Credit Alchemy: Small Business Creditworthiness Demystified

Predicting Creditworthiness of Small Businesses

Business Overview

The Small Business Administration (SBA) is a crucial organization in the United States that supports and promotes small enterprises by providing access to credit. Through loan guarantees, the SBA encourages lenders to provide credit to small businesses, which play a vital role in job creation and reducing unemployment. However, there have been instances of loan defaults, posing a challenge for the SBA in accurately assessing the creditworthiness of applicants and mitigating risks. To address this, the SBA focuses on evaluating creditworthiness by considering factors such as financial history, business plans, collateral, and projected cash flows. They also strive to adapt to market dynamics, embracing innovative approaches and technology for informed lending decisions that minimize the likelihood of defaults. Overall, the SBA's commitment to supporting small businesses goes beyond loan guarantees, encompassing mentorship programs, training resources, government contracting opportunities, and disaster recovery assistance to foster an environment conducive to small business success.

Stakeholders

- Government / Lending institutions such as the SBA
- Small Businesses

Project Overview

This project focuses on utilizing the dataset from the U.S. Small Business Administration to develop a predictive model for loan application approval. By analyzing relevant factors and historical loan data, the project aims to create a reliable system that assists the SBA in making informed decisions while minimizing the risk of defaults. The ultimate goal is to provide the SBA with a robust loan approval model that enhances their ability to make accurate and efficient lending decisions.

Project Objectives

1. Conduct a comprehensive analysis of the dataset from the U.S. Small Business Administration to identify patterns and trends for accurate loan approval predictions.

2. Develop a robust machine learning model that utilizes the identified patterns and trends to predict loan approval outcomes effectively.
3. Deploy and evaluate the developed machine learning model in the loan approval process of the U.S. Small Business Administration, continuously optimizing its predictive capabilities for informed lending decisions.

Data Understanding

The data used to conduct this project was extracted from the official US Small Business Administration open data source (https://data.sba.gov/dataset/). The data had a total of 899,164 rows and 27 columns. In this section we begin by importing the necessary libraries that will be used throughout the notebook, we then went on to load the datasets, as well as further understanding the data by displaying the summary statistics, as well as checking for missing values.

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
pd.set_option('display.max_columns', None)

# Data preprocessing and transformation
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler

# Classification model libraries
from sklearn.metrics import accuracy_score, precision_score, f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import svm
from sklearn.linear_model import SGDClassifier
from xgboost import XGBClassifier
from keras.models import Sequential
from keras.layers import Dense,Flatten

# Grid Search CV
from sklearn.model_selection import GridSearchCV

# Pipeline and feature selection
from sklearn.pipeline import Pipeline
```

In [3]:

```
# Reading SBA Data
df = pd.read_csv('data/SBAnational.csv',low_memory=False)
df.head()
```

Out[3]:

	LoanNr_ChkDgt	Name	City	State	Zip	Bank	BankState	NAICS
0	1000014003	ABC HOBBYCRAFT	EVANSVILLE	IN	47711	FIFTH THIRD BANK	OH	451120
1	1000024006	LANDMARK BAR & GRILLE (THE)	NEW PARIS	IN	46526	1ST SOURCE BANK	IN	722410
2	1000034009	WHITLOCK DDS, TODD M.	BLOOMINGTON	IN	47401	GRANT COUNTY STATE BANK	IN	621210
3	1000044001	BIG BUCKS PAWN & JEWELRY, LLC	BROKEN ARROW	OK	74012	1ST NATL BK & TR CO OF BROKEN	OK	0
4	1000054004	ANASTASIA CONFECTIONS, INC.	ORLANDO	FL	32801	FLORIDA BUS. DEVEL CORP	FL	0

In [4]:

```
# Reading Foia Data
df2 = pd.read_csv("data/foia-7afy2020-present-asof-230331.csv",encoding='latin-1', e
df2.head()
```

Out[4]:

	AsOfDate	Program	BorrName	BorrStreet	BorrCity	BorrState	BorrZip	BankName
0	20230331	7A	Allen Foot and Ankle Medicine	2919 S ELLSWORTH RD STE 124	MESA	AZ	85212	Western Alliance Bank
1	20230331	7A	Cojutepeque Restaurant	2610 W 3rd St	Los Angeles	CA	90057	Banner Bank
2	20230331	7A	Adwa LLC	3105 ALDERWOOD MALL BLVD Suite	LYNNWOOD	WA	98036	Umpqua Bank
3	20230331	7A	Town Cleaners	2700 WASHINGTON BLVD #B	ARLINGTON	VA	22201	Hanmi Bank
4	20230331	7A	Moor Inc.	524 WOODSIDE RD	REDWOOD CITY	CA	94061	JPMorgan Chase Bank, National Association

In [5]:

```
# Compatibility of dependent variable
df2.LoanStatus.value_counts()
```

Out[5]:

```
EXEMPT    121746
COMMIT     17858
CANCLD     14980
PIF        13147
CHGOFF       480
Name: LoanStatus, dtype: int64
```

In [6]:

```
# Foia Data as a percentage of SBA Data
len(df2)/len(df)
```

Out[6]:

```
0.18707488289121896
```

Since Foia data is incompatible and of negligible size, we proceeded with the SBA data only for our analysis

In [7]:

```
class Describer:

    # initialize object
    def __init__(self, df):
        self.df = df

    # method to check shape of data
    def shape(self):
        shape_df = print(f"The DataFrame has:\n\t* {self.df.shape[0]} rows\n\t* {sel
        return shape_df

    # method to check info on dataset
    def data_info(self):
        info_df = print(self.df.info(), '\n')
        return info_df

    # method to describe numerical columns
    def data_describe(self):
        num_col = self.df.describe()
        return num_col
```

In [8]:

```
# creating an instance of the class describer
describe_df = Describer(df)

# Viewing the shape of the data
describe_df.shape()
```

```
The DataFrame has:
        * 899164 rows
        * 27 columns
```

In [9]:

```
# summary information
print('Summary infomation on dataset')
print('-----------------------------------------------')
describe_df.data_info()
```

```
Summary infomation on dataset
-----------------------------------------------
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 899164 entries, 0 to 899163
Data columns (total 27 columns):
 #   Column             Non-Null Count   Dtype 
---  ------             --------------   ----- 
 0   LoanNr_ChkDgt      899164 non-null  int64 
 1   Name               899150 non-null  object
 2   City               899134 non-null  object
 3   State              899150 non-null  object
 4   Zip                899164 non-null  int64 
 5   Bank               897605 non-null  object
 6   BankState          897598 non-null  object
 7   NAICS              899164 non-null  int64 
 8   ApprovalDate       899164 non-null  object
 9   ApprovalFY         899164 non-null  object
 10  Term               899164 non-null  int64 
 11  NoEmp              899164 non-null  int64 
```

In [10]:

```
# summary statistics
describe_df.data_describe()
```

Out[10]:

	LoanNr_ChkDgt	Zip	NAICS	Term	NoEmp	NewE
count	8.991640e+05	899164.000000	899164.000000	899164.000000	899164.000000	899028.000(
mean	4.772612e+09	53804.391241	398660.950146	110.773078	11.411353	1.280
std	2.538175e+09	31184.159152	263318.312760	78.857305	74.108196	0.451
min	1.000014e+09	0.000000	0.000000	0.000000	0.000000	0.000(
25%	2.589758e+09	27587.000000	235210.000000	60.000000	2.000000	1.000(
50%	4.361439e+09	55410.000000	445310.000000	84.000000	4.000000	1.000(
75%	6.904627e+09	83704.000000	561730.000000	120.000000	10.000000	2.000(
max	9.996003e+09	99999.000000	928120.000000	569.000000	9999.000000	2.000(

In [11]:

```
# Duplicates
df.duplicated().any()
```

Out[11]:

```
False
```

In [12]:

```
# Function to display missing values

def missing_values(data):
    # identify the total missing values per column
    # sort in order
    if df.isnull().sum().any():
        print(f"The Data has {df.isnull().sum().sum()} missing values in total.")
        miss = data.isnull().sum().sort_values(ascending = False)

        # percentage of missing values
        percentage_miss = (data.isnull().sum() / len(data)).sort_values(ascending =

        # store in a dataframe
        missing = pd.DataFrame({"Missing Values": miss, "Percentage(%)": percentage_

        return missing
    else:
        print("The Data has no missing values")


missing_values(df)
```

```
The Data has 751259 missing values in total.
```

Out[12]:

	Missing Values	Percentage(%)
ChgOffDate	736465	0.819055
RevLineCr	4528	0.005036
LowDoc	2582	0.002872
DisbursementDate	2368	0.002634
MIS_Status	1997	0.002221
BankState	1566	0.001742
Bank	1559	0.001734
NewExist	136	0.000151
City	30	0.000033

Data Cleaning

At this stage of our analysis after checking the quality of the data, we proceeded to utilise different data cleaning techniques as we saw necessary, such as:

- Dealing with the missing values
- Changing data types to the appropriate formats; among others.

We decided to drop the column `ChgOffDate` , representing the **date when a loan is declared to be in default**, due to the high percentage of missing values.

In [13]:

```
# Dropping the ChgOffDate column
df.drop('ChgOffDate', axis = 1, inplace=True)
```

In [14]:

```
# Checking the unique values in the target variable
df['MIS_Status'].unique()
```

Out[14]:

```
array(['P I F', 'CHGOFF', nan], dtype=object)
```

In [15]:

```
# Checking the percentage of data retained if we choose to drop all rows with missir
len(df.dropna())/len(df)
```

Out[15]:

```
0.9856266487537313
```

Since we still have a very significant amount of data(`98.5%`) retained after dropping the missing values, we proceed and drop them.

In [16]:

```
# Dropping all rows with missing values
df.dropna(inplace=True)
```

In [17]:

```
# Confirming that we have dropped all rows with missing values
df.isnull().sum().any()
```

Out[17]:

```
False
```

Next, we removed the dollar signs to facilitate data manipulation and for consistency.

In [18]:

```
# Removing the dollar signs in the currency columns
dollar_columns = ['DisbursementGross', 'BalanceGross', 'ChgOffPrinGr', 'GrAppv', 'SB
df[dollar_columns] = df[dollar_columns].replace('[\$,]', '', regex=True).astype(floa
```

In [19]:

```
date_columns=['ApprovalDate','DisbursementDate']
for column in date_columns:
    df[column]=pd.to_datetime(df[column])
```

Checking for inconsistencies / invalid data

In [20]:

```
invalid_data=df[df.DisbursementDate<df.ApprovalDate]
invalid_data
```

Out[20]:

	LoanNr_ChkDgt	Name	City	State	Zip	Bank	BankState	NAICS	A
481	1003703008	AQUA BELLA ENTERPRISES, INC.	CHICAGO	IL	60634	NORTH COMMUNITY BANK	IL	0	
1548	1010634005	EARL'S MACHINE SHOP	ROGERS	AR	72756	IBERIABANK	AR	332312	
1655	1011204007	MACHINE SHOP SERVICES INC	NEENAH	WI	54956	AMER NATL BANK-FOX CITIES	WI	333512	
6907	1047134001	ADVANTAGE ASIA	NEW YORK	NY	10028	JPMORGAN CHASE BANK NATL ASSOC	IL	0	
13804	1098814004	LANSING CONSTRUCTION	MONTEZUMZ CREEK	UT	84510	VECTRA BK COLORADO NATL ASSOC	NM	235930	
...	...	...	...	...	...	...	...	...	

In [21]:

```
# Dropping Inconsistencies
df.drop(invalid_data.index,axis=0,inplace=True)
```

In [22]:

```
# Confirming Drop
len(df[df.DisbursementDate<df.ApprovalDate])
```

Out[22]:

```
0
```

In [23]:

```
# Sorting the DataFrame
df = df.sort_values(by=['ApprovalDate']).reset_index(drop=True)
```

In [24]:

```
# Displaying the DataFrame
df.head()
```

Out[24]:

	LoanNr_ChkDgt	Name	City	State	Zip	Bank	BankState	NAICS
0	2357951001	WINSLOW CORP	BERLIN	NJ	8009	PNC BANK, NATIONAL ASSOCIATION	NJ	0
1	4160941004	THE BUCKSKIN	QUEMADO	NM	87829	FIRST STATE BANK	NM	0
2	3527741006	MOBICENTRICS INC	BRONX	NY	10451	BANK OF AMERICA NATL ASSOC	NY	0
3	3227931002	AMERICANA INSTITUTE OF DAY CAR	GLEN ARDEN	MD	20027	BANK OF AMERICA NATL ASSOC	VA	0
4	5090841007	JAMES E SEARS	LINCOLN	NE	68502	THE BANK	KS	0

Cleaning `NewExist` Column (Whether a business is new or has been in operation), as we didn't have sufficient information on what `0.0` represented, we proceeded to backfill those values.

In [25]:

```
# Checking Value Counts
df.NewExist.value_counts()
```

Out[25]:

```
1.0    635560
2.0    248875
0.0      1021
Name: NewExist, dtype: int64
```

In [26]:

```
# Replacing 0.0 values with null
df.NewExist=df.NewExist.replace(0.0,np.nan)
```

In [27]:

```
# Backfilling null values
df.NewExist=df.NewExist.backfill()
```

In [28]:

```
# Confirming replacement
df.NewExist.isnull().sum()
```

Out[28]:

```
0
```

In [29]:

```
# Displaying value counts
df.NewExist.value_counts()
```

Out[29]:

```
1.0    636300
2.0    249156
Name: NewExist, dtype: int64
```

Feature Engineering

We proceeded to do feature engineering on the data, this included creating a new column `Industry` , that was mapped to the NAICS code (https://www.census.gov/naics/?58967?yearbck=2022), this was to create deeper understanding as well as help in the preceding sections of the analysis.

In [30]:

```
# Computing SBA Approved as a percentage of Gross Approved
df['PercentageAprroved']=(df.SBA_Appv/df.GrAppv)*100
df['PercentageAprroved']=round(df['PercentageAprroved'],1).astype(int)
df['PercentageAprroved'].mean()
```

Out[30]:

```
70.84415148804683
```

In [31]:

```
# Engineering Industry Column
df['Industry'] = df['NAICS'].astype('str').apply(lambda x: x[:2])
df['Industry'] = df['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',
    '0': 'Other'
})
```

Feature Selection

In this section of the analysis, the key columns for the project were selected, while the other columns were dropped off.

The key features that were chosen are the following `GrAppv`, `Term`, `NoEmp`, `NewExist`, `Industry`, and `MIS_Status`.

In [32]:

```
relevant_columns=['GrAppv', 'Term', 'NoEmp','NewExist', 'UrbanRural', 'Industry','MI
df=df.loc[:,relevant_columns]
df
```

Out[32]:

	GrAppv	Term	NoEmp	NewExist	UrbanRural	Industry	MIS_Status
0	200000.0	120	5	1.0	0	Other	CHGOFF
1	155000.0	230	2	1.0	0	Other	CHGOFF
2	150000.0	180	16	1.0	0	Other	CHGOFF
3	70000.0	120	1	2.0	0	Other	CHGOFF
4	18000.0	168	1	1.0	0	Other	CHGOFF
...	...	...	...	...	...	...	...
885451	350000.0	264	4	2.0	0	Other	P I F
885452	28000.0	162	3	1.0	0	Other	CHGOFF
885453	100000.0	84	40	2.0	0	Other	CHGOFF
885454	200000.0	102	30	1.0	0	Other	CHGOFF
885455	20000.0	84	5	1.0	0	Other	CHGOFF

885456 rows × 7 columns

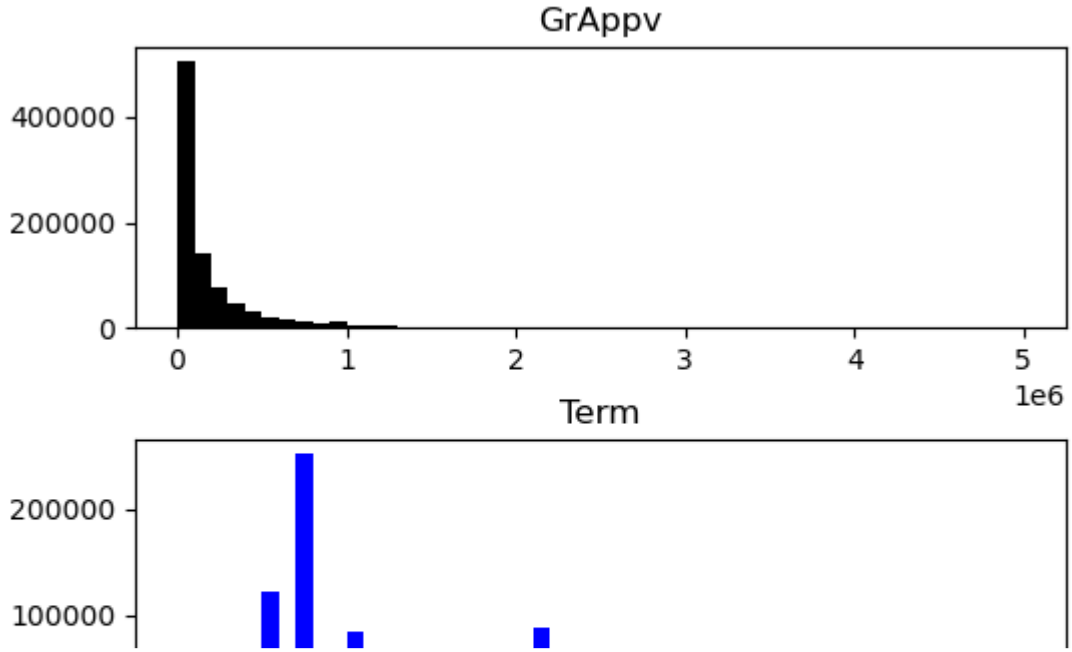
Exploratory Data Analysis

To further understand our data, we explored the variables that will be useful in the project. This included doing a univariate analysis on the numerical columns, as well as performing a multivariate analysis by using a correlation matrix.

Univariate Analysis

In [33]:

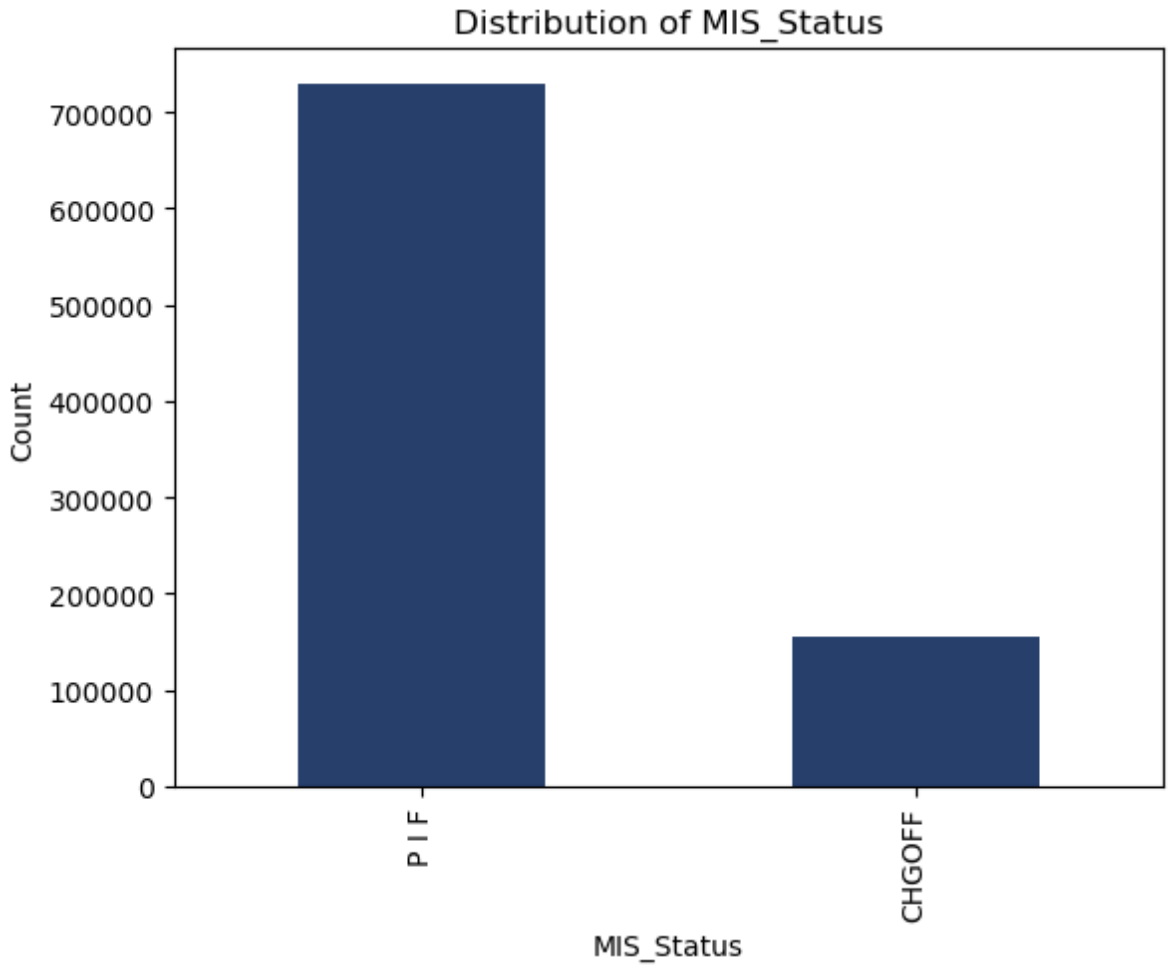
```
colors=['black','blue','green','orange','pink']
numeric_columns = df.select_dtypes(['int64','float64']).columns
def plot_histograms(df, columns):
    fig, axs = plt.subplots(ncols=1, nrows=len(columns), figsize=(6,12))
    for i, col in enumerate(columns):
        axs[i].hist(df[col], bins=50,color=colors[i])
        axs[i].set_title(col)
    plt.subplots_adjust(wspace=0.2,hspace=0.4)
    plt.show()
plot_histograms(df,numeric_columns)
```

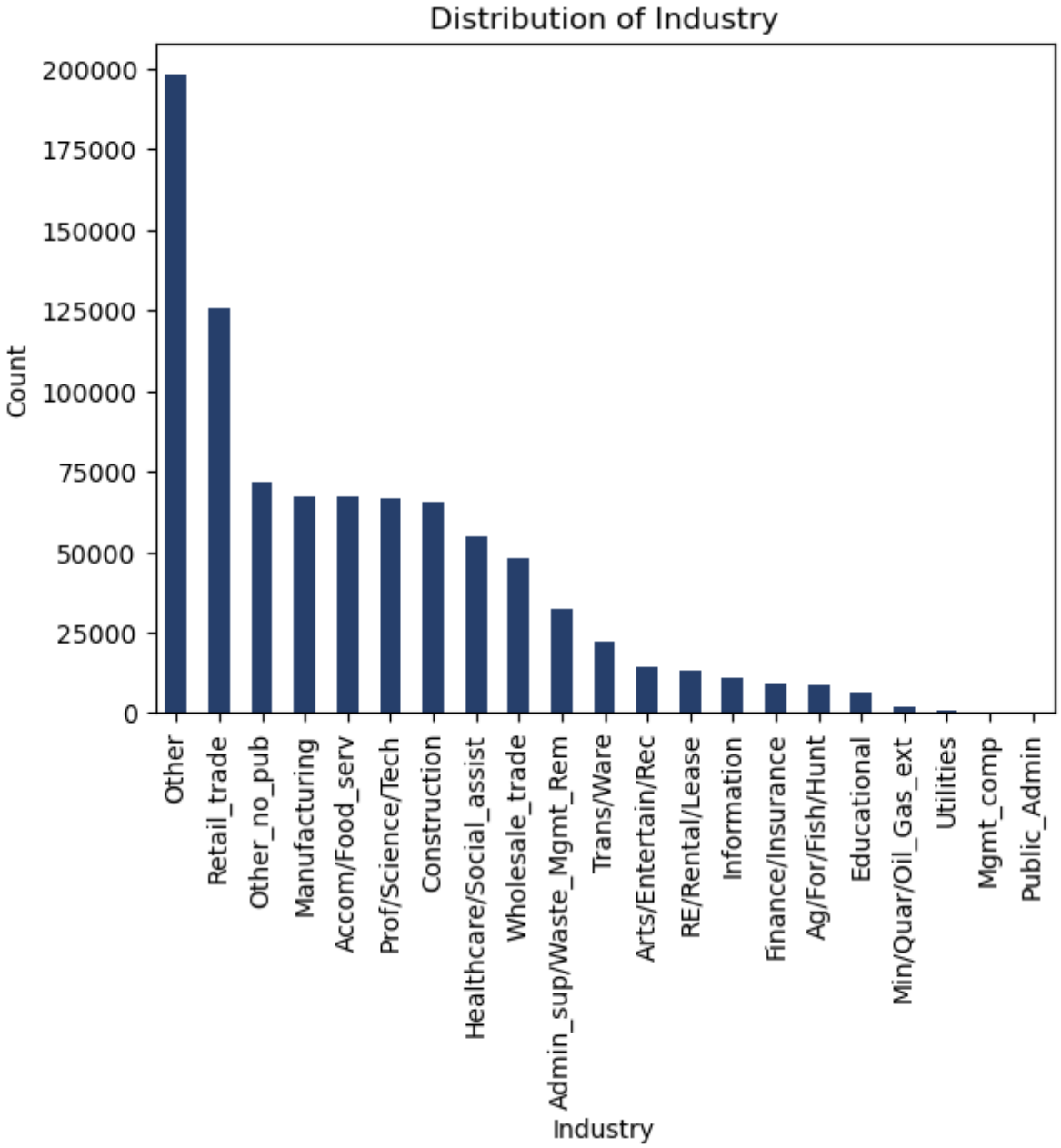


Based on this univariate analysis for the numerical columns, we note that the numerical columns are not normally distributed. In order to deal with this, we will do log transformation at the pre-processing stage.

In [56]:

```
# Define a function that plots the categorical columns
def plot_categorical_columns(df, categorical_columns):
    color = '#263F6B'

    for column in categorical_columns:
        ax = df[column].value_counts().plot(kind='bar', color=color)
        ax.set_xlabel(column)
        ax.set_ylabel('Count')
        ax.set_title(f'Distribution of {column}')
        plt.xticks(rotation=90)

        plt.show()
# Categorical Columns

categorical_columns = ['MIS_Status', 'Industry']

plot_categorical_columns(df, categorical_columns)
```



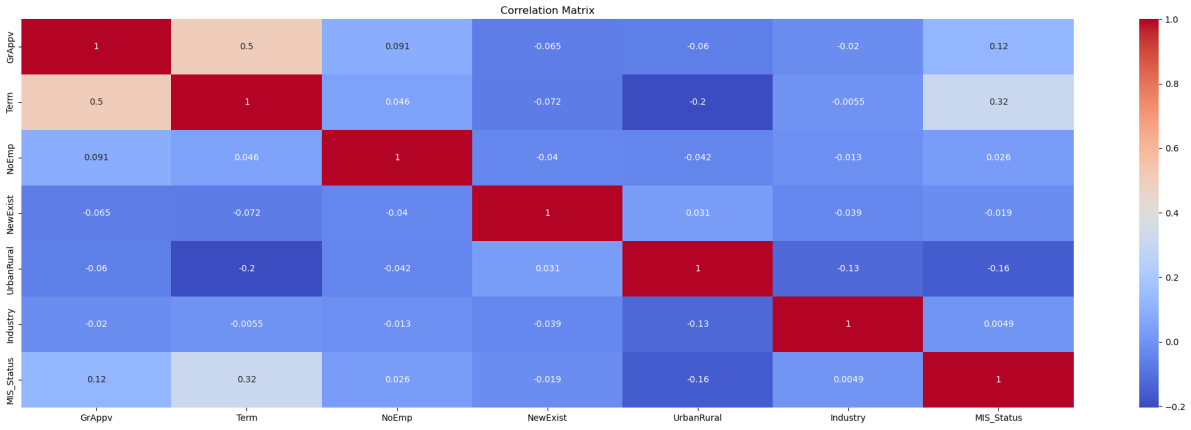


1. Other/Miscellaneous
2. Retail trade
3. Manufacturing
4. Accommodation and Foodservice

Multivariate Analysis

In [34]:

```
# Create a new DataFrame for encoded variables
# Perform label encoding on each categorical column
encoded_df=pd.DataFrame()
for column in df.columns:
    if df[column].dtype == 'object':
        encoded_df[column] = df[column].astype('category').cat.codes
    else:
        encoded_df[column] = df[column]

# Calculate the correlation matrix
correlation_matrix = encoded_df.corr()

# Plot the correlation matrix using a heatmap

plt.figure(figsize=(27, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



- From the matrix, it is clear that the higher the job creation the higher the retention rate since the two have a correlation of `99%` .

Preprocessing

Log Transformation

In [35]:

```
# added a small positive constant to avoid log(0) and negative values
small_constant = 1e-8
df[numeric_columns] = df[numeric_columns].apply(lambda x: x + small_constant)

# Log transformation function
def log_transform(x):
    return np.log(x)

# Apply log transformation to selected numeric columns
df[numeric_columns] = df[numeric_columns].apply(log_transform)
```

In [36]:

```
df[numeric_columns] # Checking the results after the log transformation
```

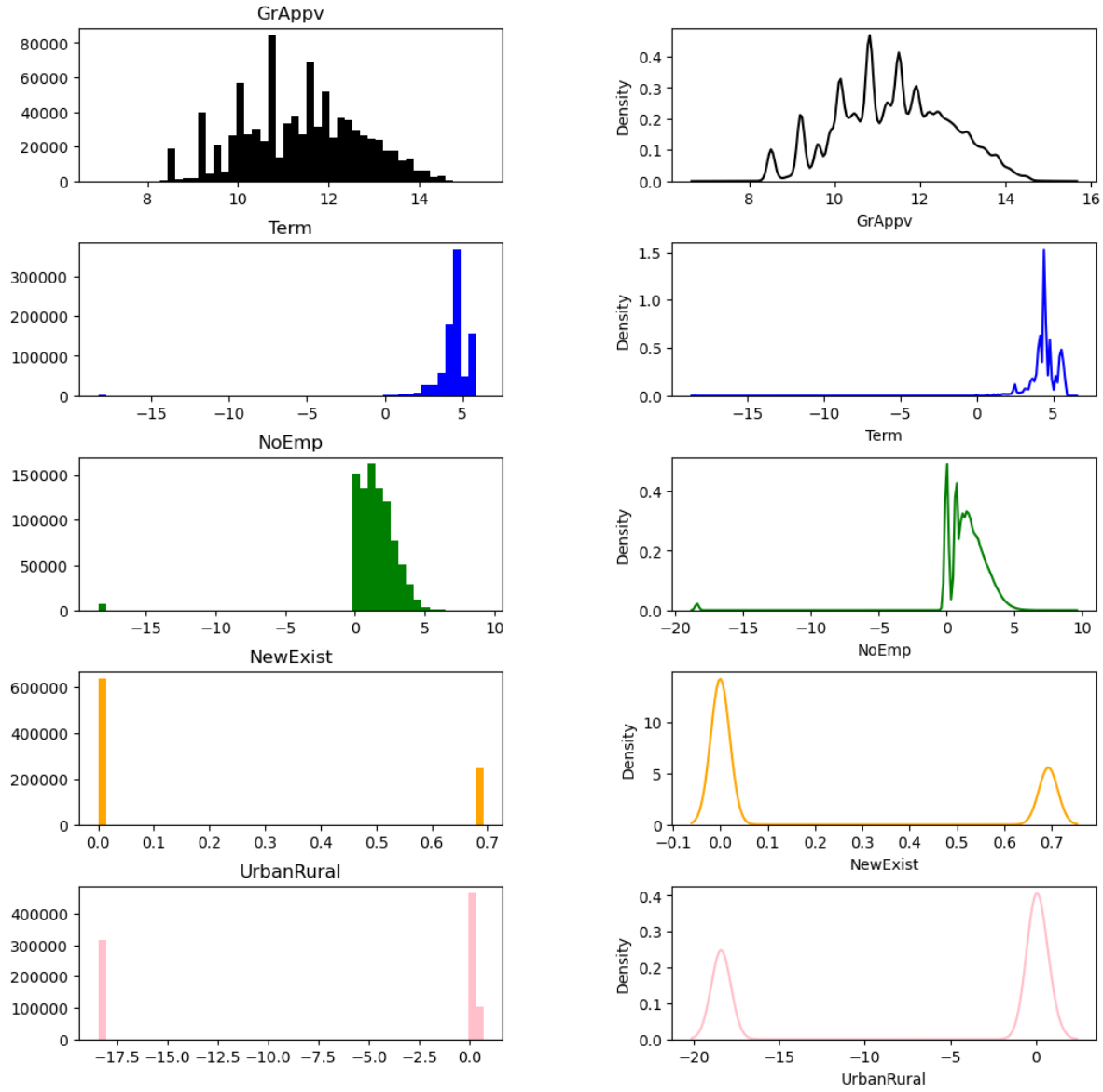
Out[36]:

	GrAppv	Term	NoEmp	NewExist	UrbanRural
0	12.206073	4.787492	1.609438e+00	1.000000e-08	-18.420681
1	11.951180	5.438079	6.931472e-01	1.000000e-08	-18.420681
2	11.918391	5.192957	2.772589e+00	1.000000e-08	-18.420681
3	11.156251	4.787492	1.000000e-08	6.931472e-01	-18.420681
4	9.798127	5.123964	1.000000e-08	1.000000e-08	-18.420681
...	...	...	...	...	...
885451	12.765688	5.575949	1.386294e+00	6.931472e-01	-18.420681
885452	10.239960	5.087596	1.098612e+00	1.000000e-08	-18.420681
885453	11.512925	4.430817	3.688879e+00	6.931472e-01	-18.420681
885454	12.206073	4.624973	3.401197e+00	1.000000e-08	-18.420681
885455	9.903488	4.430817	1.609438e+00	1.000000e-08	-18.420681

885456 rows × 5 columns

In [37]:

```
# Created a function to plot the histogram with the kde in order to inspect the dist
def plot_histograms_with_kde(df, columns):
    fig, axs = plt.subplots(ncols=2, nrows=len(columns), figsize=(12, 12))
    for i, col in enumerate(columns):
        axs[i, 0].hist(df[col], bins=50, color=colors[i])
        axs[i, 0].set_title(col)
        sns.kdeplot(data=df, x=col, ax=axs[i, 1], color=colors[i])
    plt.subplots_adjust(wspace=0.4, hspace=0.4)
    plt.show()

plot_histograms_with_kde(df, numeric_columns)
```



After conducting the log transformation, we have managed to normalise the numerical columns, the next step is to encode the categorical columns `MIS_Status` and `UrbanRural` .

Encoding

As there were categorical columns, we decided to encode the columns, as part of our preparation to conduct the modeling.

In [38]:

```
# Encoding DataFrame
encoded_df=pd.DataFrame()
for column in df.columns:
    if df[column].dtype == 'object':
        encoded_df[column] = df[column].astype('category').cat.codes
    else:
        encoded_df[column] = df[column]
```

In [39]:

```
# Displaying Info
encoded_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 885456 entries, 0 to 885455
Data columns (total 7 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   GrAppv      885456 non-null  float64
 1   Term        885456 non-null  float64
 2   NoEmp       885456 non-null  float64
 3   NewExist    885456 non-null  float64
 4   UrbanRural  885456 non-null  float64
 5   Industry    885456 non-null  int8   
 6   MIS_Status  885456 non-null  int8   
dtypes: float64(5), int8(2)
memory usage: 35.5 MB
```

As noted in the cell above, all of our columns are now in the format of either a float or integer, therefore we have successfully encoded the categorical columns.

In the next sections, we will be splitting our data, resampling and scaling it.

Train Test Split

In [40]:

```
# Extracting Independent Variables
X=encoded_df.drop('MIS_Status',axis=1)
# Extracting Dependent Values
y=encoded_df['MIS_Status']

# Splitting the data
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=42,test_size=0.2)
```

Resampling

In [41]:

```
# Initializing Sampler
sampler=RandomOverSampler(sampling_strategy='minority')
# Fitting Sampler
X_train, y_train = sampler.fit_resample(X_train,y_train)
```

In [38]:

```
# Checking Distribution
y_train.value_counts(normalize=True)
```

Out[38]:

```
1    0.5
0    0.5
Name: MIS_Status, dtype: float64
```

Scaling

In [42]:

```
from sklearn.preprocessing import StandardScaler
# Initializing Scaler
scaler=StandardScaler()
# Fitting Scaler
scaler.fit(X_train)
# Transforming Data
columns=['GrAppv', 'Term', 'NoEmp','NewExist', 'UrbanRural', 'Industry']
scaled_X_train=pd.DataFrame(scaler.transform(X_train),columns=columns)
scaled_X_test=pd.DataFrame(scaler.transform(X_test),columns=columns)

scaled_X_test.head()
```

Out[42]:

	GrAppv	Term	NoEmp	NewExist	UrbanRural	Industry
0	1.948834	0.959356	0.750443	-0.637133	-1.624924	-0.289066
1	0.652105	0.229461	0.664816	-0.637133	-1.624924	0.215674
2	1.101568	0.112563	-0.091056	-0.637133	-1.624924	-0.289066
3	0.663531	0.759344	-0.091056	1.569532	-1.624924	0.383920
4	-0.986919	0.229461	-0.281483	1.569532	0.601034	0.383920

Modeling

Expected Input

1. Amount of Loan
2. Loan Repayment Period
3. Pay Previous Loan Status

4. No.of Employees
5. Business Location
6. Business Industry
7. How long has business been in operation

Expected Output

1. Whether the loan is guaranteed or not
2. Next steps according to the outcome of the result : if **rejected** they will receive certain resources ; if **approved** they will proceed to upload the necessary documents to continue with the application.

Models used included the following:

`logistic regression` - This is the baseline model

`decision tree`

`random forest`

`support vector machine`

`XGBoost`

`neural network`

Logistic Regression

In [43]:

```
# Creates an instance of the logistic regression model
logreg_model = LogisticRegression()

# Trains the model on the scaled training data
logreg_model.fit(scaled_X_train, y_train)

# Predicts on the scaled test data
train_pred = logreg_model.predict(scaled_X_train)
test_pred = logreg_model.predict(scaled_X_test)

# Evaluates the model on the scaled test data
train_accuracy = accuracy_score(train_pred, y_train)
test_accuracy = accuracy_score(test_pred, y_test)
print('Train Accuracy:', train_accuracy)
print('Test Accuracy:' , test_accuracy)
```

```
Train Accuracy: 0.7715037367261414
Test Accuracy: 0.7666693018318163
```

Based on our success metric of at least 85% accuracy, the baseline model does not achieve it, therefore we proceeded to try and explore other models to see if the accuracy improves.

Decision Tree

In [44]:

```
# Create and train the decision tree model
tree_model = DecisionTreeClassifier()
tree_model.fit(scaled_X_train, y_train)

# Predicting on scaled data
train_pred = tree_model.predict(scaled_X_train)
test_pred = tree_model.predict(scaled_X_test)

# Evaluate the model on the scaled test data
train_accuracy = accuracy_score(train_pred, y_train)
test_accuracy = accuracy_score(test_pred, y_test)
print('Train Accuracy:', train_accuracy)
print('Test Accuracy:' , test_accuracy)
```

```
Train Accuracy: 0.9830793638827686
Test Accuracy: 0.8939760124680957
```

Random Forest

In [45]:

```
# Initializing Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, max_depth=10)

# Fitting the model
rf_model.fit(scaled_X_train, y_train)

# Making Predictions
train_pred = rf_model.predict(scaled_X_train)
test_pred = rf_model.predict(scaled_X_test)

# Evaluate the model on the scaled test data
train_accuracy = accuracy_score(train_pred, y_train)
test_accuracy = accuracy_score(test_pred, y_test)
print('Train Accuracy:', train_accuracy)
print('Test Accuracy:' , test_accuracy)
```

```
Train Accuracy: 0.8755272707336045
Test Accuracy: 0.877668104713934
```

Support Vector Machine

In [44]:

```
# Creating an instance of SGDClassifier with the 'hinge' loss function
svm = SGDClassifier(loss='hinge')

# Defining Parameter Grid
param_grid = {
    'alpha': [0.0001, 0.001, 0.01],
    'penalty': ['l2', 'l1']
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(scaled_X_train, y_train)

# Get the best parameters and best score
best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Best Parameters:", best_params)
print("Best Score:", best_score)
```

```
Best Parameters: {'alpha': 0.01, 'penalty': 'l1'}
Best Score: 0.7834377875867321
```

In [45]:

```
# Use the best model obtained from grid search
best_model = grid_search.best_estimator_

# Predict on the scaled test data
y_pred_test = best_model.predict(scaled_X_test)

# Predict on the scaled training data
y_pred_train = best_model.predict(scaled_X_train)

# Calculate the accuracy of the model on the training data
accuracy_train = accuracy_score( y_train, y_pred_train)
print("Training Accuracy:", accuracy_train)

# Calculate the accuracy of the model on the test data
accuracy_test = accuracy_score(y_test, y_pred_test)
print("Test Accuracy:", accuracy_test)
```

```
Training Accuracy: 0.7828920992485429
Test Accuracy: 0.7545964809251688
```

XGBoost

In [46]:

```
# Create an instance of the XGBoostModel class
xgboost_model = XGBClassifier()

# Train the model
xgboost_model.fit(scaled_X_train, y_train)

# Evaluate the model on the train and test set
train_pred = xgboost_model.predict(scaled_X_train)
test_pred = xgboost_model.predict(scaled_X_test)

# Evaluate the model on the scaled test data
train_accuracy = accuracy_score(train_pred, y_train)
test_accuracy = accuracy_score(test_pred, y_test)
print('Train Accuracy:', train_accuracy)
print('Test Accuracy:' , test_accuracy)
```

```
Train Accuracy: 0.9118785401239751
Test Accuracy: 0.9076864002891153
```

Neural Network

In [52]:

```
# Initializing network
network = Sequential()

# Adding Layers
network.add(Flatten(input_shape=(6, 1))) # Flatten 2D image to 1D vector
network.add(Dense(328, activation='relu'))
network.add(Dense(128,activation='relu'))
network.add(Dense(64,activation='relu'))
network.add(Dense(128,activation='relu'))
network.add(Dense(1, activation='sigmoid'))

# Compiling network
network.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
network.fit(scaled_X_train, y_train, epochs=20, batch_size=32, validation_split=0.1)

# Making binary predictions
y_pred = network.predict(scaled_X_test)
y_pred_binary = (y_pred > 0.5).astype(int)
```

```
Epoch 1/20
32832/32832 [==============================] - 130s 4ms/step - loss:
0.3768 - accuracy: 0.8453 - val_loss: 0.4253 - val_accuracy: 0.8336
Epoch 2/20
32832/32832 [==============================] - 134s 4ms/step - loss:
0.3413 - accuracy: 0.8638 - val_loss: 0.3730 - val_accuracy: 0.8627
Epoch 3/20
32832/32832 [==============================] - 132s 4ms/step - loss:
0.3272 - accuracy: 0.8680 - val_loss: 0.3399 - val_accuracy: 0.8657
Epoch 4/20
32832/32832 [==============================] - 141s 4ms/step - loss:
0.3173 - accuracy: 0.8708 - val_loss: 0.3368 - val_accuracy: 0.8783
Epoch 5/20
32832/32832 [==============================] - 131s 4ms/step - loss:
0.3098 - accuracy: 0.8734 - val_loss: 0.3494 - val_accuracy: 0.8585
Epoch 6/20
32832/32832 [==============================] - 121s 4ms/step - loss:
0.3002 - accuracy: 0.8758 - val_loss: 0.3235 - val_accuracy: 0.8684
Epoch 7/20
```

In [53]:

```
from sklearn.metrics import accuracy_score
accuracy_score(y_pred_binary,y_test)
```

Out[53]:

```
0.8989338874709191
```

Model Selection

Upon running our models, we noted that there was no significant difference when we attempted to tune them, therefore for the sake of readability, we decided to remove the tuned models from this notebook.

We therefore concluded that the best performing model was XGBoost, as it has a Test accuracy of upto **91%**

Pipeline

In the preparation to deploy our model on Streamlit (https://streamlit.io/), we built a pipeline that uses the trained XGBoost model, we then proceeded to create a pickle file.

In [54]:

```
# Declaring the steps in our pipeline
pipeline=Pipeline([
    ('scaler',StandardScaler()),
    ('model', XGBClassifier())
])
```

In [55]:

```
# Fitting the pipeline to the training data
pipeline.fit(X_train,y_train)
```

Out[55]:

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('model',
                 XGBClassifier(base_score=None, booster=None, callback
s=None,
                               colsample_bylevel=None, colsample_bynod
e=None,
                               colsample_bytree=None,
                               early_stopping_rounds=None,
                               enable_categorical=False, eval_metric=N
one,
                               feature_types=None, gamma=None, gpu_id=
None,
                               grow_policy=None, importance_type=None,
                               interaction_constraints=None, learning_
rate=None,
                               max_bin=None, max_cat_threshold=None,
                               max_cat_to_onehot=None, max_delta_step=
None,
                               max_depth=None, max_leaves=None,
                               min_child_weight=None, missing=nan,
                               monotone_constraints=None, n_estimators
=100,
                               n_jobs=None, num_parallel_tree=None,
                               predictor=None, random_state=None,
...))])
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Pickling the pileline

In [56]:

```
# Creating the pickle file for the pipeline
import pickle
filename = 'pipeline.pkl'
pickle.dump(pipeline, open(filename, 'wb'))
```

In [58]:

```
# Creating a new dataframe to encode the industry column
new_df=pd.DataFrame(df.Industry,columns=['Industry'])
new_df['Encoded'] = new_df.Industry.astype('category').cat.codes
```

In [59]:

```
unique_industries=list(pd.DataFrame(new_df.Industry.value_counts()).index)
```

In [61]:

```
encoded_df['Industry_raw']=df['Industry']
industry_df=encoded_df[['Industry','Industry_raw']]
```

In [63]:

```
value_dictionary={}
for i in range(len(industry_df)):
    row=industry_df.iloc[i]
    value_dictionary[row.Industry_raw]=row.Industry
```

In [64]:

```
value_dictionary # Showing the Industry and the corresponding encoded value
```

Out[64]:

```
{'Other': 12,
 'Construction': 4,
 'Prof/Science/Tech': 14,
 'Accom/Food_serv': 0,
 'Manufacturing': 9,
 'Wholesale_trade': 20,
 'Retail_trade': 17,
 'Other_no_pub': 13,
 'Healthcare/Social_assist': 7,
 'Admin_sup/Waste_Mgmt_Rem': 1,
 'Arts/Entertain/Rec': 3,
 'RE/Rental/Lease': 16,
 'Information': 8,
 'Ag/For/Fish/Hunt': 2,
 'Educational': 5,
 'Trans/Ware': 18,
 'Min/Quar/Oil_Gas_ext': 11,
 'Mgmt_comp': 10,
 'Utilities': 19,
 'Finance/Insurance': 6,
 'Public_Admin': 15}
```

Conclusion

In conclusion, the enhanced predictive capabilities of the SBA have revolutionized the loan application process, enabling automated filtration and empowering clients to identify crucial factors influencing their loan eligibility. This advancement not only streamlines the lending process but also fosters a supportive environment for entrepreneurs and small businesses. By encouraging more individuals to seek SBA support, this initiative contributes to reducing unemployment rates and fostering economic growth through increased business opportunities and job creation.

Recommendations

Based on the analysis that we have done, we came up with five main areas in which the SBA can dive deeper into, these areas included:

1. Loan Term

The SBA should consider advising the small businesses on the appropriate loan amount to apply for based on the loan amount and term distribution. This can be achieved by identifying the most common loan sizes and advising small businesses to apply for loans within those ranges. Additionally, we recommend that the SBA identifies the most common loan terms and set loan term limits or eligibility criteria based on those terms.

2. Size of the Business

We advise the SBA to look into the loan amount ranges and/or eligibility criteria for businesses of different sizes based on the employee distribution. Through this, we recommend that they set loan amount limits that would be appropriate for different business sizes.

3. Location

We would recommend the SBA to advise the small businesses to seek loans that are appropriate for their location and the needs of their local community. Additionally, the SBA should ensure the distribution of loans is easily accessible to those in the Rural/Semi-urban regions, in order to encourage development

4. Business Existence Period

Considering whether the business has been in existence for a long or short period of time, we recommend that the SBA advises the businesses seeking loans to apply for loans that are in alignment with their growth plans, additionally, the SBA can recommend and/or implement new training and mentorship programs to help the businesses achieve their targets.

5. Industry

Finally, we would suggest that the SBA advises their members to apply for loans that are tailored to the specific needs of their industry.