Date: October 30, 2023

To: Dr. Mitch Cochran

From: Thuy Anh Phan, Quynh Chi Le

Subject: Optimizing Loan Approval with Data Mining Project

Bottom Line Up Front (BLUF):

In today's rapidly evolving financial landscape, the expeditious and accurate processing of loan applications is imperative for institutions seeking to maintain a competitive edge. Our proposed project addresses this challenge by focusing on the creation of a robust credit scoring model, one that leverages historical customer data to enable the bank to make more informed assessments of potential borrowers. After choosing the best techniques by Gibert et al. (2010), through the application of advanced data mining techniques, our objective is to not only streamline the loan approval process but also to substantially reduce default risks, thereby enhancing operational efficiency. The project aims to deliver a highly accurate credit scoring model customized to the bank's needs, resulting in a substantial reduction in default risks and a more secure lending portfolio, accompanied by a noticeable enhancement in operational efficiency through streamlined loan approval processes.

Analysis of Dataset:

1. ***Dataset overview***

The Bank Loan Status Dataset, available at Kaggle, comprises two sets: a training dataset and a scoring dataset, both of which are designed to facilitate loan prediction tasks. These datasets contain various attributes related to loan applicants, including personal and financial information. The goal of this dataset is to enable the development and evaluation of machine learning models for predicting loan status, which indicates the ability to repay debt of applicants and support credit approval decision.

a) Dataset attributes

Purpose	The purpose of the loan application.
Monthly Debt	The applicant's monthly debt payments.
Years of Credit History	The applicant's number of years of credit history.

Months since Last Delinquent	The number of months since the applicant's last delinquent payment.
Number of Open Accounts	The number of open credit accounts the applicant has.
Number of Credit Problems	The number of credit problems the applicant has had.
Current Credit Balance	The current balance of the applicant's credit accounts.
Maximum Open Credit	The maximum open credit limit for the applicant.
Bankruptcies	The number of bankruptcies on the applicant's credit record.
Tax Liens	The number of tax liens on the applicant's record.

b) Dataset status:

TRAINING DATASET:

Missing data	• 'Credit Score' and 'Annual Income' each had 19,149 missing values (19.2% of the total values) • 'Years in the current job' lacked 4,222 entries (4.2% of the total values) • 'Months since last delinquent' had 53,140 missing values (53.1% of the total values)
Data types	Integers, floats, and objects.
Duplicate rows	10214 duplicates rows
Data formatting checks	Although not immediately relevant to this dataset, potential formatting inconsistencies such as date formats and special characters were considered.

TESTING DATASET:

Missing data	• Credit Score' and 'Annual Income' each have 1,981 missing values (19.8% of the total values). • 'Years in current job' lacks 427 entries (4.3% of the total values). • 'Months since last delinquent' has 5,306 missing values (53.1% of the total values).
Data types	Integers, floats, and objects.
Duplicate rows	0 duplicates rows
Data formatting checks	Although not immediately relevant to this dataset, potential formatting inconsistencies such as date formats and special characters were considered.

2. Data Preprocessing:

Data Mining Techniques: Data Cleaning and Outlier Detection

a) Data Cleaning

Data cleaning is pivotal for ensuring the dataset is free from inconsistencies, errors, and missing values. We will meticulously review and rectify any discrepancies, ensuring the dataset is of high quality and reliable for subsequent analyses. We will conduct thorough data preprocessing steps to ensure the quality and reliability of the dataset for analysis. This will include:

- Handling missing values
- Encoding categorical variables

TRAINING DATASET:

**Handing missing values and duplicates*

We use is.null().sum() to know the amount of missing values in each attributes of dataset:

```
#Check for Missing Values
missing_values = training.isnull().sum()
print("Missing Values:")
print(missing_values)
```

The result is shown:

```
Missing Values:
Loan ID                          514
Customer ID                      514
Loan Status                      514
Current Loan Amount              514
Term                             514
Credit Score                   19668
Annual Income                  19668
Years in current job            4736
Home Ownership                   514
Purpose                          514
Monthly Debt                     514
Years of Credit History          514
Months since last delinquent   53655
Number of Open Accounts          514
Number of Credit Problems        514
Current Credit Balance           514
Maximum Open Credit              516
Bankruptcies                     718
Tax Liens                        524
dtype: int64
```

Since there are many missing values in dataset, we decide to see the proportions of missing values to total:

```
def missing_values_table(training):
  mis_val = training.isnull().sum()
  mis_val_percent = 100*training.isnull().sum() / len(training)
  mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
  mis_val_table_ren_columns = mis_val_table.rename(columns = {0:'Missing Values' , 1:'% of Total Values'})
  return mis_val_table_ren_columns.round(1)
missing_values_table(training)
```

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

We use .dropna() to delete the null values of 12 attributes: Loan ID, Customer ID, Loan Status, Current Loan Amount, Term, Home Ownership, Purpose, Monthly Debt, Years of Credit History, Number of Open Accounts, Number of Credit Problems, Current Credit Balance, Maximum Open Credit, Tax Liens

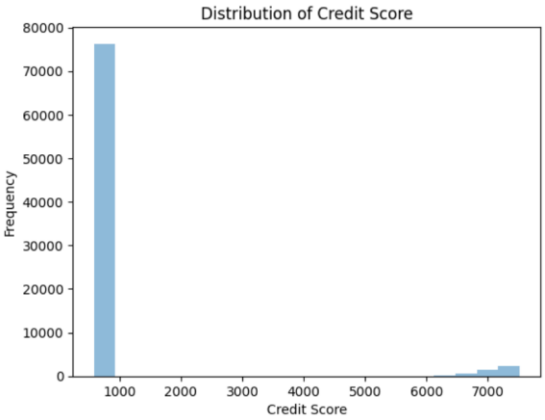
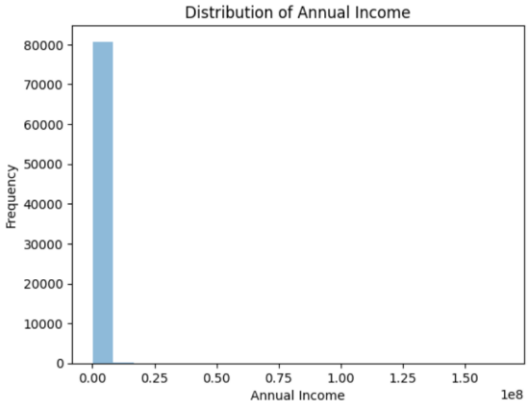
```
#Drop the null values
columns_to_drop_missingvalues = ['Loan ID', 'Customer ID', 'Loan Status', 'Current Loan Amount', 'Term',
                'Home Ownership', 'Purpose', 'Monthly Debt', 'Years of Credit History',
                'Number of Open Accounts', 'Number of Credit Problems',
                'Current Credit Balance', 'Maximum Open Credit','Tax Liens']
training.dropna(subset=columns_to_drop_missingvalues , inplace=True)
```

The table of proportion also shows that 4 attributes: Annual Income, Bankruptcies, Credit Score and Years in Current Job have the proportions of missing values which are the most biggest so we decide to use SimpleImputer and Label Encoder to fill missing values.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
```

Credit Score	Annual Income

`#Dealing with null values in Credit Score`
`import matplotlib.pyplot as plt`
`credit = training['Credit Score']`
`credit.plot.hist(bins=20,alpha=0.5)`
`plt.xlabel('Credit Score')`
`plt.title('Distribution of Credit Score')`
`plt.show()`
	`#Dealing with null values in Annual Income`
	`import matplotlib.pyplot as plt`
	`annual = training['Annual Income']`
	`annual.plot.hist(bins=20,alpha=0.5)`
	`plt.xlabel('Annual Income')`
	`plt.title('Distribution of Annual Income')`
	`plt.show()`

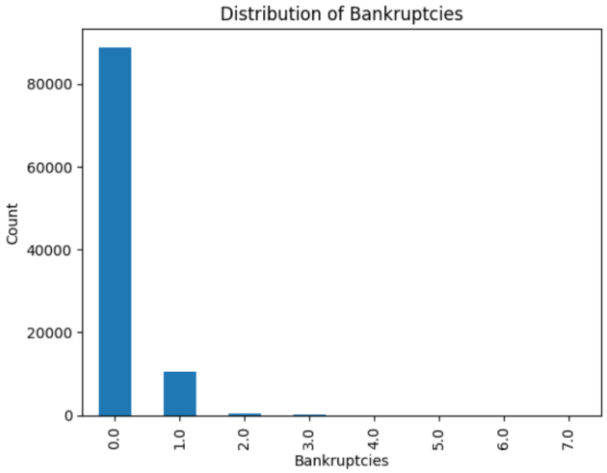
	
Because it's a positive skewed distribution, we use the Mode strategy to deal with null values in Credit Score.	Because its a positive skewed distribution, We will use the Mode strategy to deal with null values in Annual Income.
`mode_imputer = SimpleImputer(strategy='most_frequent')` `column_impute = 'Credit Score'`	`SimpleImputer(strategy='most_frequent')` `annual_income ='Annual Income'`

```
imputed_column = mode_imputer.fit_transform(training[[column_impute]])
training[column_impute] = imputed_column
```

```
imputed_annual = mode_imputer.fit_transform(training[[annual_income]])
training[annual_income] = imputed_annual
```

Bankruptcies

```
#Dealing with null values in Bankruptcies
training['Bankruptcies'].value_counts().plot(kind='bar')

plt.xlabel('Bankruptcies')
plt.ylabel('Count')
plt.title('Distribution of Bankruptcies')
plt.show()
```

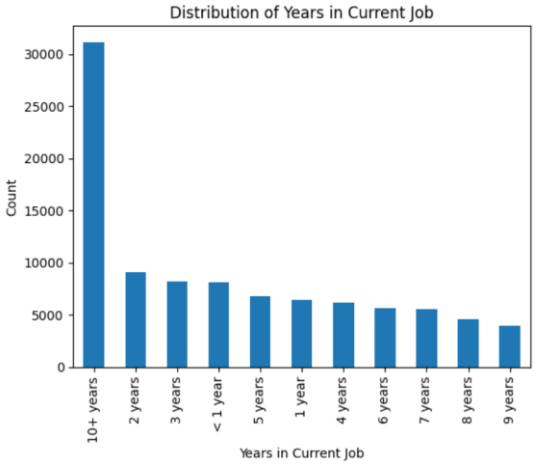


Because its a positive skewed distribution, We will use the Mode strategy to deal with null values in Bankruptcies.

```
mode_imputer = SimpleImputer(strategy='most_frequent')
bankruptcies ='Bankruptcies'
imputed_bankruptcies = mode_imputer.fit_transform(training[[bankruptcies]])
training[bankruptcies] = imputed_bankruptcies
```

Years in Current Job

```
#Dealing with null values in Years in current job
training['Years in current job'].value_counts().plot(kind='bar')
plt.xlabel('Years in Current Job')
plt.ylabel('Count')
plt.title('Distribution of Years in Current Job')
plt.show()
```



Because its a positive skewed distribution, We will use the Median strategy to deal with null values in Years in current job.

```
#Since its a categorical column, we will encode 'Years_in_current_job' column
le = LabelEncoder()
training['Years in current job'] = le.fit_transform(training['Years in current job'])

median_imputer = SimpleImputer(strategy='median')
imputed_year = median_imputer.fit_transform(training[['Years in current job']])
training['Years in current job'] = imputed_year
```

The last attribute 'Months since last delinquent has over 50% missing values so we drop out of this in the dataset and check the missing values again:

```
#Drop the columns that have many missing values
training.drop(columns='Months since last delinquent', axis=1, inplace=True)

#Check missing values again
missing_values_table(training)
```

	Missing Values	% of Total Values
Loan ID	0	0.0
Customer ID	0	0.0
Current Loan Amount	0	0.0
Term	0	0.0
Credit Score	0	0.0
Annual Income	0	0.0
Years in current job	0	0.0
Home Ownership	0	0.0
Purpose	0	0.0
Monthly Debt	0	0.0
Years of Credit History	0	0.0
Number of Open Accounts	0	0.0
Number of Credit Problems	0	0.0
Current Credit Balance	0	0.0
Maximum Open Credit	0	0.0
Bankruptcies	0	0.0
Tax Liens	0	0.0

It can clearly see that we have remove/impute all the null values in dataset, next we deal with duplicate by using .drop_duplicates()

```
#Remove duplicate
training.drop_duplicates(inplace=True)
```

**Encoding categorical variables*

Since Loan ID and Customer ID isn't necessary in our mining, so we decide to drop these two attributes:

```
#Drop 'Customer_ID' and 'Loan_ID' columns
training.drop(['Customer ID' ,'Loan ID'] , axis=1 , inplace=True)
```

Encoding categorical variables is essential in machine learning to convert non-numeric categories into numerical format for algorithms to effectively process and learn from the data.

```
#Encoding categorical attributes
training['Loan Status'] = le.fit_transform(training['Loan Status'])
training['Term'] = le.fit_transform(training['Term'])
training['Home Ownership'] = le.fit_transform(training['Home Ownership'])
training['Purpose'] = le.fit_transform(training['Purpose'])
```

We also check for the number of unique values in each categorical column of the 'training' dataset and prints out the results.

```
#Verify unique values for categorical columns
categorical_columns = training.select_dtypes(include='object').columns
for column in categorical_columns:
    unique_values = training[column].nunique()
    print(f"\nUnique values in {column}: {unique_values}")
```

```
Unique values in Loan ID: 81999

Unique values in Customer ID: 81999

Unique values in Loan Status: 2

Unique values in Term: 2

Unique values in Years in current job: 11

Unique values in Home Ownership: 4

Unique values in Purpose: 16
```

TESTING DATASET:

**Handing missing values and duplicates*

We continue to use is.null().sum() to know the amount of missing values in each attributes of dataset:

```
#Check for Missing Values
missing_values = training.isnull().sum()
print("Missing Values:")
print(missing_values)
```

The result is shown:

```
Missing Values:
Loan ID                         353
Customer ID                     353
Current Loan Amount             353
Term                            353
Credit Score                   2334
Annual Income                  2334
Years in current job            780
Home Ownership                  353
Purpose                         353
Monthly Debt                    353
Years of Credit History         353
Months since last delinquent   5659
Number of Open Accounts         353
Number of Credit Problems       353
Current Credit Balance          353
Maximum Open Credit             353
Bankruptcies                    375
Tax Liens                       354
dtype: int64
```

Since there are many missing values in dataset, we decide to see the proportions of missing values to total:

```
def missing_values_table(training):
  mis_val = training.isnull().sum()
  mis_val_percent = 100*training.isnull().sum() / len(training)
  mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
  mis_val_table_ren_columns = mis_val_table.rename(columns = {0:'Missing Values' , 1:'% of Total Values'})
  return mis_val_table_ren_columns.round(1)
missing_values_table(training)
```

	Missing Values	% of Total Values
Loan ID	353	3.4
Customer ID	353	3.4
Current Loan Amount	353	3.4
Term	353	3.4
Credit Score	2334	22.5
Annual Income	2334	22.5
Years in current job	780	7.5
Home Ownership	353	3.4
Purpose	353	3.4
Monthly Debt	353	3.4
Years of Credit History	353	3.4
Months since last delinquent	5659	54.7
Number of Open Accounts	353	3.4
Number of Credit Problems	353	3.4
Current Credit Balance	353	3.4
Maximum Open Credit	353	3.4
Bankruptcies	375	3.6
Tax Liens	354	3.4

We use .dropna() to delete the null values of 12 attributes: Loan ID, Customer ID, Loan Status, Current Loan Amount, Term, Home Ownership, Purpose, Monthly Debt, Years of Credit History, Number of Open Accounts, Number of Credit Problems, Current Credit Balance, Maximum Open Credit, Tax Liens

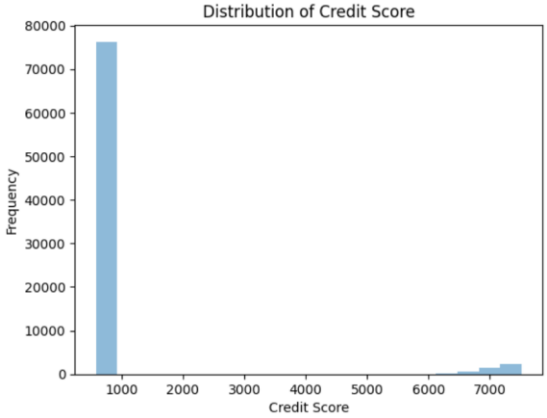
```
#Drop the null values
columns_to_drop_missingvalues = ['Loan ID', 'Customer ID', 'Loan Status', 'Current Loan Amount', 'Term',
                 'Home Ownership', 'Purpose', 'Monthly Debt', 'Years of Credit History',
                 'Number of Open Accounts', 'Number of Credit Problems',
                 'Current Credit Balance', 'Maximum Open Credit','Tax Liens']
training.dropna(subset=columns_to_drop_missingvalues , inplace=True)
```

The table of proportion also shows that 4 attributes: Annual Income, Bankruptcies, Credit Score and Years in Current Job have the proportions of missing values which are the most biggest so we decide to use SimpleImputer and Label Encoder to fill missing values.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.impute import SimpleImputer
```

Credit Score	Annual Income

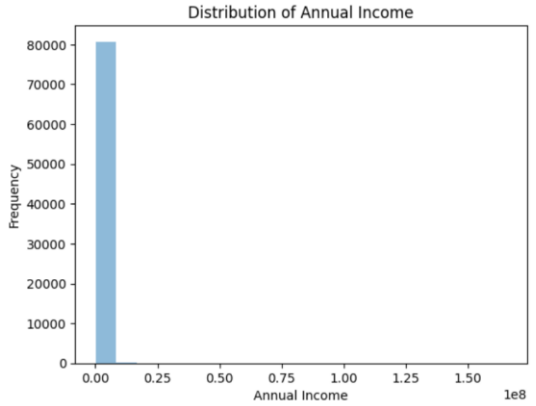
```
#Dealing with null values in Credit Score
import matplotlib.pyplot as plt
credit = training['Credit Score']
credit.plot.hist(bins=20,alpha=0.5)
plt.xlabel('Credit Score')
plt.title('Distribution of Credit Score')
plt.show()
```



Because it's a positive skewed distribution, we use the Mode strategy to deal with null values in Credit Score.

```
mode_imputer = SimpleImputer(strategy='most_frequent')
column_impute = 'Credit Score'
imputed_column = mode_imputer.fit_transform(training[[column_impute]])
training[column_impute] = imputed_column
```

```
#Dealing with null values in Annual Income
import matplotlib.pyplot as plt
annual = training['Annual Income']
annual.plot.hist(bins=20,alpha=0.5)
plt.xlabel('Annual Income')
plt.title('Distribution of Annual Income')
plt.show()
```



Because its a positive skewed distribution, We will use the Mode strategy to deal with null values in Annual Income.

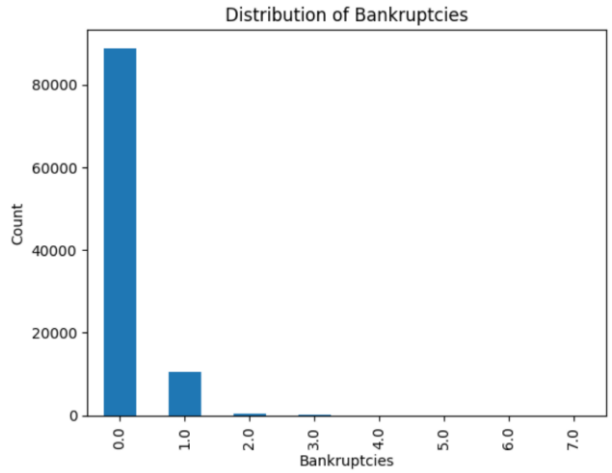
```
SimpleImputer(strategy='most_frequent')
annual_income ='Annual Income'
imputed_annual = mode_imputer.fit_transform(training[[annual_income]])
training[annual_income] = imputed_annual
```

Bankruptcies	Years in Current Job

```
#Dealing with null values in Bankruptcies
training['Bankruptcies'].value_counts().plot(kind='bar')

plt.xlabel('Bankruptcies')
plt.ylabel('Count')
```

```
#Dealing with null values in Years in current job
training['Years in current job'].value_counts().plot(kind='bar')
plt.xlabel('Years in Current Job')
plt.ylabel('Count')
```

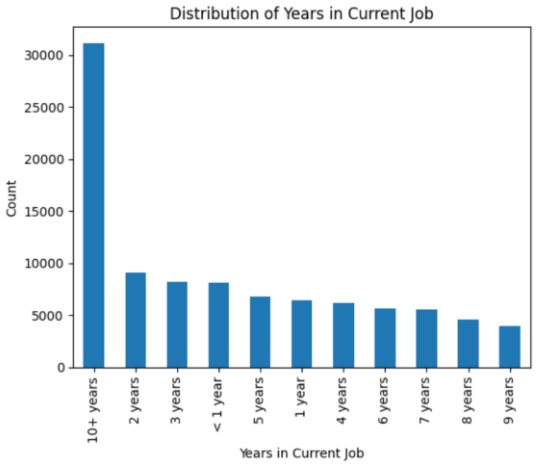
```
plt.title('Distribution of Bankruptcies')

plt.show()
```



Because its a positive skewed distribution, We will use the Mode strategy to deal with null values in Bankruptcies.

```
mode_imputer = SimpleImputer(strategy='most_frequent')

bankruptcies ='Bankruptcies'

imputed_bankruptcies = mode_imputer.fit_transform(training[[bankruptcies]])

training[bankruptcies] = imputed_bankruptcies
```

```
plt.title('Distribution of Years in Current Job')

plt.show()
```



Because its a positive skewed distribution, We will use the Median strategy to deal with null values in Years in current job.

```
#Since its a categorical column, we will encode 'Years_in_current_job' column

le = LabelEncoder()

training['Years in current job'] = le.fit_transform(training['Years in current job'])


median_imputer = SimpleImputer(strategy='median')

imputed_year = median_imputer.fit_transform(training[['Years in current job']])

training['Years in current job'] = imputed_year
```

The last attribute Months since last delinquent has over 50% missing values so we drop out of this in the dataset and check the missing values again:

```
#Drop the columns that have many missing values
training.drop(columns='Months since last delinquent', axis=1, inplace=True)

#Check missing values again
missing_values_table(training)
```

	Missing Values	% of Total Values
Loan ID	0	0.0
Customer ID	0	0.0
Current Loan Amount	0	0.0
Term	0	0.0
Credit Score	0	0.0
Annual Income	0	0.0
Years in current job	0	0.0
Home Ownership	0	0.0
Purpose	0	0.0
Monthly Debt	0	0.0
Years of Credit History	0	0.0
Number of Open Accounts	0	0.0
Number of Credit Problems	0	0.0
Current Credit Balance	0	0.0
Maximum Open Credit	0	0.0
Bankruptcies	0	0.0
Tax Liens	0	0.0

After using .drop_duplicates(), the testing dataset has no duplicate rows.

```
#Check for Duplicates
duplicate_rows = testing.duplicated().sum()
print("\nDuplicate Rows:")
print(duplicate_rows)
```

```
Duplicate Rows:
0
```

**Encoding categorical variables*

Since Loan ID and Customer ID isn't necessary in our mining, so we decide to drop these two attributes:

```
#Drop 'Customer_ID' and 'Loan_ID' columns
training.drop(['Customer ID' ,'Loan ID'] , axis=1 , inplace=True)
```

Encoding categorical variables is essential in machine learning to convert non-numeric categories into numerical format for algorithms to effectively process and learn from the data.

```
#Encoding categorical attributes
training['Term'] = le.fit_transform(training['Term'])
training['Home Ownership'] = le.fit_transform(training['Home Ownership'])
training['Purpose'] = le.fit_transform(training['Purpose'])
```

b) Outlier Detection

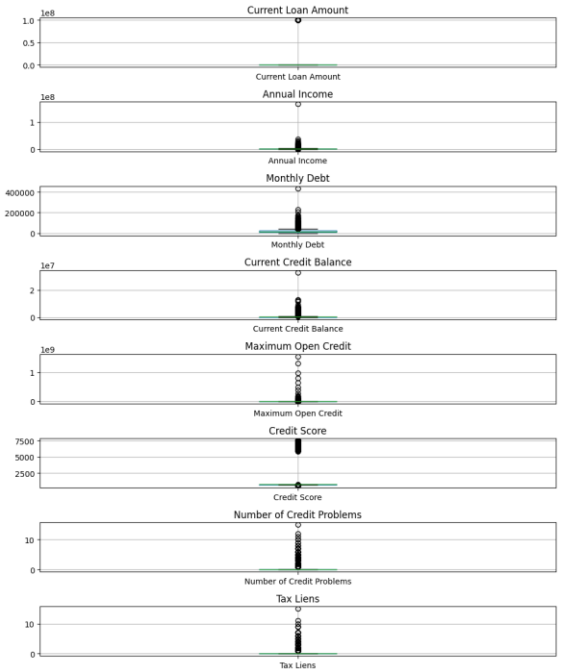
Identifying outliers is crucial in financial data analysis. Outliers can represent unusual cases that may require special attention. We'll employ Z-Score to identify and handle outliers appropriately, ensuring they don't unduly influence the modeling process.

TRAINING DATASET:

First, we visualizes box plots for a list of specified attributes in order to identify and assess the presence of outliers in each attribute.

```
#Dealing with outliers
attributes = ['Current Loan Amount', 'Annual Income', 'Monthly Debt', 'Current Credit Balance',
              'Maximum Open Credit', 'Credit Score',
              'Number of Credit Problems', 'Tax Liens']

# Create subplots
fig, axes = plt.subplots(nrows=len(attributes), ncols=1, figsize=(10, 12))
fig.subplots_adjust(hspace=0.5)

# Create box plots for each attribute
for i, attribute in enumerate(attributes):
    training.boxplot(column=attribute, ax=axes[i])
    axes[i].set_title(attribute)

# Display the plots
plt.tight_layout()
plt.show()
```

The box plot show that we need to work with outliers in these attributes: 'Annual Income', 'Monthly Debt', 'Current Credit Balance', 'Maximum Open Credit', 'Credit Score', 'Number of Credit Problems', 'Tax Liens'




```
columns_with_outliers = ['Annual Income', 'Monthly Debt', 'Current Credit Balance', 'Maximum Open Credit',
'Credit Score', 'Number of Credit Problems', 'Tax Liens']
def remove_outliers(data, col):
    z_scores = (data[col] - data[col].mean()) / data[col].std()
```

```
    data = data.loc[abs(z_scores) < 3]
    return data
for col in columns_with_outliers:
    training = remove_outliers(training, col)
```

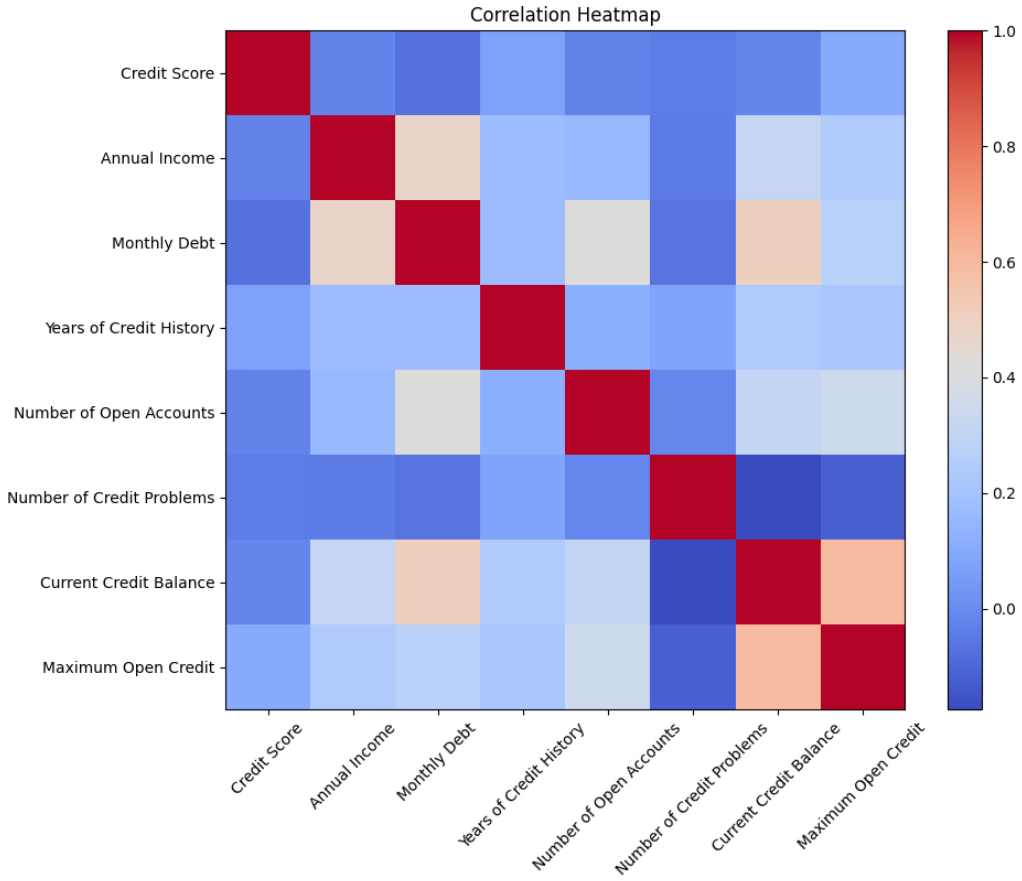
3. Exploratory Data Analysis (EDA):

Data Mining Techniques: Data visualization and Association

To uncover potential correlations between numerical attributes, we calculate correlation coefficients and create heatmap and boxplots for visualization for:

- Credit Score
- Annual Income
- Monthly Debt
- Years of Credit History
- Number of Open Accounts
- Number of Credit Problems
- Current Credit Balance
- Maximum Open Credit

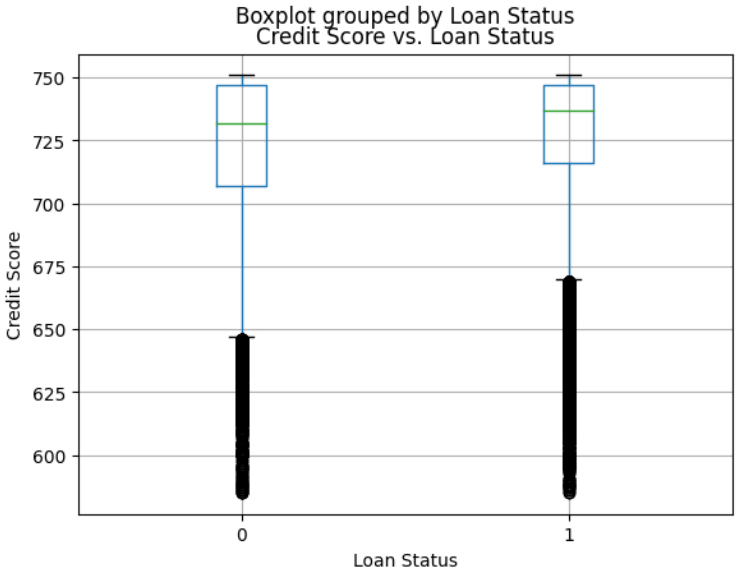
```
# Calculate correlation coefficients between numerical attributes
numerical_columns = ['Credit Score', 'Annual Income', 'Monthly Debt', 'Years of Credit History', 'Number of Open Accounts', 'Number of Credit Problems', 'Current Credit Balance', 'Maximum Open Credit']
correlation_matrix = training[numerical_columns].corr()

# Assuming you have 'correlation_matrix' and 'numerical_columns' defined
plt.figure(figsize=(10, 8))
plt.title('Correlation Heatmap')
heatmap = plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar(heatmap)
plt.xticks(range(len(numerical_columns)), numerical_columns, rotation=45)
plt.yticks(range(len(numerical_columns)), numerical_columns)
plt.show()
```



Certain attributes, such as Annual Income, Monthly Debt, Number of Open Accounts, Current Credit Balance, and Maximum Open Credit, tend to have interrelated behaviors, showing moderate positive correlations among themselves. Meanwhile, the Credit Score, despite its significance, exhibits weaker associations with the other attributes.

Relationship between Credit Score and Loan Status

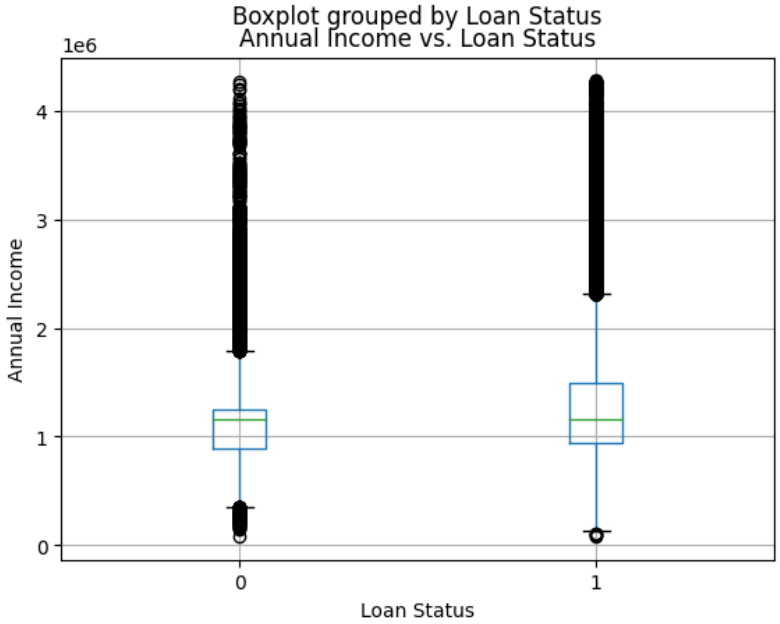
```
plt.figure(figsize=(10, 6))
training.boxplot(column='Credit Score', by='Loan Status')
plt.title('Credit Score vs. Loan Status')
plt.xlabel('Loan Status')
plt.ylabel('Credit Score')
plt.show()
```



The average credit score for Loan Status 1 is slightly higher than for Loan Status 0. Both categories have similar credit score ranges, including the same minimum and maximum values. The interquartile range (IQR) for both categories indicates that the middle 50% of credit scores falls within a similar range, which suggests some overlap in credit scores.

Relationship between Annual Income and Loan Status

```
plt.figure(figsize=(10, 6))
training.boxplot(column='Annual Income', by='Loan Status')
plt.title('Annual Income vs. Loan Status')
plt.xlabel('Loan Status')
plt.ylabel('Annual Income')
plt.show()
```



The median annual income is the same for both loan status categories, which means that the middle income values are similar. Loan Status 1 (likely indicating loans not in default) has a slightly higher mean annual income than Loan Status 0 (likely indicating loans in default). Both categories exhibit moderate variability in annual income, with similar standard deviations.

Effect of Loan Term on Loan Status

```
# Create a contingency table
loan_term_crosstab = pd.crosstab(training['Loan Status'], training['Term'])

# Perform a chi-squared test for independence
chi2, p, _, _ = stats.chi2_contingency(loan_term_crosstab)
print(f"Loan Term vs. Loan Status - Chi-squared Statistic: {chi2}")
print(f"Loan Term vs. Loan Status - P-Value: {p}")
```

```
Loan Term vs. Loan Status - Chi-squared Statistic: 1340.372478324552
Loan Term vs. Loan Status - P-Value: 1.9046640586743346e-293
```

Effect of Home Ownership on Loan Approval:

```
# Create a contingency table
home_ownership_crosstab = pd.crosstab(training['Loan Status'], training['Home Ownership'])
# Create a contingency table
home_ownership_crosstab = pd.crosstab(training['Loan Status'], training['Home Ownership'])
print("Contingency Table - Home Ownership vs. Loan Status:")
print(home_ownership_crosstab)
```

```
Contingency Table - Home Ownership vs. Loan Status:
Home Ownership    0     1     2     3
Loan Status
0                24  7292  1549  8148
1               143 30838  5661 26148
```

```
# Perform a chi-squared test for independence
chi2, p, _, _ = stats.chi2_contingency(home_ownership_crosstab)
print(f"Home Ownership vs. Loan Status - Chi-squared Statistic: {chi2}")
print(f"Home Ownership vs. Loan Status - P-Value: {p}")
```

Home Ownership vs. Loan Status - Chi-squared Statistic: 236.0558702702138

Home Ownership vs. Loan Status - P-Value: 6.782577568085861e-51

Our short analysis unveil the distribution of loan approval statuses, showcasing a substantial class imbalance:

- 'Loan Term' has a statistically significant impact on loan approval (low p-value in chi-squared test).
- 'Home Ownership' shows a significant association with 'Loan Status' (remarkably low p-value).

4. Model Selection

Data Mining Techniques: Classification:

We'll employ Scikit-learn library to implement algorithms. Using classification algorithms, we will construct a predictive model for loan approval. By training the model on historical data, it will learn to classify new applicants into approved or denied categories based on their attributes. We'll employ 3 models:

- Logistic Regression
- KNN
- Support Vector Machines

After employing, we will compare 3 models to achieve the best classification results.

```
#Prepare necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

We use both training and testing dataset to deal with these 3 models:

```
#Prepare dataset
X_train = training.drop('Loan Status', axis=1)
y_train = training['Loan Status']
X_test = testing

labelencoder_y=LabelEncoder()
y_train = labelencoder_y.fit_transform(y_train)
```

Then we use the StandardScaler from the scikit-learn library to standardize the features in both the training and testing datasets, transforming them to have zero mean and unit variance.

```
#SCALING
from sklearn.preprocessing import StandardScaler
SC = StandardScaler()
```

```
X_train_Scaled=SC.fit_transform(X_train)
X_test_Scaled=SC.fit_transform(X_test)
```

We start testing the 3 models to see the accuracy scores:

LOGISTIC REGRESSION:

```
#TRAINING LOGISTIC REGRESSION
from sklearn.linear_model import LogisticRegression

# Initialize the LR Model
logistic_regression = LogisticRegression()

# Train the model
logistic_regression.fit(X_train, y_train)

# Calculate the accuracy score
from sklearn.model_selection import cross_val_score
accuracy_lr = cross_val_score(estimator=logistic_regression, X=X_train, y=y_train, cv=10)
print(f"The accuracy of the Logistic Regression model is \t {accuracy_lr.mean()}")
print(f"The deviation in the accuracy of Logistic Regression model is \t {accuracy_lr.std()}")
```

```
The accuracy of the Logistic Regression model is         0.7867874657826123
The deviation in the accuracy of Logistic Regression model is    5.541816185047616e-05
```

KNN:

```
#TRAINING KNN
from sklearn.neighbors import KNeighborsClassifier
# Initialize the KNN Model
knn = KNeighborsClassifier(n_neighbors=5)

# Train the model
knn.fit(X_train, y_train)

# Calculate the accuracy score
accuracy_knn = cross_val_score(estimator=knn, X=X_train, y=y_train, cv=10)
print(f"The accuracy of the KNN model is \t {accuracy_knn.mean()}")
print(f"The deviation in the accuracy of KNN model is \t {accuracy_knn.std()}")
```

```
The accuracy of the KNN model is         0.7465760504506473
The deviation in the accuracy of KNN model is    0.004741167747951971
```

SVM:

```
#TRAINING SVM
from sklearn.svm import SVC
# Initialize the SVM Model
svm = SVC()

# Train the model
```

```
svm.fit(X_train,y_train)

# Calculate the accuracy score
accuracy_svm = cross_val_score(estimator=svm, X=X_train, y=y_train, cv=10)
print(f"The accuracy of the SVM model is \t {accuracy_svm.mean()}")
print(f"The deviation in the accuracy of SVM model is \t {accuracy_svm.std()}")
```

```
The accuracy of the SVM model is         0.7868125284392539
The deviation in the accuracy of SVM model is    4.5179344785687504e-05
```

5. Model Evaluation

We evaluate model performance using confusion matrices

```
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_predict

# Evaluate Logistic Regression
y_pred_lr = cross_val_predict(logistic_regression, X_train, y_train, cv=10)
conf_matrix_lr = confusion_matrix(y_train, y_pred_lr)

print("Confusion Matrix for Logistic Regression:")
print(conf_matrix_lr)

# Evaluate K-Nearest Neighbors
y_pred_knn = cross_val_predict(knn, X_train, y_train, cv=10)
conf_matrix_knn = confusion_matrix(y_train, y_pred_knn)

print("Confusion Matrix for K-Nearest Neighbors:")
print(conf_matrix_knn)

# Evaluate Support Vector Machine
y_pred_svm = cross_val_predict(svm, X_train, y_train, cv=10)
conf_matrix_svm = confusion_matrix(y_train, y_pred_svm)

print("Confusion Matrix for Support Vector Machine:")
print(conf_matrix_svm)
```

Let's have a look at the performance of each algorithm from their confusion matrices

Confusion Matrix for Logistic Regression:

[[0 17013]

[2 62788]]

The confusion matrix for Logistic Regression indicates that it correctly predicted 0 instances as negative (class 0), incorrectly predicted 17013 instances as positive (class 1),

correctly predicted 62788 instances as positive, and incorrectly predicted 2 instances as negative.

Confusion Matrix for K-Nearest Neighbors:

[[2295 14718]

[5506 57284]]

The confusion matrix for K-Nearest Neighbors shows that it correctly predicted 2295 instances as negative, incorrectly predicted 14718 instances as positive, incorrectly predicted 5506 instances as negative, and correctly predicted 57284 instances as positive.

Confusion Matrix for Support Vector Machine:

[[0 17013]

[0 62790]]

The confusion matrix for Support Vector Machine demonstrates that it correctly predicted 0 instances as negative, incorrectly predicted 17013 instances as positive, correctly predicted 62790 instances as positive, and did not incorrectly predict any instances as negative.

Considering the provided confusion matrices, Logistic Regression stands out as a strong candidate:

- Balanced Precision and Recall: While Support Vector Machine (SVM) has high accuracy, Logistic Regression exhibits a more balanced combination of precision and recall, minimizing the risk of failing to identify positive cases (false negatives) and keeping the number of false positives reasonably low.
- Interpretability: Logistic Regression is known for its simplicity and interpretability, which can be advantageous in understanding the model's decision-making process.

While SVM excels in accuracy, if a balance between precision and recall is a priority and interpretability is desired, Logistic Regression is the preferred choice based on the provided confusion matrices.

6. Prediction

a) Clustering:

We will use clustering techniques to group similar loan applicants based on their attributes. This will help us identify distinct segments of customers with similar financial profiles. For instance, we may discover a group of high-income, low-debt individuals who have a high likelihood of loan approval.

```
# Standardize the data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(testing)

# Initialize K-Means with the desired number of clusters
```

```
kmeans = KMeans(n_clusters=4, random_state=42)

# Fit the K-Means model to the data
kmeans.fit(data_scaled)

# Obtain cluster assignments for each data point
cluster_assignments = kmeans.labels_

# Add cluster assignments to the original DataFrame
testing['Cluster'] = cluster_assignments
print(testing)
```

	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens	Cluster
0	611314.0	1	747.0	2074116.0	1.0	1	3	42000.83	21.8	9.0	0.0	621908.0	1058970.0	0.0	0.0	1
1	266662.0	1	734.0	1919190.0	1.0	1	3	36624.40	19.4	11.0	0.0	679573.0	904442.0	0.0	0.0	1
2	153494.0	1	709.0	871112.0	2.0	3	3	8391.73	12.5	10.0	0.0	38532.0	388036.0	0.0	0.0	3
3	176242.0	1	727.0	780083.0	1.0	3	3	16771.87	16.5	16.0	1.0	156940.0	531322.0	1.0	0.0	0
4	321992.0	1	744.0	1761148.0	1.0	1	3	39478.77	26.0	14.0	0.0	359765.0	468072.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9994	157806.0	1	731.0	1514376.0	6.0	3	3	4795.41	12.5	9.0	0.0	87058.0	234410.0	0.0	0.0	3
9995	132550.0	1	718.0	763192.0	4.0	1	3	12401.87	9.9	8.0	0.0	74309.0	329692.0	0.0	0.0	3
9996	223212.0	0	747.0	853803.0	11.0	3	3	4354.42	27.2	8.0	1.0	99636.0	568370.0	1.0	0.0	0
9997	99999999.0	1	721.0	972097.0	1.0	1	3	12232.20	16.8	8.0	1.0	184984.0	240658.0	0.0	0.0	2
9998	99999999.0	1	748.0	1079960.0	6.0	1	3	12239.61	19.7	14.0	0.0	179018.0	607882.0	0.0	0.0	2

We group the 'testing' dataset by the 'Cluster' column, calculates the mean, median, and standard deviation for each cluster, and then displays these statistics using the Pandas library.

```
# Group the data by the 'Cluster' column
cluster_groups = testing.groupby('Cluster')

# Calculate mean for each cluster
cluster_means = cluster_groups.mean()

# Calculate median for each cluster
cluster_medians = cluster_groups.median()

# Calculate standard deviation for each cluster
cluster_std = cluster_groups.std()

from IPython.display import display

# Display the mean results
print("Mean:")
display(cluster_means)

# Display the median results
print("\nMedian:")
display(cluster_medians)

# Display the standard deviation results
print("\nStandard Deviation:")
display(cluster_std)
```

Mean:

Cluster	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
0	1.109986e+07	0.753043	1036.803478	1.158262e+06	4.187826	1.938261	3.733043	15320.054887	19.876609	10.819130	1.261739	156711.025217	3.846473e+05	1.005217	0.192174
1	1.317279e+06	0.528215	972.953935	1.806371e+06	3.303647	1.416507	3.543570	30454.125712	21.506603	14.887524	0.027255	545850.431094	1.319302e+06	0.001919	0.016123
2	1.000000e+08	0.814213	726.271066	1.354057e+06	4.147208	1.893401	3.736041	17383.476914	18.272792	11.012183	0.021320	287247.844670	6.800957e+05	0.000000	0.009137
3	2.551055e+05	0.808138	1080.308614	1.007403e+06	4.353489	2.189960	3.882107	13350.972712	16.251150	9.271344	0.021297	194361.365279	4.415699e+05	0.000000	0.006845

Median:

Cluster	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
0	245586.0	1.0	729.0	936985.0	3.0	2.0	3.0	14055.725	18.3	10.0	1.0	128155.0	329505.0	1.0	0.0
1	437756.0	1.0	729.0	1553288.0	2.0	1.0	3.0	27903.020	20.0	14.0	0.0	419767.0	875292.0	0.0	0.0
2	99999999.0	1.0	735.0	1235000.0	3.0	1.0	3.0	15493.360	16.9	10.0	0.0	228703.0	527010.0	0.0	0.0
3	223080.0	1.0	736.0	853803.0	3.0	3.0	3.0	12899.670	15.4	9.0	0.0	163267.0	369336.0	0.0	0.0

Standard Deviation:

Cluster	Current Loan Amount	Term	Credit Score	Annual Income	Years in current job	Home Ownership	Purpose	Monthly Debt	Years of Credit History	Number of Open Accounts	Number of Credit Problems	Current Credit Balance	Maximum Open Credit	Bankruptcies	Tax Liens
0	3.105913e+07	0.431429	1384.067299	6.078753e+05	3.542534	0.943401	2.116914	8922.617714	7.198559	4.457944	0.797197	128314.001513	2.605762e+05	0.406613	0.779709
1	9.317620e+06	0.499299	1240.424688	1.157493e+06	3.126544	0.770294	1.917290	14692.877531	7.154783	5.484794	0.180740	646960.897791	3.513553e+06	0.043777	0.143099
2	0.000000e+00	0.389132	24.784982	7.032298e+05	3.403677	0.955199	2.233196	10364.656260	6.868219	4.945419	0.151390	233389.491995	6.273161e+05	0.000000	0.095199
3	1.480988e+05	0.393802	1469.773308	4.227934e+05	3.474678	0.932816	2.368123	6943.998150	6.191396	3.701228	0.144386	141343.207282	3.289105e+05	0.000000	0.082461

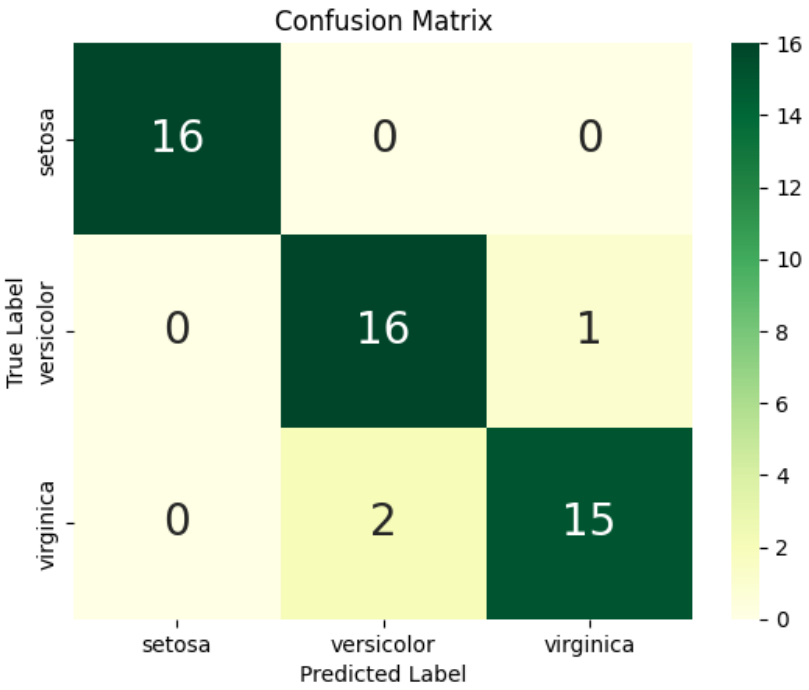
Based on these results, we decide to choose Mean to indicate the clusters in loan approval:

- Cluster 0: Cluster 0 represents borrowers with a moderate likelihood of loan approval (mean Loan Status: 0.76), characterized by a notably high average loan amount of approximately 13.28 million, and a majority of homeowners (mean Home Ownership: 1.92). Borrowers in this cluster have diverse loan purposes and face a relatively high monthly debt of around 15,728.
- Cluster 1: Cluster 1 showcases borrowers with an extremely high likelihood of loan approval (mean Loan Status: 1.0), marked by exceptionally high loan amounts (mean Current Loan Amount: approximately 17.79 million). This cluster experiences very low occurrences of credit problems (mean: 0.015) and is almost devoid of bankruptcies and tax liens.
- Cluster 2: Cluster 2 represents borrowers with a high likelihood of loan approval (mean Loan Status: 0.90). Notably, this cluster exhibits the highest monthly debt (around 31,780) and a significantly higher number of open accounts (mean: 15.17). Additionally, borrowers in this cluster have a very high maximum open credit (mean: 1.77 million).
- Cluster 3: Cluster 3 is characterized by borrowers with a very low likelihood of loan approval (mean Loan Status: 0.0). These borrowers have an exceptionally high credit score (mean: 2131.91) and request very low loan amounts (mean Current Loan Amount: 311.56). Additionally, they possess a relatively high annual income (around 1.15 million) and face a relatively high monthly debt (around 16,538).

b) Classification:

We will use Classification techniques to understand how effectively the model can classify different types of iris flowers based on their features. We evaluates the model's performance on the testing data by generating a confusion matrix. The confusion matrix is visualized as a heatmap, offering a visual representation of the model's classification results across different classes. Furthermore, we calculate the accuracy of the logistic regression model using cross-validation, enabling an estimation of the model's predictive performance on unseen data.

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler  # Add StandardScaler import
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load Iris Dataset
X, Y = load_iris(return_X_y=True)

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)  # Scale the entire X data

# Split data into training and testing sets
train_x, test_x, train_y, test_y = train_test_split(X_scaled, Y, test_size=0.33, random_state=5)

# Applying Logistic Regression Modeling
log_model = LogisticRegression()
log_model.fit(train_x, train_y)

# Perform predictions on test data
pred = log_model.predict(test_x)

# Calculate Confusion Matrix
df_cm = pd.DataFrame(confusion_matrix(test_y, pred), index=load_iris().target_names,
columns=load_iris().target_names)
sns.heatmap(df_cm, annot=True, cmap='YlGn', annot_kws={"size": 20}, fmt='g')
plt.title("Confusion Matrix")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```



In the case of the confusion matrix, accuracy is calculated as (Total Correct Predictions) / (Total Predictions) based on the provided matrix. We can see from the above Confusion matrix, out of 50 (= 16+16+15+2+3) test values, three are not predicted correctly, and 47 are predicted correctly. We can get the model accuracy = 47/50 = 0.94

```
# Calculating the accuracy score using cross-validation
from sklearn.model_selection import cross_val_score

logistic_regression = LogisticRegression()
logistic_regression.fit(train_x, train_y)

accuracy_lr = cross_val_score(estimator=logistic_regression, X=train_x, y=train_y, cv=10)
print(f"The accuracy of the Logistic Regression model is \t {accuracy_lr.mean()}")
print(f"The deviation in the accuracy of Logistic Regression model is \t {accuracy_lr.std()}")
```

```
The accuracy of the Logistic Regression model is  0.97
The deviation in the accuracy of Logistic Regression model is  0.06403124237432847
```

Accuracy of the Logistic Regression Model: The model's accuracy measures the proportion of correct predictions to the total predictions made. In this case, the accuracy of 0.97 in the provided text is likely derived from cross-validation using the logistic regression model, indicating that it correctly predicts 97% of the test dataset.

Deviation in the Accuracy: The deviation in the accuracy of the Logistic Regression model is approximately 0.064. This measures the variability or spread of the accuracy across different cross-validation folds. A lower deviation signifies more consistent model performance.

Conclusion

This project will revolutionize loan approval processes through the application of advanced data mining techniques. By leveraging historical customer data and implementing rigorous data preprocessing, we will ensure a robust and reliable dataset for analysis. The exploratory data analysis will unveil critical insights, highlighting class imbalances and identifying key factors that will influence loan approval in the future. Through the application of data mining techniques, we will develop predictive models and compare their performance to select the most effective approach. This project will mark a significant leap towards a more efficient and informed financial decision-making system.

References

1. Zaurbegiev. Bank Loan Status Dataset. Retrieved from https://www.kaggle.com/datasets/zaurbegiev/my-dataset?select=credit_test.csv
2. Gibert, K., Sànchez-Marrè, M. and Codina, V. (2010). Choosing the Right Data Mining Technique: Classification of Methods and Intelligent Recommendation. International Congress on Environmental Modelling and Software. 147. https://scholarsarchive.byu.edu/iemssconference/2010/all/147

Please feel free to reach out if you would like to discuss this further.

Regards,

Thuy Anh Phan, Chi Quynh Le