CAPSTONE PROJECT

Project Title:

MACHINE LEARNING MODELS FOR PREDICTING LOAN ELIGIBILITY

HAND IN DATE: 18 May 2023

Group Members:

1. **CHUA ROU LIN – TP056059 (Group Leader)**
2. **DEVINA WIYANI – TP061652**
3. **EDWARD LEONARDO – TP058284**
4. **ADJANI PUTRI CEMYLLA UDAYANA – TP061651**

Table of Contents

1.0 Introduction

Loan distribution is a pivotal aspect of the financial and banking sector, serving as a significant revenue generator for banks. Ensuring the eligibility of borrowers before approving loans is of utmost importance. Mehta (2022) highlights the requirement for borrowers to undergo a credit approval process, involving a thorough review of their financial statements. Multiple criteria, including cash flow, existing debts, and credit history, are considered to assess the suitability of borrowers. This rigorous evaluation typically involves professionals such as credit analysts, credit officers, and relationship managers, who collectively strive to invest bank funds in customers capable of repaying the loans.

As the volume of bank loans continues to rise, the loan validation process becomes increasingly demanding. To streamline this process, banks can adopt automated Loan Approval Systems, leveraging predictive models trained on historical data. These systems employ a defined set of features to predict the bank's decision-making process. While complete certainty in borrower eligibility cannot be achieved, patterns and key characteristics derived from past loan data help mitigate risks associated with lending.

The purpose of this research is to develop statistical analysis and predictive models to anticipate the likelihood of loans being charged off by creditors. The study encompasses a comprehensive data exploration phase aimed at better understanding the data and ensuring its integrity. Statistical analysis and data visualization techniques will be employed during the analysis. Prior to feeding the data into the models, appropriate pre-processing and data preparation steps, including cleaning and transformation, will be executed. The research will involve building seven models, namely AdaBoost, GradientBoost, CatBoost, XGBoost, Decision Tree, K-Nearest Neighbor, and Random Forest, which will be evaluated, compared, and extensively documented to determine the model that exhibits the best performance.

2.0 Problem Statements

A bank's lending activity is one of its most significant activities. According to the IMF, bank lending makes up two-thirds of all global credit. Customers are given access to funds that they otherwise would not have had, in addition to the bank benefiting from the interest charge. The system does, however, still have a few drawbacks despite being so advantageous. Discrimination is one of its reoccurring problem. As can be seen in figure 1, lenders are less likely to approve minority-group borrowers than they are the majority. NCRC (2019) discovered that even after controlling for credit score, income, and loan size, black borrowers were more likely to be denied loans than white borrowers from analysing the Home Mortgage Disclosure Act data. The survey also discovered that black borrowers were more likely than white borrowers to be offered high-cost loans, even when they qualified for better credit options.

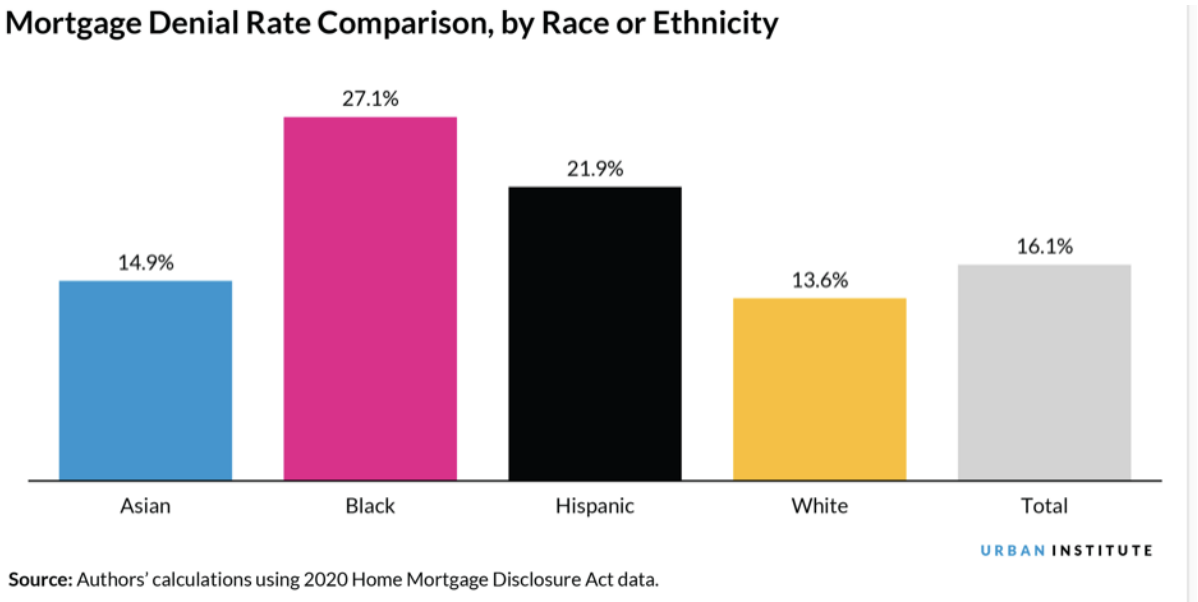

Figure 1 Mortgage Denial Rate Comparison by Race or Ethnicity (source:https://www.urban.org/urban-wire/what-different-denial-rates-can-tell-us-about-racial-disparities-mortgage-market)

Second aspect that takes a toll in the current system is effectiveness. The current system remains inefficient because it is heavily reliant on paper-based processes and manual data entry, resulting in lengthy loan approval turnaround times. Each loan application must be manually reviewed and analysed by loan officers, which takes time. Furthermore, the lending process necessitates the collection of data from various sources, such as credit scores, employment records, and financial statements, resulting in loan approval delays and inconsistencies. The paper-based system also poses as a security threat given that there is a

large possibility of errors. Loan applications may be misplaced or lost, leading to late approvals or incorrect decisions. Documentation errors may result in legal complications or incorrect loan distribution.

Lastly, paper-based loan system can be costly especially on its operational fees. On a paper-based system, a bank needs to spend a fortune for record storage and maintenance. Furthermore, the lending process necessitates a large workforce and substantial infrastructure, such as physical loan processing centres and face-to-face meetings with loan officers.

3.0 Scope & Objectives

3.1 Aim

The aim of this project is to develop a machine learning model that can predict whether a bank loan will be paid back or not to help banks or financial institutions to evaluate the creditworthiness of loan applicants and make informed lending decisions.

3.2 Objectives

1. To identify and select a suitable dataset related to bank loans and prepare the data for analysis.
2. To conduct exploratory data analysis (EDA) to gain insights into the dataset, identify any patterns or relationships between variables and perform pre-processing based on the EDA.
3. To develop one or more machine learning models that can predict whether a loan will be paid back or not based on the available data.
4. To evaluate the performance of the developed models and select the best one based on certain metrics such as accuracy, precision, recall, and F1 score.

3.3 Deliverables

The key deliverables of this project will include:

1. A predictive machine learning model to predict the customers' eligibility to get a bank loan from a bank based on their previous records
2. A simple demonstration of how the loan status predictive model could be used in a real-world scenario.

4.0 Literature Review

4.1 Similar Systems

Citation	Brief Summary	Models/ Techniques Used	Findings/ Limitations
(Gupta et al., 2020)	The paper "Bank Loan Prediction System using Machine Learning" systematically covers a machine learning-based loan prediction system in banking. Through pre-processing, feature engineering, and evaluating multiple algorithms, random forest outperformed others with high accuracy. The study highlights the practical implications for banks, automating loan approvals, and improving decision-making. Accurate loan prediction systems are crucial in enhancing efficiency and mitigating risks. The findings emphasize the significance of informed lending decisions, suggesting the adoption of random forest. Future research may explore	logistic regression, decision trees, random forests, and support vector machines, to train and evaluate the predictive models.	In terms of loan prediction in banking, the random forest method outperforms all other researched models. It demonstrates how machine learning might support lending judgement. The study has a number of flaws, including a reliance on a single dataset, the absence of workable alternatives in terms of methodologies or models, and a lack of consideration of the moral implications of loan prediction. The system's applicability and resilience will

	advanced techniques and additional data sources to further enhance prediction accuracy. Overall, the paper emphasizes the importance of accurate loan prediction systems in empowering banks with efficient decision-making processes.		improve if these issues are addressed.
(S. Sreesouthry, 2021)	The paper "Loan Prediction Using Logistic Regression in Machine Learning" presents a study on utilizing logistic regression for loan prediction in the banking sector. The authors develop a loan prediction model using logistic regression and evaluate its performance based on accuracy, precision, recall, and F1-score. The findings demonstrate the effectiveness of logistic regression in accurately classifying loan applications as approved or rejected. The paper highlights the implications for informed decision-making and risk mitigation in the banking industry. It emphasizes the	Logistic Regression	The study demonstrates that logistic regression is effective for loan prediction, achieving high accuracy, precision, recall, and F1-score. It improves decision-making and risk management in loan approvals. The study may have limited generalizability to different datasets or institutions. Alternative models are not explored, and ethical considerations are not discussed. Addressing these limitations would

	transparency and interpretability of logistic regression models in assessing loan applications. The study contributes to the field of loan prediction and suggests further research on improving models using logistic regression.		enhance the system's applicability and fairness.
(Alsaleem & Hasoon, 2020)	Maan Y Alsaleem and Safwan O Hasoon's study "Predicting Bank Loan Risks Using Machine Learning Algorithms" focuses on using machine learning algorithms to accurately predict bank loan risks. DT J48, Random Forest, Bayes' Theorem, Nave Bayes, and Multilayer Perception are among the algorithms used in the study. The author then compares 1000 data points with supervised learning and targeted data (YES and NO). After that, the authors evaluate their performance leaning towards ROC measurement. The findings emphasise machine learning's usefulness in	DT J48, Random Forest, Bayes' Theorem, Naïve Bayes, Multilayer Perception.	The paper found utilisation of the proposed methods achieve acceptable accuracy rates in predicting loan eligibility and emphasizes the superiority of neural networks for this goal. The paper, however, does not explicitly mention constraints, but potential limitations may include data availability, biases in the dataset, generalizability, and the need for continuous model updates to adapt to

	forecasting loan hazards, as well as its potential for boosting risk assessment and decision-making in the banking sector. The report emphasises the need of precise risk prediction and provides practical implications for banks looking to improve their loan portfolios.		evolving loan risk patterns.

5.0 Methodology/ Process Model

5.1 Methodology - CRISP-DM

Methodology is a framework that acts as a guide on how a person can do a task based on the specific study. In the Data Analytics field, Methodology can be a guide for the data analyst to work on their project, and by following the principle of the methodology, they can reach a satisfactory result from the project. For this project, CRISP-DM methodology is used as the framework for this project.

Cross Industry Standard Process for Data Mining (CRISP-DM) is a data analytics specific methodology. It is regarded as one of the most popular methodologies used in the industry of data mining and data analytics. CRISP-DM is very well-known due to its 6 distinct phases that are tailored specifically for data analytics jobs. Those 6 phases are:

Figure 1 CRISP-DM Overview

1. Business Understanding

   The Business Understanding phase covers the analysts' understanding of the project's objectives and requirements. In this phase, the analysts must fully understand what the project is for, what field-specific context or concept are needed to fully grasp the project, and deep dive into understanding the field of the project to ensure success. In this project, Business Understanding results are stated in the Problem Statements, Scope, and Objectives.

2. Data Understanding

   The second phase is Data Understanding. This phase focuses more on gaining an understanding of the data that is going to be used for the project. This covers the entirety of the dataset, including all of its attributes/variables, all the values, all the flaws or noise embedded inside the data, and thinking on how to clean up the data so it can be suitable for project's usage. In this case, Data Understanding covers all the variables in the Bank Loan Dataset, all the values for each variable, and how to deal with all of the problems of the data.

3. Data Preparation

   The third phase is Data Preparation. Arguably the most important phase of the methodology. Data Preparation covers all the preprocessing required for the data to be suitable for Machine Learning modelling or for Data Visualization. This covers data cleaning, imputing missing data, removing outliers, data transformation, creating new variables, reformatting the data, and much more. In this project, the data can be considered as quite dirty, thus a lot of preprocessing needs to be done before any progress can be made.

4. Modelling

   Next phase is called Modelling. After the data is fully prepared, the data then can be used following the project's purpose. Since this project's objective is to create a predictive model using the dataset, modelling then covers creating all the machine learning algorithms that are suitable for the data. This includes data splitting, and Machine Learning (ML) model creation and training. If there are multiple suitable ML Models, the developers need to do a comparison of all models to ensure the best model is used for the project.

5. Evaluation

   Before the last phase, Evaluation must be held regarding the results from all the previous phases. If there are unsatisfactory results, such as low accuracy scores from the ML models, the developers can backtrack to all the previous phases, evaluate each phase, and redo from wherever they can start making improvements. This phase is repeated until a satisfactory result can be reached from the project. For example, in this

project, an evaluation of the pre-processing steps needs to be done to make sure the model has better accuracy results, since the previous attempt resulted in lower-than-expected results.

6. Deployment

   Finally, after the project is completed, Deployment can be done to allow the project to be used by the target users. This means figuring out how the project is going to be deployed. This phase also covers monitoring and providing maintenance/updates to the project if required, producing final report and documentation of the project if needed, and taking a retrospective look toward the whole project.

5.2 Exploratory Data Analysis (EDA) and Pre-processing

```
# Pandas and numpy for data manipulation
import pandas as pd
import numpy as np

# Matplotlib visualization
import matplotlib.pyplot as plt
%matplotlib inline

# Seaborn for visualization
import seaborn as sns
sns.set(font_scale = 2)
import missingno as msno

import warnings
warnings.filterwarnings('ignore')

# Sampling Technique
from imblearn.over_sampling import SMOTE

# Encoding
from sklearn.preprocessing import LabelEncoder

# Data split
from sklearn.model_selection import train_test_split

# Model Metrics
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,precision_score,recall_score,f1_score,roc_auc_score,roc_curve

# Models
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoostingClassifier
from sklearn.model_selection import cross_val_score,GridSearchCV
```

Figure 2 Import Python Libraries

This code block imports several Python libraries and tools commonly used in data analysis, machine learning, and model evaluation, including Pandas, NumPy, Matplotlib, LabelEncoder, train_test_split, various classifiers and so on. These tools are essential for data preprocessing, modeling, and evaluation in data science projects.

```
# Read data into a dataframe
loan_data = pd.read_csv("/content/sample_data/credit_train.csv")

# Display top of dataframe
loan_data.head()
```

Figure 3 Code Snippets for Read and Display Data Frame

	Loan ID	Customer ID	Loan Status	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Months since last delinquent	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
0	14dd8831-6af5-400b-83ec-68e61888a048	981165ec-3274-42f5-a3b4-d104041a9ca9	Fully Paid	445412.0	Short Term	709.0	1167493.0	8 years	Home Mortgage	Home Improvements	5214.74	17.2	NaN	6.0	1.0	228190.0	416746.0	1.0	0.0
1	4771cc26-131a-45db-b5aa-537ea4ba5342	2de017a3-2e01-49cb-a581-08169e83be29	Fully Paid	262328.0	Short Term	NaN	NaN	10+ years	Home Mortgage	Debt Consolidation	33295.98	21.1	8.0	35.0	0.0	229976.0	850784.0	0.0	0.0
2	4eed4e6a-aa2f-4c91-8851-ce984ee8fb26	5efb2b2b-bf11-4dfd-a572-3761a2694725	Fully Paid	99999999.0	Short Term	741.0	2231892.0	8 years	Own Home	Debt Consolidation	29200.53	14.9	29.0	18.0	1.0	297996.0	750090.0	0.0	0.0

Figure 4 Output of Data Frame Display

This code block reads a CSV file named "credit_train.csv" that located in the '/content/sample_data/' Google Collab's default directory into a Pandas Data Frame called 'loan_data' and displays the top of the data frame using the 'head()' method. The output is shown in Figure above.

```
[ ]  # Check the total rows and columns
     loan_data.shape

     (100514, 19)
```

Figure 5 Code Snippets and Output for Checking Total Number of Rows and Columns

According to the figure above, it shows the dataset contains 100514 rows and 19 columns.

```
# Check data types
loan_data.dtypes

Loan ID                        object
Customer ID                    object
Loan Status                    object
Current Loan Amount           float64
Term                           object
Credit Score                  float64
Annual Income                 float64
Years in current job           object
Home Ownership                 object
Purpose                        object
Monthly Debt                  float64
Years of Credit History       float64
Months since last delinquent  float64
Number of Open Accounts       float64
Number of Credit Problems     float64
Current Credit Balance        float64
Maximum Open Credit           float64
Bankruptcies                  float64
Tax Liens                     float64
dtype: object
```

Figure 6 Code Snippets and Output for Checking Data Types for All Attributes

By using the ‘dtypes()’ method, it can be observed that the ‘loan_data’ data frame contains 7 categorical attributes and 12 numerical attributes.

```
# Drop Unnecessary Columns
loan_data.drop(labels=['Loan ID', 'Customer ID'], axis=1, inplace=True)
```

Figure 7 Code Snippets for Dropping Unnecessary Columns

According to the Figure 4, it shows that the columns ‘Loan ID’ and ‘Customer ID’ from the ‘loan_data’ data frame didn’t convey any useful information for data analysis, so the columns will be dropped.

```
# Function to calculate missing values by column
def missing_values_table(df):
  # Total missing values
  mis_val = df.isnull().sum()

  # Percentage of missing values
  mis_val_percent = 100 * df.isnull().sum() / len(df)

  # Make a table with the results
  mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

  # Rename the columns
  mis_val_table_ren_columns = mis_val_table.rename(
  columns = {0 : 'Missing Values', 1 : '% of Total Values'})

  # Sort the table by percentage of missing descending
  mis_val_table_ren_columns = mis_val_table_ren_columns[
      mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
  '% of Total Values', ascending=False).round(1)

  # Print some summary information
  print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
      "There are " + str(mis_val_table_ren_columns.shape[0]) +
        " columns that have missing values.")

  # Return the dataframe with missing information
  return mis_val_table_ren_columns

missing_values_table(loan_data)
```

Figure 8 Code Snippets for Calculating the Total Number and Percentage of Missing Value for Each Column

The function ‘missing_values_table’ is used to calculate and display the total number and percentage of missing values in each column of the given data frame in table form.

Your selected dataframe has 17 columns.
There are 17 columns that have missing values.

	Missing Values	% of Total Values
Months since last delinquent	53655	53.4
Credit Score	19668	19.6
Annual Income	19668	19.6
Years in current job	4736	4.7
Bankruptcies	718	0.7
Tax Liens	524	0.5
Maximum Open Credit	516	0.5
Current Credit Balance	514	0.5
Number of Credit Problems	514	0.5
Number of Open Accounts	514	0.5
Loan Status	514	0.5
Years of Credit History	514	0.5
Current Loan Amount	514	0.5
Purpose	514	0.5
Home Ownership	514	0.5
Term	514	0.5
Monthly Debt	514	0.5

Figure 9 Output of Total Number and Percentage of Missing Value for Each Column

The output shows that the data frame has 17 columns and all of them have missing values to some extent. The table lists each column along with the number and percentage of missing values in descending order of the percentage of missing values. The missing values are quite high in some columns such as ‘Months since last delinquent’, ‘Credit Score’, and ‘Annual Income’.

Since there are 514 missing values for most of the columns, it's possible that some rows in the CSV file were accidentally read as data rows. This could lead to inconsistencies in the data, such as having a row with all missing values. Therefore, it's worth checking the null rows in the data frame.

```
# Drop the columns with > 50% missing
loan_data.drop(columns = 'Months since last delinquent', axis=1, inplace=True)
```

Figure 10 Code Snippets for Dropping Columns with More Than 50 % Missing

According to the output in Figure 9, it shows that the ‘Months since last delinquent’ column has more than 50% of total missing values, so and if these missing values are imputed or filled in with some other value, it may bias the model results. Therefore, it's better to drop this column from the dataset rather than imputing the missing values as in Figure 10.

```
# Check null rows
loan_data[loan_data['Years of Credit History'].isnull() == True]
# It shows the last 514 observations are NaN values.
```

Figure 11 Code Snippets for Checking Empty Rows

	Loan Status	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
100000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100001	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100003	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100004	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
100509	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100510	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100511	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100512	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
100513	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

514 rows × 16 columns

Figure 12 Output for Empty Rows

According to the interpretation of Figure 9, the code in Figure 11 is checking for null rows in the 'Years of Credit History' column of the data frame. It returns the last 514 observations, indicating that these rows are entirely composed of null values in Figure 12.

```
# Drop the last 514 empty rows
loan_data.drop(loan_data.tail(514).index, inplace=True) # drop last 514 rows
missing_values_table(loan_data)
```

```
Your selected dataframe has 16 columns.
There are 6 columns that have missing values.
```

	Missing Values	% of Total Values
Credit Score	19154	19.2
Annual Income	19154	19.2
Years in current job	4222	4.2
Bankruptcies	204	0.2
Tax Liens	10	0.0
Maximum Open Credit	2	0.0

Figure 13 Code Snippets and Output for Drop the Empty Rows and Check Total Missing Value

According to the interpretation of Figure 12, it indicates that the last 514 rows of the data frame entirely composed of null values, so these rows will be removed and the output table shows there are no missing values for most of the columns as shown in Figure 13.

```
# Drop the 'Credit Score', 'Annual Income' id there is a missing data
columns_to_drop = [ 'Credit Score', 'Annual Income']

for col in columns_to_drop: loan_data = loan_data.dropna(subset=[col])

missing_values_table(loan_data)
```

```
Your selected dataframe has 16 columns.
There are 4 columns that have missing values.
```

	Missing Values	% of Total Values
Years in current job	3412	4.2
Bankruptcies	162	0.2
Tax Liens	6	0.0
Maximum Open Credit	1	0.0

Figure 14 Code Snippets and Output for Drop the Missing Data Rows and Check Total Missing Value

Based on the Figure 13, it shows that the ‘Credit Score’ and ‘Annual Income’ columns have relatively high missing values (19.6%), so removing the missing data rows will help to improve the quality of the dataset which can affect the accuracy of the model. The code in Figure 14 drops the rows with missing values in the 'Credit Score' and 'Annual Income' columns and then

prints out a table showing the number and percentage of missing values for the remaining columns in the data frame.

```
loan_data.describe()
```

	Current Loan Amount	Credit Score	Annual Income	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
count	8.084600e+04	80846.000000	8.084600e+04	80846.000000	80846.000000	80846.000000	80846.000000	8.084600e+04	8.084500e+04	80684.000000	80840.000000
mean	1.447443e+07	1076.456089	1.378277e+06	18510.579017	18.183799	11.143334	0.170064	2.950600e+05	7.351900e+05	0.118822	0.030059
std	3.480085e+07	1475.403791	1.081360e+06	12233.909830	7.006238	5.029085	0.487823	3.816747e+05	6.889569e+06	0.353492	0.264721
min	1.124200e+04	585.000000	7.662700e+04	0.000000	3.700000	1.000000	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000
25%	1.864555e+05	705.000000	8.488440e+05	10244.040000	13.500000	8.000000	0.000000	1.125750e+05	2.719200e+05	0.000000	0.000000
50%	3.246980e+05	724.000000	1.174162e+06	16256.115000	16.900000	10.000000	0.000000	2.104060e+05	4.664000e+05	0.000000	0.000000
75%	5.496150e+05	741.000000	1.650663e+06	24050.865000	21.700000	14.000000	0.000000	3.681630e+05	7.799660e+05	0.000000	0.000000
max	1.000000e+08	7510.000000	1.655574e+08	435843.280000	70.500000	76.000000	15.000000	3.287897e+07	1.539738e+09	7.000000	15.000000

Figure 15 Code Snippets and Outputs for Generating Descriptive Summary of the Dataset

The describe() function used in Figure 15 provides a summary of the statistical measures of the ‘loan_data’ data frame. It indicates most of the columns have a mean greater than the median, indicating a right-skewed distribution for those columns. Additionally, there seems to be some outliers in the 'Credit Score', 'Annual Income', 'Current Loan Amount', 'Maximum Open Credit', and 'Monthly Debt' columns since their maximum values are significantly higher than their 75th percentile values.

Moreover, the 'Current Loan Amount' column has a very large range between the minimum and maximum values, indicating that the loan amounts vary widely. The 'Number of Credit Problems' column has a low mean and a high maximum value, indicating that most borrowers have a low number of credit problems, but some borrowers have a very high number of credit problems. The 'Bankruptcies' and 'Tax Liens' columns also have relatively low means and maximum values, indicating that most borrowers have not filed for bankruptcy or had a tax lien.

```
import matplotlib.pyplot as plt

loan_data.hist(bins = 10 , figsize= (30,30))
plt.show()
```

Figure 16 Code Snippets of Histogram Plotting

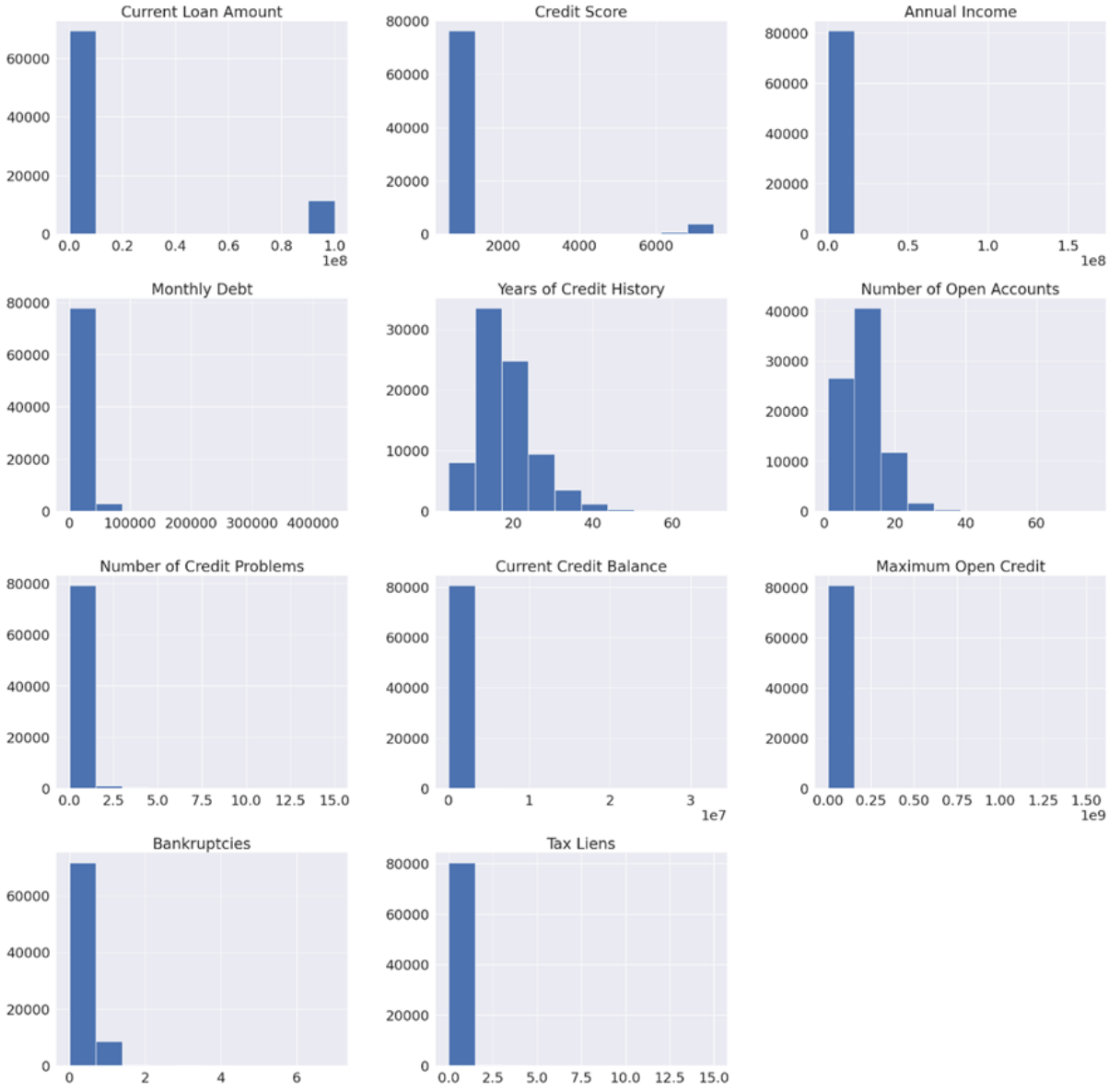

Figure 17 Histogram of All Numerical Variables

Figure 16 illustrates the code specification for generating histograms of the numerical values using the pyplot module from the matplotlib library. The resulting histograms, as depicted in Figure 17, aim to identify any potential skewness in the datasets for each feature. Based on the observations, the data appears to be relatively symmetric with minimal skewness, except for a

few outlier instances. Considering this characteristic, the chosen approach to handle missing values is mean imputation, which is deemed suitable for non-skewed data.

The following steps include imputing missing values for numeric attribute being Bankruptcies, Tax Liens, and Maximum Open Credit in which the missing values will be mean imputed.

```
# Impute missing value to mean numeric attributes such as Bankruptcies, Tax Liens, Maximum Open Credit
loan_data.fillna(loan_data.mean(), inplace=True)
missing_values_table(loan_data)
```

Figure 18 Code Snippets of Mean Imputation for Numerical Values

The figure above shows the code snippets to achieve the imputation task.

In order to address the missing values in the "Years in current job" variable, it is essential to first visualize and explore the data. This variable is not of numerical type, which necessitates the application of alternative data handling methods. To achieve this, code snippets were utilized to plot the "Years in current job" variable and assess the presence of missing values in the column. This exploration helps in gaining a deeper understanding of the variable's distribution and the extent of missing data. By visually examining the plot, appropriate strategies can be determined for handling the missing values effectively, ensuring the integrity and reliability of the data. The following figure shows code snippets to plot the “Years in current job” variable and identify the number of missing values in the column:

```
plt.figure(figsize=(20,8))
sns.countplot(x= loan_data['Years in current job'])

missing_values_table(loan_data)
```

Figure 19 Code Snippets of Variable “Years in current job” Plotting

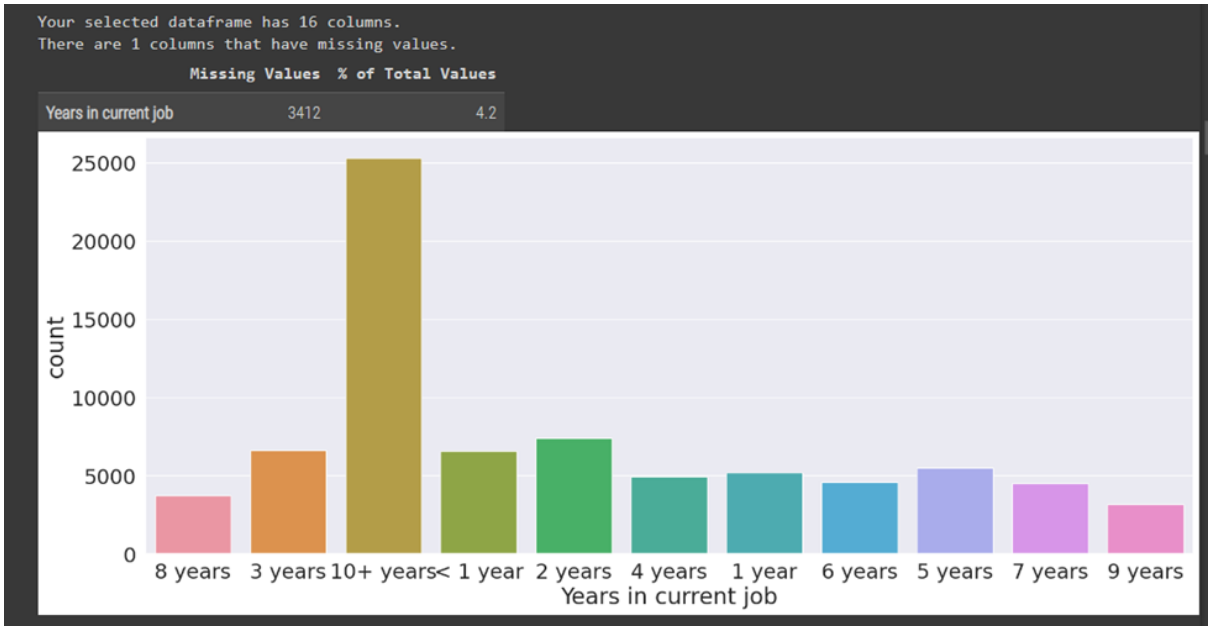

Figure 20 Count Plot Output for Variable "Years in Current Job"

The results of the plot depicted in Figure 20 provide insights into the "Years in current job" variable, including the identification of missing values. It is observed that the variable exhibits a relatively significant number of missing values, accounting for approximately 4% of the overall dataset. In order to handle this issue without drastically reducing the dataset size, an appropriate imputation method is deemed necessary. Based on the bar chart displayed in Figure 20, the mode imputation approach emerges as a suitable choice for filling in the missing values. By utilizing the mode value derived from the bar chart, the missing data in the "Years in current job" column can be imputed accordingly using the provided code snippet:

```
# Since'10+ years' is mod, fill missng value with '10+ years'.
loan_data.fillna('10+ years', inplace=True)
missing_values_table(loan_data)
```

Figure 21 Code Snippets for Mode Imputation for Categorical Variables

To ensure compatibility with the machine learning models, which exclusively accept numerical inputs, it becomes necessary to perform encoding on the categorical data. The initial phase involves encoding the "Years in current job" variable. By applying encoding techniques, the categorical values are transformed into numerical representations, allowing for seamless integration into the models. This encoding step facilitates the utilization of the "Years in current

job" variable as a valuable feature for training and predicting outcomes in the machine learning models.

```
# Change the 'Years in current job' into numeric value
loan_data['Years in current job']=loan_data['Years in current job'].replace(['< 1 year','1 year','2 years','3 years','4 years','5 years','6 years','7 years','8 years','9 years','10+ years'],[0.5,1,2,3,4,5,6,7,8,9,10])
```

Figure 22 Code Snippet of the Encoding of "Years in current job"

The approach employed to handle the "Years in current job" variable is straightforward, as depicted in the aforementioned figure. The method involves excluding all characters except for the numerical ones. However, other categorical values necessitate alternative handling techniques.

```
from sklearn.preprocessing import LabelEncoder
label=LabelEncoder()
loan_data['Loan Status']=label.fit_transform(loan_data['Loan Status'])
loan_data['Term']=label.fit_transform(loan_data['Term'])
loan_data['Home Ownership']=label.fit_transform(loan_data['Home Ownership'])
dummy=pd.get_dummies(loan_data['Purpose'],prefix='Purpose')
loan_data=pd.concat([loan_data,dummy],axis=1)
loan_data.drop(['Purpose'],axis=1,inplace=True)
```

Figure 23 Code Snippets of the Encoding of Other Categorical Variables

The remaining categorical values are encoded using the LabelEncoder function from the preprocessing module of the scikit-learn library, as illustrated in Figure 23. This function employs dummy values for each category within each variable, transforming the categorical data into Boolean values.

The sample output of the whole encoding process are as follows:

	Loan Status	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Monthly Debt	Years of Credit History	Number of Open Accounts	...	Purpose_Medical Bills	Purpose_Other	Purpose_Take a Trip	Purpose_major_purchase	Purpose_moving	Purpose_other	Purpose_renewable_energy	Purpose_small_business	Purpose_vacation	Purpose_wedding
0	1	445412.0	1	709.0	1167493.0	8.0	1	5214.74	17.2	6.0	...	0	0	0	0	0	0	0	0	0	0
2	1	99999999.0	1	741.0	2231892.0	8.0	2	29200.53	14.9	18.0	...	0	0	0	0	0	0	0	0	0	0
3	1	347666.0	0	721.0	806949.0	3.0	2	8741.90	12.0	9.0	...	0	0	0	0	0	0	0	0	0	0
5	0	206602.0	1	7290.0	896857.0	10.0	1	16367.74	17.3	6.0	...	0	0	0	0	0	0	0	0	0	0
6	1	217646.0	1	730.0	1184194.0	0.5	1	10855.08	19.6	13.0	...	0	0	0	0	0	0	0	0	0	0
8	1	548746.0	1	678.0	2559110.0	2.0	3	18660.28	22.6	4.0	...	0	0	0	0	0	0	0	0	0	0
9	1	215952.0	1	739.0	1454735.0	0.5	3	39077.75	13.9	20.0	...	0	0	0	0	0	0	0	0	0	0
10	1	99999999.0	1	728.0	714628.0	3.0	3	11931.04	16.0	16.0	...	0	0	0	0	0	0	0	0	0	0
12	1	99999999.0	1	740.0	778198.0	0.5	2	11978.22	8.5	6.0	...	0	0	0	0	0	0	0	0	0	0
13	1	99999999.0	1	743.0	1560907.0	4.0	3	17560.37	13.3	10.0	...	0	0	0	0	0	0	0	0	0	0

10 rows × 31 columns

Figure 24 Output of Encoding Process

Next is to proceed with a common data exploration task which is to identify correlations between variables, where a heatmap can be used.

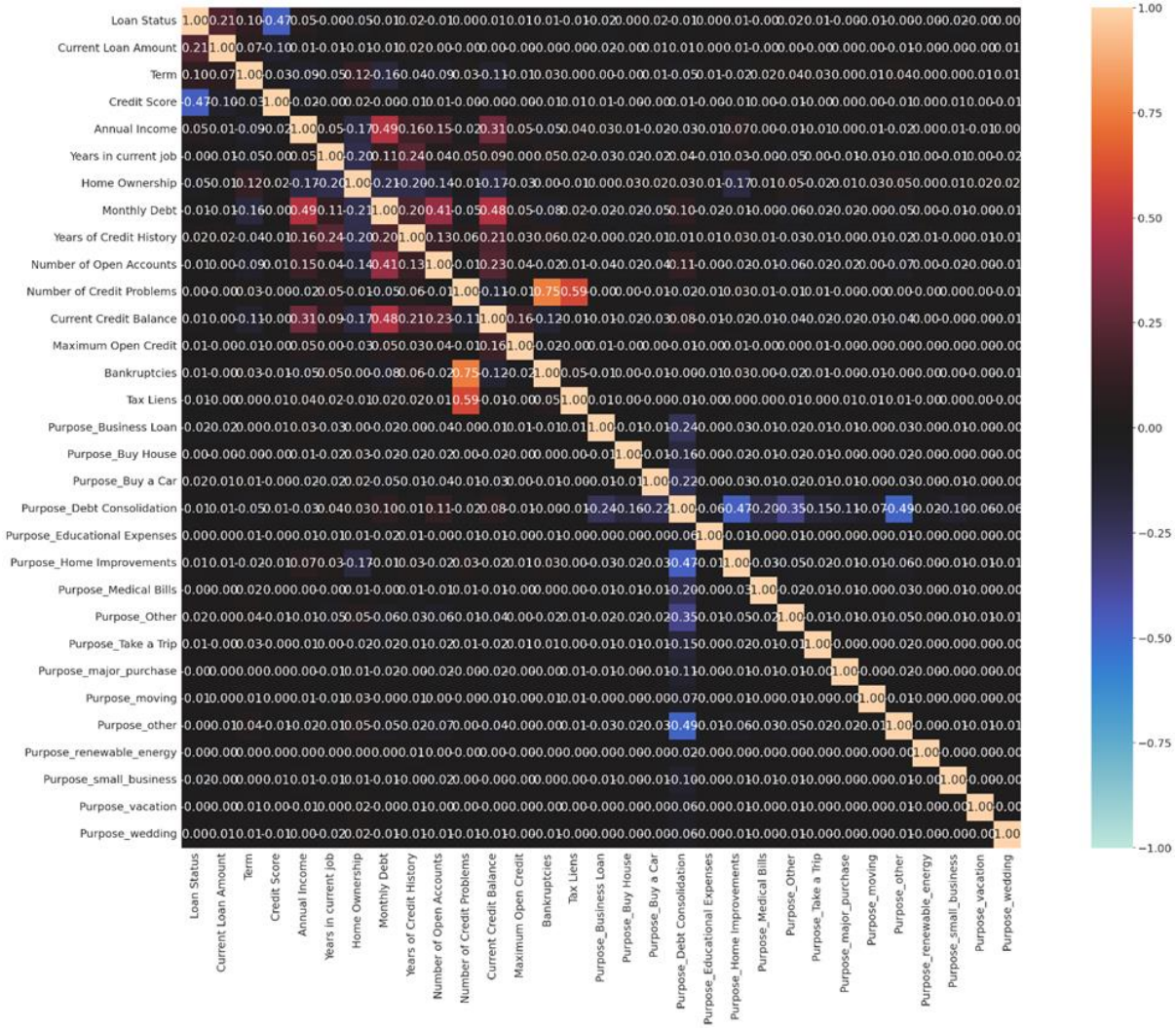

Figure 25 Heatmap of Dataset Variables

Figure 25 presents the correlation among the variables, with particular emphasis on the relationship between each input variable and the target variable. Notably, the variable "Credit Score" exhibits the highest correlation with the target variable, indicating its significance in influencing the target variable's values.

```
!pip install -U numpy
!pip install dython
import numpy as np
from dython.nominal import associations
associations(encoded_loan_data, figsize=[50,30])
```

Figure 26 Code Snippet of Heatmap Plotting

The library dython is utilized to plot the heatmap as it allows to include correlations for categorical values as well. With all numerical values in hand, the seaborn library can alternatively be used to plot the heatmap. Lastly is to explore the target variable through visualization. The target variable will be visualized as a count plot.

```
#Checking the balance of the target variable

plt.figure(figsize=(20,8))
sns.countplot(x= encoded_loan_data['Loan Status'])

missing_values_table(encoded_loan_data)

print(encoded_loan_data['Loan Status'].value_counts())
```

Figure 27 Code Snippet of the Plotting of "Loan Status" Bar Chart

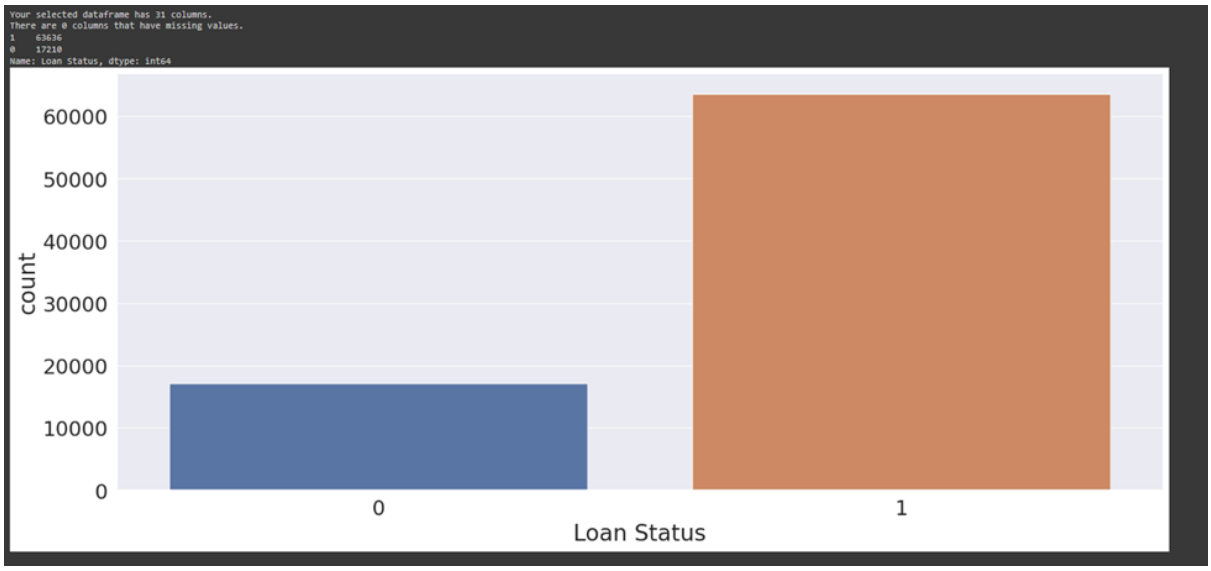

Figure 28 Bar Chart of the Distribution of Target Variable

Figure 27 showcases the code snippet employing seaborn and matplotlib libraries to plot the graph, while Figure 28 portrays the resulting output. The plot reveals an imbalanced distribution of the target variable, indicating the potential for bias towards the majority class if the imbalance is left unaddressed. To mitigate this issue, two sampling methods can be employed: under-sampling and oversampling. In this scenario, over-sampling is chosen as the approach to handle the class imbalance. Specifically, the Synthetic Minority Over-sampling Technique (SMOTE) is employed as the oversampling technique. SMOTE generates synthetic

samples to augment the minority class, effectively balancing the class distribution and alleviating the impact of class imbalance.

```
from collections import Counter
from imblearn.over_sampling import SMOTE

#Separating features and labels
x = encoded_loan_data.drop(['Loan Status'], axis=1)
y = encoded_loan_data['Loan Status']

#split train-test data
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.2,random_state=0,stratify=y)
print("Before SMOTE sampling: ", Counter(y_train))
# Apply SMOTE only to the training set
sm = SMOTE()
x_smote_train, y_smote_train = sm.fit_resample(x_train, y_train)

print("After SMOTE sampling: ", Counter(y_smote_train))
```

Figure 29 Code Snippet of Dataset Splitting and SMOTE Sampling

To ensure the integrity of the oversampling process, it is crucial to split the data into training and testing datasets prior to performing oversampling. The recommended ratio for this split is 80:20, allocating 80% of the data for traininfeg and 20% for testing. Following the data split, the oversampling technique can be applied to the training dataset using the SMOTE() function from the imblearn.over_sampling library. This function effectively generates synthetic samples to augment the minority class, addressing the class imbalance issue and preparing the data for subsequent modeling and evaluation.

```
Before SMOTE sampling:  Counter({1: 50908, 0: 13768})
After SMOTE sampling:  Counter({0: 50908, 1: 50908})
```

Figure 30 Output of SMOTE Sampling

Both categories have the same number of values after oversampling.

5.3 Model Building and Evaluation of Results

Following the pre-processing steps, multiple models are built and tested to compare and find out which one has the best result for this project.

```
def cross_val(model):
    accuracies=cross_val_score(estimator=model,X=x_smote_train,y=y_smote_train,cv=10)
    return accuracies.mean()*100

def fit_evaluate(model):
    name=model.__class__.__name__
    model.fit(x_smote_train,y_smote_train)
    y_pred=model.predict(x_test)
    cross=cross_val(model)
    a_s=accuracy_score(y_test,y_pred)*100
    pre_sc=precision_score(y_test,y_pred)*100
    rec_sc=recall_score(y_test,y_pred)*100
    f1_sc=f1_score(y_test,y_pred)*100
    roc_sc=roc_auc_score(y_test,y_pred)*100
    result=pd.DataFrame([[name,cross,a_s,pre_sc,rec_sc,f1_sc,roc_sc]],columns=['model','accuracy_train_cv','accuracy_test','precision_score','recall_score','f1_score','roc_auc_score'])
    return result
```

Figure 31 Code Snippet of Two Functions for Model Evaluation

The two functions, named cross_val and fit_evaluate are used to automatically train, test, and evaluate the accuracy results of all the models used in the function. The next ML models to be tested are Logistic Regression, XGBoost, Decision Tree, K-Nearest Neighbour, Random Forest, Ada boost, and Gradient Boosting.

```
models=[LogisticRegression(),XGBClassifier(),DecisionTreeClassifier(),KNeighborsClassifier(),RandomForestClassifier(),AdaBoostClassifier(),GradientBoostingClassifier()]
result_models=pd.DataFrame(columns=['model','accuracy_train_cv','accuracy_test','precision_score','recall_score','f1_score','roc_auc_score'])
for model in models:
  print(model.__class__.__name__)
  results = fit_evaluate(model)
  result_models=pd.concat([result_models,results])
result_models.sort_values(by='accuracy_test',ascending=False)
```

Figure 32 Code Snippet of Model Training and Testing Functions

```
LogisticRegression
XGBClassifier
DecisionTreeClassifier
KNeighborsClassifier
RandomForestClassifier
AdaBoostClassifier
GradientBoostingClassifier
```

	model	accuracy_train_cv	accuracy_test	precision_score	recall_score	f1_score	roc_auc_score
0	XGBClassifier	89.459790	84.075448	83.841077	98.813639	90.713693	64.194734
0	RandomForestClassifier	89.173893	83.104515	84.540428	96.110937	89.955144	65.559826
0	GradientBoostingClassifier	85.177498	82.022263	83.933384	95.427404	89.312107	63.939733
0	AdaBoostClassifier	82.599258	79.369202	84.780033	89.935575	87.281738	65.115957
0	DecisionTreeClassifier	84.892591	77.884972	86.105413	85.740101	85.922368	67.288993
0	KNeighborsClassifier	77.906268	61.688312	82.740303	64.857008	72.715261	57.413978
0	LogisticRegression	67.458006	61.069882	87.414214	59.043055	70.480657	63.803921

Figure 33 Output of Model Training and Testing Functions

After the function is finished running, the results are printed in a table, sorted according to the accuracy_test variable from highest to lowest. This means that the table is sorted with the

accuracy result from the test dataset. As the objective of the predictive model is to accurately predict if a bank loan can be approved or not, thus accuracy score is the most crucial metrics to be considered during model building. From the results above, it shows that XGBoost, Random Forest, Gradient Boosting, Ada Boost, and Decision Tree has a very good accuracy results for the training accuracy. For the testing accuracy, XGBoost boast the highest number with 84.07% accuracy. This high accuracy, combined with respectable scores for other metrics, shows that XGBoost is the definitive Machine Learning model to be used for the deployment of this project.

```
# Performance metrics for Best Model
# Best model with highest accuaracy : XGBoost
best_model=XGBClassifier()
best_model.fit(x_smote_train,y_smote_train)
best_model_prediction=best_model.predict(x_test)
print("Performance Metrics before Tuning for XGBClassifier")
print(classification_report(y_test,best_model_prediction))
print (confusion_matrix(y_test,best_model_prediction))
```

Figure 34 Snippet Code of XGBoost Classification Report

```
Performance Metrics before Tuning for XGBClassifier
              precision    recall  f1-score   support

           0       0.85      0.30      0.44      3426
           1       0.84      0.99      0.91     12660

    accuracy                           0.84     16086
   macro avg       0.84      0.64      0.68     16086
weighted avg       0.84      0.84      0.81     16086

[[ 1031  2395]
 [  185 12475]]
```

Figure 35 XGBoost Classification Report Before Tuning

After XGBoost Classification has been chosen to be the best model for the project, model tuning is done to make sure that the model has the highest score numbers. The tuning is done using RandomizeSearchCV from Scikitlearn library.

```
from sklearn.model_selection import RandomizedSearchCV
# Random hyperparameter tuning for best model
params={'n_estimators':[100,200,300,400,500],'max_depth':[1,2,3,4,5],
        "learning_rate":[0.01,0.05,0.1,0.2,0.3]}
randSearchCV=RandomizedSearchCV(best_model,params,scoring='accuracy',cv=10)
randSearchCV.fit(x_smote_train,y_smote_train)
prediction=randSearchCV.predict(x_test)
print("Performance Metrics after Tuning for XGBClassifier")
print(classification_report(y_test,prediction))
print (confusion_matrix(y_test,prediction))
```

Figure 36 Snippet Code of XGBoost Classification Report With Tuning

```
Performance Metrics after Tuning for XGBClassifier
              precision    recall  f1-score   support

           0       0.88      0.29      0.43      3426
           1       0.84      0.99      0.91     12660

    accuracy                           0.84     16086
   macro avg       0.86      0.64      0.67     16086
weighted avg       0.84      0.84      0.81     16086

[[  982  2444]
 [  140 12520]]
```

Figure 37 XGBoost Classification Report After Tuning

After tuning, a noticeable increase in class 0 precision can be seen, from 0.85 into 0.88. This means that the deployed model is better at correctly identifying the bank loan that might be charged off.

```
# Feature importance
feature_df=pd.DataFrame({'feature':x.columns,'importance':best_model.feature_importances_})
feature_df.sort_values(by='importance',ascending=False).head(10)
```

Figure 38 : Code Snippet of Feature Importance

	feature	importance
2	Credit Score	0.129643
1	Term	0.110463
19	Purpose_Home Improvements	0.099339
25	Purpose_other	0.093295
4	Years in current job	0.091341
12	Bankruptcies	0.079465
21	Purpose_Other	0.048770
16	Purpose_Buy a Car	0.038046
14	Purpose_Business Loan	0.037180
0	Current Loan Amount	0.036992

Figure 39 Results: Output of Feature Importance

Finally, feature importance can be used as an evaluation of which attributes are the most important that can have an impact on the results of the prediction. From the results, it shows the top ten variables that has the highest impact towards the prediction.

5.4 Real-world Application of the Model

For the real-world application of the project's result, a user input prompt can be run from the python code to get the data from the user. After the data has been inputted, the data will be fitted into the best model, which in this case XGBoost, and the result of the prediction will be shown, either the loan is likely to be approved or unlikely to be approved.

```
# Prompt the user to enter input values
print("Please enter the following information:")
loan_amount = float(input("Current Loan Amount: "))
term = int(input("Term: (0= Short Term, 1= Long Term) "))
credit_score = float(input("Credit Score: "))
annual_income = float(input("Annual Income: "))
years_in_current_job = float(input("Years in current job: "))
home_ownership = int(input("Home Ownership ( 1= Home Mortgage, 2= Own Home, 3= Rent): "))
monthly_debt = float(input("Monthly Debt: "))
years_of_credit_history = float(input("Years of Credit History: "))
num_open_accounts = float(input("Number of Open Accounts: "))
num_credit_problems = float(input("Number of Credit Problems: "))
current_credit_balance = float(input("Current Credit Balance: "))
max_open_credit = float(input("Maximum Open Credit: "))
bankruptcies = float(input("Bankruptcies: (0= No, 1= Yes)"))
tax_liens = float(input("Tax Liens: "))

purpose = input("Purpose (Business Loan, Buy House, Buy Car, Medical, Other): ")

# Set purpose variables based on user input
purpose_business_loan = 1 if purpose == 'Business Loan' else 0
purpose_buy_house = 1 if purpose == 'Buy House' else 0
purpose_buy_a_car = 1 if purpose == 'Buy a Car' else 0
purpose_debt_consolidation  = 1 if purpose == 'Debt Consolidation' else 0
purpose_educational_expenses = 1 if purpose == 'Educational Expenses' else 0
purpose_home_improvements = 1 if purpose == 'Home Improvements' else 0
purpose_medical_bills = 1 if purpose == 'Medical Bills' else 0
purpose_Other = 1 if purpose == 'Other' else 0
purpose_take_a_trip = 1 if purpose == 'Take a Trip' else 0
purpose_major_purchase = 1 if purpose == 'Major Purchase' else 0
purpose_moving = 1 if purpose == 'Moving' else 0
purpose_other = 1 if purpose == 'other' else 0
purpose_renewable_energy = 1 if purpose == 'Renewable Energy' else 0
purpose_small_business  = 1 if purpose == 'Small Business' else 0
purpose_vacation = 1 if purpose == 'Vacation' else 0
purpose_wedding = 1 if purpose == 'Wedding' else 0
```

Figure 40 Code Snippet of Deployment Input

```
# Create input array for the model
input_data = np.array([
    loan_amount, term, credit_score, annual_income,years_in_current_job, home_ownership,
    monthly_debt, years_of_credit_history, num_open_accounts, num_credit_problems, current_credit_balance,
    max_open_credit, bankruptcies, tax_liens, purpose_business_loan, purpose_buy_house, purpose_buy_a_car, purpose_debt_consolidation,
    purpose_educational_expenses, purpose_home_improvements, purpose_medical_bills, purpose_Other, purpose_take_a_trip,
    purpose_major_purchase, purpose_moving, purpose_other, purpose_renewable_energy, purpose_small_business, purpose_vacation, purpose_wedding
]).reshape(1, -1)

# Make prediction with the model
prediction = best_model.predict(input_data)

# Print the prediction result
if prediction == 1:
  print("The loan is likely to be approved.")
  print("prediction: ", prediction)
else:
  print("The loan is unlikely to be approved.")
  print("prediction: ", prediction)
```

Figure 41 Code Snippet of Deployment Processing and Output

```
Please enter the following information:
Current Loan Amount: 445412
Term: (0= Short Term, 1= Long Term) 0
Credit Score: 709
Annual Income: 1167493
Years in current job: 8
Home Ownership ( 1= Home Mortgage, 2= Own Home, 3= Rent): 1
Monthly Debt: 5214.74
Years of Credit History: 17.2
Number of Open Accounts: 6
Number of Credit Problems: 1
Current Credit Balance: 228190
Maximum Open Credit: 416746
Bankruptcies: (0= No, 1= Yes)1
Tax Liens: 0
Purpose (Business Loan, Buy House, Buy Car, Medical, Other): Home Improvements
The loan is likely to be approved.
prediction:  [1]
```

Figure 42 Code Snippet of Deployment Output

An example of the user input prompt deployment can be seen from the image above, where based on the data fitted into the XGBoost model, the loan is likely to be approved.

6.0 Conclusion

In conclusion, this project aimed to develop a high accuracy machine learning model to predict the likelihood of loans being charged off by creditors. To achieve this, the project involved the exploration of a comprehensive dataset related to bank loans, which was pre-processed, cleaned, and transformed. The data was then subjected to exploratory data analysis, employing statistical analysis and data visualization techniques, to identify patterns and relationships between variables.

Afterward, seven machine learning models, namely Logistic Regression, XGBoost, Decision Tree, K-Nearest Neighbor, Random Forest, Ada boost, and Gradient Boosting were developed, evaluated, compared, and documented to determine the model that exhibits the best performance. The results showed that the XGBoost model outperformed the other models, with an accuracy for testing set of 84.08%, precision of 83.84%, recall of 98.81%, and F1 score of 90.71%.

The developed model is intended to assist banks and financial institutions in evaluating the creditworthiness of loan applicants and making informed lending decisions. The project's success suggests that machine learning can be leveraged to automate the loan approval process and mitigate risks associated with lending. However, it is essential to note that the model's performance is subject to the quality and integrity of the data, and further testing and validation are necessary before deployment.

Furthermore, the project highlighted the issue of discrimination in the loan approval process, where borrowers from minority groups are less likely to be approved for loans than those from the majority. This underscores the need for banks and financial institutions to adopt fair and unbiased lending practices, leveraging machine learning to automate loan processing and minimize human biases.

In short, this project demonstrates the potential of machine learning to enhance the loan approval process's efficiency, accuracy, and fairness, benefiting both banks and customers alike.

7.0 References

Alsaleem, M. Y., & Hasoon, S. O. (2020). Predicting Bank Loan Risks Using Machine Learning Algorithms. *AL-Rafidain Journal of Computer Sciences and Mathematics, 14*(1). https://doi.org/10.33899/CSMJ.2020.164686

Gupta, A., Pant, V., Kumar, S., & Bansal, P. K. (2020). Bank Loan Prediction System using Machine Learning. *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*, 423-426. https://doi.org/10.1109/SMART50582.2020.9336801

Mehta, S. (6 May, 2022). Credit Approval. FinancialEdge. Retrieved 14 May, 2023, from FinancialEdge: https://www.fe.training/free-resources/credit/credit-approval/

S. Sreesouthry, A. A. (2021). Loan Prediction Using Logistic Regression in Machine Learning. *Annals of the Romanian Society for Cell Biology, 25*(4), 2790. https://www.annalsofrscb.ro/index.php/journal/article/view/2818