ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Predictive Modeling of Loan Default: A Data Mining Approach

by
João Figueiredo, 1230194
João Araújo 1200584

Course: Masters in Software Engineering
Subject: Mineração de Dados

Supervisors: Maria de Fátima Coutinho Rodrigues, Catarina Figueiredo

Date: Sunday 26th November, 2023

Academic Year: 2023/2024

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the field of data mining, the extraction of valuable insights and patterns from extensive datasets is a key objective. This chapter introduces the context, motivation, and goals of the 1st Practical Work in the 2023/2024 1st semester of the Mineração de Dados course at the Dep. de Eng. Informática.

1.1 Context

The project revolves around the crucial task of predicting whether bank clients will default on their loans. Defaulting on loans can lead to significant financial losses for banks, impacting economic growth. The project focuses on analyzing various factors, such as funded amount, location, loan balance, and more, to develop predictive models.

1.2 Motivation

The motivation behind this project stems from the economic implications of loan defaults. When clients fail to make timely payments, banks face financial losses, and the subsequent debt collection process becomes essential. By leveraging data mining techniques, the project aims to provide insights that can help mitigate the impact of loan defaults.

1.3 Goals

The primary goals of this project include:

Data Exploration and Preparation: Follow the CRISP-DM methodology to explore and prepare the dataset for analysis.

Data Pre-processing: Perform necessary data pre-processing tasks to enhance the quality of the dataset for model training.

Model Development: Develop predictive models capable of forecasting whether a bank client is likely to default on a loan.

Model Evaluation: Evaluate the performance of the developed models using appropriate metrics.

1.4 Document Structure

This section aims to explain the purpose of each subsequent chapter following the introduction of this work.

1.4.1 Chapter 2: Data Exploration

In this chapter, we will delve into the data exploration process. This includes an in-depth analysis of the dataset, examining patterns, trends, and insights obtained through various exploratory data analysis techniques.

1.4.2 Chapter 3: Data Preprocessing

Chapter 3 focuses on the crucial step of data preprocessing. We will discuss the methods employed to clean and prepare the data for modeling. This involves handling missing values, encoding categorical variables, and addressing any other data quality issues.

1.4.3 Chapter 4: Creation of Models using Data Mining Algorithms

The fourth chapter is dedicated to the creation of predictive models using data mining algorithms. We will detail the selection of algorithms, the training process, and the fine-tuning of model parameters to achieve optimal performance.

1.4.4 Chapter 5: Evaluation of the Models Created

In Chapter 5, we will evaluate the models created in the previous chapter. This includes assessing their accuracy, precision, recall, and other relevant metrics. We will also discuss any challenges encountered during the evaluation process.

1.4.5 Chapter 6: Conclusions, Limitations, and Future Work

The final chapter provides a conclusion to the study, summarizing the key findings and insights. We will also discuss the limitations of the study and propose directions for future research or improvements to the methodology.

Chapter 2

Data Exploration

2.1 Dataset Overview

Before delving into the details of our data exploration process, it is essential to provide an overview of the dataset. The dataset used in this study consists of data collected from several bank clients, containing samples of clients who defaulted on a loan and clients who didn't. . Below is an explanation of the key columns in the dataset:

Column	Description
ID	Unique ID of the representative
Loan Amount	Loan amount applied
Funded Amount	Loan amount funded
Funded Amount Investor	Loan amount approved by the investors
Term	Term of the loan (in months)
Batch Enrolled	Batch numbers assigned to representatives
Interest Rate	Interest rate (%) on the loan
Grade	Grade assigned by the bank
Sub Grade	Sub-grade assigned by the bank
Employment Duration	Duration of employment
Home Ownership	Ownership status of the home
Verification Status	Income verification status by the bank
Payment Plan	If any payment plan has started against the loan
Loan Title	Loan title provided
Debit to Income	Ratio of representative's total monthly debt repayment divided by self-reported monthly income excluding mortgage
Delinquency - two years	Number of 30+ days delinquency in the past 2 years
Inquiries - six months	Total number of inquiries in the last 6 months
Open Account	Number of open credit lines in the representative's credit line
Public Record	Number of derogatory public records
Revolving Balance	Total credit revolving balance
Revolving Utilities	Amount of credit a representative is using relative to revolving balance
Total Accounts	Total number of credit lines available in the representative's credit line
Initial List Status	Unique listing status of the loan - W(Waiting), F(Forwarded)
Total Received Interest	Total interest received till date
Total Received Late Fee	Total late fee received till date
Recoveries	Post charge-off gross recovery

Collection Recovery Fee	Post charge-off collection fee
Collection 12 months Medical	Total collections in the last 12 months excluding medical collections
Application Type	Indicates whether the representative is an individual or joint
Last week Pay	Indicates how long (in weeks) a representative has paid EMI after batch enrolled
Accounts Delinquent	Number of accounts on which the representative is delinquent
Total Collection Amount	Total collection amount
Total Current Balance	Total current balance from all accounts
Total Revolving Credit Limit	Total revolving credit limit
Loan Status	1 = Defaulter, 0 = Non-Defaulters

Table 2.1: Description of Dataset Columns

During the initial exploration of the dataset, an inconsistency was identified in the CSV file. Specifically, Column 10 and Column 11 were in each other's places.

Before Correction:

- Column 10 ("Employment Duration"): Expected to contain numerical data but instead contained values such as "Mortgage," "Rent," and "Own," which were atypical for this column.
- Column 11 ("Home Ownership"): Contained numerical data, which was inconsistent with its label.

To rectify this inconsistency, it was determined that the labels on each column were switched directly in the CSV file. The corrected configuration is as follows:

After Correction:

- Column 10 ("Home Ownership"): Now contains data related to home ownership.
- Column 11 ("Employment Duration"): Now contains numerical data representing employment duration.

This correction ensures that the dataset aligns with the expected data types and improves its overall coherence.

2.2 Exploratory Data Analysis and Initial Cleaning

First, we started by loading the dataset and printing out some basic information about it, such as, number of rows, columns and statistics about the columns.

```
df = pd.read_csv('train.csv')
print(df.describe())
print(f"Number of rows: {df.shape[0]}")
print(f"Number of columns: {df.shape[1]}")
```

Figure 2.1: Code for Loading and Initial Exploration

With this, we were able to obtain multiple statistics regarding the numerical columns such as: count, mean, standard deviation, minimum and maximum values and percentage distributions. We also found out that the dataset has 35 columns and 67463 rows.

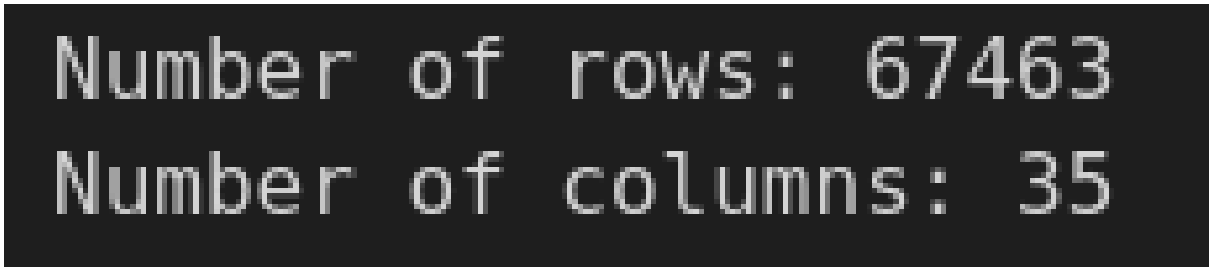


Figure 2.2: Code for displaying statistics regarding the dataset

Prior to delving into data visualization, we identified columns for potential exclusion. This involved targeting columns with only one unique value, indicating a lack of variation, as well as the ID column. These columns were deemed insignificant in relation to the target variable, allowing for their removal without the need for correlation analysis. Here's how we achieved that and the columns that were found to fit the criteria:

```
#Let's check the ammount of unique values a column can have
#if the unque_values == 1, the column is irrevelant ( novariation -> column does not have an impact on the target variable)
irrelevant_columns = []

irrelevant_columns.append('ID') #ID is just a number that identifies a client, completely irrelevant

for column in df.columns:
    unique_values = df[column].nunique() #if the number of unique values in a column is 1, then that column doesn't affect the target varibale
    if unique_values <= 1:
        irrelevant_columns.append(column)

print("Irrelevant Columns:", irrelevant_columns)

#Let's drop these columns as they don't serve any purpose, regarding the goal of this project
df = df.drop(irrelevant_columns, axis=1)
```

Figure 2.3: Code for finding irrelevant columns

```
Irrelevant Columns: ['ID', 'Payment Plan', 'Accounts Delinquent']
```

Figure 2.4: List of insignificant columns

2.3 Data Visualization

Now, that columns deemed insignificant were removed, we can get down to visualize our data and its respective columns. First we gathered into 2 different lists, categorical columns and numerical columns.

```python
#imprimir algumas informações básicas do dataframe
print(df.columns.to_list())
print(f"Number of columns in the dataset: {len(df.columns.to_list())}")

#descobrir quais as colunas que representam dados numéricos e que colunas representam dados categóricos
# Get numerical columns
numerical_cols = df.select_dtypes(include=['int64', 'float64']).columns.to_list()
print("Numerical columns:", numerical_cols)

# Get categorical columns
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.to_list()
print("Categorical columns:", categorical_cols)



#Mortgage e Employment Duration estão trocados
```

Python

```
['Loan Amount', 'Funded Amount', 'Funded Amount Investor', 'Term', 'Batch Enrolled', 'Interest Rate', 'Grade', 'Sub Grade', 'Home Ownership', 'Employment Duration',
Number of columns in the dataset: 32
Numerical columns: ['Loan Amount', 'Funded Amount', 'Funded Amount Investor', 'Term', 'Interest Rate', 'Employment Duration', 'Debit to Income', 'Delinquency - two y
Categorical columns: ['Batch Enrolled', 'Grade', 'Sub Grade', 'Home Ownership', 'Verification Status', 'Loan Title', 'Initial List Status', 'Application Type']
```

Figure 2.5: Finding categorical and numerical columns

2.3.1 Numerical columns' visualization

We automated the generation of three types of plots for our numerical columns: histograms, box plots, and kernel density plots. Here are the results:

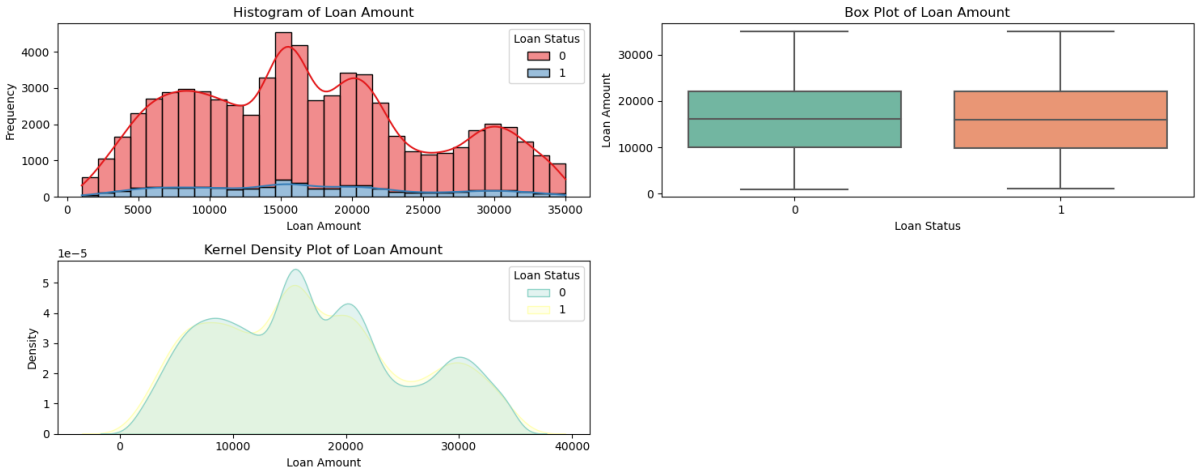


Figure 2.6: Loan Amount

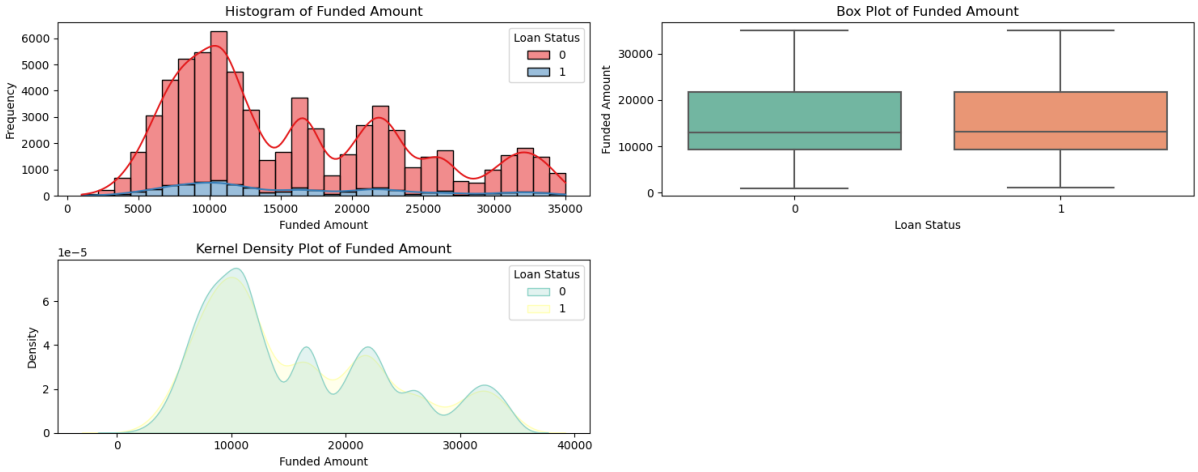


Figure 2.7: Funded Amount

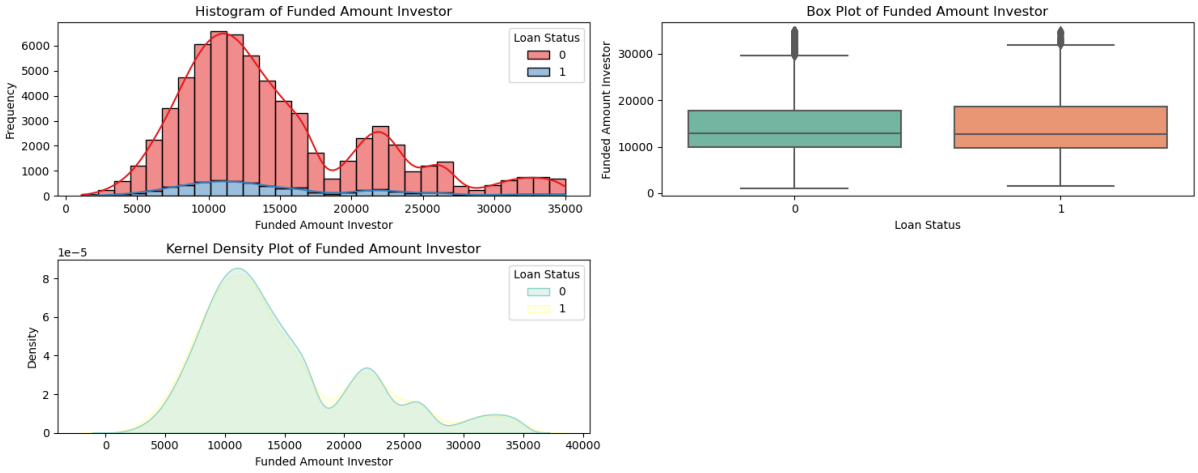


Figure 2.8: Funded Amount Investor

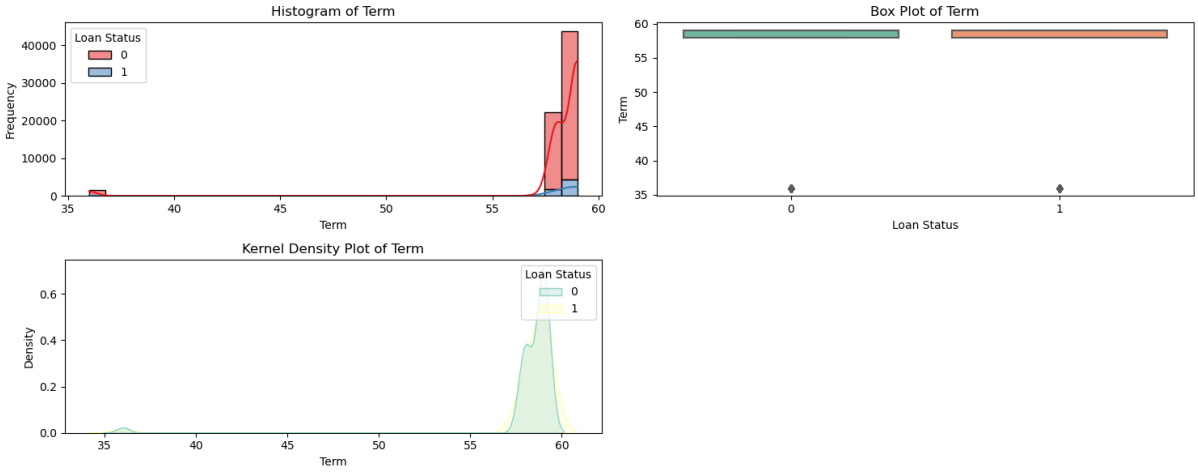


Figure 2.9: Term

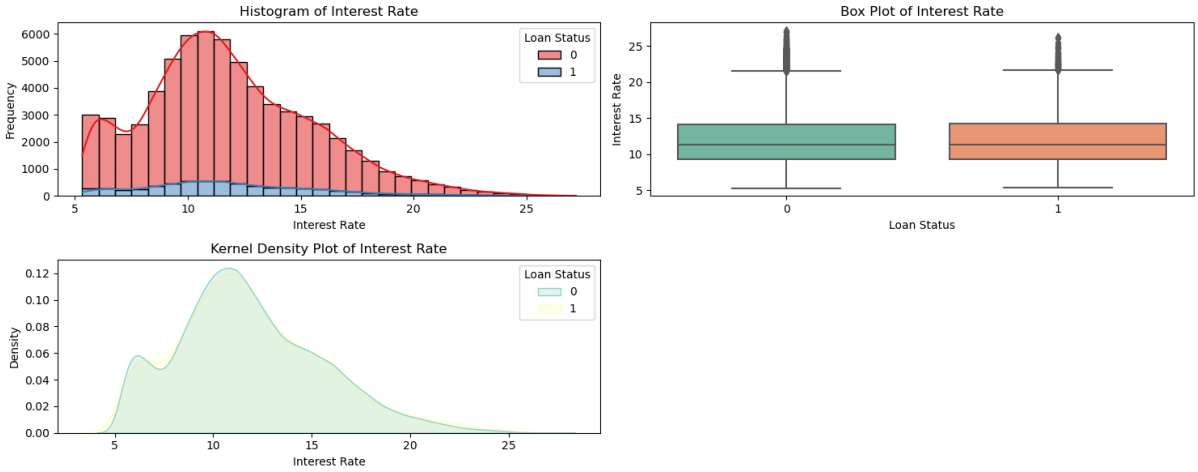


Figure 2.10: Interest Rate

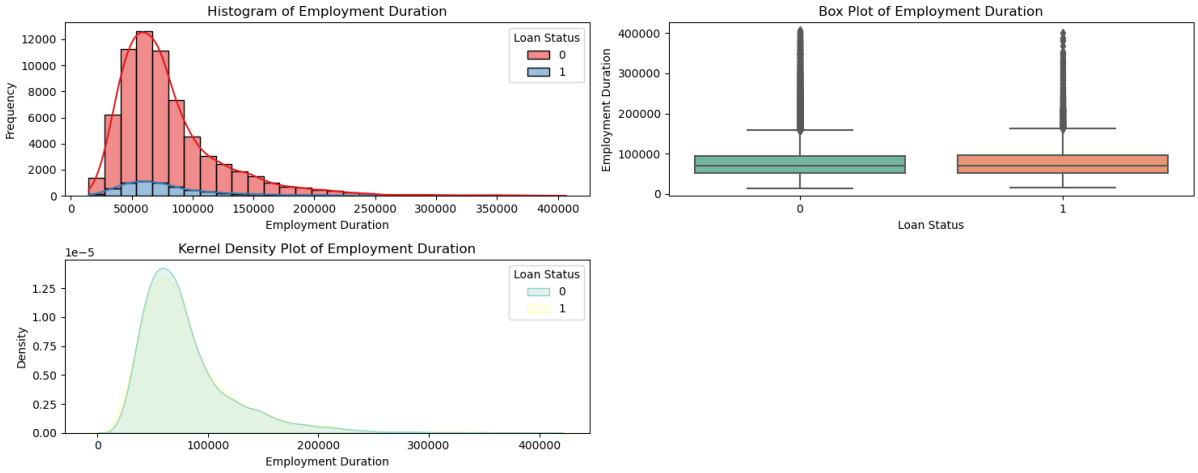


Figure 2.11: Employment Duration

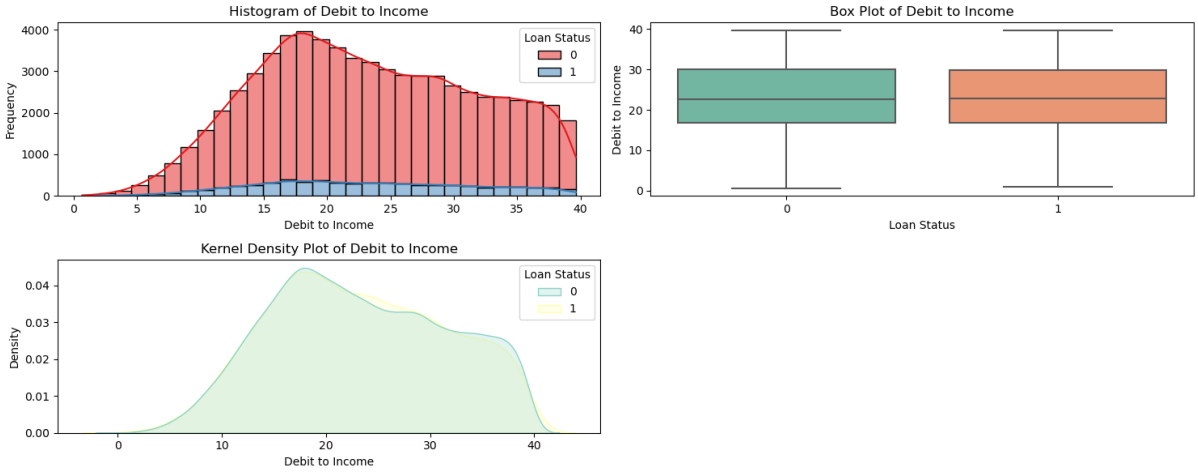


Figure 2.12: Debit To Income

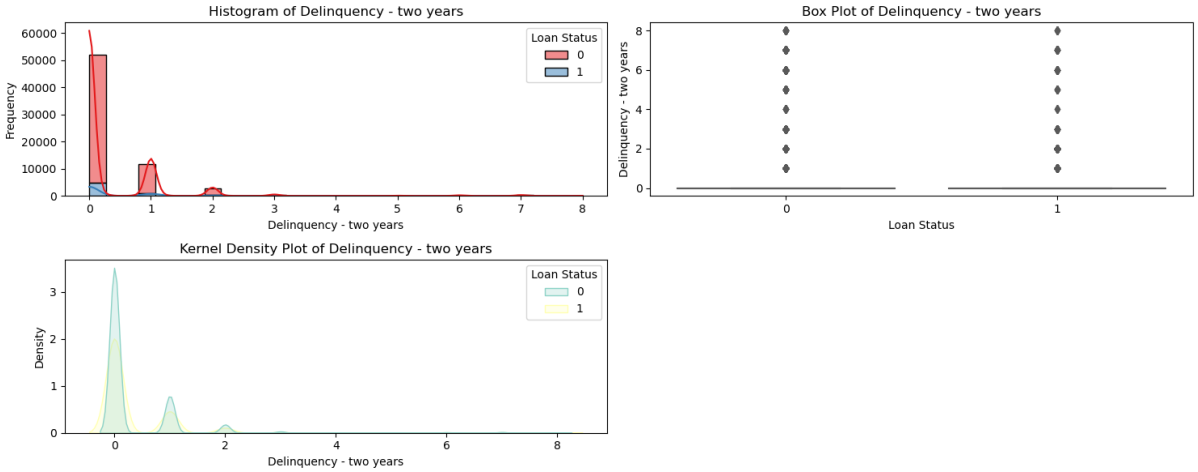


Figure 2.13: Delinquency - 2 years

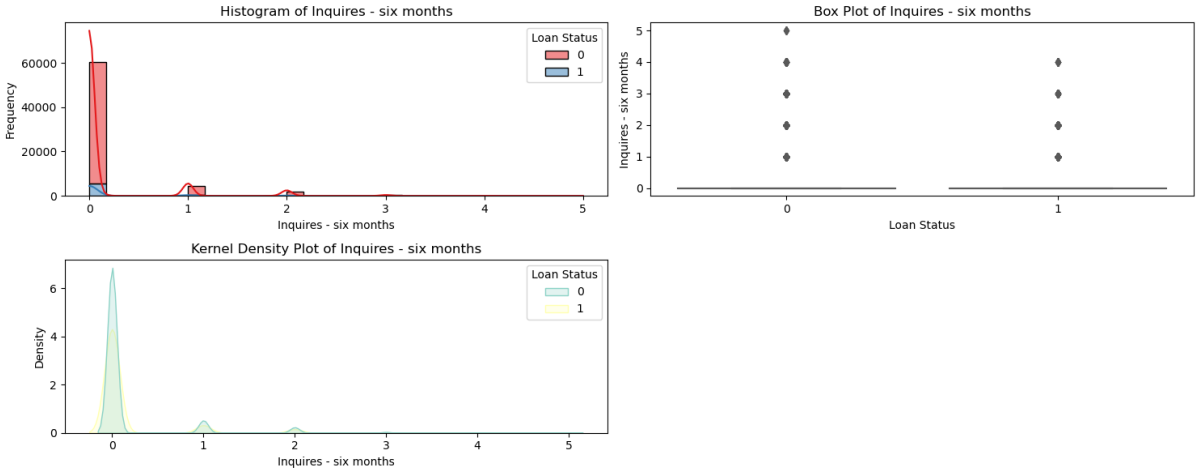


Figure 2.14: Inquires - 6 months

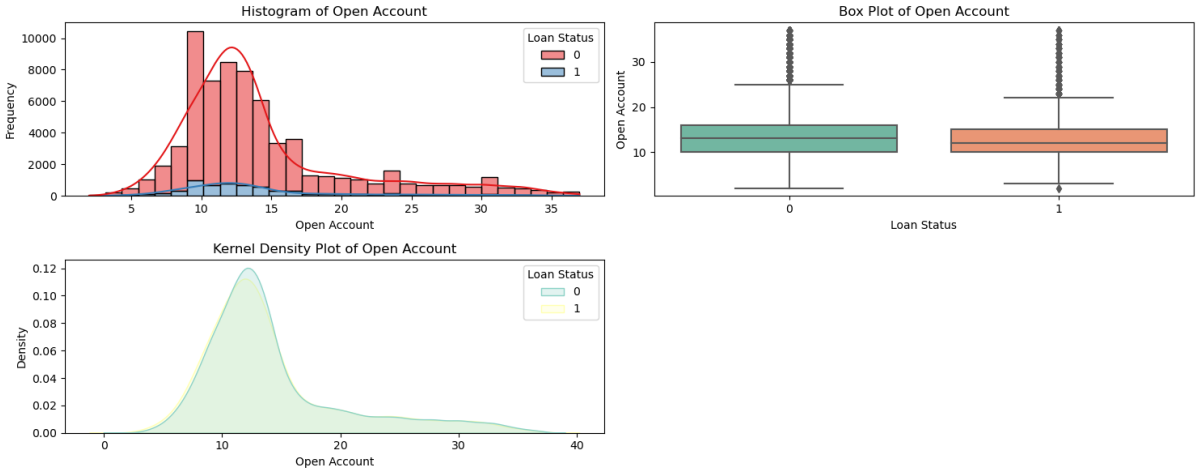


Figure 2.15: Open Account

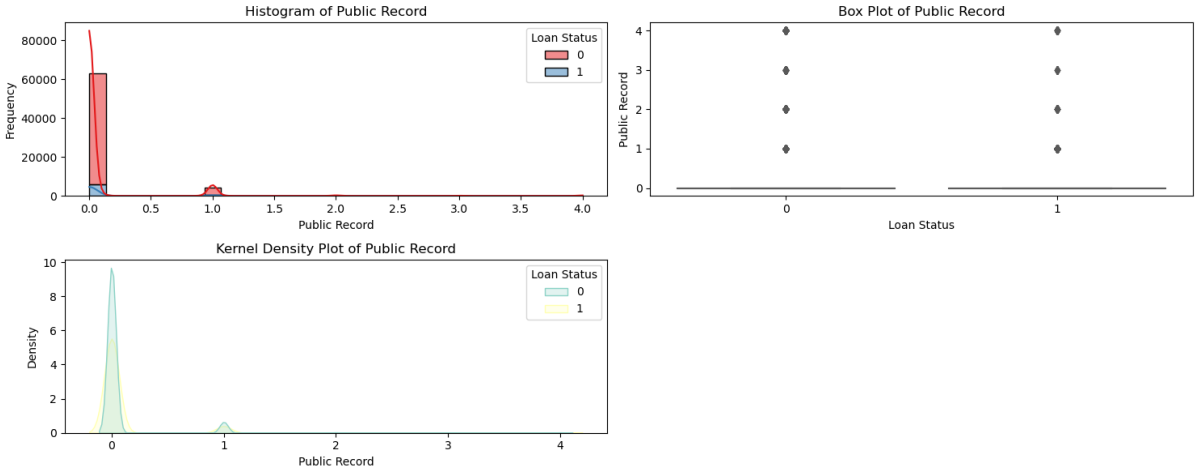


Figure 2.16: Public Record

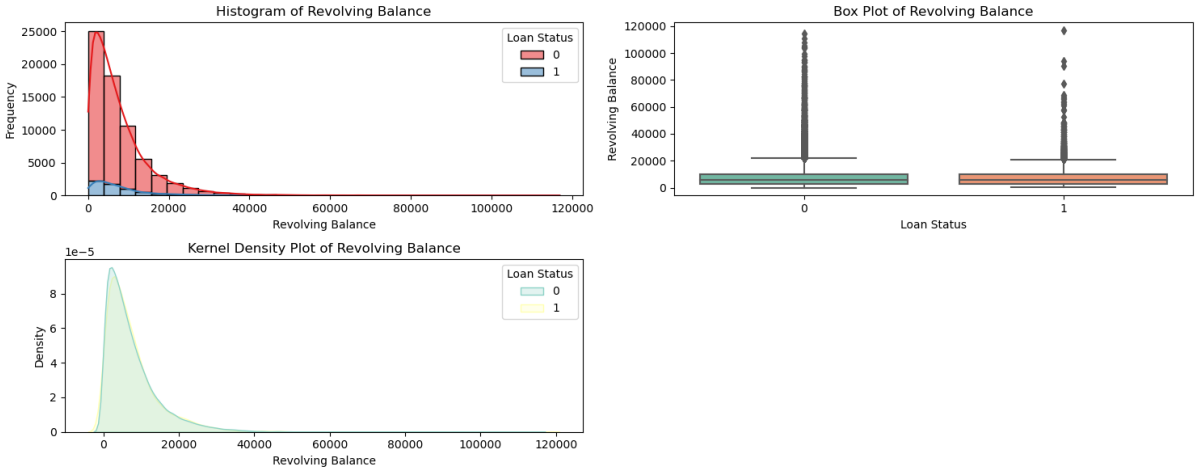


Figure 2.17: Revolving Balance

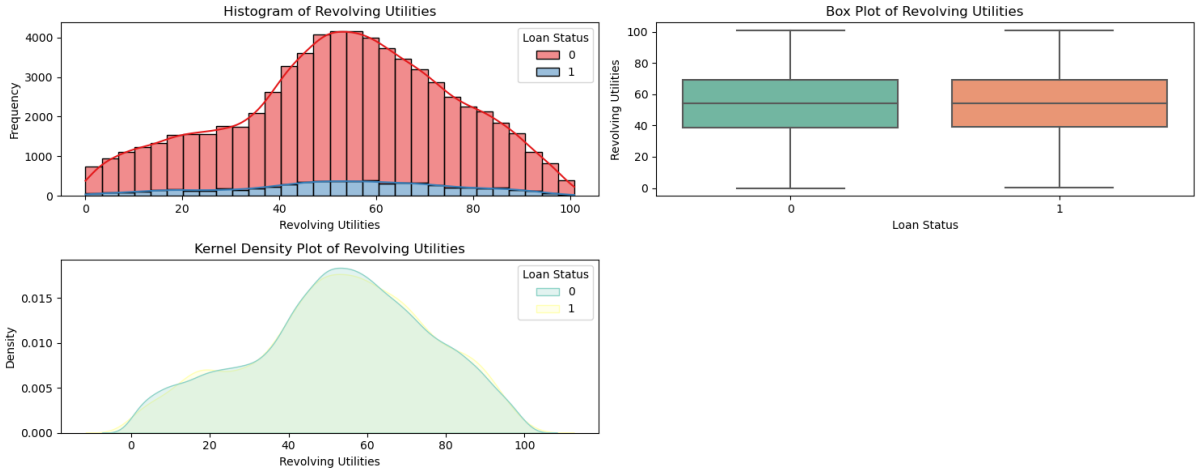


Figure 2.18: Revolving Utilities

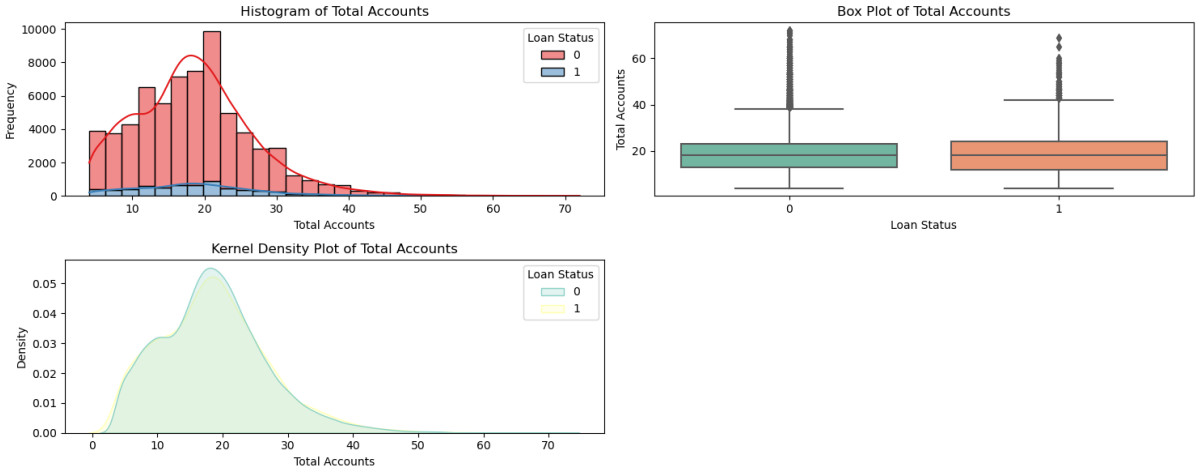


Figure 2.19: Total Accounts

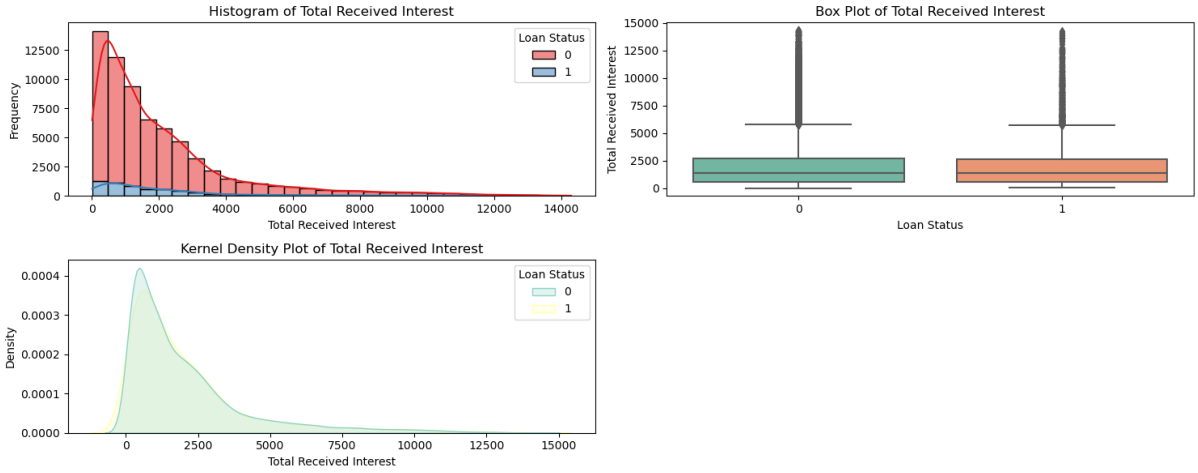


Figure 2.20: Total Received Interest

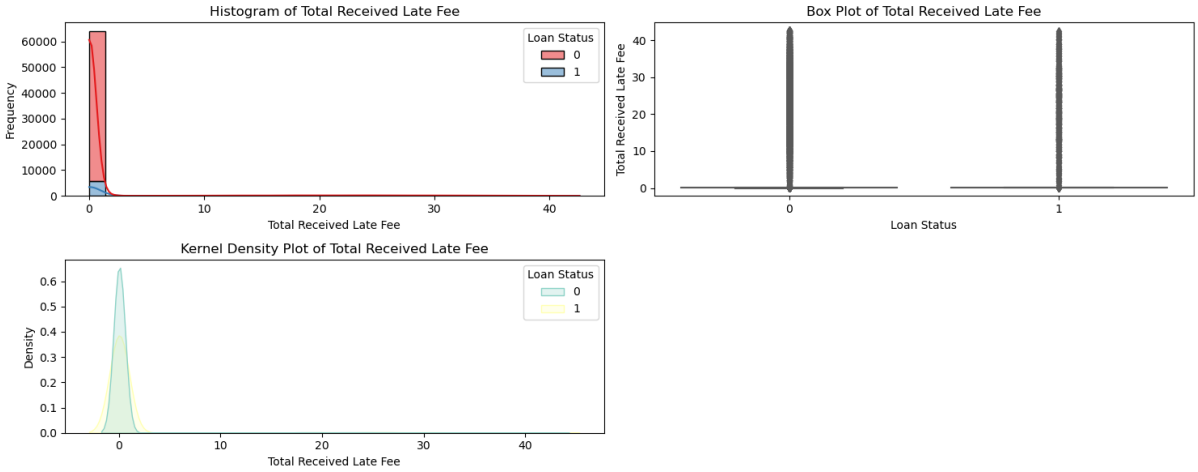


Figure 2.21: Total Received Late Fee

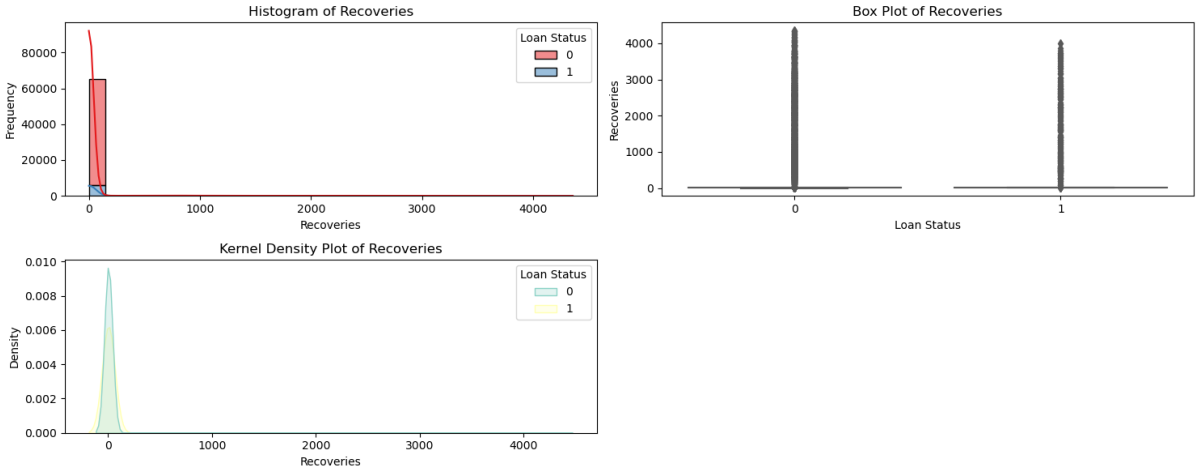


Figure 2.22: Recoveries

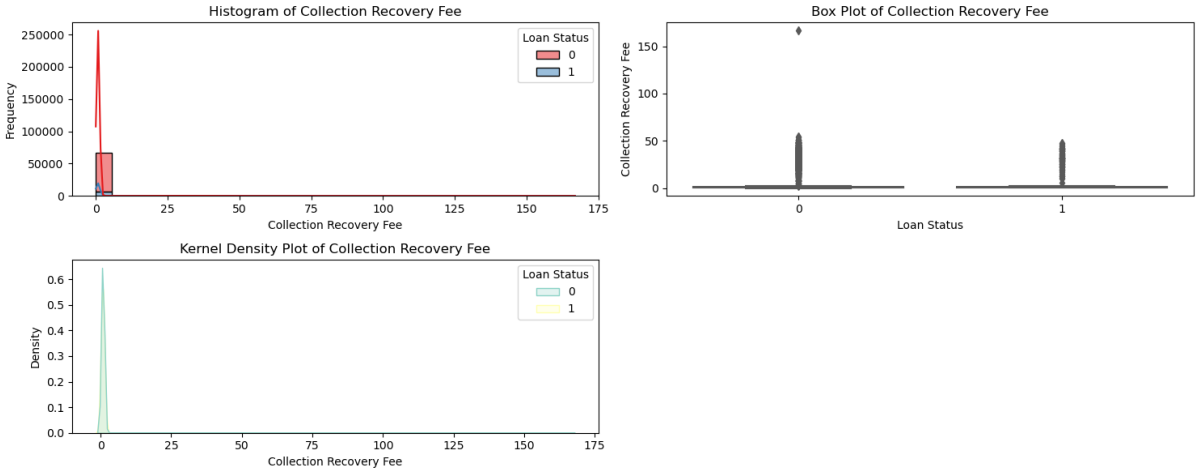


Figure 2.23: Collection Recovery Fee

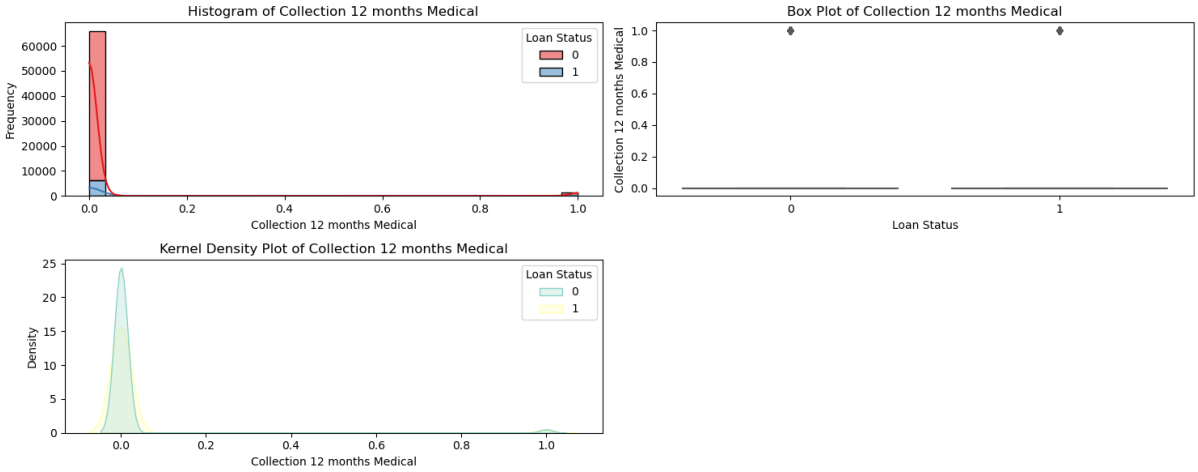


Figure 2.24: Collection 12 Months Medical

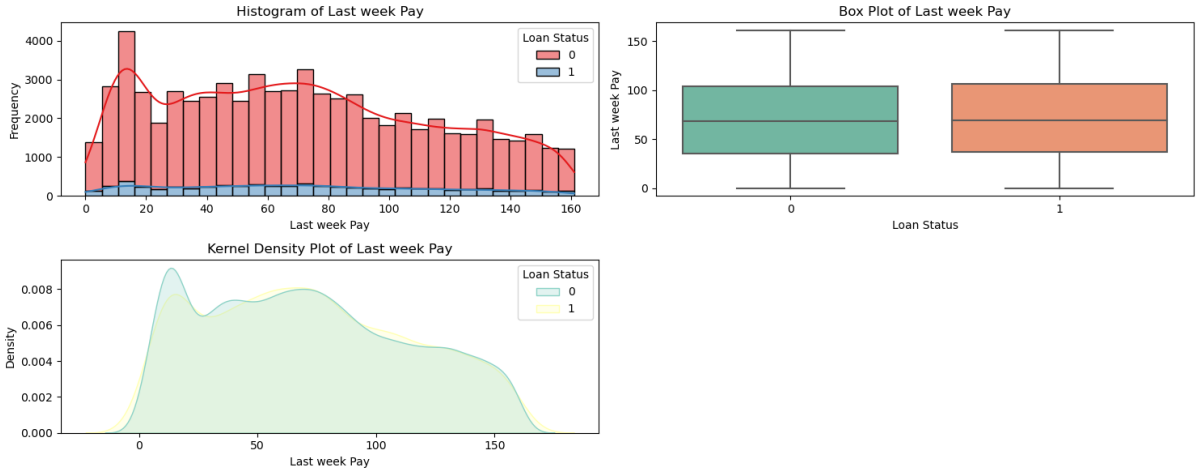


Figure 2.25: Last Week Pay

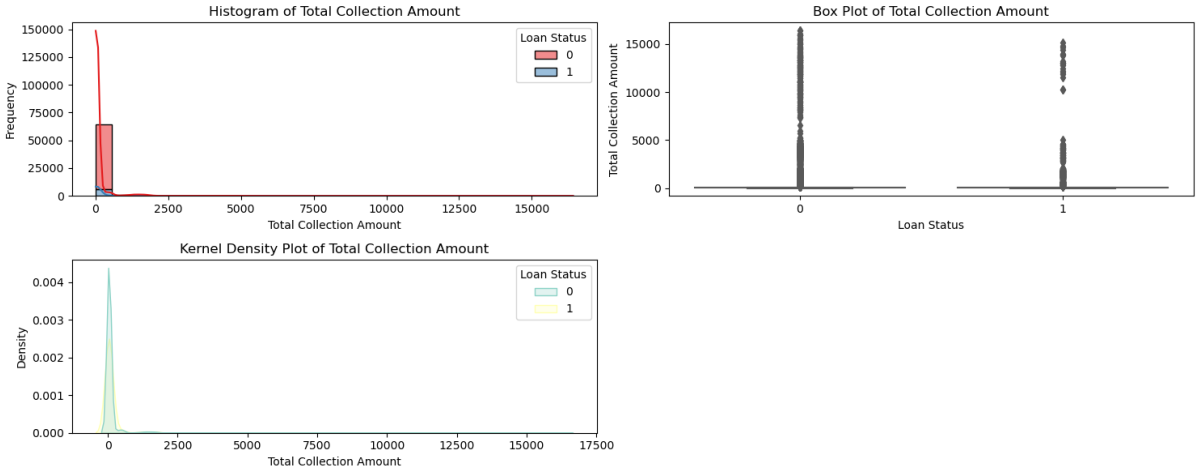


Figure 2.26: Total Collection Amount

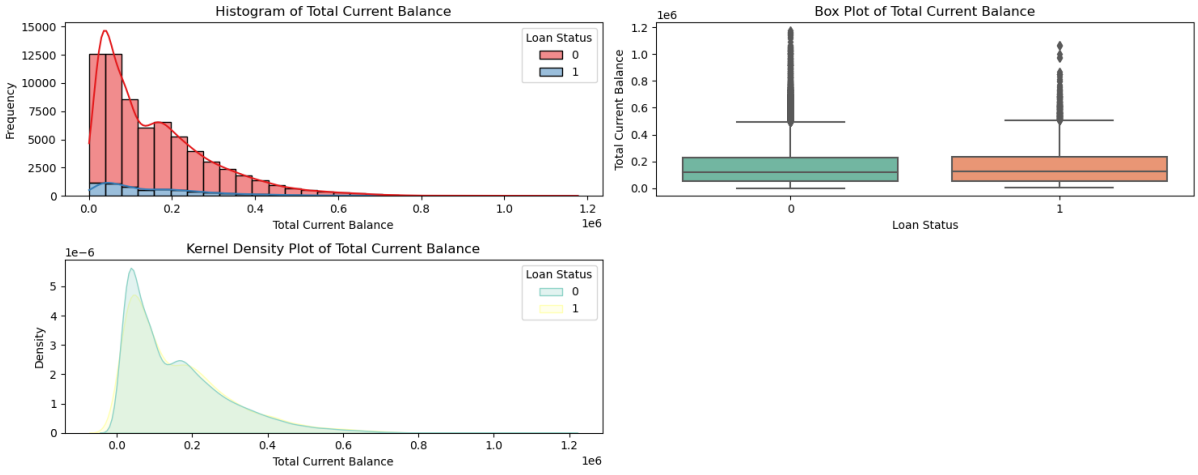


Figure 2.27: Total Current Balance

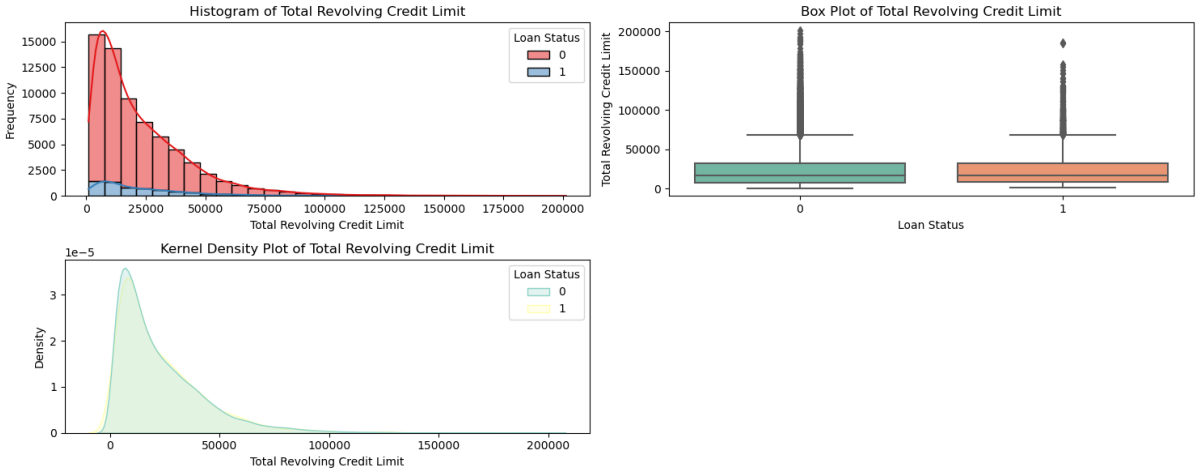


Figure 2.28: Total Revolving Credit Limit

2.3.2 Categorical columns' visualization

We adopted a varied approach for our categorical columns, utilizing heatmaps, bar plots, pie charts, and word clouds based on the number of unique values in each column. Here are the outcomes:

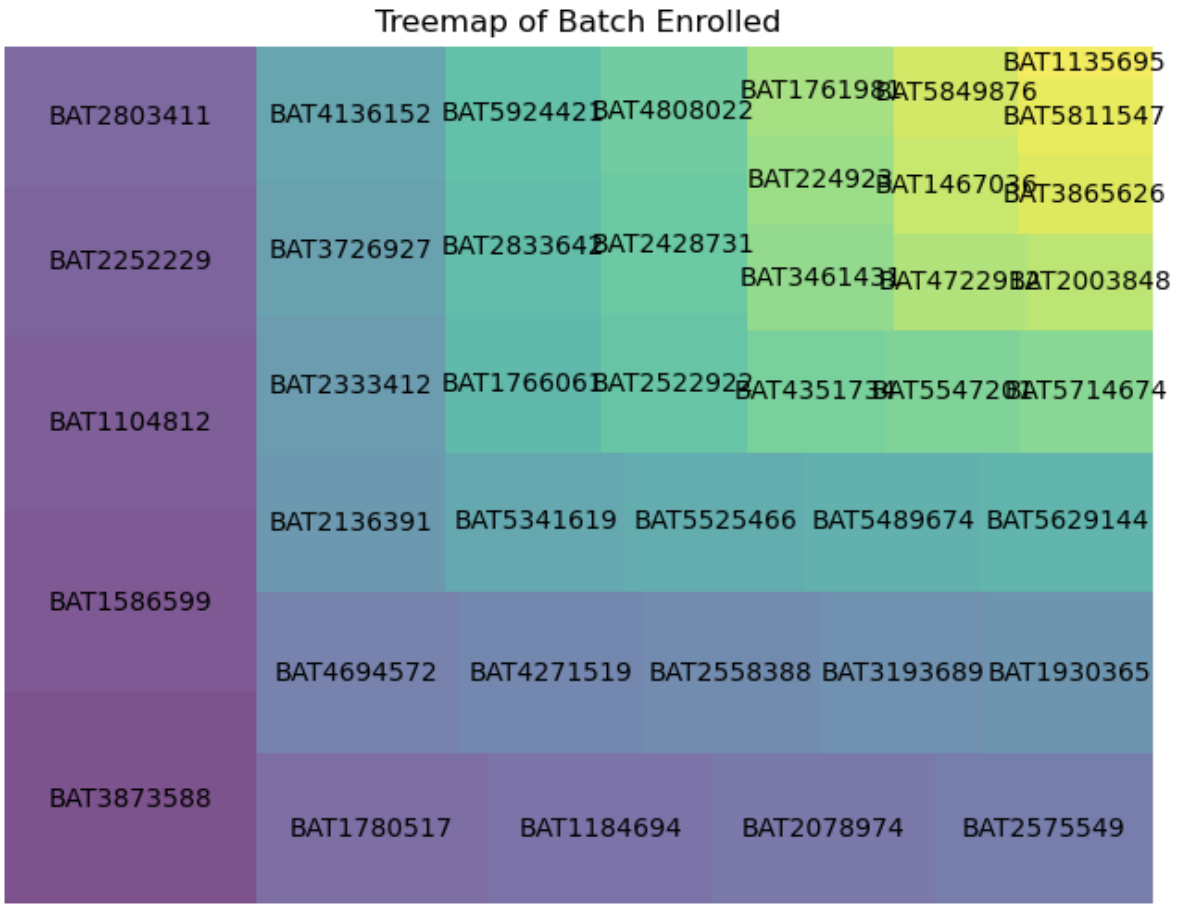


Figure 2.29: Batch Enrolled

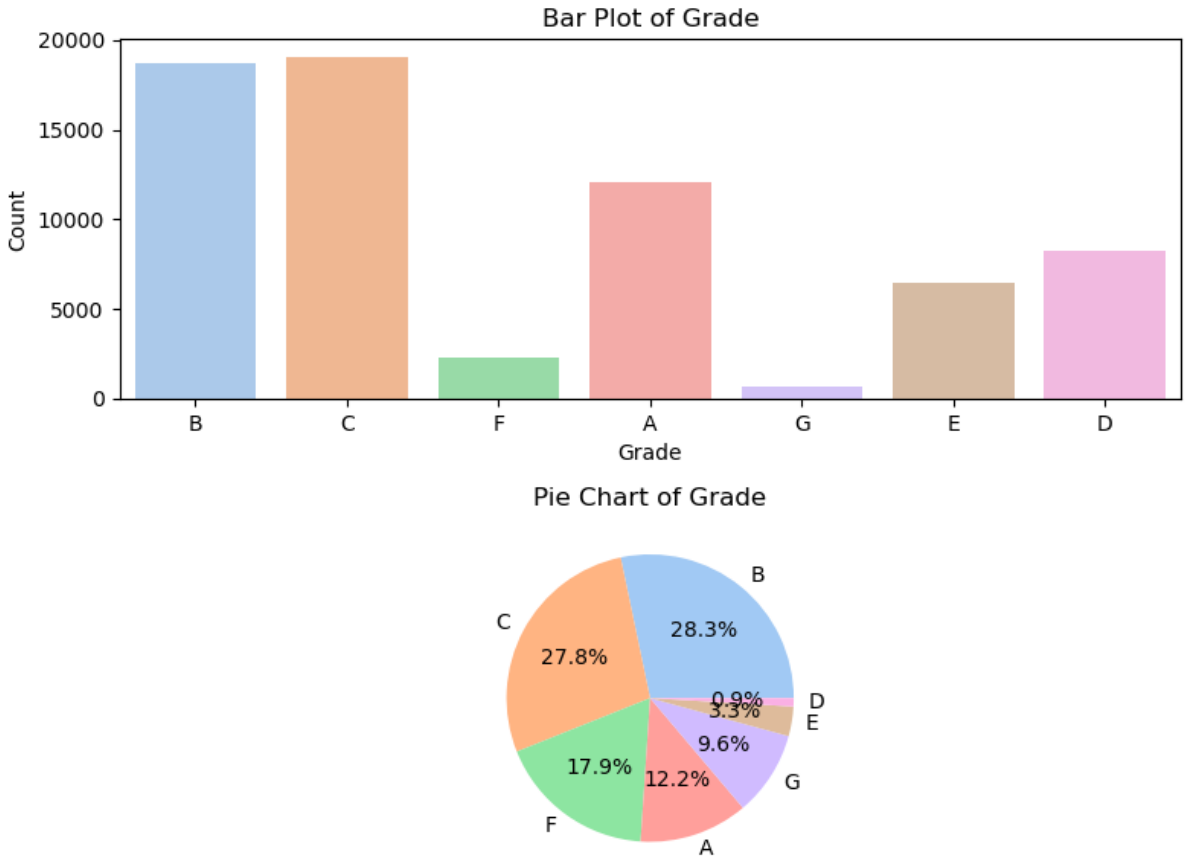


Figure 2.30: Grade

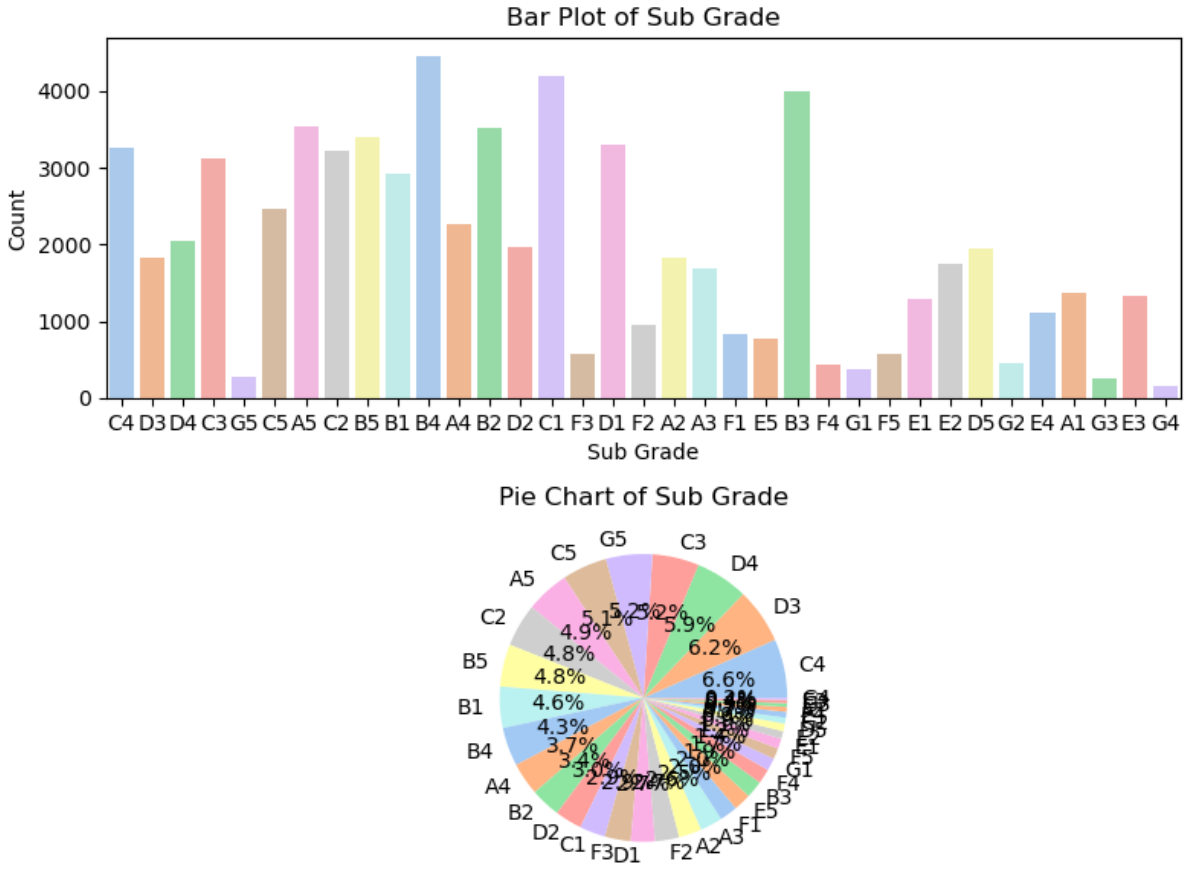


Figure 2.31: Sub Grade

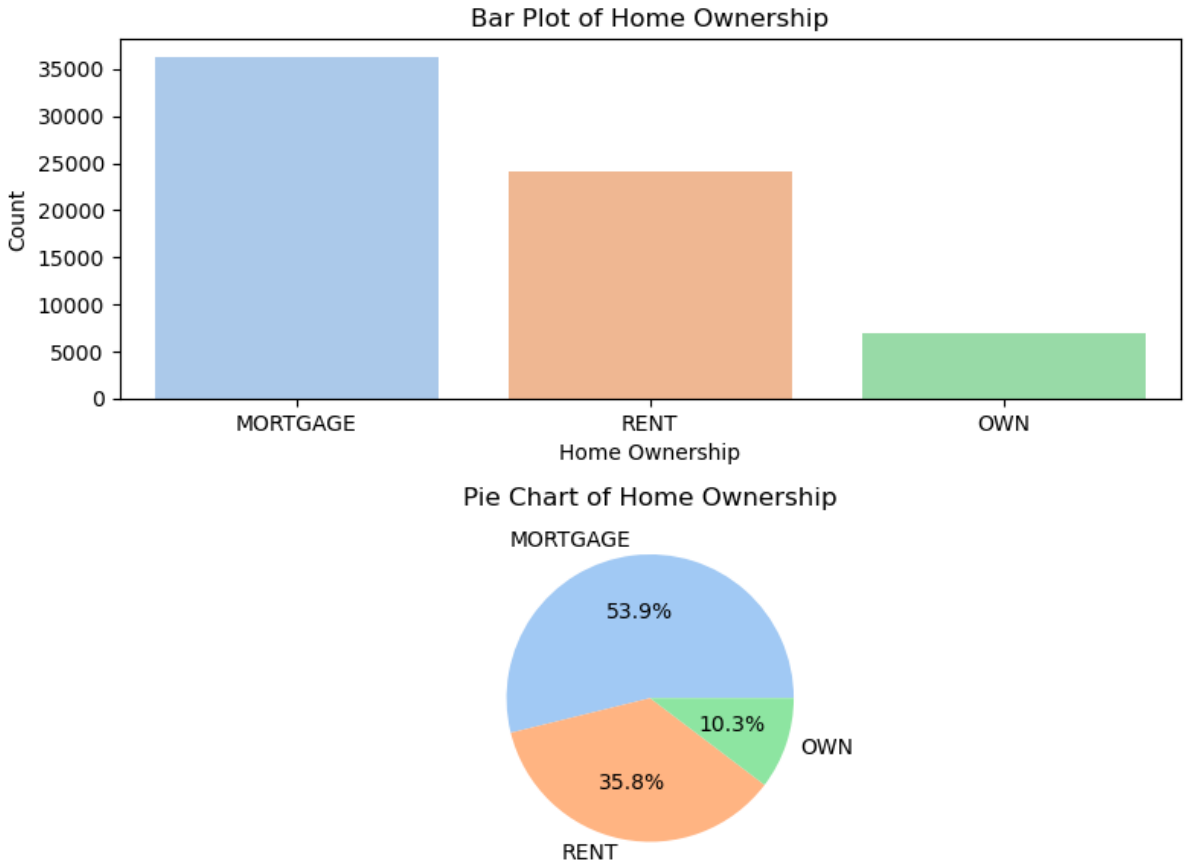


Figure 2.32: Home OwnerShip

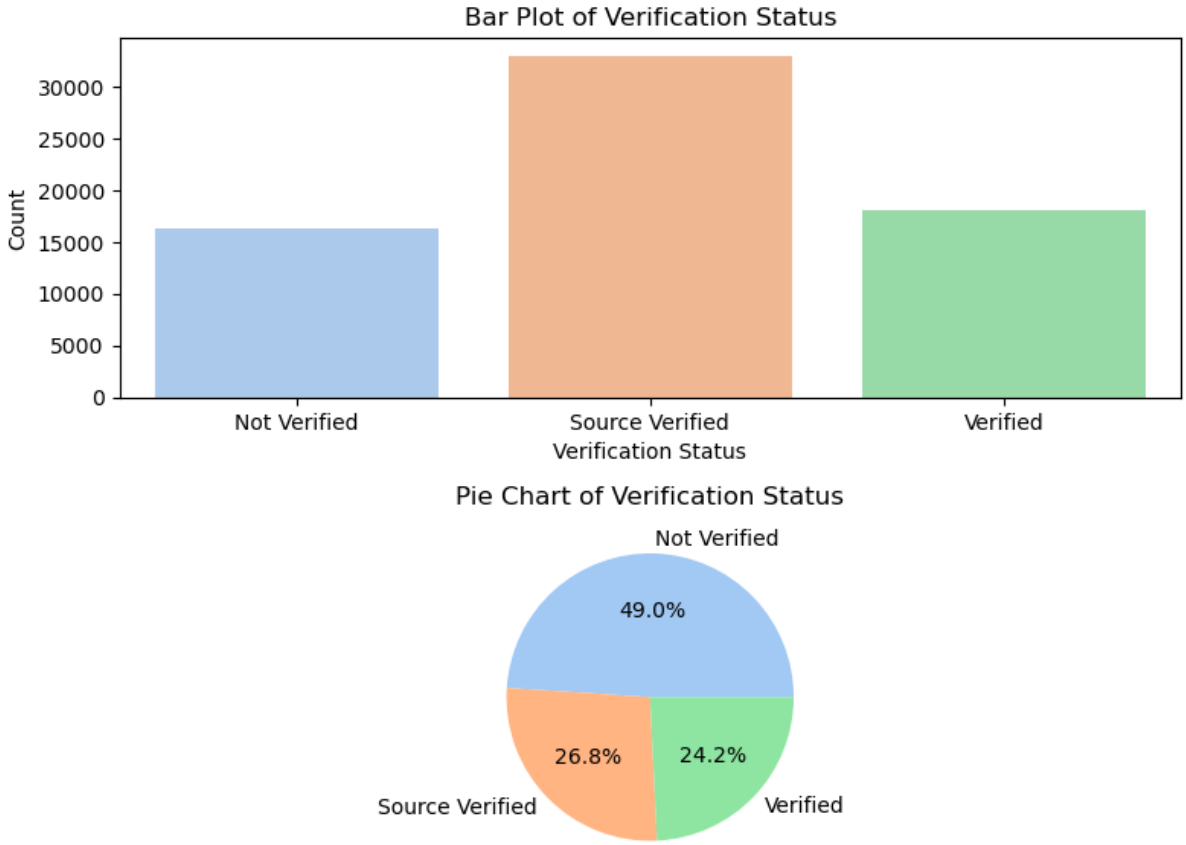


Figure 2.33: Verification Status

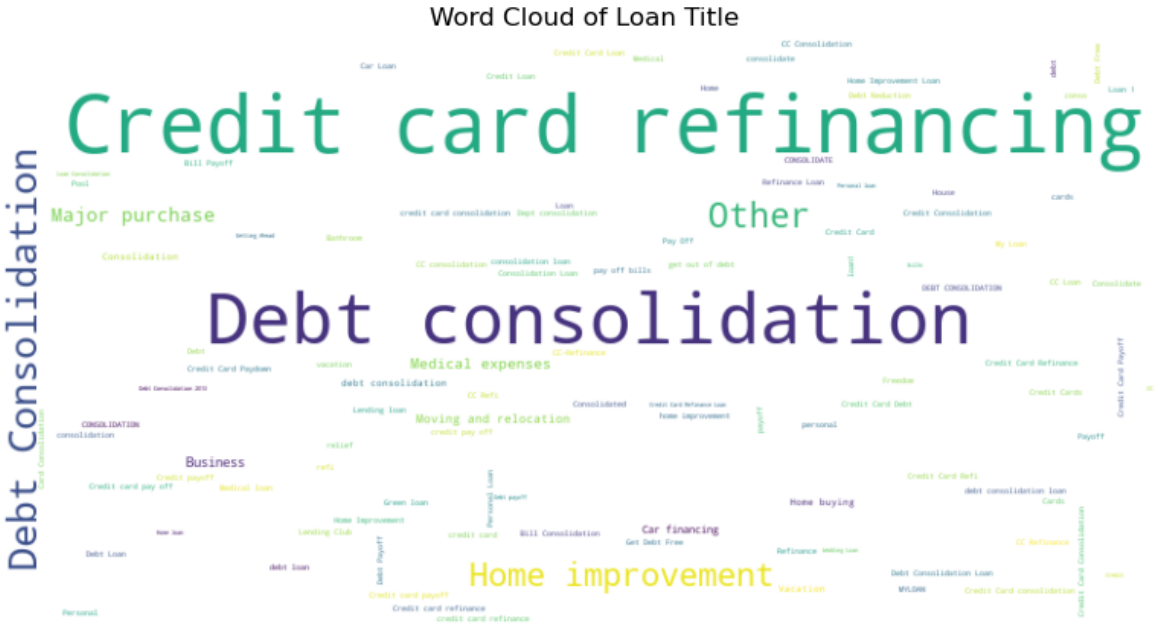


Figure 2.34: Loan Title

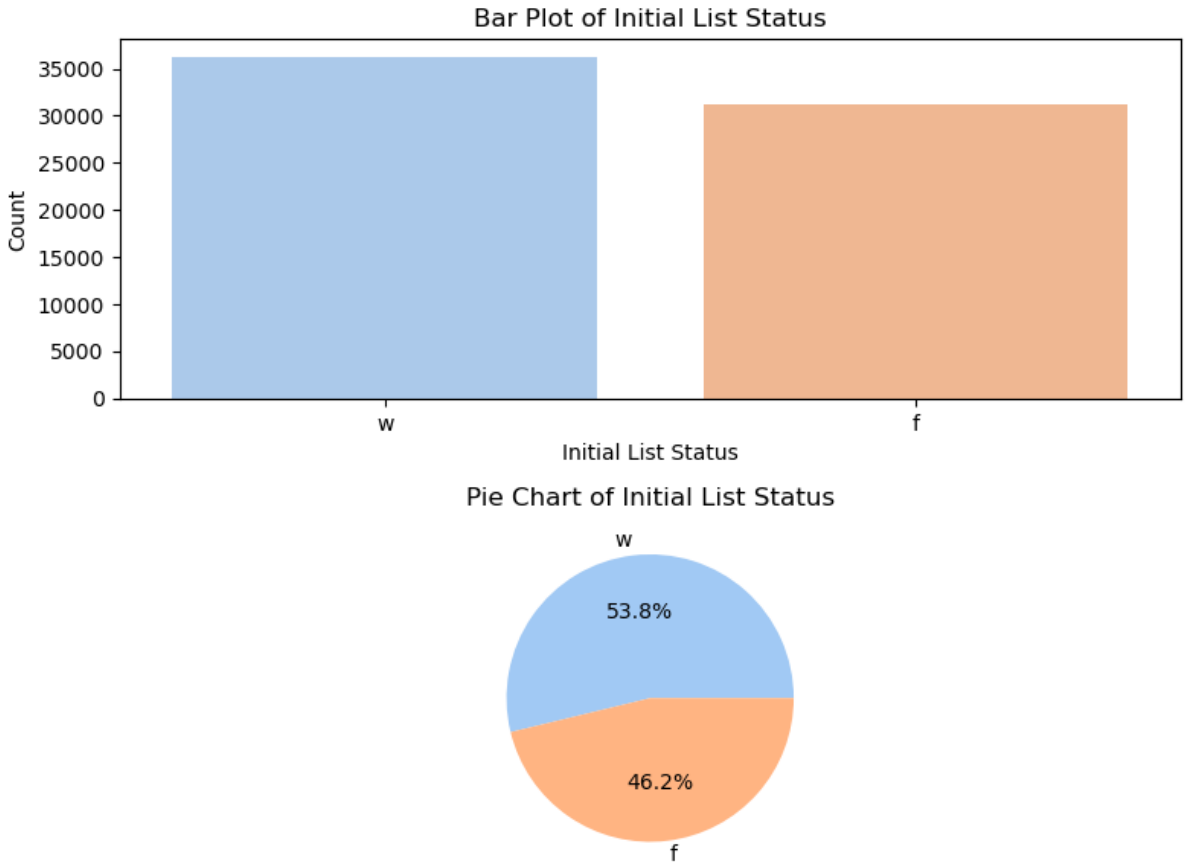


Figure 2.35: Initial List Status

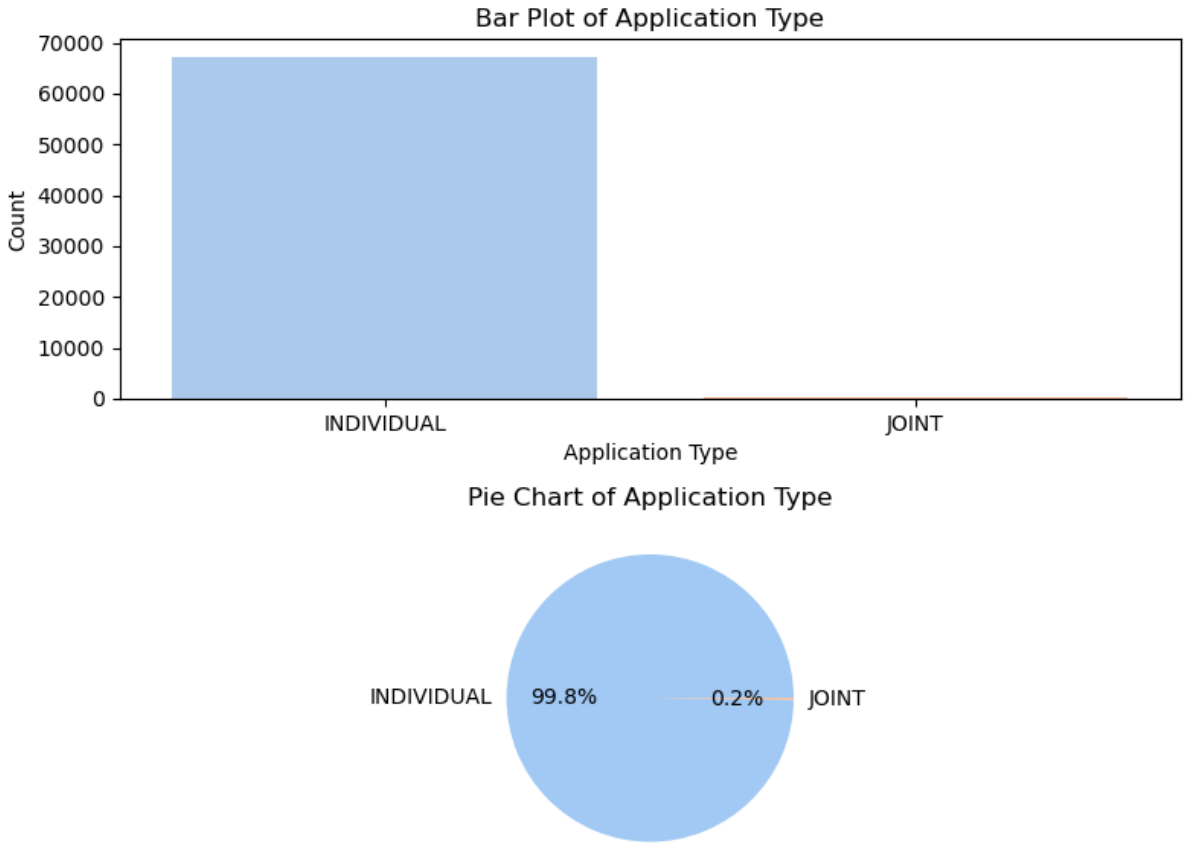


Figure 2.36: Application Type

2.3.3 Data Balancing

We posed a crucial question: 'Is our data balanced concerning the target variable, Loan Status?' Visualizing its distribution revealed imbalance; '0' constitutes 90.75% of our data, while '1' only amounts to 9.25%." This will be adressed in chapter 4.

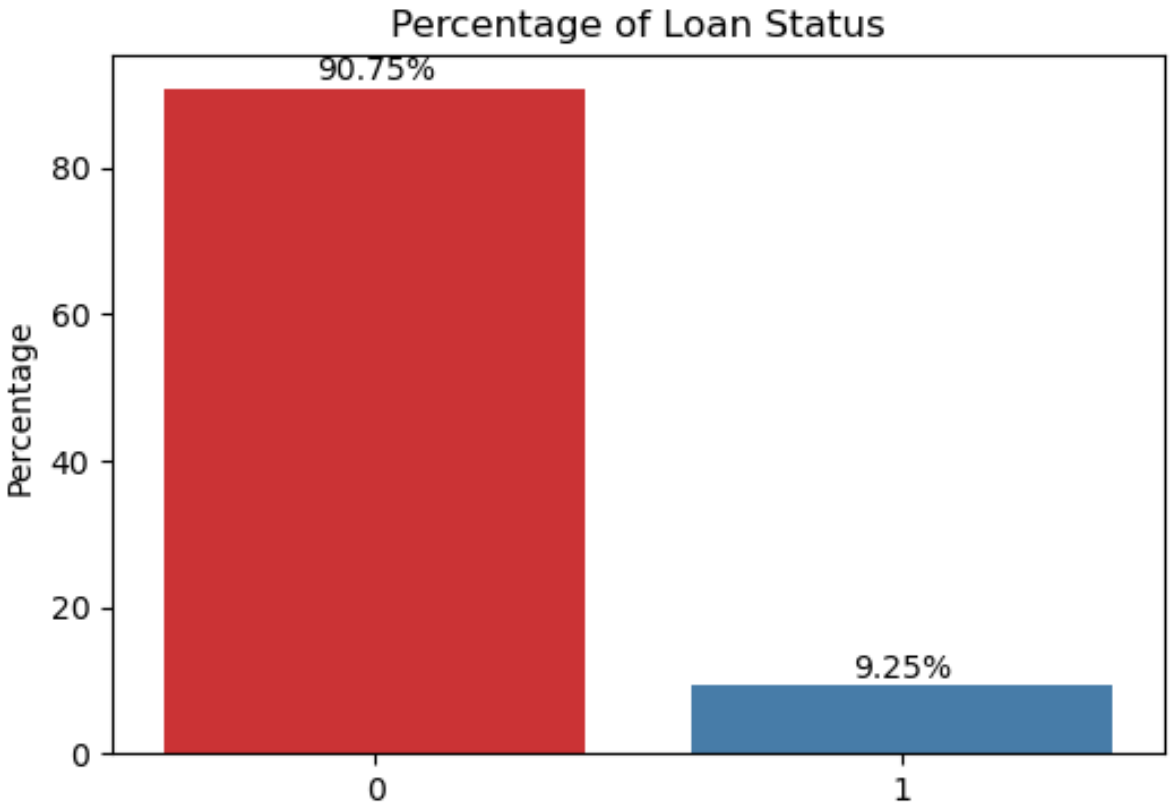


Figure 2.37: Loan Status Distribution

2.3.4 Conclusions from visualization

Visualizing our data through diverse plots has been highly beneficial. In the next chapter, we will address and analyze outliers identified during this visualization.

Chapter 3

Data Preprocessing

This chapter delves into the crucial phase of data preprocessing, an essential step in preparing raw data for effective analysis and modeling. In the following sections, we will explore various preprocessing techniques, handling missing values, dealing with outliers, and ensuring the dataset is optimized for subsequent stages in our analysis. The ultimate goal is to enhance the quality and reliability of our data, laying a solid foundation for meaningful insights and accurate modeling.

3.1 Applying correlation techniques

In this section, we utilize statistical methods, including Analysis of Variance (ANOVA) and Chi-squared tests, to discern correlations between various columns and the target variable. ANOVA is effective for evaluating the correlation between numerical variables and categorical targets, while the Chi-squared test is employed to assess associations among categorical variables. Identified correlations aid in making informed decisions during subsequent analyses.

It is important to note that columns found to be unrelated will be considered for exclusion. However, the exclusion will not be immediate, as the presence of outliers must first be addressed in the preprocessing stage. This strategic approach ensures a comprehensive and accurate refinement of the dataset.

Here's the results we got from each technique:

```
##### ANOVA Results #####

Loan Amount is NOT correlated with Loan Status | P-Value: 0.24532743370121238
Funded Amount is NOT correlated with Loan Status | P-Value: 0.7231120961020374
Funded Amount Investor is NOT correlated with Loan Status | P-Value: 0.9811486184162733
Term is NOT correlated with Loan Status | P-Value: 0.37582542132472074
Interest Rate is NOT correlated with Loan Status | P-Value: 0.4513394637979872
Employment Duration is NOT correlated with Loan Status | P-Value: 0.3344917102292555
Debit to Income is NOT correlated with Loan Status | P-Value: 0.42714577391898256
Delinquency - two years is correlated with Loan Status | P-Value: 0.009462634922992178
Inquires - six months is NOT correlated with Loan Status | P-Value: 0.8806176446325595
Open Account is NOT correlated with Loan Status | P-Value: 0.066207019877204O4
Public Record is correlated with Loan Status | P-Value: 0.005946386493849328
Revolving Balance is NOT correlated with Loan Status | P-Value: 0.780435356532883
Revolving Utilities is NOT correlated with Loan Status | P-Value: 0.284592197210366
Total Accounts is NOT correlated with Loan Status | P-Value: 0.954029173097918
Total Received Interest is NOT correlated with Loan Status | P-Value: 0.6626603145173544
Total Received Late Fee is correlated with Loan Status | P-Value: 0.014999219404735661
Recoveries is NOT correlated with Loan Status | P-Value: 0.8656143713719425
Collection Recovery Fee is NOT correlated with Loan Status | P-Value: 0.32013763521270544
Collection 12 months Medical is NOT correlated with Loan Status | P-Value: 0.8585316354726124
Last week Pay is NOT correlated with Loan Status | P-Value: 0.11211457498515208
Total Collection Amount is correlated with Loan Status | P-Value: 0.04037045067395023
Total Current Balance is correlated with Loan Status | P-Value: 0.010689900265710124
Total Revolving Credit Limit is NOT correlated with Loan Status | P-Value: 0.7057585361554068
Selected Prdictors: ['Delinquency - two years', 'Public Record', 'Total Received Late Fee', 'Total Collection Amount', 'Total Current Balance']
```

Figure 3.1: ANOVA's results

```
Chi-Square Test Results

Batch Enrolled vs Loan Status
Chi2 value: 40.33203503167361, P-value: 0.4555732749097418

Grade vs Loan Status
Chi2 value: 14.024785837884542, P-value: 0.0293605144624493

Sub Grade vs Loan Status
Chi2 value: 33.22818832002119, P-value: 0.5052675609219769

Home Ownership vs Loan Status
Chi2 value: 16.101583241640988, P-value: 0.000318849414713897

Verification Status vs Loan Status
Chi2 value: 0.6606479264176166, P-value: 0.718690866314581

Loan Title vs Loan Status
Chi2 value: 126.9083832920198, P-value: 0.1032824198045893

Initial List Status vs Loan Status
Chi2 value: 12.101443058821943, P-value: 0.0005038281525924869

Application Type vs Loan Status
Chi2 value: 0.0, P-value: 1.0

Significantly associated variables with Status: ['Grade', 'Home Ownership', 'Initial List Status']
```

Figure 3.2: Chi-Squared's results

We assessed the correlation between Grade and Sub Grade. While Sub Grade isn't directly correlated with the target variable, retaining this feature is crucial if it correlates with Grade to ensure optimal performance in subsequent analyses.

```
#grade seems to be correlated to "Loan Status", so let's check if subgrade is correlated with grade
# Create a contingency table
contingency_table = pd.crosstab(df['Grade'], df['Sub Grade'])

# Chi-square test
chi2, p, _, _ = chi2_contingency(contingency_table)
print(f"Chi-square statistic: {chi2}, p-value: {p}")
```

```
✓ 0.0s
Chi-square statistic: 736.1805835653637, p-value: 2.9253049030321995e-61
```

Figure 3.3: Correlation between Grade and Sub Grade

Since the p-value is lower than 0.05, we can say that both variables are correlated. Because of all this, we now know which columns are important to keep.

```
#let's make sure we keep the important columns in a list so we can use later on

columns_to_keep = SelectedPredictors + significant_predictors
columns_to_keep.append("Sub Grade")
print(columns_to_keep)
```

```
✓ 0.0s                                                                                    Python
['Delinquency - two years', 'Public Record', 'Total Received Late Fee', 'Total Collection Amount', 'Total Current Balance', 'Grade', 'Home Ownership', 'Initial List
```

Figure 3.4: Important columns

3.2 Handling missing values

We examined the presence of missing values (NaN or null) and, if detected, took corrective actions. Fortunately, no missing values were found.

```python
#let's check for null values
#if they exist, we must handle them
null_values_df = df.isnull().sum()
if (null_values_df.sum() == 0):
    print('There are no NULL values in the dataframe!')
else:
    print('The dataframe contains NULL values!')


#let's check for NaN values
#if they exist, we must handle them
NaN_values_df = df.isna().sum()
if (NaN_values_df.sum() == 0):
    print('There are no NaN values in the dataframe!')
else:
    print('The dataframe contains NaN values!')


#let's check for duplicates and if there are any, let's drop them
#let's print it's size before and after the drop_duplicates
before_duplicates_drop = df.size
df.drop_duplicates(inplace=True) #the inplace =True, ensures the changes are aplied tot eh variable and don't need to call/create a new one or another reference
if (df.size == before_duplicates_drop):
    print('There were no duplicates in the dataframe')
else:
    print('The dataframe had duplicates and they were removed')
```

```
There are no NULL values in the dataframe!
There are no NaN values in the dataframe!
There were no duplicates in the dataframe
```

Figure 3.5: Missing values

3.3 Outlier Handling

After meticulous data examination, we employed the Interquartile Range (IQR) method to target and remove outliers from specific columns. The IQR is a statistical measure representing the range within which the middle 50% of a dataset lies, calculated as the difference between the third quartile (Q3) and the first quartile (Q1). The chosen columns, including 'Employment Duration,' 'Interest Rate,' 'Inquiries - six months,' 'Total Accounts,' 'Total Received Interest,' 'Total Received Late Fee,' 'Recoveries,' 'Collection Recovery Fee,' 'Collection 12 months Medical,' 'Total Collection Amount,' and 'Total Revolving Credit Limit,' were selected based on insights derived from data visualization in Chapter 2, Sections 2.1 and 2.2. This focused approach, utilizing the IQR method, ensures the precision and relevance of our outlier management strategy.

```python
# these columns are worth handling outliers
columns_outliers_remove = ['Employment Duration', 'Interest Rate', 'Inquires - six months', 'Total Accounts',
                           'Total Received Interest', 'Total Received Late Fee', 'Recoveries',
                           'Collection Recovery Fee', 'Collection 12 months Medical',
                           'Total Collection Amount', 'Total Revolving Credit Limit']
```

Figure 3.6: Columns with outliers

```python
outliers_removed_info = {}

for column in columns_outliers_remove:
    # IQR method
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1

    outliers_iqr = df.loc[(df[column] < (Q1 - 1.5 * IQR)) | (df[column] > (Q3 + 1.5 * IQR)), column]

    # Store information about outliers in the dictionary
    outliers_removed_info[column] = {
        'outliers_count': outliers_iqr.size,
        'outliers_values': outliers_iqr.tolist()
    }

    # Remove outliers from the dataframe
    df = df.loc[~df.index.isin(outliers_iqr.index)]

# Print information about outliers removed for each column
for column, info in outliers_removed_info.items():
    if info['outliers_count'] == 0:
        continue
    print(f"Outliers for {column} using IQR method ({info['outliers_count']})\n")
```

Figure 3.7: Outlier handling

```
Outliers for Employment Duration using IQR method (4115)

Outliers for Interest Rate using IQR method (793)

Outliers for Inquires - six months using IQR method (6436)

Outliers for Total Accounts using IQR method (1147)

Outliers for Total Received Interest using IQR method (4026)

Outliers for Total Received Late Fee using IQR method (2502)

Outliers for Recoveries using IQR method (1578)

Outliers for Collection Recovery Fee using IQR method (546)

Outliers for Collection 12 months Medical using IQR method (973)

Outliers for Total Collection Amount using IQR method (3108)

Outliers for Total Revolving Credit Limit using IQR method (1659)
```

Figure 3.8: Outlier removal

Simultaneously, a strategic decision was made to exclude rows where the 'Application Type' equaled 'Joint,' representing a mere 0.2% of the dataset. This rigorous preprocessing ensured a refined dataset.

```python
    # lets's remove the rows in which application type == Joint
    # only 0.2% are joint, so let's remove them as they are outliers
    df= df[df['Application Type'] != 'Joint']
```

Figure 3.9: CRemoval of Joint Aplication type

Subsequently, a thorough reassessment of the Loan Status distribution was conducted to fortify

our understanding of the data's integrity and distribution characteristics. It was concluded that the distribution did not change much.

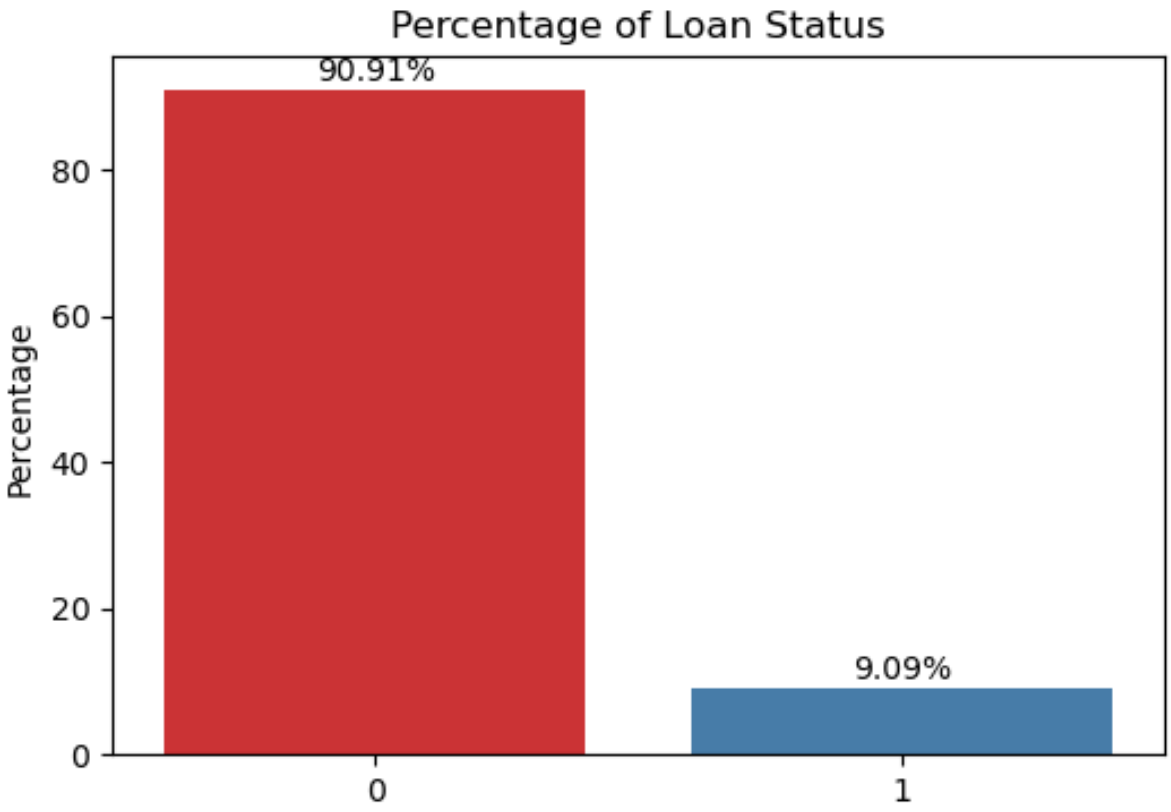


Figure 3.10: Loan Status after outlier removal

And to conclude this section, a total of 26883 outliers (rows) were removed!

3.4 Column Removal

After addressing outliers, we eliminated columns deemed insignificant. First, we identified and dropped columns with only 1 unique value. The removal process is illustrated in Figure 3.2.

```
#Now that we have dealt with outliers, let's check if there are any columns with 1 unique value only.
#if so, let's drop them

columns_to_drop = []

for col in df.columns.to_list():
    if df[col].nunique() == 1:
        columns_to_drop.append(col)

if len(columns_to_drop) > 0:
    print(f"Columns dropped: {columns_to_drop}")
    df = df.drop(columns_to_drop, axis=1)
else:
    print('No columns were dropped!')
```

Python

```
Columns dropped: ['Inquires - six months', 'Collection 12 months Medical']
```

Figure 3.11: Removal of column with only 1 unique value

Following correlation techniques applied in Section 3.1 and outlier handling, we excluded columns not deemed essential for further analysis.

```python
df_columns_list = df.columns.to_list()
df_columns_list.remove('Loan Status')
#df_columns_list.remove('Sub Grade')
```

Python

```python
# This will be used to make modifications to the dataframe, in hopes it improves the models' performance
for col in df_columns_list:
    if col not in columns_to_keep:
        df.drop(col, axis=1, inplace=True)

print(columns_to_keep)
```

Python

Figure 3.12: Removal of columns deemed insignificant

3.5 Categorical columns encoding

The code snippet presented in Figure 3.13 illustrates the encoding of binary and categorical columns in the dataset. Utilizing a Label Encoder for binary columns and one-hot encoding for non-binary categorical columns, this preprocessing step transforms categorical data into a format suitable for machine learning models, enhancing their interpretability and performance.

```python
#let's encode the binary columns:

# Assuming df is your DataFrame
label_encoder = LabelEncoder()

for column in binary_columns:
    df[column] = label_encoder.fit_transform(df[column])

categorical_cols = df.select_dtypes(include=['object', 'category']).columns.to_list()


#Let's encode the categorical columns that are not binary
df_encoded = pd.get_dummies(df, columns=categorical_cols, prefix=categorical_cols, drop_first=True)
```

0.0s Python

Figure 3.13: Categorical Columns enconding

With the completion of the preprocessing steps outlined in Chapter 3, the dataset has undergone meticulous cleaning, handling of outliers, and encoding of categorical variables. These crucial preprocessing procedures set the stage for the subsequent modeling phase. The refined dataset is now poised for the application of machine learning algorithms to build predictive models in the pursuit of our research objectives.

Chapter 4

Creation of Models using Data Mining Algorithms

In this pivotal chapter, our focus shifts to the application of cutting-edge data mining algorithms for the creation of predictive models. Building upon the foundation laid in the preceding chapters, where we explored and prepared our dataset, we now harness the power of advanced algorithms to unearth patterns, relationships, and insights. Through a systematic and methodical approach, we aim to construct robust models capable of predicting loan default with accuracy and precision. Join us on this journey as we navigate the intricacies of data mining and leverage its capabilities to unlock valuable predictions in the realm of loan default.

4.1 Data Preprocessing for Model Training

This section delves into the crucial steps of preparing our data for the creation of predictive models. We initiated the process by segregating features and the target variable, followed by employing sampling strategies such as NearMiss to address class imbalance. Furthermore, to ensure consistency across features, we applied standardization using StandardScaler.

The initial data split involved allocating 20% for testing and 80% for training, maintaining a balanced distribution of the target variable through stratification. Result reproducibility was guaranteed with a random state of 42. Recognizing the substantial imbalance in our target variable, we utilized NearMiss undersampling for effective balancing.

In our experimentation, NearMiss demonstrated superior performance compared to alternative methods like random undersampling and tomek links, showcasing its effectiveness in handling class imbalance.

```python
# Separate features and target variable
X = df_encoded.drop('Loan Status', axis=1)
y = df_encoded['Loan Status']


# undersampler = RandomUnderSampler(random_state = 42)
undersampler = NearMiss(sampling_strategy='auto')

# smote = SMOTE(sampling_strategy='auto', random_state=42)
# X, y = smote.fit_resample(X, y)




# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,  random_state=42)

X_train, y_train = undersampler.fit_resample(X_train, y_train)


##Min-Max Normalization
# scaler = MinMaxScaler()
# X_train = scaler.fit_transform(X_train)
# X_test = scaler.transform(X_test)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 4.1: Data Preparation

4.2 Cross-Validation and balancing techniques comparison

In this section, we conducted an extensive evaluation of various machine learning models using cross-validation techniques. The models considered include Logistic Regression, Decision Tree, Naive Bayes, Random Forest, Gradient Boosting, AdaBoost, K-NN, Support Vector Machine (SVM), XGBoost, and a Neural Network implemented with Keras. Utilizing K-Fold cross-validation with 10 splits, the models were systematically assessed for their performance on the training data. The evaluation metric employed was the Receiver Operating Characteristic Area Under the Curve (ROC AUC), providing insights into each model's ability to discriminate between positive and negative instances. These were the results obtained:

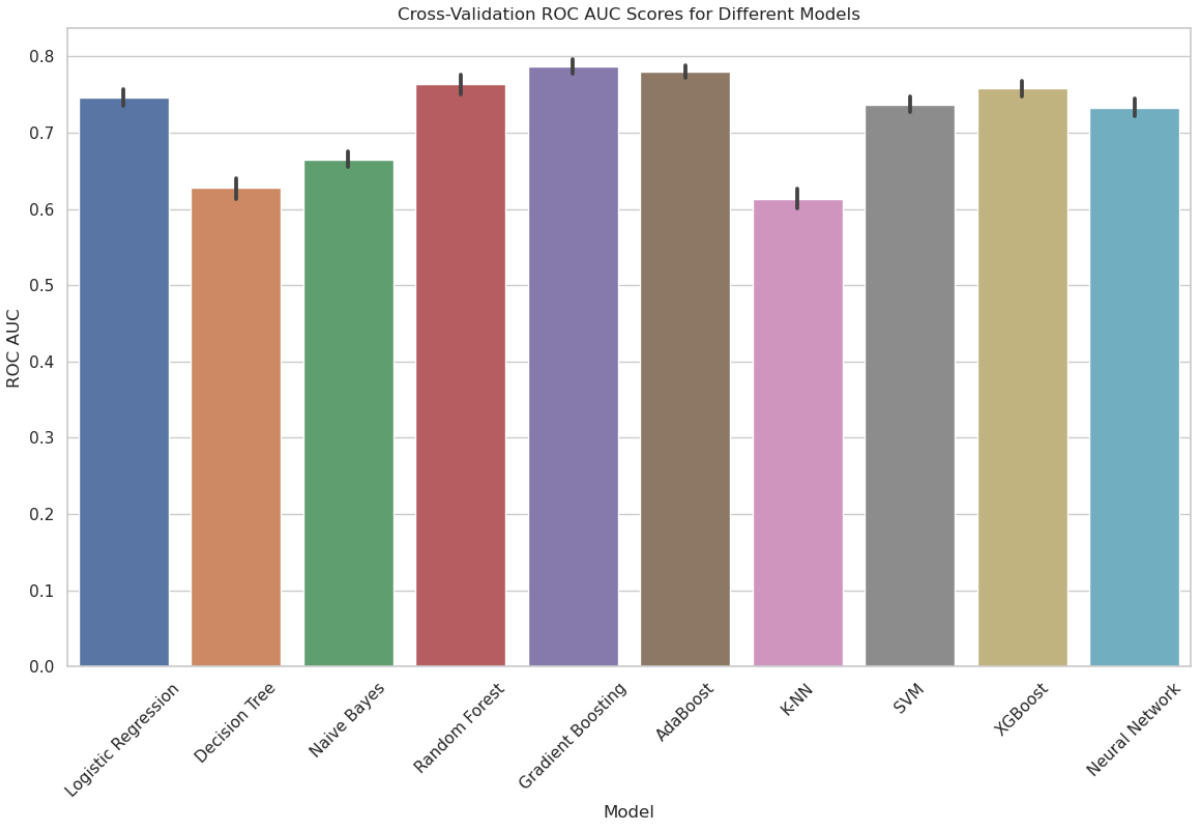


Figure 4.2: Cross-Validation NearMiss Undersampler

.

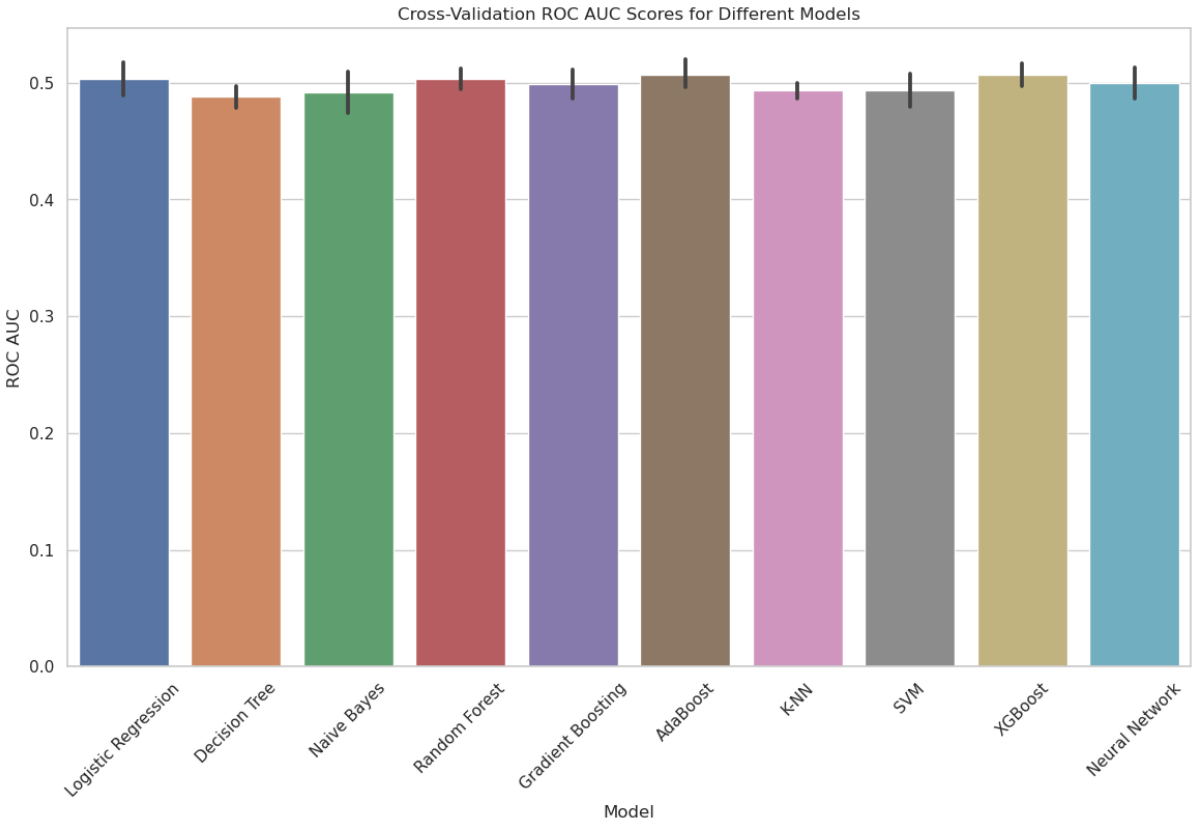


Figure 4.3: Cross-Validation RandomUndersampler

4.3 Model Creation

4.3.1 Boosting Algorithms

Gradient Boosting

Gradient Boosting utilizes sequential weak learners to enhance predictive accuracy. It builds models in a stage-wise fashion, each correcting errors made by the previous ones.

```
#Gradient Boosting
gradient_boosting_model = GradientBoostingClassifier(random_state=42)
gradient_boosting_model.fit(X_train, y_train)
y_pred_gb = gradient_boosting_model.predict(X_test)
```

Figure 4.4: Gradient Boosting

AdaBoost

AdaBoost, or Adaptive Boosting, focuses on misclassified instances by assigning more weight to them in subsequent iterations. It adapts to improve performance based on the difficulty of instances.

```
#AdaBoost
adaboost_model = AdaBoostClassifier(random_state=42)
adaboost_model.fit(X_train, y_train)
y_pred_adaboost = adaboost_model.predict(X_test)
```

Figure 4.5: AdaBoost

XGBoost

XGBoost is an extreme gradient boosting implementation known for its speed and performance. It employs a regularization term to control overfitting and is widely used in machine learning competitions.

```
#XGBoost
xgboost_model = XGBClassifier(random_st  (variable) X_test: Any | ndarray | spmatri
xgboost_model.fit(X_train, y_train)
y_pred_xgboost = xgboost_model.predict(X_test)
✓ 0.1s
```

Figure 4.6: XGBoost

LightGBM

LightGBM is a gradient boosting framework that uses tree-based learning algorithms. It is designed for efficiency and scalability, making it suitable for large datasets.

```
#Lightgbm
lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_train, y_train)
y_pred_lgbm = lgbm_model.predict(X_test)
✓ 0.1s
```

Figure 4.7: Visualization of LightGBM

CatBoost

In addition to the previously mentioned models, we explored the inclusion of CatBoost, a powerful gradient boosting algorithm known for its ability to handle categorical features without manual encoding. While the categorical data in our dataset has already been encoded, CatBoost still offers advantages such as efficient training, resistance to overfitting, and the boosting framework's capacity to capture complex patterns in the data.

We'll evaluate the performance of CatBoost alongside other models to determine its impact on overall predictive accuracy. This inclusion allows us to explore potential synergies with the ensemble of models and uncover any additional insights CatBoost may provide.

```
#Catboost
catboost_model = CatBoostClassifier(random_state=42, verbose=0)
catboost_model.fit(X_train, y_train)
y_pred_catboost = catboost_model.predict(X_test)
✓ 2.4s
```

Figure 4.8: CatBoost

4.3.2 Linear Algorithms

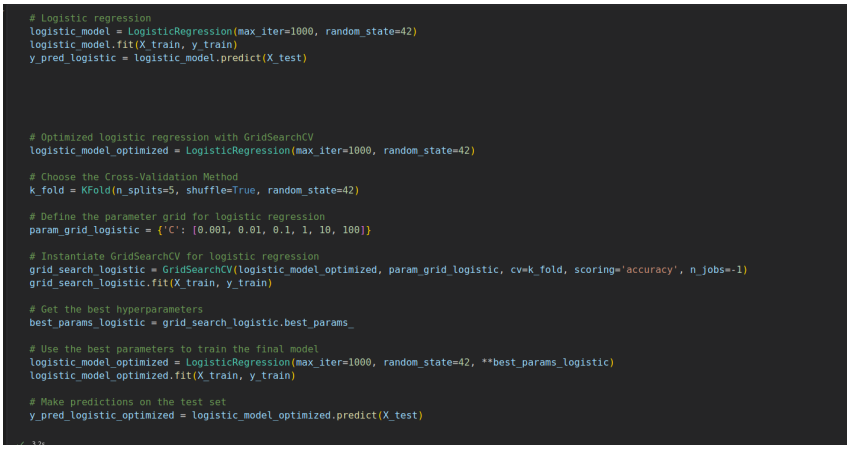
Within the realm of linear algorithms, we employed logistic regression, a widely used method for binary classification, and an optimized version achieved through GridSearchCV.

Logistic Regression

Logistic regression is a linear model suitable for binary classification tasks. It estimates the probability that an instance belongs to a particular class.

Optimized Logistic Regression

To enhance the performance of logistic regression, we utilized GridSearchCV for hyperparameter tuning. The best hyperparameters were determined through cross-validation, leading to an optimized logistic regression model.



```
# Logistic regression
logistic_model = LogisticRegression(max_iter=1000, random_state=42)
logistic_model.fit(X_train, y_train)
y_pred_logistic = logistic_model.predict(X_test)



# Optimized logistic regression with GridSearchCV
logistic_model_optimized = LogisticRegression(max_iter=1000, random_state=42)

# Choose the Cross-Validation Method
k_fold = KFold(n_splits=5, shuffle=True, random_state=42)

# Define the parameter grid for logistic regression
param_grid_logistic = {'C': [0.001, 0.01, 0.1, 1, 10, 100]}

# Instantiate GridSearchCV for logistic regression
grid_search_logistic = GridSearchCV(logistic_model_optimized, param_grid_logistic, cv=k_fold, scoring='accuracy', n_jobs=-1)
grid_search_logistic.fit(X_train, y_train)

# Get the best hyperparameters
best_params_logistic = grid_search_logistic.best_params_

# Use the best parameters to train the final model
logistic_model_optimized = LogisticRegression(max_iter=1000, random_state=42, **best_params_logistic)
logistic_model_optimized.fit(X_train, y_train)

# Make predictions on the test set
y_pred_logistic_optimized = logistic_model_optimized.predict(X_test)
```

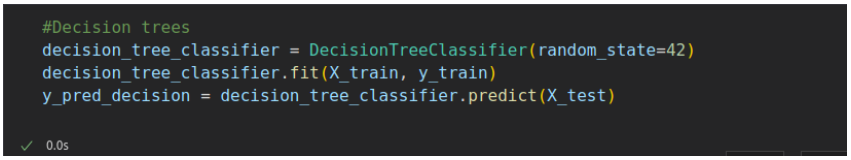
Figure 4.9: Logistic Regression

The optimized logistic regression model was then trained using the best parameters and applied to make predictions on the test set.

4.3.3 Decision Trees Algorithms

Decision Tree

A decision tree is a predictive model that maps features to outcomes by recursively partitioning the data based on conditions. It is a non-linear model capable of capturing complex relationships in the data.



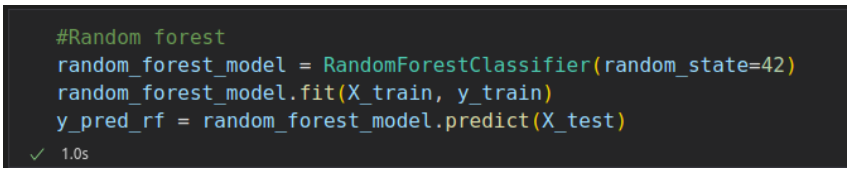
```
#Decision trees
decision_tree_classifier = DecisionTreeClassifier(random_state=42)
decision_tree_classifier.fit(X_train, y_train)
y_pred_decision = decision_tree_classifier.predict(X_test)

✓ 0.0s
```

Figure 4.10: Decision Tree

Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training. It combines the predictions of these trees to enhance model robustness and reduce overfitting.



```
#Random forest
random_forest_model = RandomForestClassifier(random_state=42)
random_forest_model.fit(X_train, y_train)
y_pred_rf = random_forest_model.predict(X_test)

✓ 1.0s
```

Figure 4.11: Random Forest

4.3.4 Instance-Based Learning

We applied Instance-Based Learning using the **K-Nearest Neighbors (KNN)** algorithm. KNN is a simple, yet effective, algorithm that classifies a data point based on the majority class of its k-nearest neighbors. In our experimentation, we tested various odd numbers of neighbors between 1 and 50, and the optimal number was found to be 49, providing the best performance.

```python
#K-NN

knn_model = KNeighborsClassifier(n_neighbors=49)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
```

Figure 4.12: K-Nearest Neighbors (KNN)

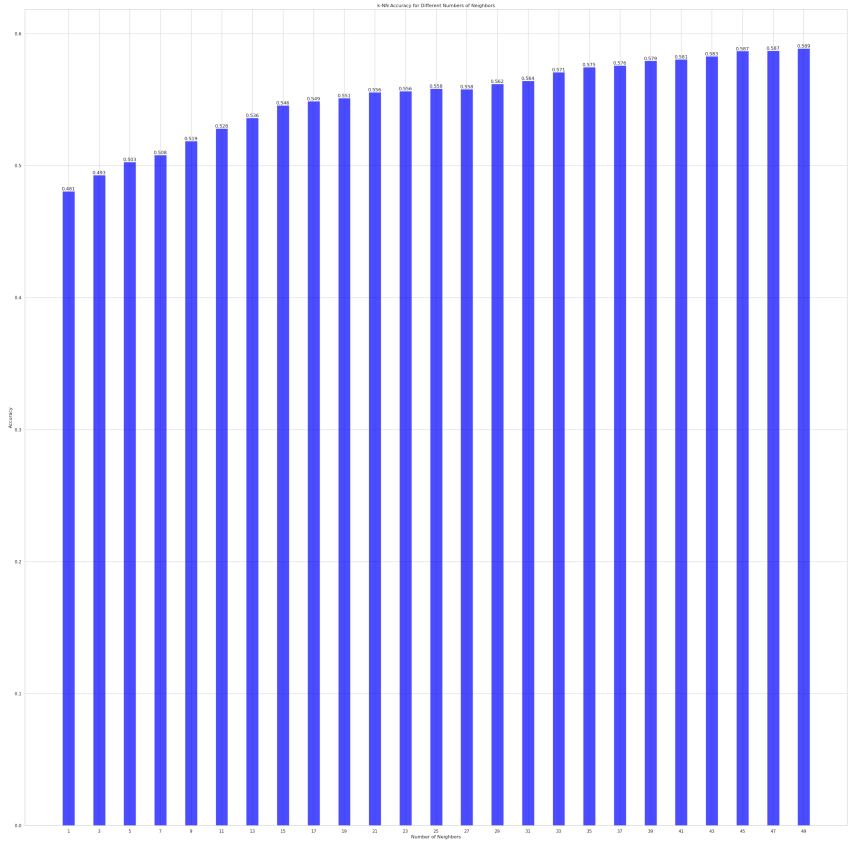


Figure 4.13: Knn number of neighbours

4.3.5 Neural Networks

In this section, we employed the Keras library to implement our neural network models. Keras is a high-level neural networks API written in Python, designed to be user-friendly and capable of running on top of various deep learning frameworks, including TensorFlow and Theano. Its simplicity and flexibility make it a popular choice for both beginners and experienced deep learning practitioners.

We utilized a straightforward neural network architecture, consisting of three layers: an input layer with 64 neurons and ReLU activation function, a hidden layer with 32 neurons and ReLU activation, and an output layer with one neuron using a sigmoid activation function. The model was compiled with the Adam optimizer and binary cross-entropy loss, while accuracy served as the evaluation metric.

To mitigate overfitting, EarlyStopping with a patience of 3 epochs and restoration of the best weights was implemented. The neural network underwent training for 20 epochs with a batch size of 32, and 20% of the training data was utilized for validation. Predictions were made on the test set, and a threshold of 0.5 was applied to convert probabilities into binary labels.

```python
#Neural Networks
nn_model = Sequential()
nn_model.add(Dense(64, activation='relu', input_dim=X_train.shape[1]))
nn_model.add(Dense(32, activation='relu'))
nn_model.add(Dense(1, activation='sigmoid'))

nn_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Use EarlyStopping to prevent overfittings
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

nn_model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2, callbacks=[early_stopping])

# Predict using the neural network
y_pred_nn_probs = nn_model.predict(X_test).flatten()
# Apply a threshold (e.g., 0.5) to convert probabilities to binary labels
y_pred_nn = (y_pred_nn_probs > 0.5).astype(int)
```

Figure 4.14: KNeural Networks (Keras)

4.3.6 Probabilistic Models

In the realm of probabilistic models, we explored the Naive Bayes algorithm. Naive Bayes is a classification technique based on Bayes' theorem with an assumption of independence between features. Despite its simplistic assumption, it has proven to be effective, especially in text classification and spam filtering.

For our analysis, we implemented the standard Naive Bayes classifier and also explored an optimized version through grid search. The optimization process involved tuning hyperparameters to enhance the model's performance. Naive Bayes models are particularly suitable for scenarios where feature independence can be assumed, making them efficient and interpretable for various applications.

```python
#Naive Bayes
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)

#----------------------------------------

# Define the Gaussian Naive Bayes model
nb_model = GaussianNB()

# Choose the Cross-Validation Method
k_fold = KFold(n_splits=5, shuffle=True, random_state=42)

# Define the parameter grid for Gaussian Naive Bayes
param_grid_nb = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}

# Instantiate GridSearchCV for Gaussian Naive Bayes
grid_search_nb = GridSearchCV(nb_model, param_grid_nb, cv=k_fold, scoring='accuracy', n_jobs=-1)
grid_search_nb.fit(X_train, y_train)

# Get the best hyperparameters
best_params_nb = grid_search_nb.best_params_

# Use the best parameters to train the final model
nb_model_optimized = GaussianNB(**best_params_nb)
nb_model_optimized.fit(X_train, y_train)

# Make predictions on the test set
y_pred_nb_optimized = nb_model_optimized.predict(X_test)
```

Figure 4.15: Naive Bayes

4.3.7 Support Vector Machines (SVM)

Support Vector Machines (SVM) is a powerful supervised learning algorithm used for classification and regression tasks. The primary goal of SVM is to find a hyperplane in an N-dimensional space that distinctly classifies data points. In a binary classification scenario, SVM aims to maximize the margin between classes, i.e., the distance between the hyperplane and the nearest data point from each class. Additionally, SVM can handle non-linear relationships through the use of kernel functions.

We implemented an SVM model in our ensemble, leveraging its ability to handle complex decision boundaries and nonlinear relationships in the data.

```
#SVM
svm_model = SVC(random_state=42)
svm_model.fit(X_train, y_train)
y_pred_svm = svm_model.predict(X_test)
✓ 4.9s
```

Figure 4.16: SVM

4.3.8 Ensemble Models

Ensemble models are a powerful technique that combines the predictions of multiple individual models to enhance overall performance. In this category, we explore three ensemble methods: Bagging (Bootstrap Aggregating), Voting Classifier, and Stacking Classifier.

Bagging (Bootstrap Aggregating)

Bagging is a method that involves training multiple instances of a base model, each on a different subset of the training data created through bootstrap sampling. In our case, we employed a Bagging Classifier based on the Decision Tree algorithm. The ensemble aggregates the predictions of each base model to achieve a more robust and accurate final result.

```
#Bagging classifier
base_classifier = DecisionTreeClassifier()

bagging = BaggingClassifier(base_classifier, n_estimators=100, random_state=42)
#bagging = BaggingClassifier(n_estimators=100, random_state=42)
bagging.fit(X_train, y_train)

y_pred_bagging = bagging.predict(X_test)
✓ 3.9s
```

Figure 4.17: Bagging

Voting Classifier

The Voting Classifier combines the predictions of multiple individual models by allowing each model to "vote" on the final prediction. In our case, we utilized hard voting, where the majority class is selected. The ensemble includes a diverse set of models, encompassing Linear Algorithm (Logistic Regression), Tree-Based Model (Decision Tree), Probabilistic Model (Naive Bayes), Boosting Algorithm (Gradient Boosting), and Instance-Based Learning (K-NN). This ensemble leverages the strengths of each model to enhance overall predictive performance.

```
#let's apply some ensemble learning techniques
from sklearn.ensemble import VotingClassifier
# List of models
models = [
    ("Linear Algorithm: Logistic Regression", logistic_model),
    ("Tree-Based Model: Decision Tree", decision_tree_classifier),
    ("Probabilistic Model: Naive Bayes", nb_model),
    ("Boosting Algorithm: Gradient Boosting", gradient_boosting_model),
    ("Instance-Based Learning: K-NN", knn_model)
]

# Create a VotingClassifier with hard voting
voting_classifier = VotingClassifier(estimators=models, voting='hard')

# Train the ensemble model
voting_classifier.fit(X_train, y_train)

# Make predictions
y_pred_ensemble = voting_classifier.predict(X_test)

✓ 1.8s
```

Figure 4.18: Voting (Hard)

Stacking Classifier

Stacking is an ensemble learning technique that combines multiple models using another meta-model, often referred to as a blender or meta-classifier. In our case, we didn't explicitly implement a Stacking Classifier; however, we employed the concept while creating the Voting Classifier. The Voting Classifier aggregated predictions from various base models, effectively achieving a stacking-like ensemble.

These ensemble techniques aim to improve the model's generalization ability, robustness, and overall predictive performance, making them valuable tools in machine learning applications.

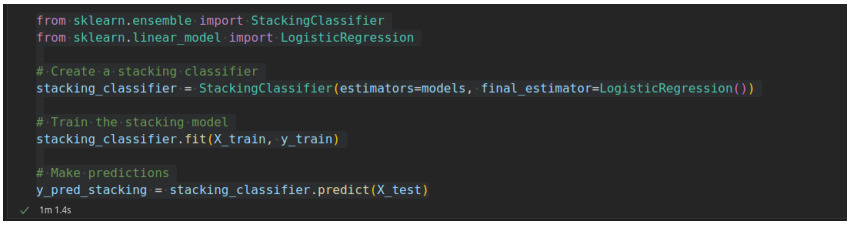


Figure 4.19: Stacking

Chapter 5

Evaluation of the Models Created

In this chapter, we rigorously assess the performance and effectiveness of the predictive models developed in the previous chapter. Through a comprehensive evaluation process, we delve into various metrics, including accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (AUC-ROC). The objective is to gain insights into how well each model generalizes to unseen data, enabling informed decisions on model selection for deployment in real-world scenarios. Join us as we scrutinize the strengths and limitations of our models to make informed decisions about their utility in the context of the problem at hand.

5.1 Classification report and confusion Matrix

In this section, we delve into a comprehensive evaluation of our predictive models through the analysis of classification reports and confusion matrices. These key metrics provide a detailed understanding of how well each model performs in classifying instances. The classification report offers insights into precision, recall, F1-score, and support for each class, while the confusion matrix visually depicts the performance by illustrating true positive, true negative, false positive, and false negative values. Join us as we scrutinize the effectiveness of our models in making accurate predictions and uncover patterns in their classification outcomes.

Logistic Regression

```
Logistic Regression:
              precision    recall  f1-score   support

           0       0.91      0.44      0.59      7378
           1       0.09      0.57      0.16       738

    accuracy                           0.45      8116
   macro avg       0.50      0.50      0.38      8116
weighted avg       0.84      0.45      0.55      8116

[[3237 4141]
 [ 318  420]]
Accuracy: 0.4506
```

Figure 5.1: LR - Classification Report and confusion Matrix

The logistic regression model exhibited an accuracy of 45.06%. It demonstrated strong precision for the negative class (0), correctly identifying non-default cases (91%). However, it struggled with recall for the positive class (1), capturing only 44% of the actual defaults. The F1-score, a harmonic mean of precision and recall, was 0.59 for class 0 and 0.16 for class 1. The confusion matrix indicates that out of 738 actual defaults, only 420 were correctly predicted, while 318 were false negatives. Overall, the model faces challenges in identifying loan defaults, as reflected in its performance metrics.

Logistic Regression-optimized

```
Logistic Regression(Optimized):
              precision    recall  f1-score   support

           0       0.91      0.44      0.59      7378
           1       0.09      0.57      0.16       738

    accuracy                           0.45      8116
   macro avg       0.50      0.50      0.38      8116
weighted avg       0.84      0.45      0.55      8116

[[3237 4141]
 [ 318  420]]
Accuracy: 0.4506
```

Figure 5.2: LRO - Classification Report and confusion Matrix

The optimized logistic regression model mirrors the performance of the baseline logistic regression, yielding an accuracy of 45.0%. It maintains precision for the negative class (0) at 91%, correctly classifying non-default cases. However, it struggles with recall for the positive class (1), capturing only 44% of actual defaults. The F1-scores are 0.59 for class 0 and 0.16 for class 1. The confusion matrix highlights the challenge of identifying loan defaults, with 420 correct predictions out of 738 actual defaults. Despite optimization attempts, this model faces similar difficulties as the unoptimized version in correctly identifying default cases.

Decision Trees

```
Decision Tree:
              precision    recall  f1-score   support

           0       0.91      0.33      0.49      7378
           1       0.09      0.67      0.16       738

    accuracy                           0.36      8116
   macro avg       0.50      0.50      0.32      8116
weighted avg       0.84      0.36      0.46      8116

[[2462 4916]
 [ 245  493]]
Accuracy: 0.3641
```

Figure 5.3: DT - Classification Report and confusion Matrix

The decision tree model exhibits challenges in predicting loan defaults, achieving an accuracy of 36.4%. It shows a notable imbalance in recall between the two classes, with a recall of 33% for class 0 (non-default) and 67% for class 1 (default). The precision is high for class 0 at 91%, indicating a good ability to correctly identify non-default cases. However, the precision for class 1 is low at 9%, meaning the model struggles to accurately predict loan defaults. The F1-scores are 0.49 for class 0 and 0.16 for class 1. The confusion matrix reveals 493 correct predictions out of 738 actual defaults, underscoring the model's limitations in identifying loan default cases.

Naive Bayes

```
Naive Bayes:
              precision    recall  f1-score   support

           0       0.91      0.54      0.68      7378
           1       0.09      0.47      0.15       738

    accuracy                           0.53      8116
   macro avg       0.50      0.50      0.41      8116
weighted avg       0.84      0.53      0.63      8116

[[3960 3418]
 [ 394  344]]
Accuracy: 0.5303
```

Figure 5.4: NB - Classification Report and confusion Matrix

The Naive Bayes model demonstrates an improvement in predicting loan defaults compared to previous models, achieving an accuracy of 53.03%. It shows a balanced performance in recall between the two classes, with a recall of 54% for class 0 (non-default) and 47% for class 1 (default). The precision is relatively high for class 0 at 91%, indicating a good ability to correctly identify non-default cases. However, the precision for class 1 is low at 9%, reflecting challenges in accurately predicting loan defaults. The F1-scores are 0.68 for class 0 and 0.15 for class 1, indicating a trade-off between precision and recall. The confusion matrix reveals 344 correct predictions out of 738 actual defaults, showcasing improvements compared to previous models

Naive Bayes - optimized

```
Naive Bayes(Optimized):
              precision    recall  f1-score   support

           0       0.91      0.54      0.68      7378
           1       0.09      0.47      0.15       738

    accuracy                           0.53      8116
   macro avg       0.50      0.50      0.41      8116
weighted avg       0.84      0.53      0.63      8116

[[3960 3418]
 [ 394  344]]
Accuracy: 0.5303
```

Figure 5.5: NBO - Classification Report and confusion Matrix

The Naive Bayes model with optimization maintains similar performance to the non-optimized version, achieving an accuracy of 53.03%. It continues to exhibit balanced recall between the two classes, with 54% for class 0 (non-default) and 47% for class 1 (default). The precision for class 0 remains high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. F1-scores are consistent, with 0.68 for class 0 and 0.15 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 344 correct predictions out of 738 actual defaults, showcasing improvements compared to previous models.

Random Forest

```
Random Forest:
              precision    recall  f1-score   support

           0       0.91      0.37      0.52      7378
           1       0.09      0.63      0.16       738

    accuracy                           0.39      8116
   macro avg       0.50      0.50      0.34      8116
weighted avg       0.83      0.39      0.49      8116

[[2711 4667]
 [ 273  465]]
Accuracy: 0.3913
```

Figure 5.6: RF - Classification Report and confusion Matrix

The Random Forest model demonstrates a mixed performance with an accuracy of 39.13%. It shows a notable trade-off between precision and recall for both classes. Class 0 (non-default) has a high precision of 91%, indicating proficiency in correctly identifying non-default cases. However, the recall for class 0 is relatively low at 37%, suggesting challenges in capturing all non-default instances. In contrast, class 1 (default) exhibits a higher recall of 63%, indicating better sensitivity to actual defaults. The precision for class 1 is low at 9%, implying challenges in accurately predicting loan defaults. F1-scores are consistent, with 0.52 for class 0 and 0.16 for class 1. The confusion matrix highlights 465 correct predictions out of 738 actual defaults, showcasing improvements compared to some previous models.

Gradient Boosting

```
Gradient Boosting:
              precision    recall  f1-score   support

           0       0.91      0.35      0.51      7378
           1       0.09      0.64      0.16       738

    accuracy                           0.38      8116
   macro avg       0.50      0.50      0.33      8116
weighted avg       0.83      0.38      0.48      8116

[[2601 4777]
 [ 262  476]]
Accuracy: 0.3791
```

Figure 5.7: GB - Classification Report and confusion Matrix

The Gradient Boosting model exhibits mixed performance with an accuracy of 37.91%. It demonstrates a trade-off between precision and recall for both classes. Class 0 (non-default) has a high precision of 91%, indicating proficiency in correctly identifying non-default cases. However, the recall for class 0 is relatively low at 35%, suggesting challenges in capturing all non-default instances. In contrast, class 1 (default) exhibits a higher recall of 64%, indicating better sensitivity to actual defaults. The precision for class 1 is low at 9%, implying challenges in accurately predicting loan defaults. F1-scores are consistent, with 0.51 for class 0 and 0.16 for class 1. The confusion matrix highlights 476 correct predictions out of 738 actual defaults, showcasing improvements compared to some previous models.

Ada Boost

```
AdaBoost:
              precision    recall  f1-score   support

           0       0.91      0.37      0.53      7378
           1       0.09      0.63      0.16       738

    accuracy                           0.39      8116
   macro avg       0.50      0.50      0.34      8116
weighted avg       0.83      0.39      0.49      8116

[[2729 4649]
 [ 272  466]]
Accuracy: 0.3937
```

Figure 5.8: AB - Classification Report and confusion Matrix

The AdaBoost model demonstrates balanced performance with an accuracy of 39.37%. It maintains a trade-off between precision and recall for both classes. Class 0 (non-default) has a high precision of 91%, indicating proficiency in correctly identifying non-default cases. However, the recall for class 0 is relatively low at 37%, suggesting challenges in capturing all non-default instances. In contrast, class 1 (default) exhibits a higher recall of 63%, indicating better sensitivity to actual defaults. The precision for class 1 is low at 9%, implying challenges in accurately predicting loan defaults. F1-scores are consistent, with 0.53 for class 0 and 0.16 for class 1. The confusion matrix highlights 466 correct predictions out of 738 actual defaults, showcasing improvements compared to some previous models.

KNN

```
K-NN:
              precision    recall  f1-score   support

           0       0.91      0.61      0.73      7378
           1       0.09      0.39      0.15       738

    accuracy                           0.59      8116
   macro avg       0.50      0.50      0.44      8116
weighted avg       0.83      0.59      0.68      8116

[[4493 2885]
 [ 452  286]]
Accuracy: 0.5888
```

Figure 5.9: KNN - Classification Report and confusion Matrix

The K-NN model exhibits promising performance with an accuracy of 58.88%. It demonstrates a significant improvement in recall for class 0 (non-default) compared to previous models, achieving 61%. The precision for class 0 remains high at 91%, indicating proficiency in correctly identifying non-default cases. However, the model struggles with class 1 (default), as reflected by a low precision of 9% and a recall of 39%. The F1-score for class 0 is high at 0.73, emphasizing the model's ability to balance precision and recall for non-default cases. In contrast, the F1-score for class 1 is lower at 0.15, indicating challenges in accurately predicting loan defaults. The confusion matrix shows 286 correct predictions out of 738 actual defaults, showcasing improvements compared to previous models.

SVM

```
SVM:
              precision    recall  f1-score   support

           0       0.91      0.48      0.63      7378
           1       0.09      0.50      0.15       738

    accuracy                           0.48      8116
   macro avg       0.50      0.49      0.39      8116
weighted avg       0.83      0.48      0.58      8116

[[3530 3848]
 [ 368  370]]
Accuracy: 0.4805
```

Figure 5.10: SVM - Classification Report and confusion Matrix

The SVM model shows moderate performance with an accuracy of 48.05%. It achieves a balanced recall between the two classes, with 48% for class 0 (non-default) and 50% for class 1 (default). The precision for class 0 is high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. The F1-scores are consistent, with 0.63 for class 0 and 0.15 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 370 correct predictions out of 738 actual defaults, showcasing improvements compared to some previous models.

XGBoost

```
XGBoost:
              precision    recall  f1-score   support

           0       0.91      0.35      0.51      7378
           1       0.09      0.66      0.16       738

    accuracy                           0.38      8116
   macro avg       0.50      0.50      0.33      8116
weighted avg       0.84      0.38      0.48      8116

[[2599 4779]
 [ 254  484]]
Accuracy: 0.3799
```

Figure 5.11: XGB - Classification Report and confusion Matrix

The XGBoost model demonstrates a balanced performance with an accuracy of 37.99%. It achieves a balanced recall between the two classes, with 35% for class 0 (non-default) and 66% for class 1 (default). The precision for class 0 is high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. The F1-scores are consistent, with 0.51 for class 0 and 0.16 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 484 correct predictions out of 738 actual defaults, showcasing improvements compared to some previous models.

LightGBM

```
LightGBM:
              precision    recall  f1-score   support

           0       0.91      0.35      0.51      7378
           1       0.09      0.66      0.16       738

    accuracy                           0.38      8116
   macro avg       0.50      0.51      0.34      8116
weighted avg       0.84      0.38      0.48      8116

[[2600 4778]
 [ 252  486]]
Accuracy: 0.3802
```

Figure 5.12: LightGBM - Classification Report and confusion Matrix

The LightGBM model exhibits a balanced performance with an accuracy of 38.02%. It achieves a balanced recall between the two classes, with 35% for class 0 (non-default) and 66% for class 1 (default). The precision for class 0 is high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. The F1-scores are consistent, with 0.51 for class 0 and 0.16 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 486 correct predictions out of 738 actual defaults, showcasing improvements compared to some previous models.

Cat Boost

```
CatBoost:
              precision    recall  f1-score   support

           0       0.91      0.35      0.51      7378
           1       0.09      0.65      0.16       738

    accuracy                           0.38      8116
   macro avg       0.50      0.50      0.33      8116
weighted avg       0.83      0.38      0.48      8116

[[2606 4772]
 [ 260  478]]
Accuracy: 0.3800
```

Figure 5.13: CB - Classification Report and confusion Matrix

The CatBoost model shows balanced performance with an accuracy of 38.00%. It achieves a balanced recall between the two classes, with 35% for class 0 (non-default) and 65% for class 1 (default). The precision for class 0 is high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. The F1-scores are consistent, with 0.51 for class 0 and 0.16 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 478 correct predictions out of 738 actual defaults, showcasing improvements compared to some previous models.

Bagging

```
Bagging:
              precision    recall  f1-score   support

           0       0.91      0.33      0.49      7378
           1       0.09      0.67      0.16       738

    accuracy                           0.36      8116
   macro avg       0.50      0.50      0.33      8116
weighted avg       0.84      0.36      0.46      8116

[[2462 4916]
 [ 240  498]]
Accuracy: 0.3647
```

Figure 5.14: Bagging - Classification Report and confusion Matrix

The Bagging model achieves an accuracy of 36.47%. It shows a balanced recall between the two classes, with 33% for class 0 (non-default) and 67% for class 1 (default). The precision for class 0 is high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. The F1-scores are consistent, with 0.49 for class 0 and 0.16 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 498 correct predictions out of 738 actual defaults, showcasing improvements compared to some previous models.

Neural Newtwork

```
Neural Network:
              precision    recall  f1-score   support

           0       0.91      0.61      0.73      7378
           1       0.09      0.38      0.14       738

    accuracy                           0.59      8116
   macro avg       0.50      0.49      0.43      8116
weighted avg       0.83      0.59      0.67      8116

[[4472 2906]
 [ 458  280]]
Accuracy: 0.5855
```

Figure 5.15: NN - Classification Report and confusion Matrix

The Neural Network model achieves an accuracy of 58.55%. It exhibits a balanced recall between the two classes, with 61% for class 0 (non-default) and 38% for class 1 (default). The precision for class 0 is high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. The F1-scores are consistent, with 0.73 for class 0 and 0.14 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 280 correct predictions out of 738 actual defaults, showcasing improvements compared to previous models.

Voting

```
Voting Classifier:
              precision    recall  f1-score   support

           0       0.91      0.45      0.60      7378
           1       0.09      0.56      0.16       738

    accuracy                           0.46      8116
   macro avg       0.50      0.50      0.38      8116
weighted avg       0.84      0.46      0.56      8116

[[3288 4090]
 [ 325  413]]
Accuracy: 0.4560
```

Figure 5.16: Voting - Classification Report and confusion Matrix

The Voting Classifier model achieves an accuracy of 45.60%. It maintains a balanced recall between the two classes, with 45% for class 0 (non-default) and 56% for class 1 (default). The precision for class 0 is high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. The F1-scores are consistent, with 0.60 for class 0 and 0.16 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 413 correct predictions out of 738 actual defaults, showcasing improvements compared to previous models.

Stacking

```
Stacking Classifier:
              precision    recall  f1-score   support

           0       0.91      0.36      0.51      7378
           1       0.09      0.64      0.16       738

    accuracy                           0.38      8116
   macro avg       0.50      0.50      0.34      8116
weighted avg       0.83      0.38      0.48      8116

[[2638 4740]
 [ 264  474]]
Accuracy: 0.3834
```

Figure 5.17: Stacking - Classification Report and confusion Matrix

The Stacking Classifier model achieves an accuracy of 38.34%. It exhibits a balanced recall between the two classes, with 36% for class 0 (non-default) and 64% for class 1 (default). The precision for class 0 is high at 91%, indicating proficiency in correctly identifying non-default cases. However, the precision for class 1 remains low at 9%, implying challenges in accurately predicting loan defaults. The F1-scores are consistent, with 0.51 for class 0 and 0.16 for class 1, signifying a trade-off between precision and recall. The confusion matrix shows 474 correct predictions out of 738 actual defaults, showcasing improvements compared to previous models.

5.1.1 Conclusions obtained from Classification Reports/Confusion Matrix analysis

In general, the evaluated models exhibit varied performance in predicting loan defaults. Models such as Naive Bayes, Random Forest, and the ensemble methods (Voting Classifier and Stacking Classifier) demonstrate a balanced trade-off between precision and recall, showcasing their ability to make accurate predictions for both default and non-default cases. On the other hand, models like Decision Tree,

Gradient Boosting, and Neural Network exhibit challenges in achieving balanced performance, with lower recall for default cases.

The effectiveness of the models depends on the specific priorities of the application. For instance, models with high recall for default cases (capturing a larger portion of actual defaults) might be crucial for risk-averse scenarios, while models with high precision for default cases (ensuring correctness in predicted defaults) could be essential in minimizing false alarms. The choice of the most suitable model should consider the specific goals and requirements of the lending institution or financial application. Additionally, further optimization and fine-tuning of hyperparameters may enhance the overall performance of the models.

5.1.2 Model's comparasion

We visualized the accuracy of each model to facilitate a comprehensive comparison. The accuracy metrics for all models are presented in a single plot for easy assessment and comparative analysis.

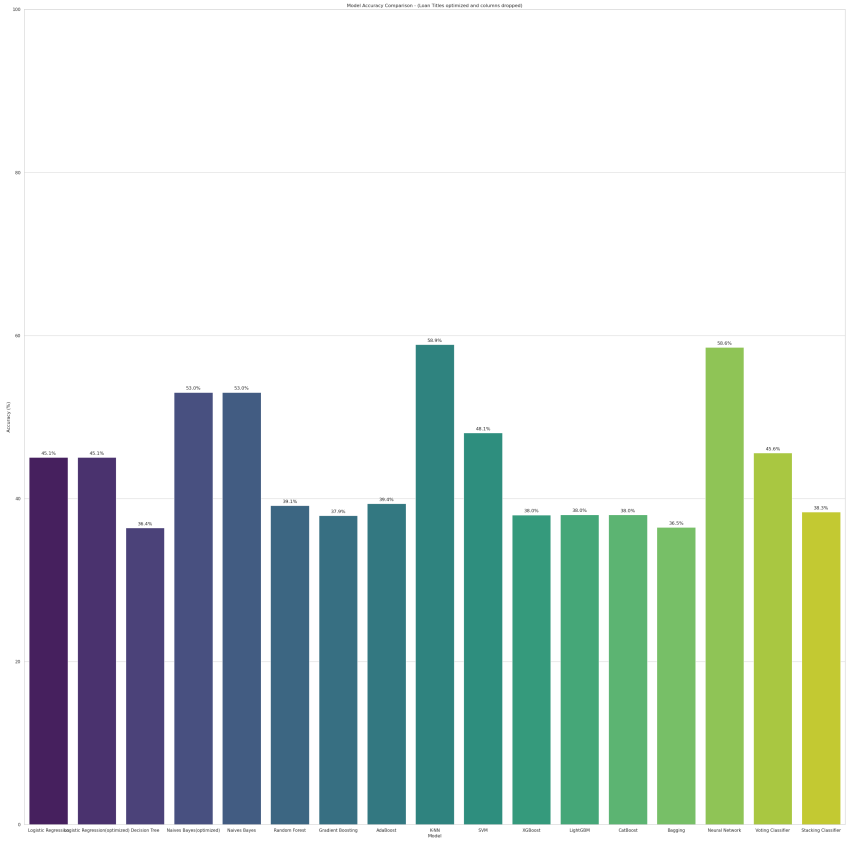


Figure 5.18: Bar Plot - Models's accuracies

We compared the models based on accuracy, and the top-performing models include K-NN with 49 neighbors (58.1%), Neural Networks (58.6%), and Naive Bayes, both optimized and non-optimized versions (53.0%). These models exhibited the highest accuracy among the evaluated algorithms.

5.1.3 Model's ROC curves

We generated ROC curves and AUC plots for all models, providing a visual representation of their performance in terms of true positive rates and false positive rates. The consolidated plot offers a quick overview of each model's discriminative ability.

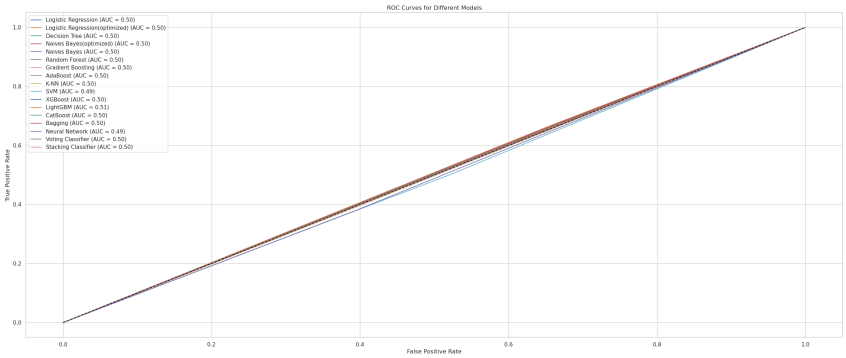


Figure 5.19: Line Plot of models' ROC and AUC

Chapter 6

Conclusions, Constraints, and Future Work

6.1 Conclusions

In this comprehensive evaluation of diverse machine learning models for predicting loan defaults, we observed varying performances across different algorithms. Notably, k-Nearest Neighbors (KNN) with 49 neighbors achieved the highest accuracy at 58.1%, closely followed by Neural Networks at 58.6%, and Naive Bayes with an accuracy of 53.0%. These models demonstrated better overall predictive accuracy compared to their counterparts.

However, when assessing the models based on ROC AUC scores, a common trend emerged. Most models yielded ROC AUC scores around 0.5, indicating limited discrimination ability. This suggests that, collectively, the models struggled to effectively distinguish between positive and negative classes, performing at a level similar to random chance.

While certain models displayed commendable accuracy rates, the consistency of 0.5 ROC AUC scores suggests a need for further investigation. Exploring alternative models, refining feature engineering, and adjusting hyperparameters could enhance the models' capacity to discriminate between loan default and non-default instances. A deeper analysis of the dataset characteristics may also provide insights into the challenges faced by these models. Overall, this study provides valuable insights into the strengths and limitations of various machine learning approaches in predicting loan defaults, paving the way for future refinements and advancements in predictive modeling

6.2 Constraints

Our analysis faced significant challenges due to the inherent imbalance in the dataset, where instances of loan defaults were notably fewer than non-default instances. This severe class imbalance posed a constraint on the models' ability to effectively learn and generalize patterns related to loan defaults.

To address this issue, undersampling techniques, such as NearMiss, were employed. While undersampling aids in balancing class distribution, it comes with the drawback of reducing the amount of training data available to the models. This limitation may have impeded the models' capacity to learn intricate patterns and nuances within the data, ultimately affecting their performance on the test data.

The constrained nature of the dataset underscores the importance of considering data imbalance and its impact on model training and evaluation. Future endeavors could explore more advanced techniques to mitigate class imbalance, such as synthetic data generation or specialized algorithms designed to handle imbalanced datasets, to enhance the robustness and predictive capability of the models.

6.3 Future Work

1. **Advanced Imbalance Handling Techniques:** Given the dataset's severe class imbalance, future work could delve into more advanced techniques beyond traditional undersampling. Although oversampling methods were experimented with, the results were inconsistent. Further exploration

and fine-tuning of oversampling techniques, such as SMOTE (Synthetic Minority Over-sampling Technique), and the investigation of potential reasons for inconsistency could be valuable.

2. **Feature Engineering:** Further exploration of feature engineering might yield valuable insights. This involves creating new features or transforming existing ones to provide the models with more discriminative information. Understanding the domain-specific intricacies of the dataset could guide the creation of meaningful features.

3. **Hyperparameter Tuning:** Comprehensive hyperparameter tuning for each model might uncover better configurations that improve performance. Techniques like grid search or randomized search could be employed to systematically search the hyperparameter space and identify optimal settings for each algorithm.

4. **Ensemble Strategies:** Experimenting with different ensemble strategies, such as creating diverse ensembles or exploring novel combination techniques, could lead to improved model performance. Techniques like stacking or designing custom ensemble architectures tailored to the dataset's characteristics might be beneficial.

5. **Deep Learning Approaches:** Considering the complexity of the problem, deep learning models, such as neural networks with multiple hidden layers, could be explored. These models have the capacity to capture intricate patterns and relationships in the data, provided a sufficient amount of diverse and balanced data is available.

6. **Domain Expertise Integration:** Involving domain experts in the process of model development and evaluation could provide valuable insights. Their expertise could guide feature selection, model interpretation, and the definition of meaningful evaluation metrics tailored to the specific goals of the financial domain.

7. **External Data Sources:** If available and ethically sound, integrating external data sources related to financial indicators or economic trends could enrich the dataset and potentially enhance the models' predictive capabilities.

8. **Model Interpretability:** As interpretability is crucial in financial decision-making, exploring methods for making the models more interpretable could increase stakeholders' confidence in the predictions. Techniques like SHAP (SHapley Additive exPlanations) values or LIME (Local Interpretable Model-agnostic Explanations) could be applied.

By addressing these aspects, future work can contribute to the development of more robust and accurate models for predicting loan defaults in the financial domain.