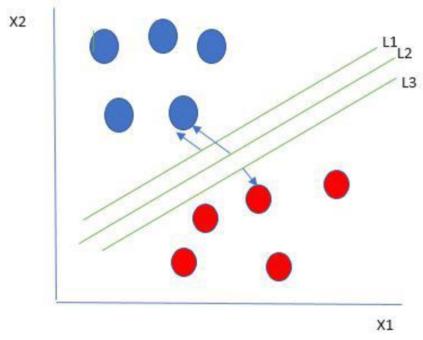
Each model was built with hyperparameter tuning which is a technique used to find correct hyperparameters for machine learning or deep learning models. The goal of hyperparameter tuning is to get the maximum performance out of models. Grid Search hyperparameter tuning method has been used here.
For each model testing and training accuracies and other metrics like precision, recall, and F1 scores are derived.

Support Vector Machine Algorithm

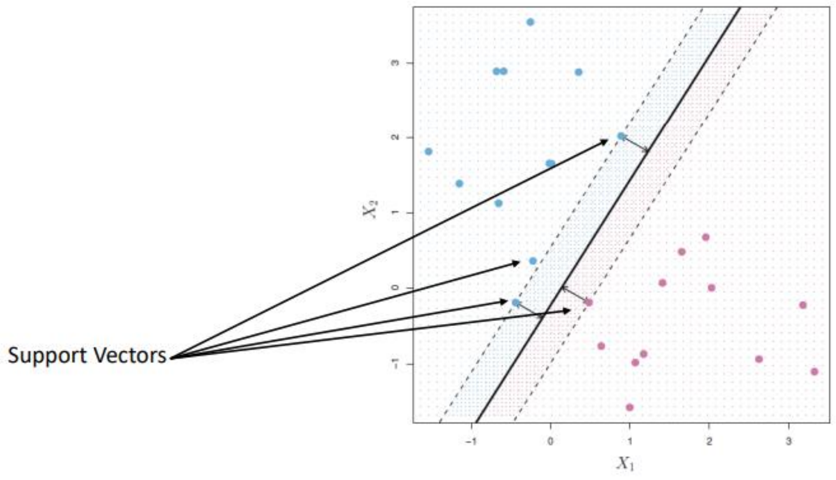
Support Vector Machine (SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well it's best suited for classification. The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points. The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three (Ray, 2017).

So how do we choose the best line or in general the best hyperplane that segregates our data points?

- One reasonable choice as the best hyperplane is the one that represents the largest separation or margin between the two classes.

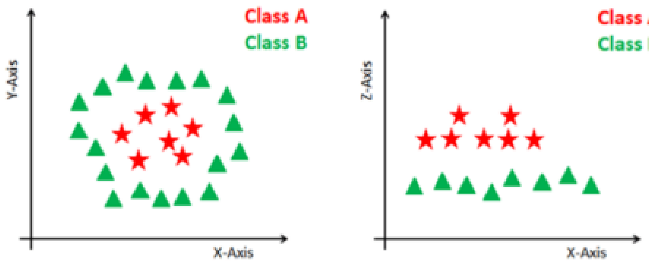


- Points on the margin or close to the margin are called Support Vectors.



So, we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the maximum-margin hyperplane/hard margin. So, from the above figure, we choose L2.

Some problems can’t be solved using linear hyperplane, as shown in the figure below. In such a situation, SVM uses a kernel trick to transform the input space to a higher dimensional space as shown on the right.



SVM Kernel

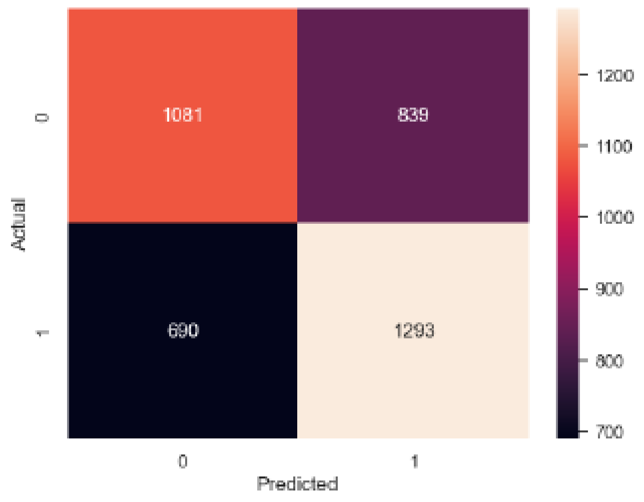
The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, i.e. it converts non separable problems to separable problems. It is mostly useful in non-linear separation problems. Simply put the kernel does some extremely complex data transformations and then finds out the process to separate the data based on the labels or outputs defined (Ray, 2017).

- Some popular kernels are,
    1. Linear Kernel
    2. Polynomial Kernel
    3. Radial Basis Function Kernel

8.1 SVM Model

```
[72]: # Initialize the SVM model
model = SVC(kernel='rbf', C=1, gamma='scale')  # You can adjust kernel, C,␣
 ↪gamma as needed

# Call the classify function with the SVM model
classify(model, x, y)
```

```
Train Accuracy - : 0.660
Test Accuracy - : 0.608
```



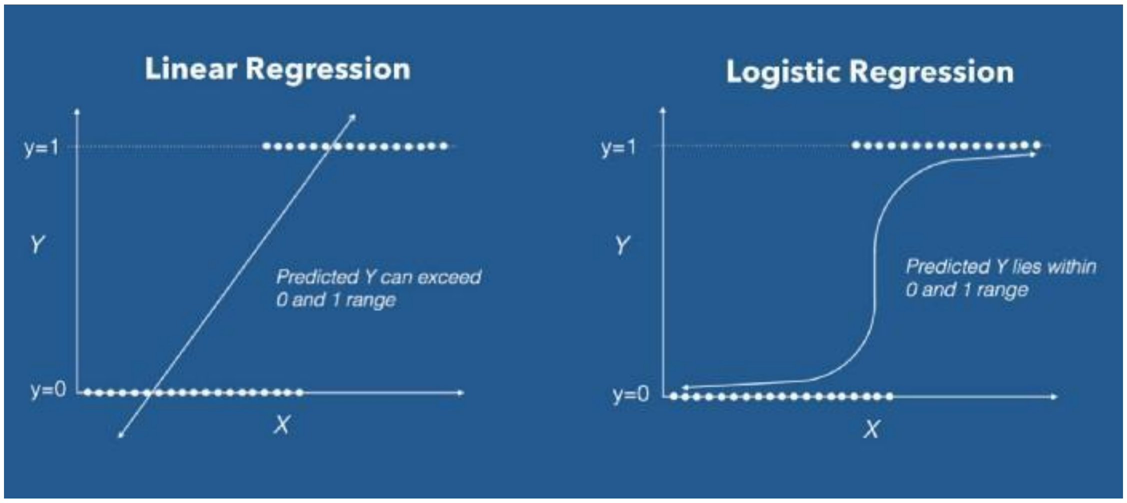
```
              precision    recall  f1-score   support

           0       0.61      0.56      0.59      1920
           1       0.61      0.65      0.63      1983

    accuracy                           0.61      3903
```

- We have used the ‘rbf’ kernel and gamma ‘Scale’ for hyperparameter optimization in this use case

Logistic Regression

Logistic regression is a Machine Learning classification algorithm that is used to predict the probability of certain classes based on some dependent variables. In short, the logistic regression model computes a sum of the input features and calculates the logistic of the result.

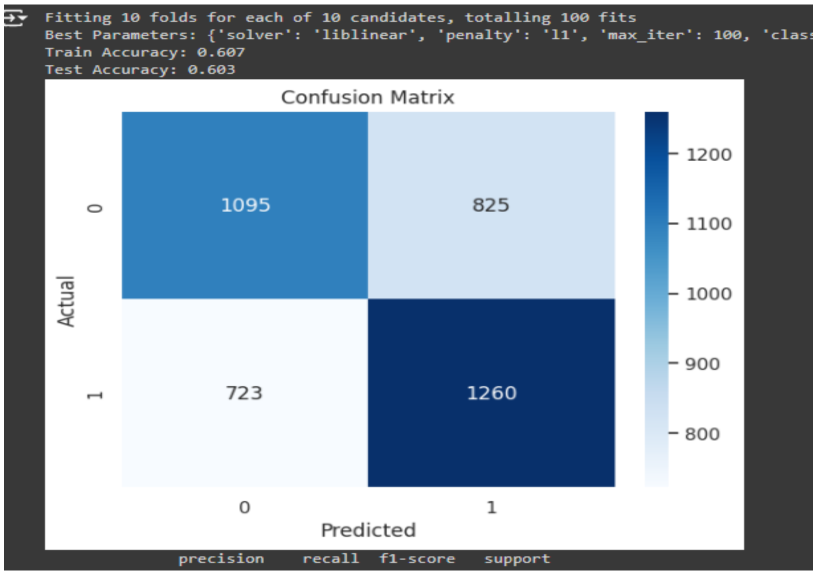
The output of logistic regression is always between (0, and 1), which is suitable for a binary classification task. The higher the value, the higher the probability that the current sample is classified as class=1, and vice versa (Swaminathan, 2018).



We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the '**Sigmoid function**' or also known as the 'logistic function' instead of a linear function.

The equation of the sigmoid function is:

$$p(x) = \frac{1}{1 + \mathrm{e}^{-(\beta_0 + \beta_1 x)}}$$



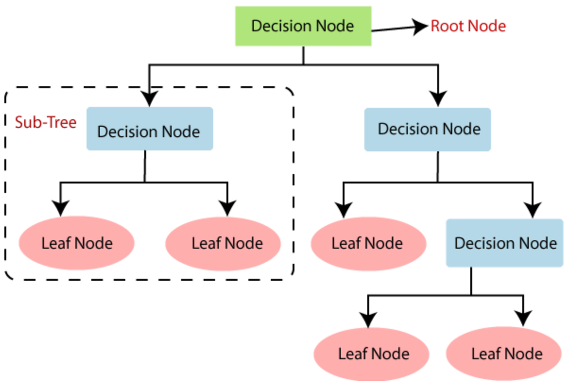
```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Best Parameters: {'solver': 'liblinear', 'penalty': 'l1', 'max_iter': 100, 'clas
Train Accuracy: 0.607
Test Accuracy: 0.603
```

```
            precision    recall  f1-score   support
```

Decision Tree

Decision Tree is one of the most popular classification models available.

A decision tree is a non-parametric supervised learning method for classification and regression. To create a model that predicts the value of a target variable, the goal is to learn simple decision rules generated from the data attributes (scikit-learn, scikit-learn, n.d.).



The main challenge of the decision tree is to find the best splitting. This process is known as attribute selection. The most popular attribute selection option in this situation is information gain. Information gain is assessed using the Gini index and entropy. Gini and entropy are both measures of a node's impurity. When a node has one class, it is said to be pure; when it has numerous classes, it is said to be impure.

$$Gini(t) = 1 - \sum_{i=1}^{j} P(i|t)^2$$

Where,

The j represents the number of classes in the label, and

The P represents the ratio of class at the ith node.

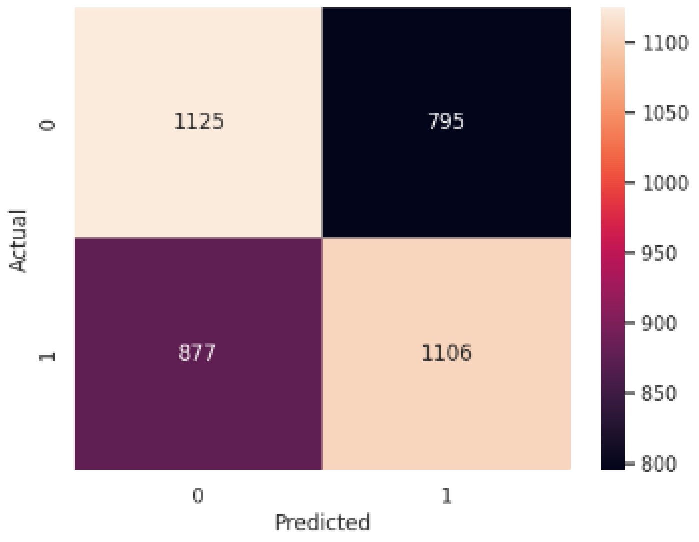
$$E = -\sum_{i=1}^{N} p_i log_2 p_i$$

Below is the model and accuracy that was returned with the model when the Decision Tree model was built, trained, and tested against the dataset.

Decision Tree Classifier Model

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV
import numpy as np
# Initialize the Decision Tree model
model = DecisionTreeClassifier()

model = DecisionTreeClassifier(criterion='gini'
                               , max_depth=20
                               , max_features='sqrt'
                               , min_samples_leaf= 1
                               , min_samples_split=2)

classify(model, X_resampled, y_resampled)
```

```
Train Accuracy - : 0.965
Test Accuracy - : 0.572
```



```
              precision    recall  f1-score   support

           0       0.56      0.59      0.57      1920
           1       0.58      0.56      0.57      1983

    accuracy                           0.57      3903
   macro avg       0.57      0.57      0.57      3903
weighted avg       0.57      0.57      0.57      3903

Cross-validation mean score: 0.572
```

Train Accuracy: 0.966 - This indicates that the model performs very well on the training set (96.6% accuracy), which might suggest overfitting because of the large difference with test accuracy.

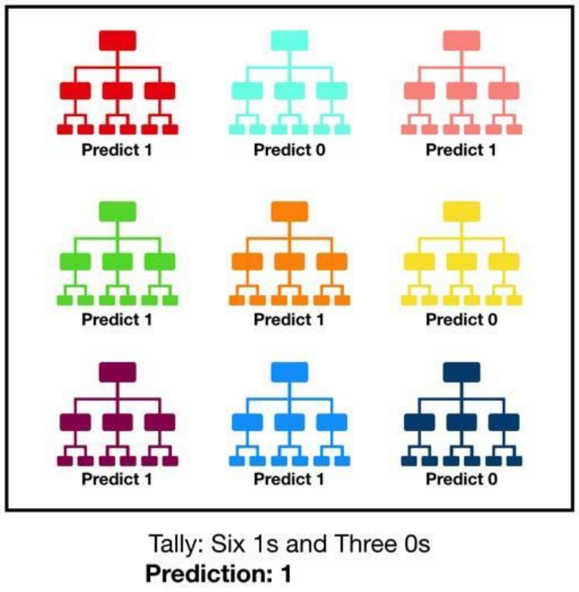
Test Accuracy: 0.567 - The model's accuracy on the test data is lower, at 56.7%, showing that the model struggles to generalize well to unseen data, which is much lower compared to training accuracy.

Overfitting is likely present, as indicated by the large difference between train accuracy (96.6%) and test accuracy (56.7%).

Random Forest Classifier

A popular algorithm for classification and regression issues is the supervised machine learning technique known as random forest. It creates decision trees from various samples, relying on their majority for categorization and average for regression. One of the key characteristics of the Random Forest Algorithm is its ability to handle data sets with both continuous variables, as in regression, and categorical variables, as in classification. For categorization issues, it performs better (scikit-learn, scikit-learn, n.d.).

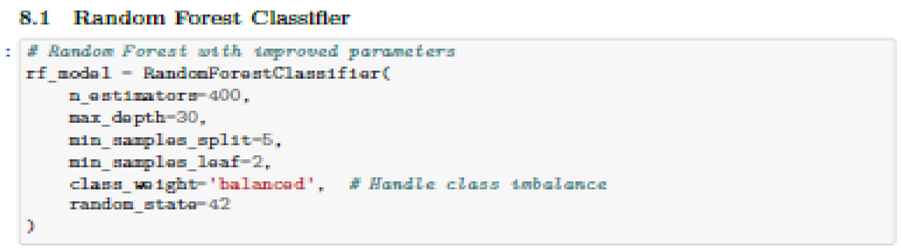
The below picture will provide an understanding of the basic structure of the Random Forest Classifier.



Here, several Decision trees are considered as samples and the predictive result given by each decision tree is considered. Then the predictive results which get the highest number of decision trees are selected. According to the above example, out of the 9 decision trees considered, 6 of them predict as 1 and the other 3 decision trees predict 0. Since 1 is predicted by most of the decision trees, the final prediction is 1.

The below image will provide an instance where we used the Random Forest classifier. It also includes the training and test accuracy of the said classifier in our chosen dataset.

- **NOTE**: After Tunning with XGBoost Random Forest Classifier model was fit into the dataset chosen, as it was the model which gave the highest accuracy.



8.1 Random Forest Classifier

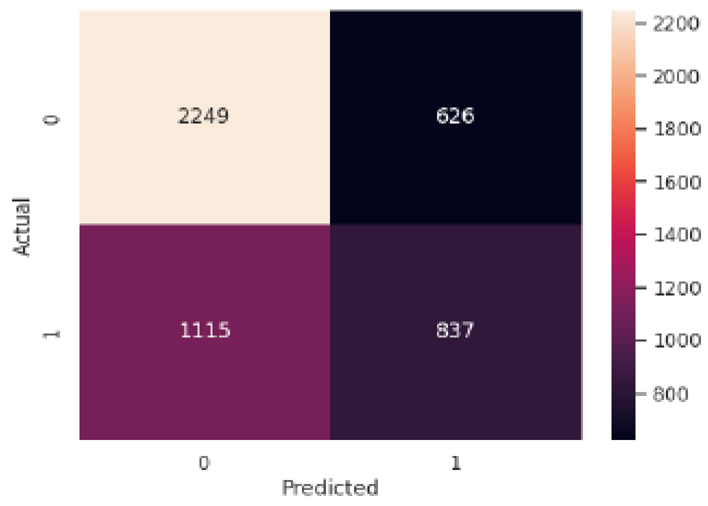
```
# Random Forest with improved parameters
rf_model = RandomForestClassifier(
    n_estimators=400,
    max_depth=30,
    min_samples_split=5,
    min_samples_leaf=2,
    class_weight='balanced',  # Handle class imbalance
    random_state=42
)
```

9 Tunning with XGBoost

```
[ ]: # XGBoost Model (updated, removing the deprecated "use_label_encoder" parameter)
     xgb_model = xgb.XGBClassifier(
         objective='binary:logistic',
         eval_metric='auc',
         n_estimators=200,
         learning_rate=0.05,
         max_depth=7,
         colsample_bytree=0.8,
         subsample=0.8,
         random_state=42
     )

     # RandomizedSearchCV for XGBoost hyperparameter tuning
     param_grid_xgb = {
         'n_estimators': [100, 200, 300],
         'learning_rate': [0.01, 0.05, 0.1],
         'max_depth': [3, 5, 7],
         'colsample_bytree': [0.7, 0.8, 1.0],
         'subsample': [0.7, 0.8, 1.0],
         'gamma': [0, 0.1, 0.3, 0.5]
     }

     random_search_xgb = RandomizedSearchCV(
         estimator=xgb_model,
         param_distributions=param_grid_xgb,
         n_iter=100,
         scoring='roc_auc',
         cv=6,
         verbose=1,
         random_state=42,
         n_jobs=-1
     )
```

```
[ ]: # Run the RandomizedSearchCV for XGBoost
     random_search_xgb.fit(X_smote, y_smote)

     # Output the best parameters from RandomizedSearchCV
     print(f'Best Parameters for XGBoost: {random_search_xgb.best_params_}')
     best_xgb_model = random_search_xgb.best_estimator_

     # Classification with best XGBoost model
     classify(best_xgb_model, X_smote, y_smote)
```

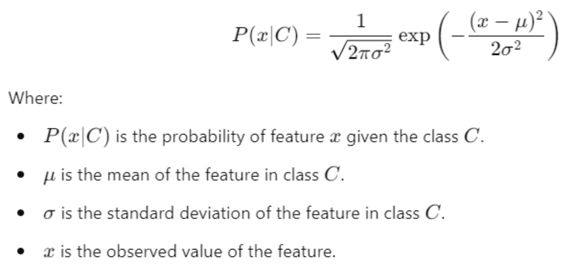
```
Train Accuracy: 0.811
Test Accuracy: 0.639
```



```
              precision    recall  f1-score   support

           0       0.67      0.78      0.72      2875
           1       0.57      0.43      0.49      1952

    accuracy                           0.64      4827
   macro avg       0.62      0.61      0.61      4827
weighted avg       0.63      0.64      0.63      4827
```

Naive Bayes Classifier

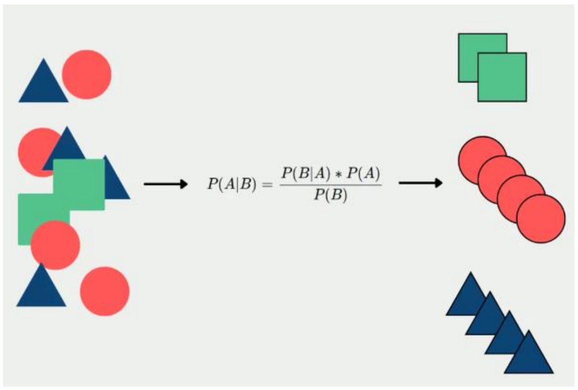
The Naive Bayes classifier is a simple but powerful probabilistic machine learning algorithm used primarily for classification tasks. Naive Bayes classifiers are a group of classification algorithms based on the Bayes' Theorem. Instead of being a single technique, it is a family of algorithms that are all based on the core assumption of conditional independence, meaning that every pair of features (or qualities) is independent of one another, given the class label. This assumption greatly simplifies the calculations, but it is often unrealistic in practice, as features may be correlated.



$$P(x|C) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

Where:

- $P(x|C)$ is the probability of feature x given the class C.
- μ is the mean of the feature in class C.
- σ is the standard deviation of the feature in class C.
- x is the observed value of the feature.



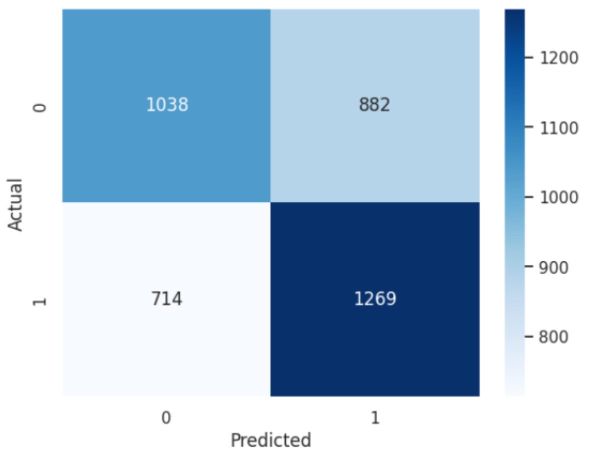
When using the Naive Bayes classifier with continuous data, a common approach is to assume that the values for each feature are normally (or Gaussian) distributed. This is known as Gaussian Naive Bayes. The algorithm calculates probabilities using the Gaussian distribution for each class and then applies Bayes' Theorem to make predictions. (Gandhi, 2018).

By assuming a Gaussian distribution and independence among features, Gaussian Naive Bayes can effectively model and classify data. This method allows for the rapid fitting of a model based on the calculated mean and standard deviation for each feature, making it particularly useful for applications involving high-dimensional datasets or scenarios where computational efficiency is critical.

The Gaussian Naive Bayes classifier can be highly efficient compared to other models, particularly for problems where simplicity, speed, and interpretability are essential.

The accuracy of the Naive Bayes classification model, as determined by the mean score obtained through cross-validation, is presented below after constructing, training, and evaluating the model using the dataset.

```
Train Accuracy - : 0.595
Test Accuracy - : 0.591
```

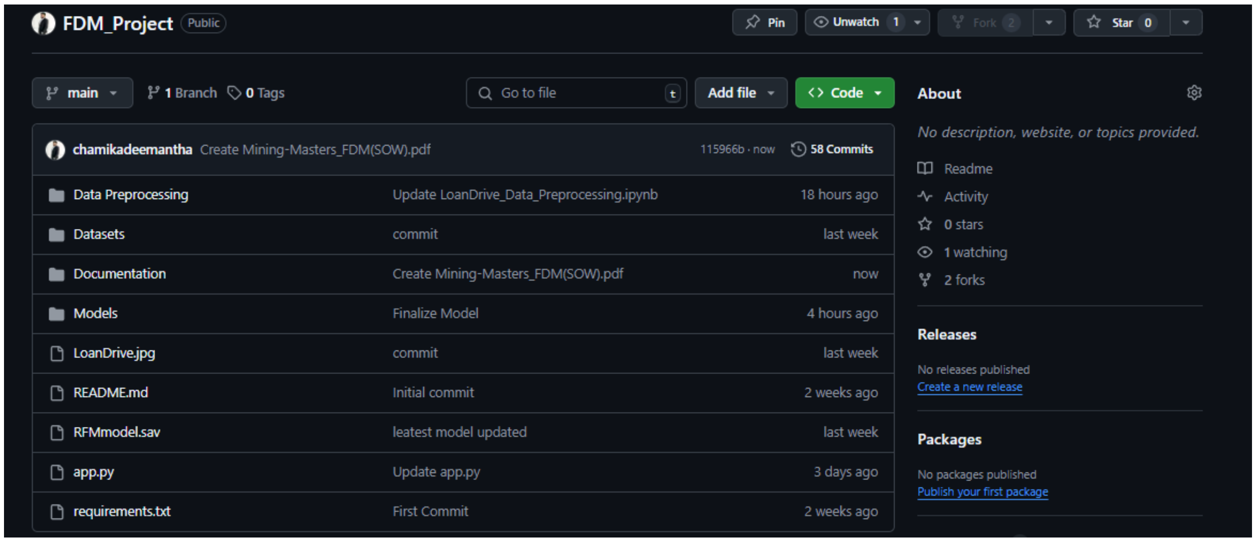


```
              precision    recall  f1-score   support

           0       0.59      0.54      0.57      1920
           1       0.59      0.64      0.61      1983

    accuracy                           0.59      3903
   macro avg       0.59      0.59      0.59      3903
weighted avg       0.59      0.59      0.59      3903

Cross-validation mean score: 0.572
```

Deployment of the Implemented Model

Model deployment is the process of putting machine learning models into production. This makes the model's predictions available to users, developers, or systems, so they can make business decisions based on data, and interact with their applications.

In this case, the project was deployed by pushing the model and the necessary metadata into a Git repository and hosting the application via the Streamlit cloud platform.



chamikadeemantha Create Mining-Masters_FDM(SOW).pdf 115966b · now 58 Commits

Data Preprocessing	Update LoanDrive_Data_Preprocessing.ipynb	18 hours ago
Datasets	commit	last week
Documentation	Create Mining-Masters_FDM(SOW).pdf	now
Models	Finalize Model	4 hours ago
LoanDrive.jpg	commit	last week
README.md	Initial commit	2 weeks ago
RFMmodel.sav	leatest model updated	last week
app.py	Update app.py	3 days ago
requirements.txt	First Commit	2 weeks ago

No description, website, or topics provided.

No releases published
Create a new release

No packages published
Publish your first package

User Interfaces

The user interface (UI) is the point of human-computer interaction and communication in a device.

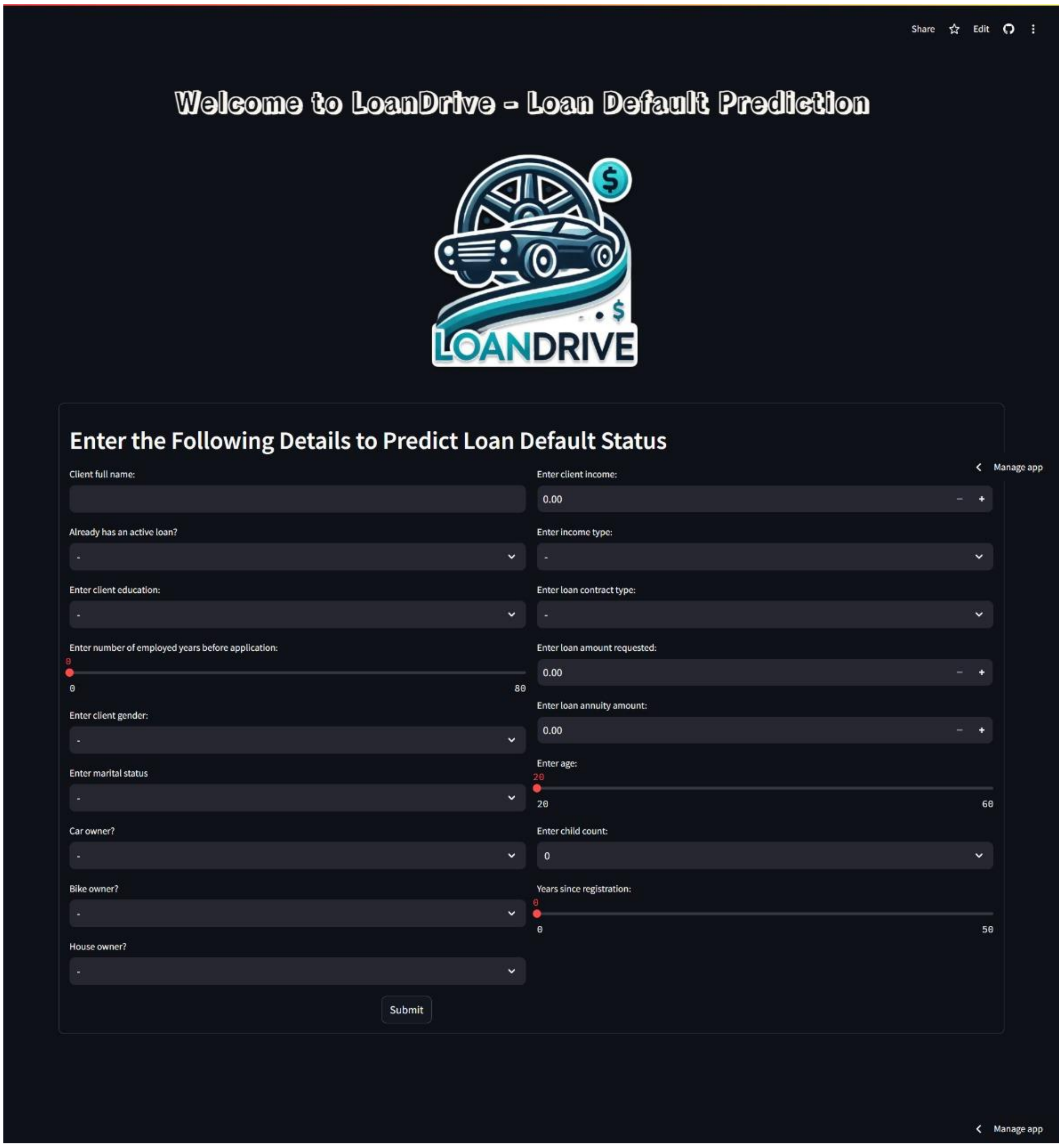
Thus, a very interactive, user-friendly user interface was built for the application. In the user interface build we have chosen the theme of Yellow and White which was continued as the application's theme throughout this project.

The user interface was built with a minimalist theme which requires near to no expertise in IT, hence training the staff would not be an issue for our clients.

The interface obtains the necessary information about the loan request via input fields and feeds them into the trained model. Then a text notification would be displayed which would inform the client prediction and whether to accept or reject the loan request.

All the necessary field validations have been done and testing of the interface has been done for user-friendliness and user experience via a third party.

● The following is a screen capture that represents the UI form that receives the necessary data for the prediction



Welcome to LoanDrive - Loan Default Prediction

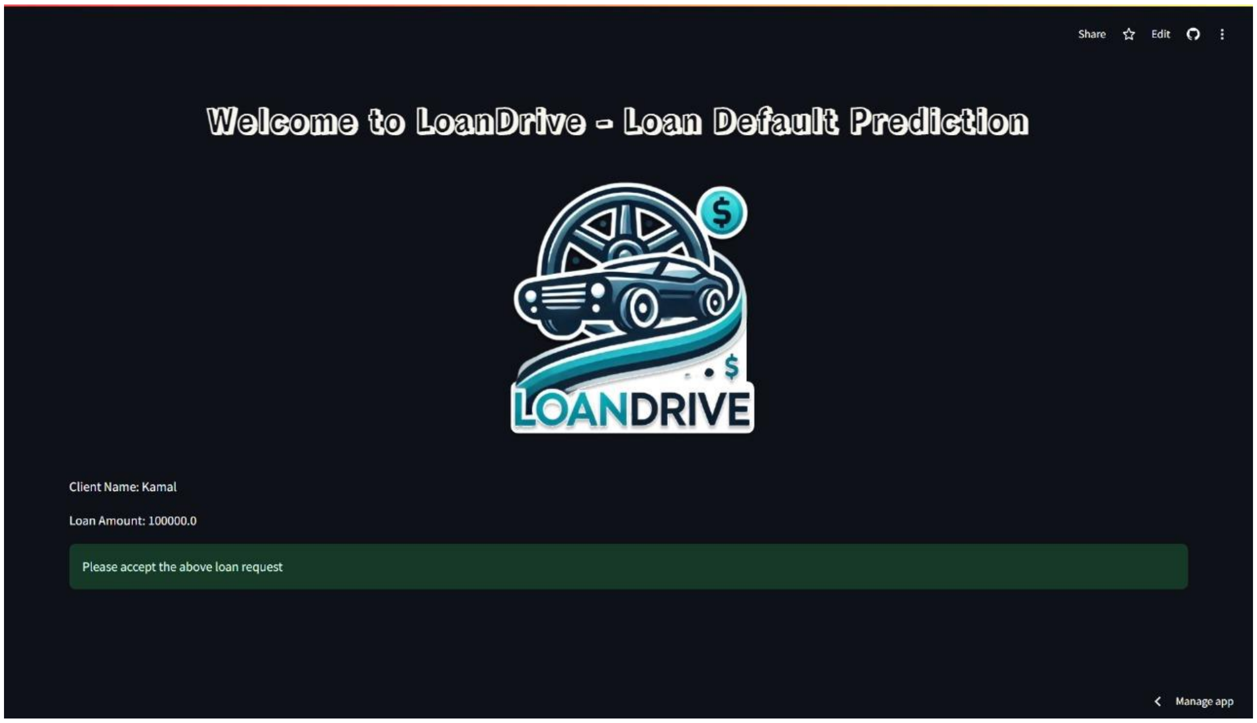
LOANDRIVE

Enter the Following Details to Predict Loan Default Status

Client full name:	Enter client income: 0.00
Already has an active loan? -	Enter income type: -
Enter client education: -	Enter loan contract type: -
Enter number of employed years before application: 0 (0–80)	Enter loan amount requested: 0.00
Enter client gender: -	Enter loan annuity amount: 0.00
Enter marital status -	Enter age: 20 (20–60)
Car owner? -	Enter child count: 0
Bike owner? -	Years since registration: 0 (0–50)
House owner? -	

Submit

- The following is a screen capture that represents the acceptance of a loan request by the model.

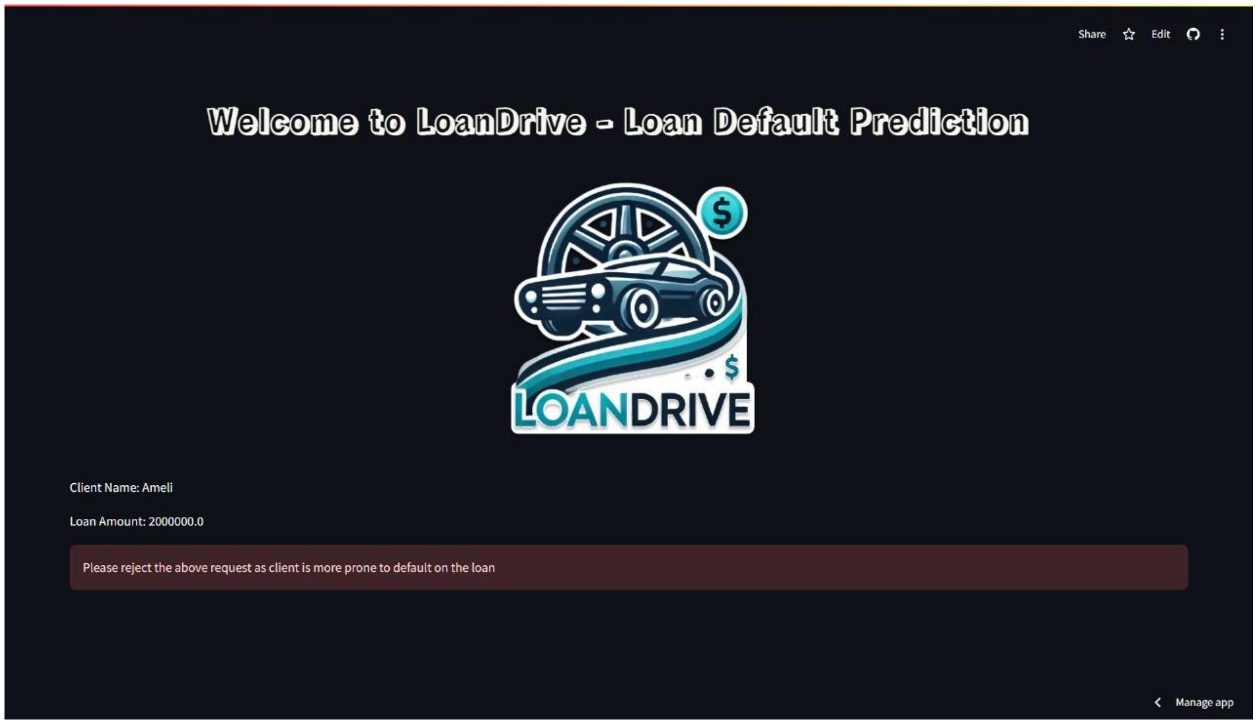


Client Name: Kamal

Loan Amount: 100000.0

Please accept the above loan request

- The following is a screen capture that represents the rejection of a loan request by the model.



Client Name: Ameli

Loan Amount: 2000000.0

Please reject the above request as client is more prone to default on the loan

Conclusion

This project was implemented for a Non-Banking Financial Institution that operates without oversight from international or national monetary authorities. The goal was to develop a predictive model to assess the likelihood of loan repayment by borrowers. The dataset selected for this task focused on the topic of ‘automobile loan default.’

One of the most challenging aspects of the project was data preprocessing. The raw data contained numerous issues such as missing values, inconsistencies, and categorical data that needed conversion to numerical formats. The preprocessing steps included handling missing values, normalizing and reducing the data, and converting categorical variables into numerical ones. Despite these efforts, initial model accuracy was low, prompting further enhancement of the dataset through additional preprocessing and data balancing techniques.

We evaluated five different classification models to determine which provided the highest accuracy for our dataset. The Random Forest Classifier emerged as the best performer, delivering the highest test accuracy after extensive preprocessing.

Following the development, training, and deployment of the model, we created a user interface that allows users to input necessary data to get predictions on loan repayment probabilities. This functionality enables the institution to better assess the creditworthiness of potential borrowers, helping to identify favorable customers who are likely to repay their loans. As a result, the company can minimize losses, increase efficiency, and target profitable customer segments more effectively.

References

Brownlee, J. (2020, August 20). *machinelearningmastery*. Retrieved from https://machinelearningmastery.com/feature-selection-with-real-and-categorical-data/

galaktika-soft. (n.d.). *galaktika*. Retrieved from https://galaktika-soft.com/blog/data-mining-normalization.html

Gandhi, R. (2018, May 5). *towardsdatascience*. Retrieved from https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c

geeksforgeeks. (2022, September 27). *geeksforgeeks*. Retrieved from https://www.geeksforgeeks.org/dimensionality-reduction/

Kumar, A. (2022, April 16). *vitalflux*. Retrieved from https://vitalflux.com/correlation-heatmap-with-seaborn-pandas

Ray, S. (2017, September 13). *analyticsvidhya*. Retrieved from https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

S, R. A. (n.d.). *simplilearn*. Retrieved from www.simplilearn.com: https://www.simplilearn.com/tutorials/python-tutorial/data-visualization-in-python

scikit-learn. (n.d.). *scikit-learn*. Retrieved from scikit-learn.org: https://scikit-learn.org/stable/modules/tree.html

scikit-learn. (n.d.). *scikit-learn*. Retrieved from scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

Sucky, R. N. (n.d.). *Medium*. Retrieved from https://towardsdatascience.com/6-tips-for-dealing-with-null-values-e16d1d1a1b33

Swaminathan, S. (2018, March 15). *towardsdatascience*. Retrieved from https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Wu, Y. (2022, August 24). *kdnuggets*. Retrieved from https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html

Xenonstack. (2018, December 23). *Medium*. Retrieved from https://medium.com/@xenonstack/data-preparation-process-preprocessing-and-data-wrangling-6c4068f2fcd1