SBA Loan Approval Analysis & Prediction

Team: Dhwanil Ranpura and Santhosh Ravindrabharathy

Introduction

1. **Background of the Data:** This data set is from the U.S. Small Business Administration (SBA) and provides historical data from 1987 through 2014, containing 27 variables and 899,164 observations. Each observation represents a loan that was guaranteed to some degree by the SBA. Included is a variable [MIS_Status] which indicates if the loan was paid in full (PIF) or defaulted/charged off. (ChgOff).

2. **Goal:** The objective of this project is to classify SBA-approved loans into "Paid In Full" and "Charged-Off" categories through machine learning, aiming to provide banks with a more precise framework for risk assessment to optimize the loan approval process and enhance the overall efficiency of the small business credit market. Specifically, we want to assist the Small Business Administration (SBA) Loan Guarantee Program and banks in accurately pinpointing which borrowers are at a higher risk of default. If time-permitted, we want to analyze the main factors influencing the default rate of guaranteed loans, and assess the program's effectiveness in creating jobs and promoting social benefits.

Method

1. Data Cleaning

Identify and correct errors and inconsistencies in the data to improve its quality. This includes:

- **Removing Duplicates:** Check for and eliminate any duplicate records that may skew the model training.
- **Handling Missing Values:** Identify columns with missing values and decide on a strategy for handling them (e.g., imputation, deletion).
- **Converting Data Types:** Ensure that each column is of the correct data type (e.g., numeric, categorical). Implement data type conversion when necessary.
- **Addressing Outliers:** Looking for any anomalies in the data that might need to be addressed.

2. Exploratory Data Analysis (EDA)

Begin the EDA by exploring the distribution and relationships of factors that include loan sizes and industry types through visualizations and statistical summaries. This will help in identifying the main drivers of loan risk, outliers, and assumptions to be made, hence informing hypothesis and model development.

3. Feature Engineering

One-hot encoding changes categorical data into numerical, makes new features based on assumptions and characteristics of the data, standardizes the values, and explores feature interaction to enhance predictive performance. We used the chi-squared test for feature selection by selecting the 10 features that are most closely related with our label. This preprocesses the data for machine learning, hence improving the prediction accuracy for loan approvals.

4. Model Selection and Training

Such a classification problem, where the goal is to predict a binary outcome-for example, a loan will be either Paid in Full or Charged-Off-several machine learning models could be suitable. Some of the models we used for this project are as follows:

- **Random Forest:** Random Forests are particularly effective for predicting borrower defaults because they can process complex datasets with many features, naturally handle imbalanced data, and maintain high accuracy by averaging multiple decision trees to reduce overfitting.
- **Logistic Regression:** Logistic regression estimates the probability that a given input point belongs to a certain class. It could effectively handle binary classification of loan Paid In Full or Charged-Off.
- **K- Nearest Neighbours:** It is a supervised algorithm for classification and regression, using distance metrics like Euclidean to find the k closest data points. It can be implemented in Python with scikit-learn.
- **Decision Tree:** It is a supervised machine learning algorithm used for classification and regression tasks. It works by splitting the dataset into branches based on feature conditions, forming a tree-like structure.

5. Model Evaluation and Tuning

Evaluate model performance using appropriate metrics (e.g., accuracy, precision, recall, F1 score) and employ techniques such as cross-validation for more robust evaluation. Perform hyperparameter tuning to optimize the model's performance.

Description of the Dataset

1. Description

This dataset from the U.S. Small Business Administration (SBA) comprises 899,164 records and 27 columns, totaling 179.43MB, detailing various aspects of SBA loans.

These are the columns from the dataset:

VARIABLE NAME	DESCRIPTION
LoanNr_ChkDgt	Identifier primary key
Name	Borrower Name
City	Borrower City
State	Borrower State
Zip	Borrower Zip
Bank	Bank Name
Bank State	Bank State
NAICS	North American Industry Classification System code
Approval Date	Date SBA Commitment Issued
ApprovalFY	Fiscal Year of Commitment
Term	Loan term in months
NoEmp	Number of Business Employees
NewExist	1 = Existing Business, 2 = New Business
CreateJob	Number of jobs created
Retained Job	Number of jobs retained
FranchiseCode	Franchise Code 00000 or 00001 = No Franchise
UrbanRural	1= Urban, 2= Rural, 0 = Undefined
RevLineCr	Revolving Line of Credit: Y = Yes

LowDoc	LowDoc Loan Program: Y = Yes, N = No
ChgOffDate	The date when a loan is declared to be in default
DisbursementDate	Disbursement Date
DisbursementGross	Amount Disbursed
BalanceGross	Gross amount outstanding
MIS_Status	Loan Status, “CHGOFF” (charged off or defaulted) or “P I F” (paid in full)
ChgOffPrintGr	Charged-off Amount
GrApprv	Gross Amount of Loan Approved by Bank
SBA_Apprv	SBA’s Guaranteed Amount of Approved Loan

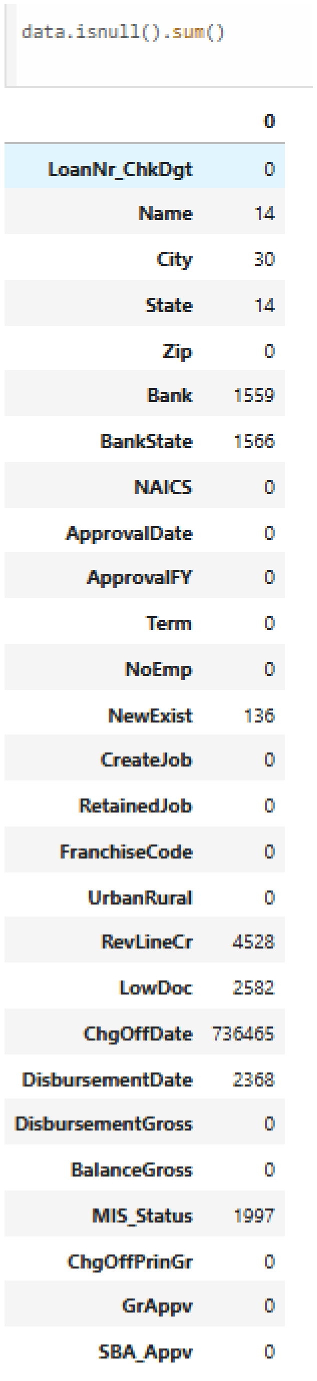
It is worth noting that some of the features are specialized in the loans within this dataset. For example, RevLineCr(Revolving line of credit) refers to a credit arrangement where the borrower can access funds up to a pre-approved limit, repay, and borrow again. LowDoc (LowDoc Loan Program) - Low Documentation Loan Program; streamlines and expedites the loan application process for smaller loan amounts, making it easier for small businesses to gain access to credit. NAICS - North American Industry Classification System code. This is an industry code system used to classify North American businesses.

Analysis

1. Data Cleaning and Feature Engineering

(1) Removing Duplicates and Handling Missing Values

Our project begins with data cleaning, which involves addressing missing values and data type conversion. Since there's no duplicate in our dataset, we directly proceed to the missing value part. **Figure below** shows the information about our missing values.



```
data.isnull().sum()
```

```
                       0
LoanNr_ChkDgt          0
Name                  14
City                  30
State                 14
Zip                    0
Bank                1559
BankState           1566
NAICS                  0
ApprovalDate           0
ApprovalFY             0
Term                   0
NoEmp                  0
NewExist             136
CreateJob              0
RetainedJob            0
FranchiseCode          0
UrbanRural             0
RevLineCr           4528
LowDoc              2582
ChgOffDate        736465
DisbursementDate    2368
DisbursementGross      0
BalanceGross           0
MIS_Status          1997
ChgOffPrinGr           0
GrAppv                 0
SBA_Appv               0
```

In the label **"Mis_Status"**, we drop those missing values. Since features **"Name", "City", "State", "Bank", "BankState", "NewExist", "RevLineCr", "LowDoc", "DisbursementDate"** have relatively few of these missing values with regards to the sample size of the dataset, we choose to remove the rows that include missing values to preserve the integrity of the data.

For sure, one of the emphasis is on the column called "ChgOffDate": the date when a borrower fails to pay and is formally declared in default. Compared against the number of blanks in "ChgOffDate" was the amount of loans classified as "PIF" - or Paid in Full. These latter two were quite comparable:. Having seen this similarity, we might infer that missing values are predominant because of fully repaid loans on "ChgOffDate", since the date is immaterial for loans that never defaulted. Using this clue, due to the high correlation of missing values on "ChgOffDate" with fully repaid loans, we decided this column can be excluded to shorten our dataset to focus our analysis on the more impact-providing variables only. After the previous steps mentioned, all the missing values in our dataset have been properly handled.

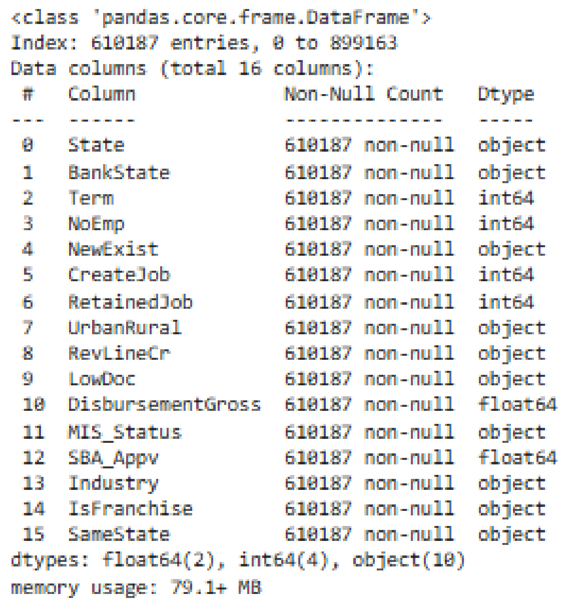
(2) Drop Features That Are Apparently Irrelevant to Our Prediction

Next, we applied some feature selection. We dropped the features that are apparently irrelevant to our prediction, such as "LoanNr_ChkDgt" and "Name".

We then looked at columns like "ChgOffPrintGr" for charged-off amounts and "BalanceGross" for gross amounts outstanding. We realized that these features would only materialize when a company has been marked as default, or in other words, the loan status is charged off, which may be a potential leakage of data if included in our model. To maintain the integrity of our predictive analysis and prevent overfitting, we chose to exclude these variables.

Also, to make sure that our model's predictive power is not biased by temporal factors and can reliably forecast future defaults, we removed time-related attributes such as "ApprovalDate," "ApprovalFY," and "DisbursementDate." This is in line with our objective of developing a time-agnostic model on the likelihood of a company paying in full or charging off, independent of the influence of specific time periods or fiscal years.

To handle the geographic information for the company, we've decided to reduce the dimensionality of the data by maintaining only the 'State' attribute. Because of the presence of a large number of unique values, we are going to exclude 'City' and 'Zip' to avoid problems with too much categorical data representation. Similarly, we will exclude the 'Bank' feature and retain only the 'BankState'.



```
<class 'pandas.core.frame.DataFrame'>
Index: 610187 entries, 0 to 899163
Data columns (total 16 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   State              610187 non-null  object
 1   BankState          610187 non-null  object
 2   Term               610187 non-null  int64
 3   NoEmp              610187 non-null  int64
 4   NewExist           610187 non-null  object
 5   CreateJob          610187 non-null  int64
 6   RetainedJob        610187 non-null  int64
 7   UrbanRural         610187 non-null  object
 8   RevLineCr          610187 non-null  object
 9   LowDoc             610187 non-null  object
 10  DisbursementGross  610187 non-null  float64
 11  MIS_Status         610187 non-null  object
 12  SBA_Appv           610187 non-null  float64
 13  Industry           610187 non-null  object
 14  IsFranchise        610187 non-null  object
 15  SameState          610187 non-null  object
dtypes: float64(2), int64(4), object(10)
memory usage: 79.1+ MB
```

(3) Convert Data Types and Create New Columns

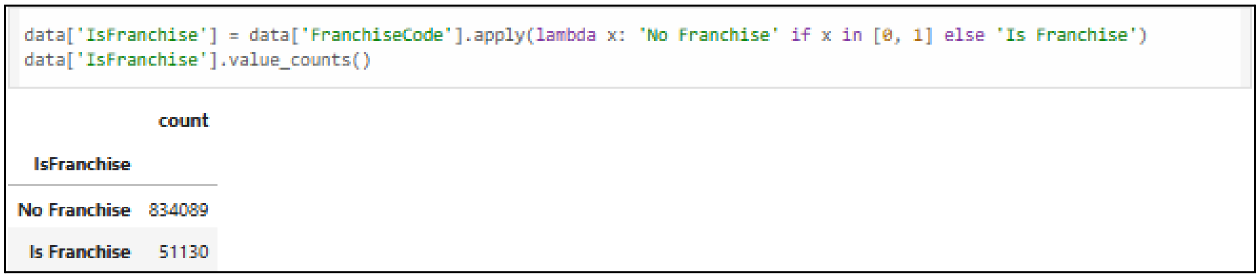
In the next step of data preprocessing, we changed the data type of certain attributes to string for better interpretability. Among these were the "NewExist" and "UrbanRural" features.

There seems to be some sort of overlap between "DisbursementGross" and "GrAppv". We kept the "DisbursementGross". Considering that the amount of the loan payment should be a float instead of a string, we used code to remove the "$" sign from the original string type and convert it into a float type.

For the "NAICS" codes, we created a new column called "Industry" that classifies businesses based on the two-digit North American Industry Classification System codes by descriptive industry sector names.

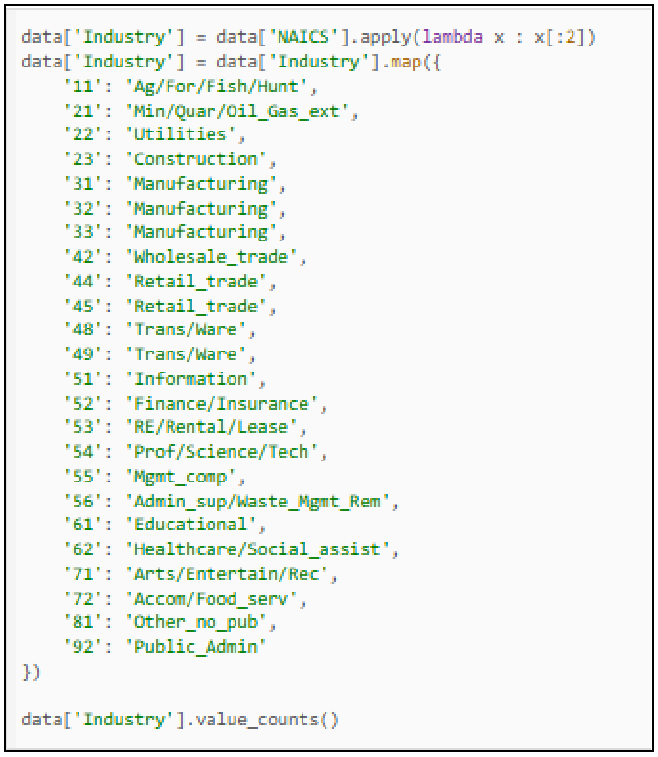
Regarding "FranchiseCode," we added a binary column called "IsFranchise" to distill the original information into an easy-to-handle form for the subsequent analysis steps.

To check whether the state of the borrower is the same as the bank's state, a new column will be introduced that indicates whether the borrower and the bank are in the same state.

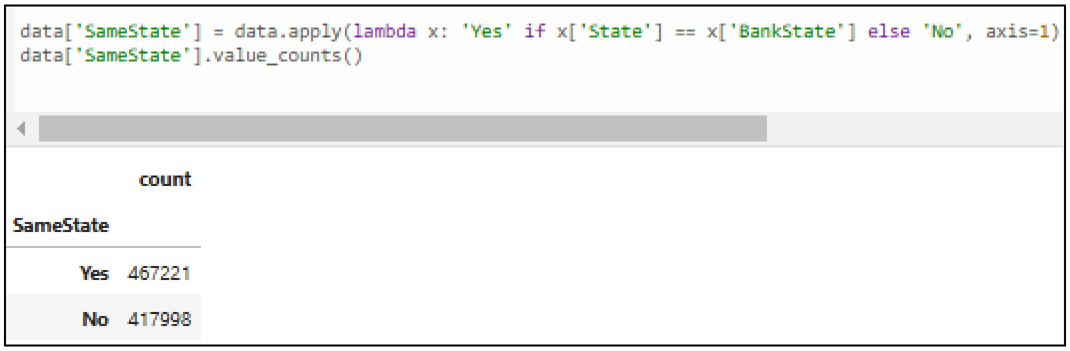


```
data['IsFranchise'] = data['FranchiseCode'].apply(lambda x: 'No Franchise' if x in [0, 1] else 'Is Franchise')
data['IsFranchise'].value_counts()
```

	count
IsFranchise	
No Franchise	834089
Is Franchise	51130



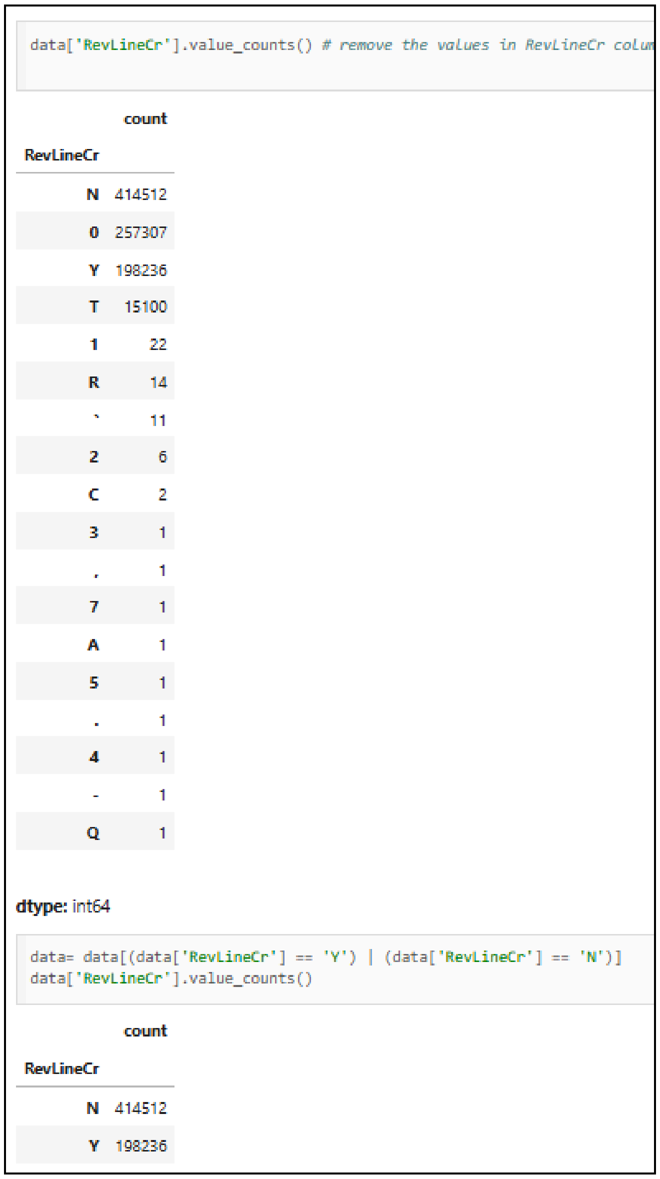
```
data['Industry'] = data['NAICS'].apply(lambda x : x[:2])
data['Industry'] = data['Industry'].map({
    '11': 'Ag/For/Fish/Hunt',
    '21': 'Min/Quar/Oil_Gas_ext',
    '22': 'Utilities',
    '23': 'Construction',
    '31': 'Manufacturing',
    '32': 'Manufacturing',
    '33': 'Manufacturing',
    '42': 'Wholesale_trade',
    '44': 'Retail_trade',
    '45': 'Retail_trade',
    '48': 'Trans/Ware',
    '49': 'Trans/Ware',
    '51': 'Information',
    '52': 'Finance/Insurance',
    '53': 'RE/Rental/Lease',
    '54': 'Prof/Science/Tech',
    '55': 'Mgmt_comp',
    '56': 'Admin_sup/Waste_Mgmt_Rem',
    '61': 'Educational',
    '62': 'Healthcare/Social_assist',
    '71': 'Arts/Entertain/Rec',
    '72': 'Accom/Food_serv',
    '81': 'Other_no_pub',
    '92': 'Public_Admin'
})

data['Industry'].value_counts()
```



```
data['SameState'] = data.apply(lambda x: 'Yes' if x['State'] == x['BankState'] else 'No', axis=1)
data['SameState'].value_counts()
```

	count
SameState	
Yes	467221
No	417998

The next step was the filtering out of as many irregular entries in both 'RevLineCr' and 'LowDoc'. We cleared all entries in the 'RevLineCr' column that did not map to the standard 'Y' and 'N', as can be seen in the before-and-after count of value frequencies. We also did a similar thing in the 'LowDoc' column by keeping only the 'Y' and 'N' entries and discarding the others. This will make sure that both columns are now filled with consistent binary indicators, which will most likely be more relevant for any further analysis-machine learning modeling or data visualization-while reducing noise and possible errors due to non-standard entries.

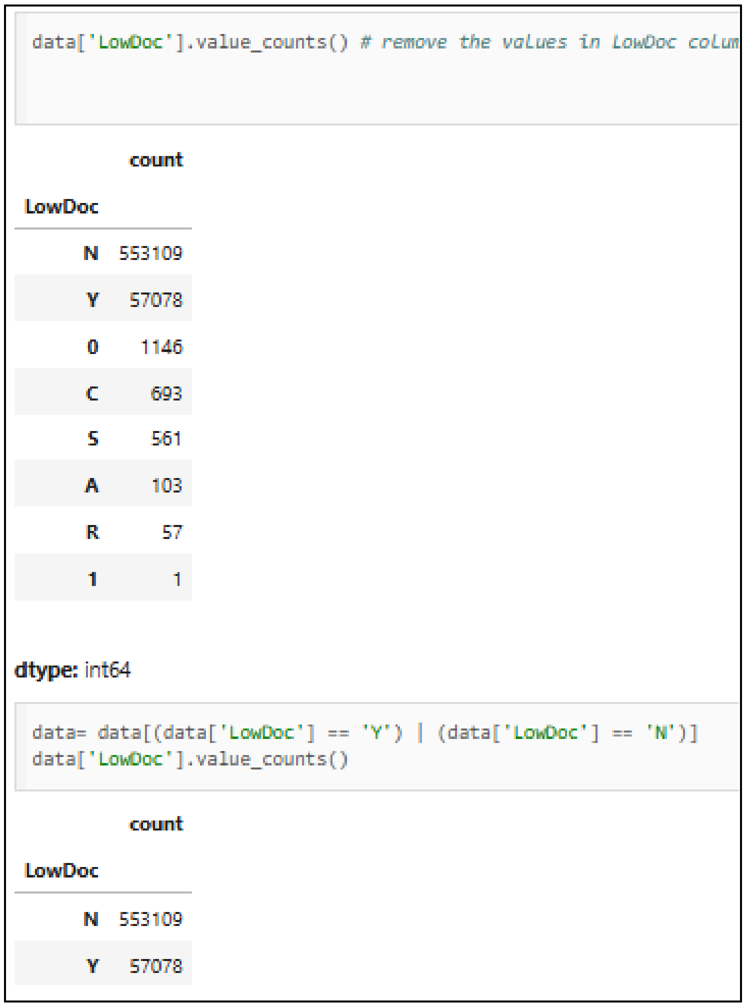


```
data['RevLineCr'].value_counts() # remove the values in RevLineCr colum
```

```
           count
RevLineCr
        N  414512
        0  257307
        Y  198236
        T   15100
        1      22
        R      14
        `      11
        2       6
        C       2
        3       1
        ,       1
        7       1
        A       1
        5       1
        .       1
        4       1
        -       1
        Q       1

dtype: int64
```

```
data= data[(data['RevLineCr'] == 'Y') | (data['RevLineCr'] == 'N')]
data['RevLineCr'].value_counts()
```

```
           count
RevLineCr
        N  414512
        Y  198236
```



```
data['LowDoc'].value_counts() # remove the values in LowDoc colum
```

```
        count
LowDoc
     N  553109
     Y   57078
     0    1146
     C     693
     S     561
     A     103
     R      57
     1       1

dtype: int64
```

```
data= data[(data['LowDoc'] == 'Y') | (data['LowDoc'] == 'N')]
data['LowDoc'].value_counts()
```

```
        count
LowDoc
     N  553109
     Y   57078
```

After cleaning the data and feature engineering, we process and fill missing values, delete the duplicates, standardize the data and filter the noise to a certain extent and obtain a relatively high-quality data set to improve the accuracy of the model. The processed dataset now has 610187 samples and 17 columns.

2. Exploratory Data Analysis (EDA)

By preliminary data processing, we got a general understanding of the data set and asked some questions:

- Does the higher the SBA guarantee mean the company has better credit and is more likely to repay the loan in full?
- How are companies applying for SBA loans geographically distributed, and what does that mean?
- How does the loan's term affect the payoff decision of the company?
- Will whether the location of the borrowing company and bank will be consistent or not would affect whether the company repays the loan in the end? How to explain
- The greater a company can create new jobs, then the greater the likelihood of it repaying?
- Does it take having a franchise business code to say whether the company will repay the loan or not? If yes, what advice does it give the banks while approving the loans?

(1) Univariate Analysis for Numeric Features

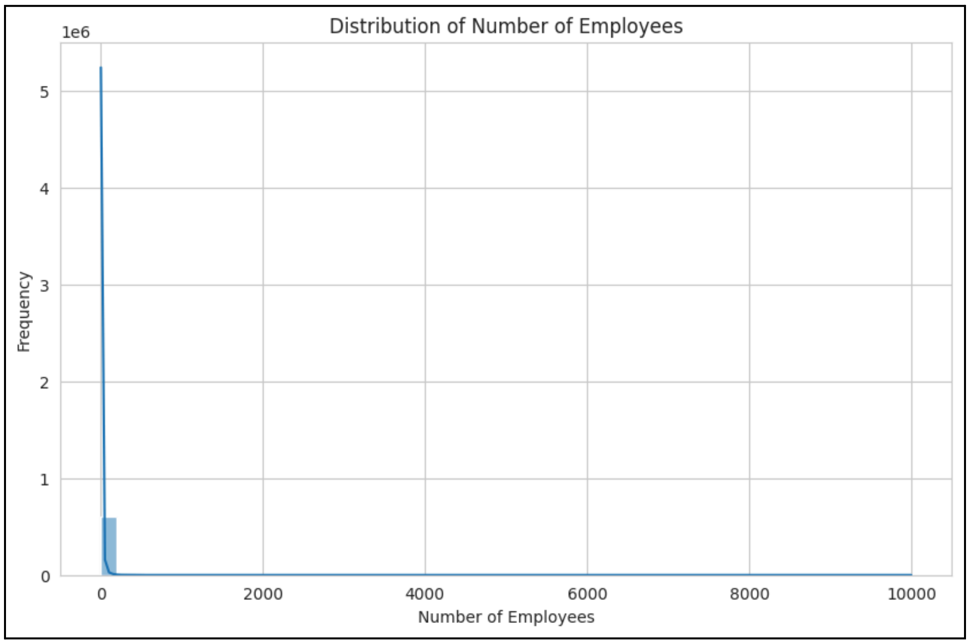


Figure 1

Figure 1 shows a left-skewed distribution of the number of employees. This corresponds to the reality because only a few large companies will apply for SBA loans. Firms with over 60 employees were filtered out as the project was focused on small companies. A revised histogram, **Figure 2**, shows a normal distribution.

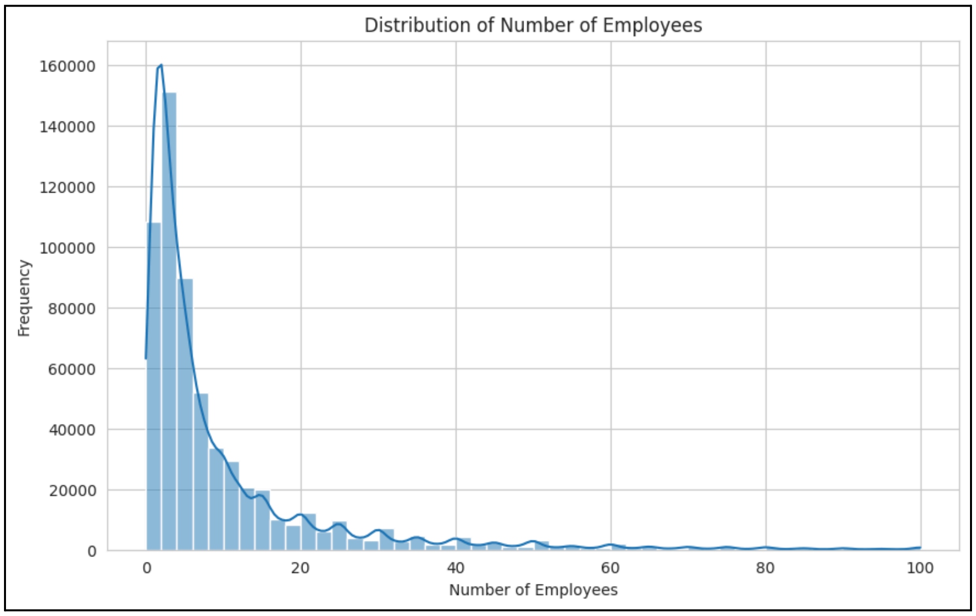


Figure 2

DISBURSEMENT GROSS

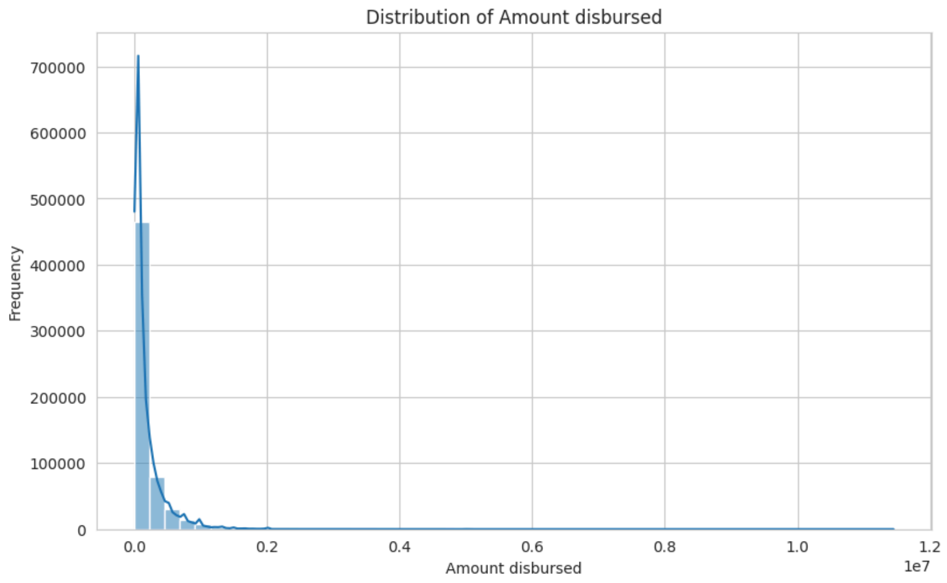
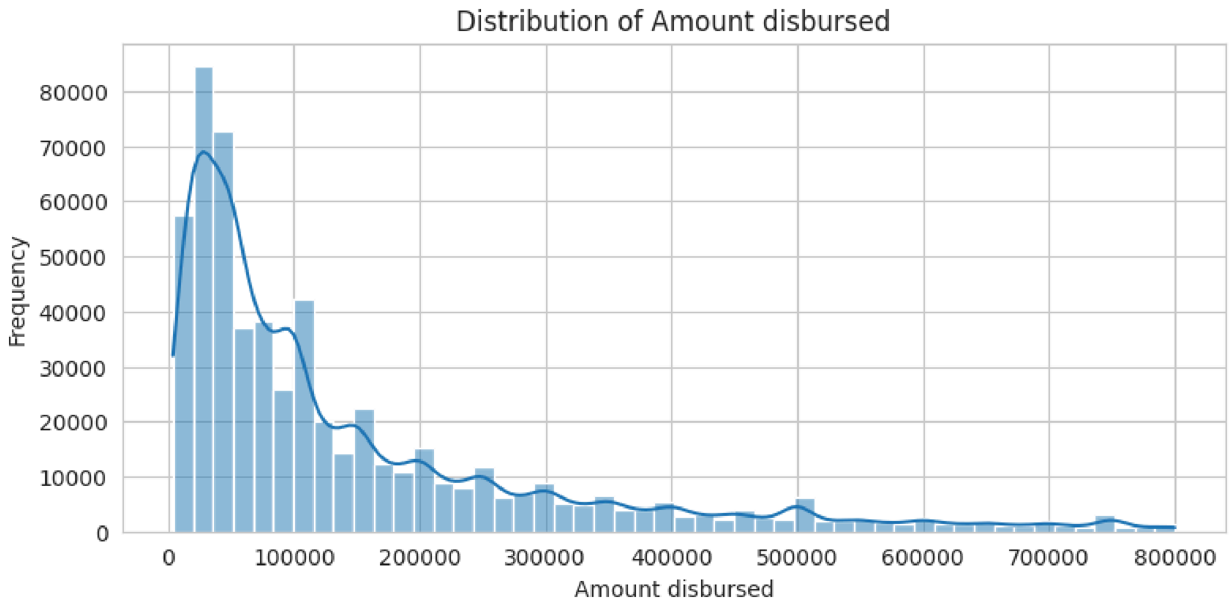


Figure 3

The distribution of "DisbursementGross" is shown in **Figure 3**. It is imbalanced, but that is realistic since some large companies also apply for SBA loans. To focus on small businesses, we filtered out loans over $1,000,000, assuming such a high amount is unlikely for small businesses. **Figure 4** shows the updated histogram.



Distribution of Amount disbursed
Frequency
Amount disbursed
0
10000
20000
30000
40000
50000
60000
70000
80000
100000
200000
300000
400000
500000
600000
700000
800000

Figure 4

(2) Univariate Analysis for Discrete Features

The dataset includes discrete features like **"State," "BankState," "NewExist," "SameState," "UrbanRural," "RevLineCr," "LowDoc," "Industry," and "IsFranchise,"** though not all are equally informative. Examining the label **"Mis_Status"** reveals a significant imbalance, with much more Paid-In-Full data than Charged-Off. This imbalance could skew model performance toward Paid-In-Full predictions, requiring strategies to address it during model training.

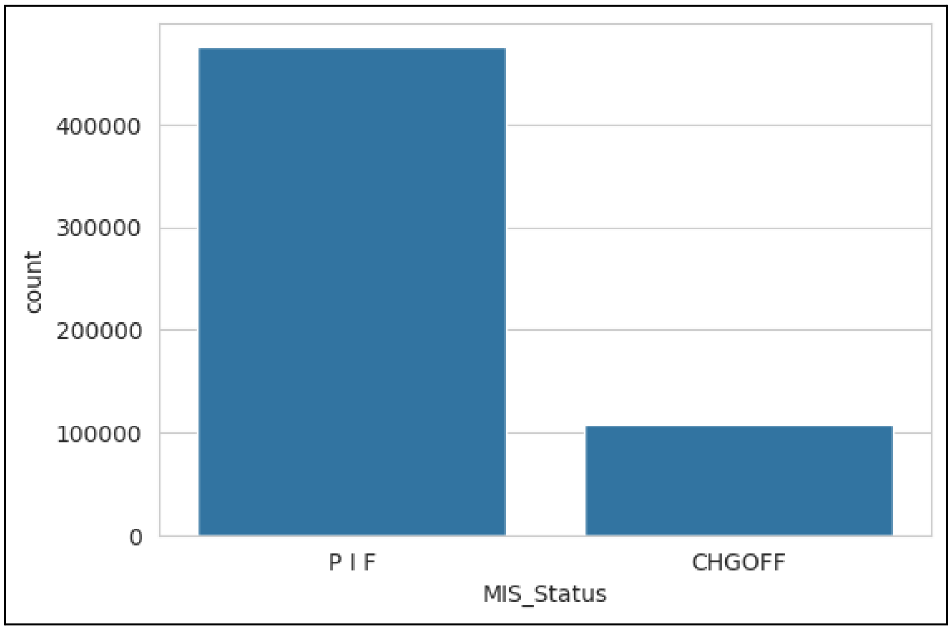


Figure 5

(3) Bivariate Analysis

Figure 6 suggests that longer-term loans are more likely to be paid in full, while shorter-term loans are more likely to be charged off.

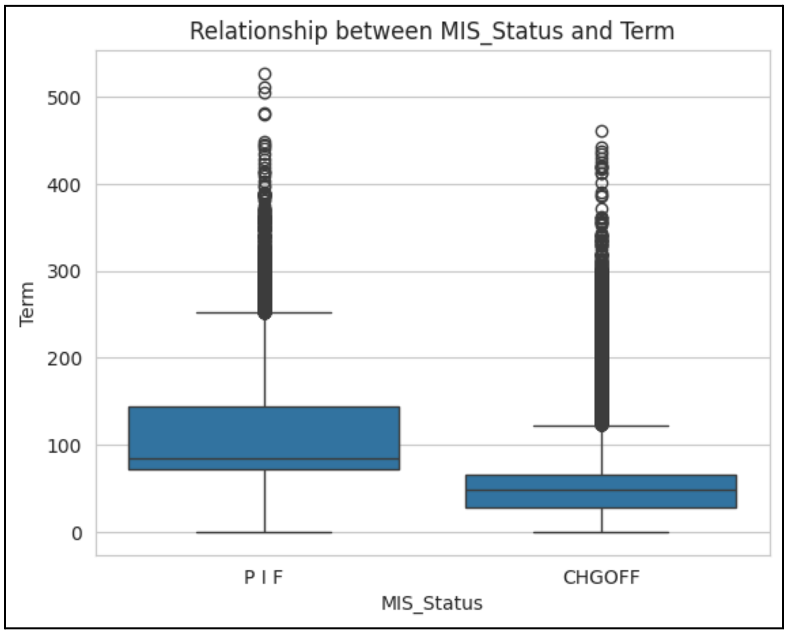


Figure 6

When we check the relationship between **“MIS_Status”** and **“DisbursementGross”**, we found that loans with higher disbursement amounts are more likely to be paid in full, whereas loans with lower disbursement amounts are more likely to be charged off (**figure 7**).

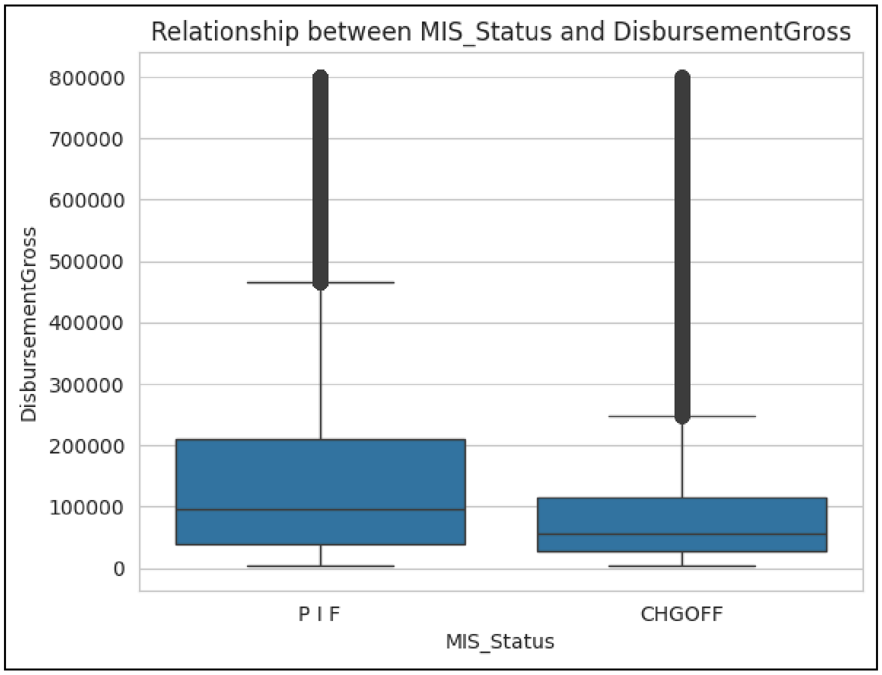
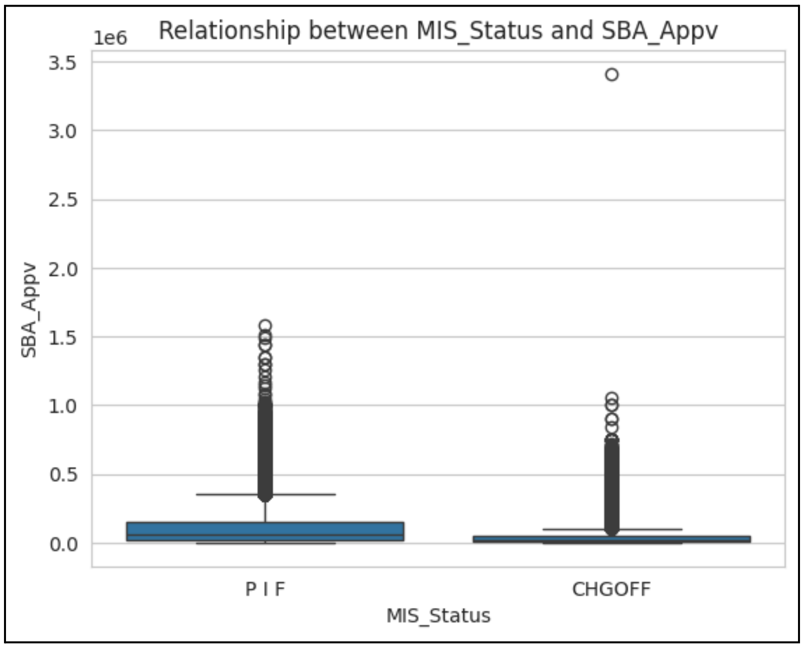


Figure 7

The analysis of "MIS_Status" and "DisbursementGross" in **Figure 8** reveals that companies with higher SBA-guaranteed loan amounts are more likely to pay in full. It means the higher guarantees, the lower probability of defaults. This hypothesis was an important one in our initial EDA. The SBA guarantee is positively related to loan repayment and development of small businesses.

Figure 8 —-- >



Through visual analysis of the relationship between "SameState" and "Mis_Status", we discovered that if the bank and the borrower company are in different states, there tends to be a higher proportion of unpaid loans **(figure 9)**. This effectively answers another question we had at the beginning of our EDA. Based on this result, we have reason to believe that SBA-approved loans where the bank and borrower company are in the same location are more conducive to reducing the risk of bad debt.

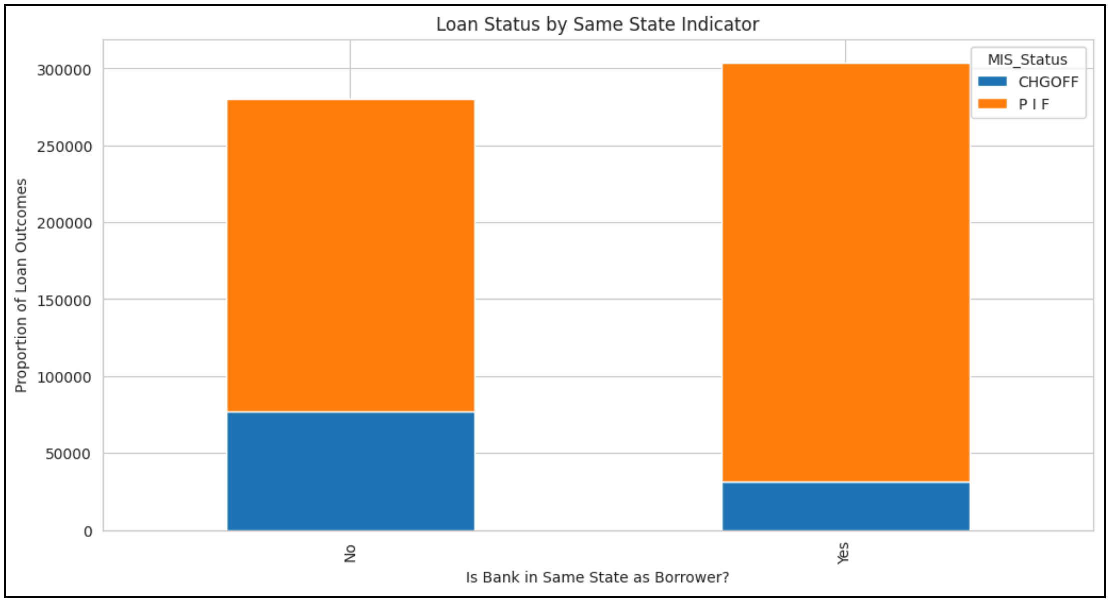


Figure 9

Next, we analyzed the relationship between "IsFranchise" and "Mis_Status" **(figure 10)**. We discovered that borrower companies without a franchise code have a higher proportion of non-repayment. To some extent, the franchise code signifies a company's reputation and background. This suggests that banks, before approving loans, can focus on examining whether the borrowing business has a franchise code and review the company's historical operational data through that code to mitigate the risk of the loan.

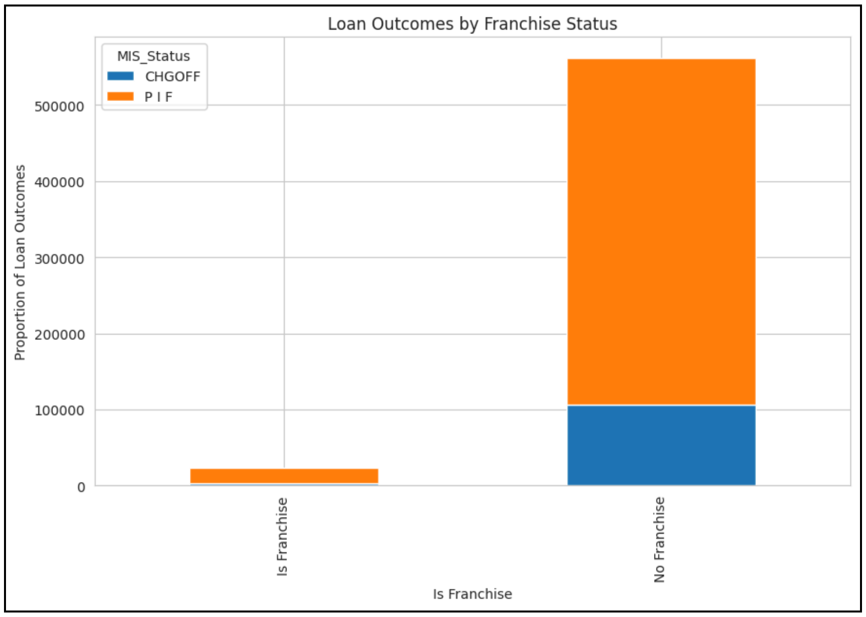


Figure 10

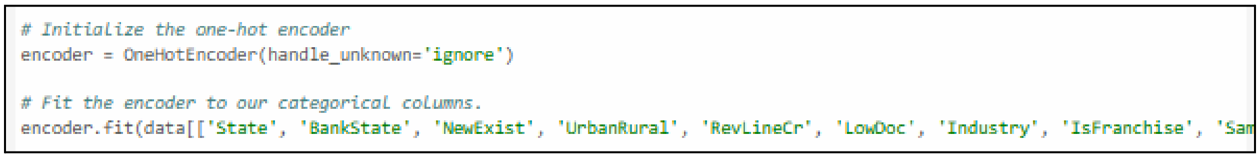
After completing the bivariate analysis between the discrete features and the target feature, we also explored the relationships between some numerical variables but did not find any strong correlations or additional information that warranted further exploration.

3. Modeling

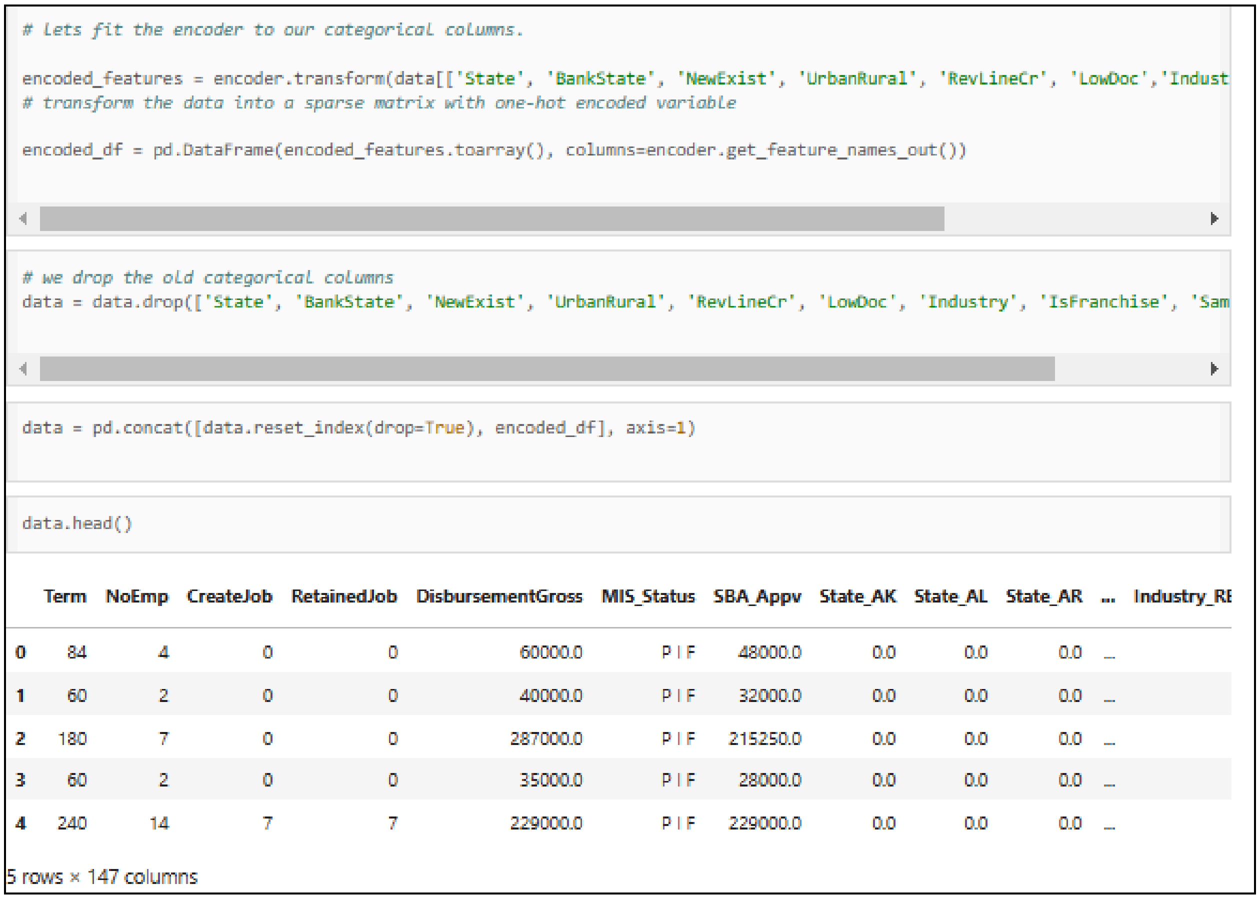
(1) Data Preprocessing

We start the pre-processing step by one-hot encoding of categorical features into a numerical format for the convenience of subsequent operations. This was achieved using the OneHotEncoder from the preprocessing module of scikit-learn. The encoder was initialized in a way that it could create a binary column for each category across the given categorical fields. These binary vectors denote the presence with a "1" or absence with a "0" of each category, effectively translating qualitative values into a structured quantitative format. Employing OneHotEncoder not only facilitates the interpretative capabilities of our models but also strengthens the integrity and precision of our predictions (**figure 11).**

As a further preprocessing step, we have clearly separated the features and the target for our predictive modeling. We will create the feature matrix by excluding the column 'MIS_Status', which we have chosen as the target variable, and set 'MIS_Status' column as the target array. The segregation of features and targets is the preparatory step required in any supervised learning algorithm.



```
# Initialize the one-hot encoder
encoder = OneHotEncoder(handle_unknown='ignore')

# Fit the encoder to our categorical columns.
encoder.fit(data[['State', 'BankState', 'NewExist', 'UrbanRural', 'RevLineCr', 'LowDoc', 'Industry', 'IsFranchise', 'Sam
```



```
# lets fit the encoder to our categorical columns.

encoded_features = encoder.transform(data[['State', 'BankState', 'NewExist', 'UrbanRural', 'RevLineCr', 'LowDoc','Indust
# transform the data into a sparse matrix with one-hot encoded variable

encoded_df = pd.DataFrame(encoded_features.toarray(), columns=encoder.get_feature_names_out())
```

```
# we drop the old categorical columns
data = data.drop(['State', 'BankState', 'NewExist', 'UrbanRural', 'RevLineCr', 'LowDoc', 'Industry', 'IsFranchise', 'Sam
```

```
data = pd.concat([data.reset_index(drop=True), encoded_df], axis=1)
```

```
data.head()
```

```
   Term  NoEmp  CreateJob  RetainedJob  DisbursementGross  MIS_Status  SBA_Appv  State_AK  State_AL  State_AR  ...  Industry_RE
0    84      4          0            0            60000.0         P I F   48000.0       0.0       0.0       0.0  ...
1    60      2          0            0            40000.0         P I F   32000.0       0.0       0.0       0.0  ...
2   180      7          0            0           287000.0         P I F  215250.0       0.0       0.0       0.0  ...
3    60      2          0            0            35000.0         P I F   28000.0       0.0       0.0       0.0  ...
4   240     14          7            7           229000.0         P I F  229000.0       0.0       0.0       0.0  ...

5 rows × 147 columns
```

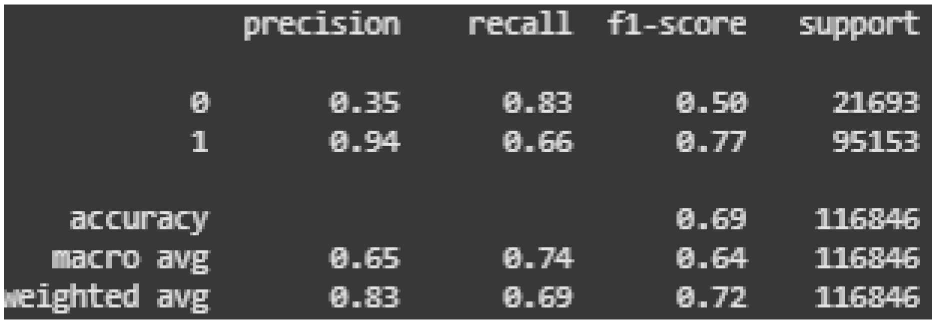
Figure 11

Then, the dataset is divided into training and testing sets for model validation, ensuring that our model generalizes well to unseen data. For this division, the function train_test_split is used, with 80% of the data kept for training and 20% for testing. The random_state parameter is set to 42 to ensure reproducibility of the split.

The last preprocessing step taken in this study was numerical transformation of the categorical labels for the target variable using the LabelEncoder. Then, we call LabelEncoder on these labels to convert them into a numeric array. This avoided computational discrepancies in the upcoming model training process. We transformed both the training and test targets into y_train and y_test, respectively.

(2) Logistic Regression

Due to the widespread application of logistic regression in classification problems, especially its remarkable effect in binary classification, we have chosen the logistic regression model to train and predict on our dataset. Initially, we established our model using the LogisticRegression class from sklearn.linear_model. The resulting model exhibits an accuracy of 91%, which seemingly indicates good performance. However, we observed a significant imbalance in the prediction of positive and negative classes.



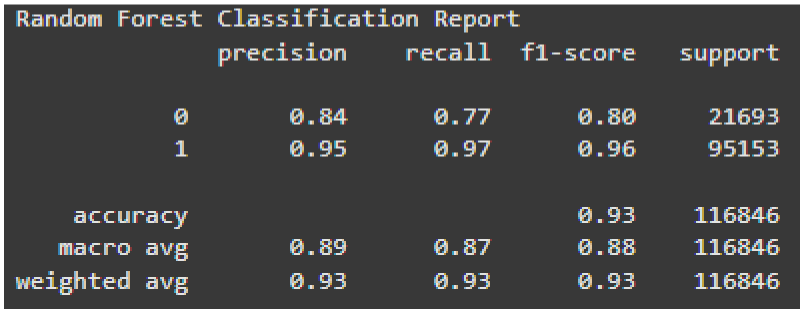
```
              precision    recall  f1-score   support

           0       0.35      0.83      0.50     21693
           1       0.94      0.66      0.77     95153

    accuracy                           0.69    116846
   macro avg       0.65      0.74      0.64    116846
weighted avg       0.83      0.69      0.72    116846
```

Figure 12

The occurrence of this situation is largely due to the presence of class imbalance in our dataset, which was also observed during our earlier EDA. To address this issue, we set the class_weight attribute to 'balanced', which will automatically adjust the weight of each category according to the frequency of the labels in the data. This means that less frequently occurring categories (Charged-off) will be assigned higher weights, while more frequently occurring categories (Paid-In-Full) will be assigned lower weights, with the expectation of countering the impact of class imbalance during the training process. Subsequently, the model is trained (fit) on the training set X_train and y_train, and the trained model is used to predict on the test set X_test, resulting in y_predicted_lg.

(3) Random Forest

The Random Forest Classifier was implemented with 100 estimators and a fixed random state for reproducibility. After training the model on the dataset, it achieved a high accuracy of **93%**, demonstrating strong overall performance. The classification report revealed detailed metrics for both classes: for class 0 ('Charged Off'), the model achieved a precision of **84%**, a recall of **77%**, and an F1-score of **80%**, indicating room for improvement in identifying all actual "Charged Off" cases. For class 1 ('Paid in Full'), the model performed exceptionally well, with a precision of **95%**, recall of **97%**, and F1-score of **96%**. The weighted averages exceeded 90%, reflecting the model's effectiveness in handling the class imbalance. Additionally, the ROC-AUC score of **0.97** signifies excellent capability in distinguishing between the two classes. While the model demonstrates robust predictive power, improving the recall for "Charged Off" cases through hyperparameter tuning or resampling techniques could further enhance performance.

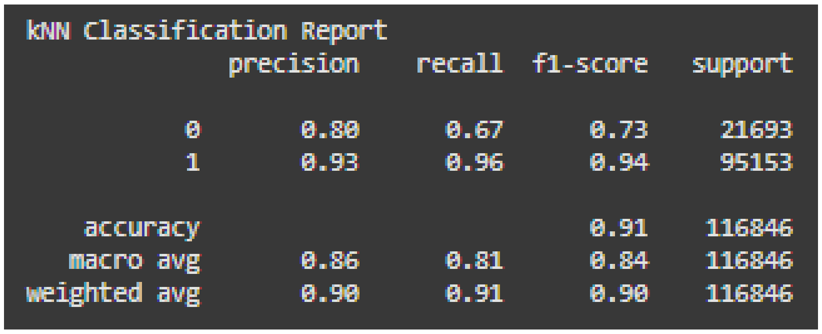


```
Random Forest Classification Report
              precision    recall  f1-score   support

           0       0.84      0.77      0.80     21693
           1       0.95      0.97      0.96     95153

    accuracy                           0.93    116846
   macro avg       0.89      0.87      0.88    116846
weighted avg       0.93      0.93      0.93    116846
```

Figure 13

(4) k-Nearest Neighbors

The k-Nearest Neighbors (kNN) algorithm was implemented with 5 neighbors to classify the dataset. After training the model on the training set (X_train, y_train), predictions were made on the test set (X_test). The classification report shows an overall accuracy of **91%**, with detailed performance metrics for each class. For class 0 ('Charged Off'), the model achieved a precision of **80%**, a recall of **67%**, and an F1-score of **73%**, indicating moderate performance in identifying "Charged Off" cases. For class 1 ('Paid in Full'), the precision was **93%**, recall was **96%**, and F1-score was **94%**, reflecting strong performance in identifying "Paid in Full" cases. The macro average F1-score of **84%** shows a slight imbalance in performance between the classes, while the weighted average F1-score of **90%** aligns with the overall accuracy.



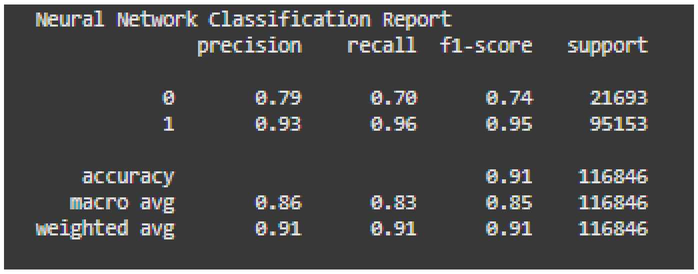
```
kNN Classification Report
              precision    recall  f1-score   support

           0       0.80      0.67      0.73     21693
           1       0.93      0.96      0.94     95153

    accuracy                           0.91    116846
   macro avg       0.86      0.81      0.84    116846
weighted avg       0.90      0.91      0.90    116846
```

Figure 14

(5) Neural Networks

The Neural Network model was implemented using TensorFlow's Keras API for a multiclass classification task. To ensure consistent feature scaling, the input data was normalized using StandardScaler. The target variable for training (y_train) was converted to one-hot encoding to match the multiclass format required for the neural network. The model architecture consisted of three layers: an input layer with 64 neurons, a hidden layer with 32 neurons (both using ReLU activation for non-linearity), and an output layer with 3 neurons and a softmax activation function for multiclass probabilities. The model was compiled with the Adam optimizer, categorical cross-entropy loss (appropriate for multiclass problems), and accuracy as the evaluation metric. Class weights were specified to address any potential imbalance between the classes.

The model was trained for 30 epochs with a batch size of 32, using class weights to give more importance to underrepresented classes. Predictions on the test set were generated, and the class with the highest probability for each prediction was selected. The classification report revealed an overall accuracy of **91%**. For class 0, the precision, recall, and F1-score were **79%**, **70%**, and **74%**, respectively, indicating moderate performance in identifying this class. For class 1, precision was **93%**, recall was **96%**, and F1-score was **95%**, showing strong performance in identifying the majority class. The macro average F1-score of **85%** reflects some imbalance across classes, while the weighted average F1-score of **91%** aligns with the overall accuracy, demonstrating the model's effectiveness. However, further tuning of class weights or architecture could enhance the recall for minority classes.



```
Neural Network Classification Report
              precision    recall  f1-score   support

           0       0.79      0.70      0.74     21693
           1       0.93      0.96      0.95     95153

    accuracy                           0.91    116846
   macro avg       0.86      0.83      0.85    116846
weighted avg       0.91      0.91      0.91    116846
```

Figure 15

(6) ROC Curve for Logistic Regression with Lasso

The Receiver Operating Characteristic (ROC) curve calculates the Area Under the Curve (AUC) for a logistic regression model, handling both binary and multiclass scenarios. It begins by obtaining the predicted probabilities (y_proba_logit) from the logistic regression model. For multiclass classification, the target variable (y_train) is binarized using label_binarize to compute ROC curves for each class individually. The false positive rate (FPR), true positive rate (TPR), and AUC are calculated for each class using roc_curve and auc, and the results are plotted with individual ROC curves labeled by class and their respective AUC values. For binary classification, the probabilities of the positive class are used to compute a single ROC curve and AUC. The diagonal line in the plot represents random guessing, serving as a baseline. After plotting, the script prints the AUC for each class in a multiclass scenario or the single AUC value for binary classification. In this specific case, the binary classification achieved an **AUC of 84.37%**, indicating strong model performance in distinguishing between the two classes.

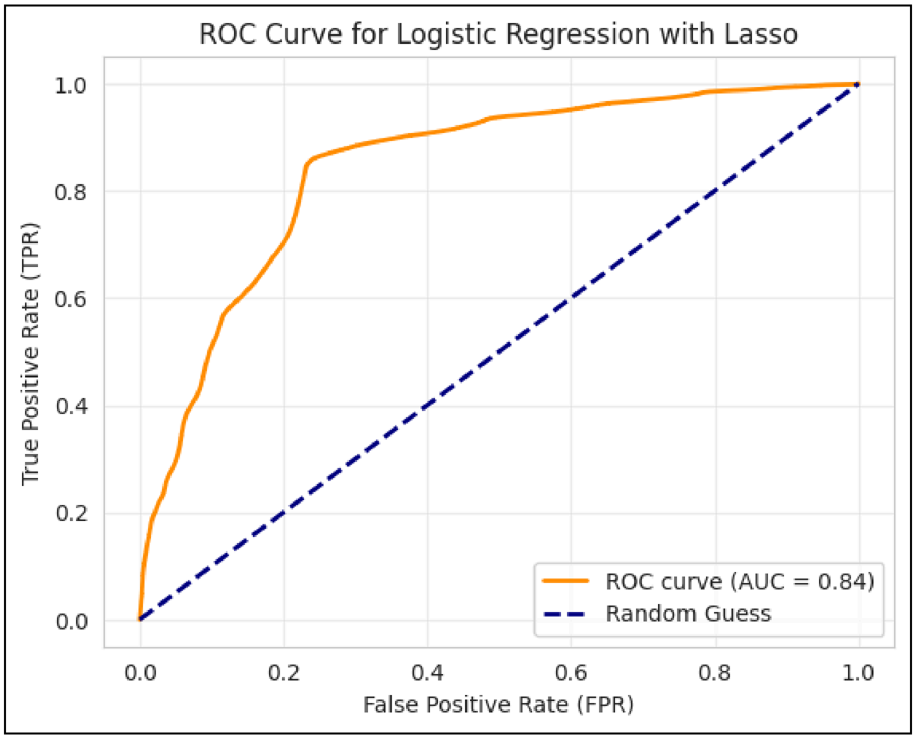


Figure 16

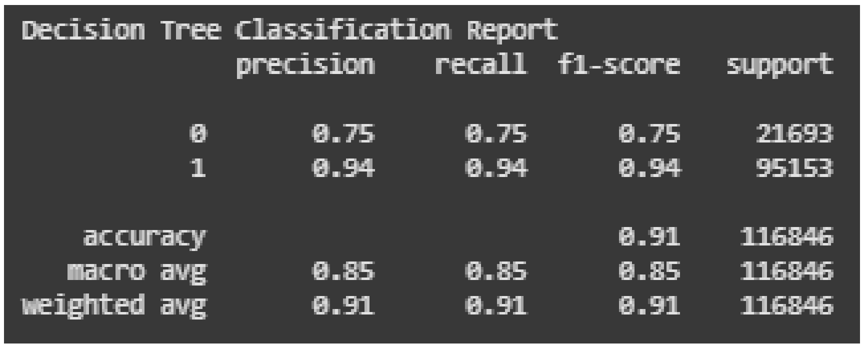
(7) Feature Selection

The provided code uses feature selection and cross-validation to determine the optimal number of features to retain when training a **Logistic Regression** model for classification. The key idea is to identify the most informative features by employing the **SelectKBest** method with the f_classif statistical test, which evaluates the relationship between each feature and the target variable. The process involves iterating over a range of predefined k values (number of features to select), selecting the top k features based on their ANOVA F-value, and training a Logistic Regression model on the reduced feature set. For each k, the model's performance is evaluated using **5-fold cross-validation**, scoring the model based on its **recall**, which prioritizes minimizing false negatives in predictions. The recall scores are averaged over all folds, and the k value with the highest average score is chosen as the optimal number of features.

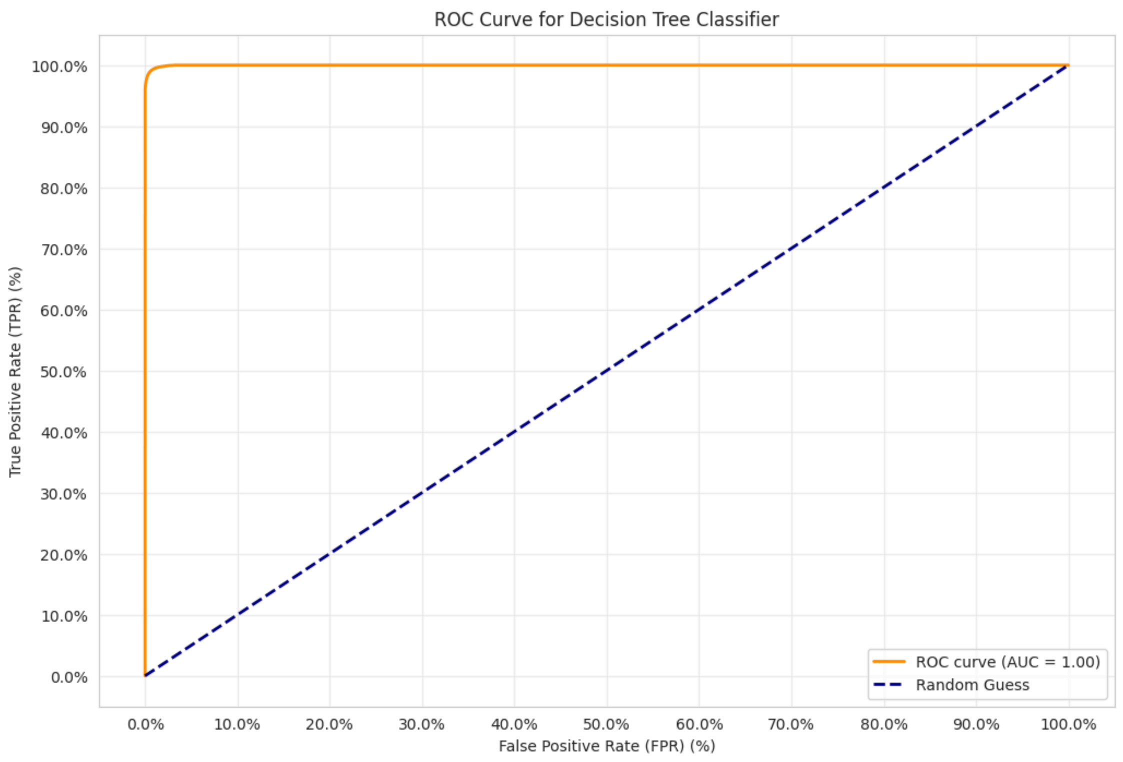
The results of the execution indicate that selecting the top **10 features** (out of the tested values: 5, 10, 15, 20, and 50) yielded the highest average recall during cross-validation. This suggests that these 10 features provide a balance between retaining sufficient information for the model to make accurate predictions while avoiding overfitting or including irrelevant features. By reducing the feature set to the most impactful ones, the model's interpretability improves, and computational complexity decreases without compromising recall performance, which is especially critical in scenarios like disease diagnosis or fraud detection where false negatives are costly.

(8) Decision tree

Using the training set (X_train, y_train) and making predictions on the test set (X_test). A cost matrix is defined to account for the profit or loss of correctly and incorrectly predicted classes. For correctly predicted labels, the associated profit is added to the total, while misclassifications incur a penalty by adding half the profit of the predicted class. The classification report shows that the model performs well for the majority class (Class 1, "Paid in Full") with a precision and recall of 0.94, and an F1-score of 0.94, indicating strong performance in identifying this class. However, the performance for the minority class (Class 0, "Charged Off") is more moderate, with precision and recall of 0.75. This leads to a balanced overall accuracy of 91%, with a macro average F1-score of 0.85, reflecting slightly lower performance for Class 0. The weighted average F1-score of 0.91 demonstrates the model's general effectiveness, though improvements could be made, especially in identifying the minority class more accurately.



```
Decision Tree Classification Report
              precision    recall  f1-score   support

           0       0.75      0.75      0.75     21693
           1       0.94      0.94      0.94     95153

    accuracy                           0.91    116846
   macro avg       0.85      0.85      0.85    116846
weighted avg       0.91      0.91      0.91    116846
```



Conclusion

The dataset was subjected to different machine learning models in the current work, with specific focus on techniques of overcoming class imbalance and further tuning model performance. The models employed for these predictions were: Logistic Regression, Random Forest, k-Nearest Neighbors (kNN), Neural Networks and Decision tree - all with strengths and weaknesses of their own.

- **Logistic Regression** initially showed a strong accuracy of 91%, but the severe class imbalance needed some tweaks. Setting the class_weight attribute to 'balanced' took care of this, so the infrequent classes were given more importance during training, hence enhancing the robustness of the model in handling imbalanced data.
- **The Random Forest** algorithm attained a remarkable accuracy of 93%, especially with a highly performing majority class ('Paid in Full'). However, there was room for improvement in the performance regarding the minority class ('Charged Off'). Further hyperparameter tuning or resampling techniques may be used to achieve higher recall for the minority class without compromising overall model performance. This was further confirmed with an excellent ROC-AUC score of 0.97.
- **k-Nearest Neighbors:** The accuracy of kNN was 91%, while the recall for the minority class was lower, which shows that this model has moderate performance in finding all the relevant cases. However, its weighted average F1-score was 90%, showing overall efficacy.
- **The neural networks** built using the Keras API in TensorFlow yielded an accuracy of 91% and showed quite a balanced performance across classes, although recall for the minority class remained a challenge. Some of the class imbalance was somewhat offset by using class weights, but further tuning of the network architecture and class weights might improve performance for underrepresented classes.
- **The Decision Tree Classifier** performed well overall, with an accuracy of 91% and strong results for the majority class. However, similar to the other models, it showed moderate performance for the minority class, highlighting the challenge of class imbalance across all models. The introduction of a cost matrix helped address misclassifications by incorporating profit and penalty factors, but further tuning could improve the recall for the "Charged Off" class.
- **Lastly, the ROC Curve for Logistic Regression** provided valuable insight into the model's discriminative power with an AUC of 84.37%, thus confirming that it is capable of differentiating between the two classes. Moreover, the feature selection process has identified that selecting only the most relevant features is necessary to improve the interpretability and performance of the models. We have narrowed it down to the top 10, ensuring that we do not lose any necessary information by decreasing computational complexity.

Overall, the model predictions were strong, and RF proved to be the best performer overall. However, for handling the minority class, one may see further improvements by implementing hyperparameter tuning, resampling methods, or utilizing higher-end ensemble techniques.