LOAN DEFAULT PREDICTION USING MACHINE LEARNING

Vamshi Krishna Katika(G01386372), Varsha Monala(G01408810), Sai Anirudh Reddy(G01388477)

Team 03, GMU ECE552 Big Data Technologies, Fall 2023

Abstract: The project aims to tackle loan defaults, a major financial risk for banks, by using data science methodologies like logistic regression and random forest in a binary classification framework. The goal is to improve decision-making processes for banks, enabling proactive engagement with borrowers at higher risk of default. The project uses precision, recall, F1-score, and AUC-ROC metrics to predict loan defaults and guide targeted marketing strategies. These strategies aim to convert potential risks into sustainable customer relationships and financial stability, contributing positively to the economic landscape.

I. INTRODUCTION

This project aims to tackle the financial losses caused by customer loan defaults, which negatively impact the country's economic growth. Using a data science approach, the project analyzes extensive loan data to identify key factors contributing to defaults. Techniques like logistic regression and decision trees are used to predict loan default likelihood. The insights will guide banks in making informed lending decisions and develop a strategic marketing plan to minimize losses. The project also aims to refine the identification of high-risk borrowers, enabling banks to engage proactively. This proactive engagement is expected to reduce defaults, safeguard financial institutions' interests, and contribute to economic stability.

II. HARDWARE

The entire project has been performed and executed in an Azure Virtual Machine, whose specifications and configurations are as listed below:

Specification	Value
Operating System	Windows 10 Pro
Number of CPU Cores	4
RAM	16

Table 1: Hardware used by VM.

III. SOFTWARE USED

Using PySpark technology, the entire project was carried out in Jupyter Notebook. In addition, MongoDb Comapss was used to import the data utilized for the analysis. Most of the data analysis and modeling within Jupyter Notebook has been performed with PySpark. As PySpark does not support complex visualizations due to which the Pandas, Matplotlib and few other libraries have been used exclusively for visualizations.

IV. DATASET: [1]

This dataset was sourced from Kaggle which consists of data 67,463 rows and 35 columns offers a comprehensive overview of loan applications. Each record is uniquely identified by an 'ID' and includes financial and personal information about the borrower. It includes critical credit history metrics like 'Debit to Income', 'Delinquency in the past two years', 'Credit Inquiries in the last six months', and 'Total Accounts' to assess applicant creditworthiness. The dataset also encapsulates loan characteristics like 'Term', 'Grade', 'Sub Grade', 'Loan Title', 'Verification Status', and 'Initial List Status'. It also provides insights into the borrower's current financial standing, such as 'Total Current Balance', 'Total Revolving Credit Limit', 'Debt-to-Income Ratio', and 'Credit Utilization Ratio'.

Information on the individual attributes can be found in the table below:

Attribute Name	Description
ID	unique identifier for each loan application
Loan Amount	the amount requested by the borrower
Funded Amount	the amount funded by investors for the loan
Funded Amount Investor	the amount funded by investors for the loan
Term	the length of the loan term in months
Batch Enrolled	the batch enrollment method for the loan
Interest Rate	the interest rate on the loan
Grade	the Lending Club assigned loan grade
Sub Grade	the Lending Club assigned loan subgrade
Employment Duration	the employment length of the borrower
Home Ownership	the type of home ownership of the borrower
Verification Status	indicates if the borrower's income was verified
Loan Title	the title of the loan as provided by the borrower
Debit to Income	the borrower's debt-to-income ratio
Delinquency - two years	the number of times the borrower has been delinquent in the past two years
Inquires - six months	the number of inquiries made on the borrower's credit in the past six months
Open Account	the number of open credit lines in the borrower's credit file
Public Record	the number of derogatory public records on the borrower's credit file
Revolving Balance	the balance on the borrower's revolving credit accounts
Revolving Utilities	the amount of the borrower's revolving credit that is currently in use
Total Accounts	the total number of credit lines the borrower has
Initial List Status	the initial listing status of the loan
Total Received Interest	total interest received to date
Total Received Late Fee	total late fees received to date
Recoveries	post charge off gross recovery
Collection Recovery Fee	post charge off collection fee
Collection 12 months Medical	number of collections in medical categories in the last 12 months
Application Type	indicates whether the loan is an individual or joint application
Last week Pay	the last week's payment on the loan
Total Collection Amount	total amount due after the charged off
Total Current Balance	total current balance of all accounts
Total Revolving Credit Limit	total revolving credit limit
Loan Status	current status of the loan
Debt-to-Income Ratio	the borrower's debt-to-income ratio
Credit Utilization Ratio	the borrower's credit utilization ratio
Time Since Last Delinquency	the number of months since the borrower's last delinquency

Table 2: Attributes in the dataset.

V. DATAFLOW AND ARCHITECTURE

The flow of the data that has been employed in this project can be viewed in the dataflow diagram below.

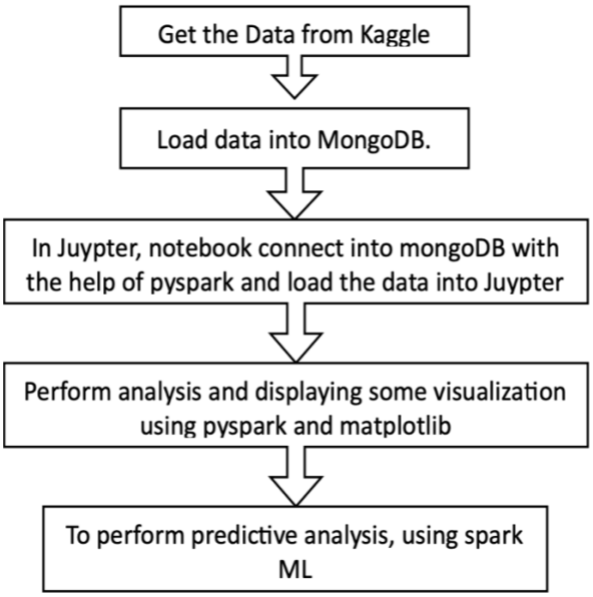

Figure 1: Flow diagram of the project

The dataset, originally obtained from Kaggle in CSV format, was first imported into a MongoDB database using a Docker container. This setup enabled the storage and management of the data in a structured and efficient manner. Subsequently, the data from the MongoDB container was accessed through Jupyter Notebook by leveraging a connector that integrates MongoDB with PySpark. To optimize the dataset for analysis, it was converted into the Parquet format, which is known for its efficiency in handling large datasets. The analysis and visualization of the data were carried out using PySpark, a powerful tool for big data processing, in combination with Matplotlib, a popular library for creating graphs and charts in Python. The core of this project involved employing machine learning algorithms available in PySpark to conduct predictive analysis, aiming to extract meaningful insights from the data.

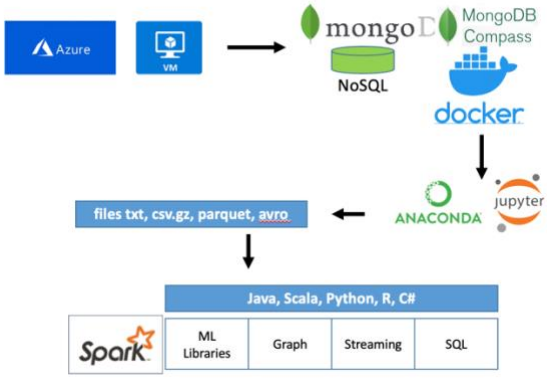

Figure 2: Software architecture of the project.

VI. LOADING THE DATA [2]

The first step in the data processing workflow involved setting up a database named 'SampleDatasets' in MongoDB. Following this, a container, also named 'train', was created to store the data. This container was then used to import the CSV dataset. In the MongoDB Compass, which is the GUI for MongoDB, a container representing a database was established and subsequently populated with the loaded data.

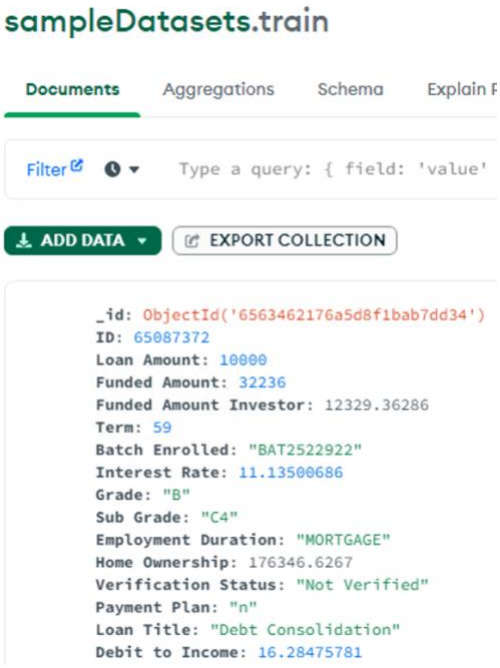

Figure 3: train into MongoDB.

The required libraries have been imported into Jupyter Notebook to execute the project as shown below:

```
In [1]: import findspark
        findspark.init()

In [2]: import pyspark
        from pyspark.context import SparkContext
        from pyspark.sql import SparkSession
        sc = SparkContext.getOrCreate()
        spark = SparkSession.builder.getOrCreate()
        print(sc.version)
        print(spark.version)

        2.4.8
        2.4.8
```

Figure 4 Libraries imported for this project.

After the above step, a connection is established between mongoDb and Juypter notebook, then the data is loaded into Juypter notebook:

Displaying the schema of the dataset:

```
root
 |-- Accounts Delinquent: integer (nullable = true)
 |-- Application Type: string (nullable = true)
 |-- Batch Enrolled: string (nullable = true)
 |-- Collection 12 months Medical: integer (nullable = true)
 |-- Collection Recovery Fee: double (nullable = true)
 |-- Debit to Income: double (nullable = true)
 |-- Delinquency - two years: integer (nullable = true)
 |-- Employment Duration: string (nullable = true)
 |-- Funded Amount: integer (nullable = true)
 |-- Funded Amount Investor: decimal(18,13) (nullable = true)
 |-- Grade: string (nullable = true)
 |-- Home Ownership: decimal(18,12) (nullable = true)
 |-- ID: integer (nullable = true)
 |-- Initial List Status: string (nullable = true)
 |-- Inquires - six months: integer (nullable = true)
 |-- Interest Rate: double (nullable = true)
 |-- Last week Pay: integer (nullable = true)
 |-- Loan Amount: integer (nullable = true)
 |-- Loan Status: integer (nullable = true)
 |-- Loan Title: string (nullable = true)
 |-- Open Account: integer (nullable = true)
 |-- Payment Plan: string (nullable = true)
 |-- Public Record: integer (nullable = true)
 |-- Recoveries: double (nullable = true)
 |-- Revolving Balance: integer (nullable = true)
 |-- Revolving Utilities: decimal(18,16) (nullable = true)
 |-- Sub Grade: string (nullable = true)
 |-- Term: integer (nullable = true)
 |-- Total Accounts: integer (nullable = true)
 |-- Total Collection Amount: integer (nullable = true)
 |-- Total Current Balance: integer (nullable = true)
 |-- Total Received Interest: double (nullable = true)
 |-- Total Received Late Fee: decimal(20,18) (nullable = true)
 |-- Total Revolving Credit Limit: integer (nullable = true)
 |-- Verification Status: string (nullable = true)
 |-- _id: struct (nullable = true)
 |    |-- oid: string (nullable = true)
```

Figure 5 Schema of the dataset.

VII. DATA PREPROCESSING

Analyzing and preparing the existing data for analysis is a crucial step. To ensure the data was ready for analysis, a series of preprocessing operations were applied to the dataset.

A. Transforming the data from the CSV format to Parquet was undertaken to enhance the efficiency of data storage and retrieval. [3]

```
#transform data to parquet.
loan_df.write.mode('overwrite').parquet('C:/Users/sthotir/Documents/ECE552_Final_Project/train.parquet')
loan_df = spark.read.parquet('C:/Users/sthotir/Documents/ECE552_Final_Project/train.parquet')

loan_df.printSchema()

root
 |-- Accounts_Delinquent: integer (nullable = true)
 |-- Application_Type: string (nullable = true)
 |-- Batch_Enrolled: string (nullable = true)
 |-- Collection_12_months_Medical: integer (nullable = true)
 |-- Collection_Recovery_Fee: double (nullable = true)
 |-- Debit_to_Income: double (nullable = true)
 |-- Delinquency_2_years: integer (nullable = true)
 |-- Employment_Duration: string (nullable = true)
 |-- Funded_Amoun: integer (nullable = true)
 |-- Funded_Amount_Investor: decimal(18,13) (nullable = true)
 |-- Grade: string (nullable = true)
 |-- Home_Ownership: decimal(18,12) (nullable = true)
 |-- ID: integer (nullable = true)
 |-- Initial_List_Status: string (nullable = true)
 |-- Inquires_6_months: integer (nullable = true)
 |-- Interest_Rate: double (nullable = true)
 |-- Last_week_Pay: integer (nullable = true)
 |-- Loan_Amount: integer (nullable = true)
 |-- Loan_Status: integer (nullable = true)
 |-- Loan_Title: string (nullable = true)
 |-- Open_Account: integer (nullable = true)
 |-- Payment_Plan: string (nullable = true)
 |-- Public_Record: integer (nullable = true)
 |-- Recoveries: double (nullable = true)
 |-- Revolving_Balance: integer (nullable = true)
 |-- Revolving_Utilities: decimal(18,16) (nullable = true)
 |-- Sub_Grade: string (nullable = true)
 |-- Term: integer (nullable = true)
 |-- Total_Accounts: integer (nullable = true)
 |-- Total_Collection_Amount: integer (nullable = true)
 |-- Total_Current_Balance: integer (nullable = true)
 |-- Total_Received_Interest: double (nullable = true)
 |-- Total_Received_Late_Fee: decimal(20,18) (nullable = true)
 |-- Total_Revolving_Credit_Limit: integer (nullable = true)
 |-- Verification_Status: string (nullable = true)
 |-- _id: struct (nullable = true)
 |    |-- oid: string (nullable = true)
```

Figure 6: Schema after converting the data into parquet.

B. The columns 'Account Delinquent' and 'Payment Plan', which contain only a single, irrelevant value, were removed. Additionally, new columns for the 'Debt-to-Income Ratio' and 'Credit Utilization Ratio' were created. Moreover, we updated certain data to reflect the current year.

```
root
 |-- Application_Type: string (nullable = true)
 |-- Batch_Enrolled: string (nullable = true)
 |-- Collection 12 months Medical: integer (nullable = true)
 |-- Collection_Recovery_Fee: double (nullable = true)
 |-- Debit_to_Income: double (nullable = true)
 |-- Delinquency_2_years: integer (nullable = true)
 |-- Employment_Duration: string (nullable = true)
 |-- Funded_Amoun: integer (nullable = true)
 |-- Funded_Amount_Investor: decimal(18,13) (nullable = true)
 |-- Grade: string (nullable = true)
 |-- Home_Ownership: decimal(18,12) (nullable = true)
 |-- ID: integer (nullable = true)
 |-- Initial_List_Status: string (nullable = true)
 |-- Inquires_6_months: integer (nullable = true)
 |-- Interest_Rate: double (nullable = true)
 |-- Last_week_Pay: integer (nullable = true)
 |-- Loan_Amount: integer (nullable = true)
 |-- Loan_Status: integer (nullable = true)
 |-- Loan_Title: string (nullable = true)
 |-- Open_Account: integer (nullable = true)
 |-- Public_Record: integer (nullable = true)
 |-- Recoveries: double (nullable = true)
 |-- Revolving_Balance: integer (nullable = true)
 |-- Revolving_Utilities: decimal(18,16) (nullable = true)
 |-- Sub_Grade: string (nullable = true)
 |-- Term: integer (nullable = true)
 |-- Total_Accounts: integer (nullable = true)
 |-- Total_Collection_Amount: integer (nullable = true)
 |-- Total_Current_Balance: integer (nullable = true)
 |-- Total Received Interest: double (nullable = true)
 |-- Total_Received_Late_Fee: decimal(20,18) (nullable = true)
 |-- Total_Revolving_Credit_Limit: integer (nullable = true)
 |-- Verification_Status: string (nullable = true)
 |-- _id: struct (nullable = true)
 |    |-- oid: string (nullable = true)
 |-- Debt_to_Income_Ratio: double (nullable = true)
 |-- Credit_Utilization_Ratio: double (nullable = true)
 |-- Time_Since_Last_Delinquency: integer (nullable = true)
```

Figure 7: Schema After Data Cleaning

VIII. DATA EXPLORATION

The data converted to parquet will been explored using pySpark's functions like groupby(), count() and agg() to find patterns between different variables

We have imported a few additional libraries for data exploration. [4]

```
from pyspark.sql import functions as F
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

Figure 8 Importing libraries for Data exploration.

The visualization shows a bar chart displaying the distribution of public records categorized by loan status. The 'Public Record' category indicates the number of derogatory public records borrowers have, ranging from bankruptcies to court judgments. The tallest bar corresponds to borrowers with '0' public records, while a smaller subset has '1' or more. The overlay of loan status '0' and '1' across these categories may suggest a pattern relating to loan default likelihood, with a visible difference in default count as the number of public records increases.

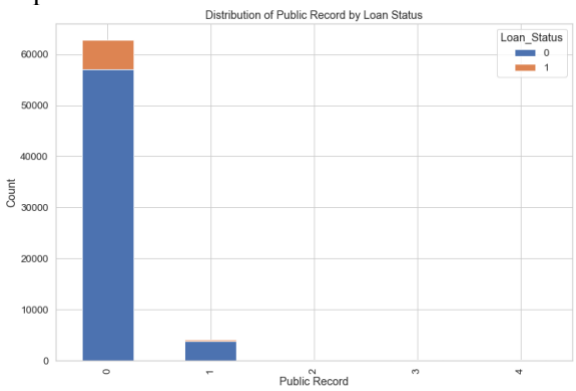

Figure 9: Distribution of Public Records by Loan status

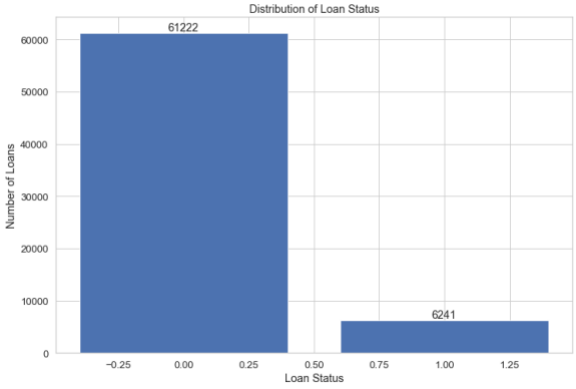

Figure 10: Distribution of Loan status

The bar chart displays how many loans are paid or ongoing versus those that have defaulted. On the bottom axis ('x'), you have '0' for all the loans that are up-to-date or finished, and '1' for the ones not paid back. The side axis ('y') counts the loans. There's a really tall bar at '0', showing that most loans, 61,222 to be exact, are being paid on time. A much shorter bar at '1' tells us that fewer, 6,241 in total, are defaults. The big difference in the height of the bars clearly shows that there are many more loans being paid back than those that aren't.

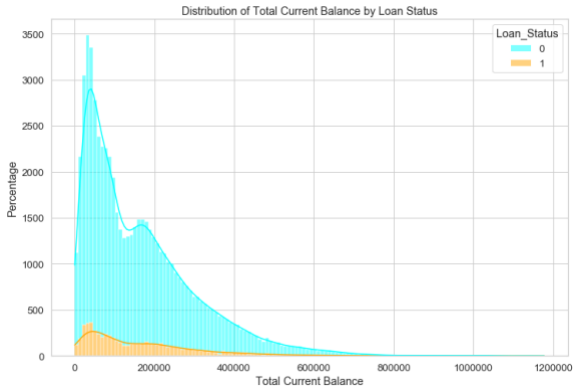

Figure 11: Distribution of Total Current Balance by Loan Status

The graph shows the distribution of borrowers' total current balances, based on their loan status. The x-axis represents the range of total current balances, while the y-axis measures the percentage of total loans. There are two lines: one for non-default loans (status '0') and one for default loans (status '1'). The non-default line peaks sharply and declines, indicating a majority of borrowers have lower balances. The default line follows a similar pattern but at a lower magnitude.

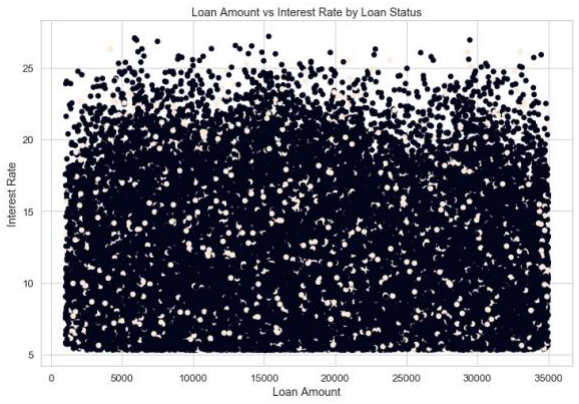

Figure 12: Loan Amount vs Interest Rate by Loan Status

The scatter plot shows the relationship between loan amount and interest rate, based on loan status. Each dot represents a loan, with the x-axis showing the loan amount and the y-axis showing the interest rate. The color differentiation indicates loan status, with one color representing current loans and another for default loans. The distribution suggests a variety of interest rates are applied across different loan amounts, with no clear distinction between good standing and default loans. The plot shows a dense clustering of loans in the mid-range of loan amounts and interest rates.

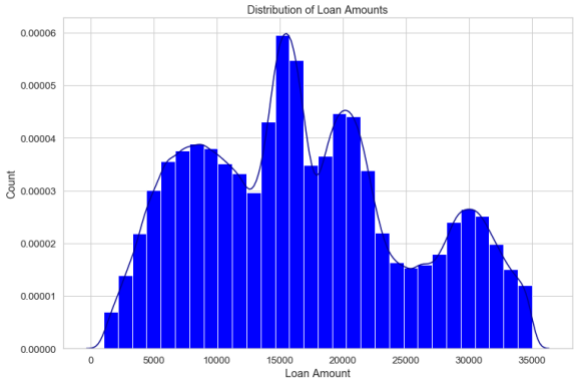

Figure 13: Distribution of Loan Amount

The histogram shows the distribution of loan amounts across financial data, with peaks at various points indicating more common loan amounts. The bars show a multi-modal distribution, with prominent peaks in regular intervals suggesting standardized loan amounts offered by the lending institution. The most frequent loan amounts are clustered in the lower to middle range, with fewer loans at the highest amounts.

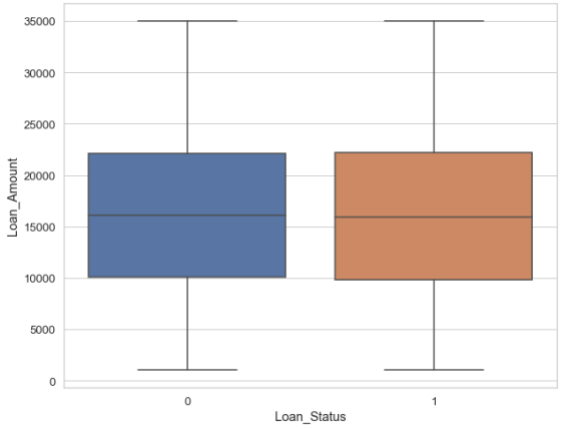

Figure 14: Loan Amount vs Loan Status

The visualization presents a boxplot comparing loan amounts based on loan status, with '0'

indicating current or paid-off loans and '1' indicating defaulted ones. The boxplots show a similar median loan amount and spread between the two categories, suggesting that loan amount may not be a distinguishing factor between loans that will default and those that will not. The absence of visible outliers in both categories indicates consistency in loan distribution across the dataset.

IX. DATA PREPARATION

The further analysis requires regression hence the dataset needs to be further cleaned and arranged in a way that is suitable so that it can be fed to the algorithms for results.

The dataset preparation consists of the following:

A) String Indexing and OneHotEncoder [5]

String Indexing is the feature transformation process that helps in encoding the column of string labels into a column of corresponding label indices. A demonstration of this process would be the transformation of Employment_Duration, Verification_Status, Initial_List_Statusand Application_Type columns.

```
from pyspark.ml.feature import StringIndexer, VectorAssembler, StandardScaler
from pyspark.sql.functions import when, lit
from pyspark.ml import Pipeline


loan_df=loan_df.withColumn("Employment_Duration_Index",\
                           when((loan_df.Employment_Duration == 'MORTGAGE'),lit(0))\
                           .when((loan_df.Employment_Duration == 'RENT'),lit(1))\
                           .when((loan_df.Employment_Duration == 'OWN'),lit(2)))
loan_df=loan_df.withColumn("Verification_Status_Index",\
                           when((loan_df.Verification_Status == 'Not Verified'),lit(0))\
                           .when((loan_df.Verification_Status == 'Verified'),lit(1))\
                           .when((loan_df.Verification_Status == 'Source Verified'),lit(2)))
loan_df=loan_df.withColumn("Initial_List_Status_Index",\
                           when((loan_df.Initial_List_Status == 'w'),lit(0))\
                           .when((loan_df.Initial_List_Status == 'f'),lit(1)))

loan_df=loan_df.withColumn("Application_Type_Index",\
                           when((loan_df.Application_Type == 'INDIVIDUAL'),lit(0))\
                           .when((loan_df.Application_Type == 'JOINT'),lit(1)))
```

Figure 15: Code executed for string indexing.

The OneHotEncoder is a method that converts categorical data into numerical format by creating binary columns for each variable category. This allows for the use of categorical data in mathematical and statistical models where only numbers can be inputted, thereby enhancing the efficiency of data analysis.

B) Dropping the unnecessary columns.

There are few columns which won't be necessary for answering the rest of the analysis, thus their columns have been removed for simplification of analysis.

```
drop_columns=['Loan_Title','Batch_Enrolled','Application_Type','Employment_Duration','Grade','Initial_List_Status','Sub_Grade',
              'Verification_Status','oid','_id']
loan_df=loan_df.drop(*drop_columns)
loan_df.printSchema()
```

```
root
 |-- Collection_12_months_Medical: integer (nullable = true)
 |-- Collection_Recovery_Fee: double (nullable = true)
 |-- Debit_to_Income: double (nullable = true)
 |-- Delinquency_2_years: integer (nullable = true)
 |-- Funded_Amoun: integer (nullable = true)
 |-- Funded_Amount_Investor: decimal(18,13) (nullable = true)
 |-- Home_Ownership: decimal(18,12) (nullable = true)
 |-- ID: integer (nullable = true)
 |-- Inquires_6_months: integer (nullable = true)
 |-- Interest_Rate: double (nullable = true)
 |-- Last_week_Pay: integer (nullable = true)
 |-- Loan_Amount: integer (nullable = true)
 |-- Loan_Status: integer (nullable = true)
 |-- Open_Account: integer (nullable = true)
 |-- Public_Record: integer (nullable = true)
 |-- Recoveries: double (nullable = true)
 |-- Revolving_Balance: integer (nullable = true)
 |-- Revolving_Utilities: decimal(20,17) (nullable = true)
 |-- Term: integer (nullable = true)
 |-- Total_Accounts: integer (nullable = true)
 |-- Total_Collection_Amount: integer (nullable = true)
 |-- Total_Current_Balance: integer (nullable = true)
 |-- Total_Received_Interest: double (nullable = true)
 |-- Total_Received_Late_Fee: decimal(20,18) (nullable = true)
 |-- Total_Revolving_Credit_Limit: integer (nullable = true)
 |-- Debt_to_Income_Ratio: double (nullable = true)
 |-- Credit_Utilization_Ratio: double (nullable = true)
 |-- Time_Since_Last_Delinquency: integer (nullable = true)
 |-- Application_Type_Indexed_encoded: vector (nullable = true)
 |-- Batch_Enrolled_Indexed_encoded: vector (nullable = true)
 |-- Employment_Duration_Indexed_encoded: vector (nullable = true)
 |-- Grade_Indexed_encoded: vector (nullable = true)
 |-- Initial_List_Status_Indexed_encoded: vector (nullable = true)
 |-- Loan_Title_Indexed_encoded: vector (nullable = true)
 |-- Sub_Grade_Indexed_encoded: vector (nullable = true)
 |-- Verification_Status_Indexed_encoded: vector (nullable = true)
 |-- Employment_Duration_Index: integer (nullable = true)
 |-- Verification_Status_Index: integer (nullable = true)
 |-- Initial_List_Status_Index: integer (nullable = true)
 |-- Application_Type_Index: integer (nullable = true)
```

Figure 16: Code for dropping the unwanted columns.

C) Vector Assembling [6]

The VectorAssembler is a feature transformer in Apache Spark that consolidates multiple input columns into a single vector column, enabling efficient processing by machine learning algorithms. This tool simplifies the process of preparing datasets for models by merging necessary features into a unified column, streamlining the data for efficient processing by models that require a singular vector format for inputs.

```
assembler = VectorAssembler(inputCols=['Employment_Duration_Index','Verification_Status_Index','Application_Type_Index',
                                       'Initial_List_Status_Index','Collection_12_months_Medical', 'Collection_Recovery_Fee',
                                       'Debit_to_Income', 'Delinquency_2_years', 'Funded_Amoun', 'ID', 'Inquires_6_months',
                                       'Interest_Rate', 'Last_week_Pay', 'Loan_Amount', 'Open_Account', 'Public_Record',
                                       'Recoveries', 'Revolving_Balance', 'Term', 'Total_Accounts', 'Total_Collection_Amount',
                                       'Total_Current_Balance', 'Total_Received_Interest', 'Total_Revolving_Credit_Limit',
                                       'Debt_to_Income_Ratio', 'Credit_Utilization_Ratio', 'Time_Since_Last_Delinquency'],
                            outputCol='features')
```

Figure 17: Code for vector assembler.

D) Splitting data for training and testing.

The dataset is suitable for training a model and can be divided for evaluation. However, it's not practical to use the entire dataset for training. Instead, a smaller segment is dedicated for validation to assess the model's predictive accuracy. The training data is

randomly selected to represent 70 percent of the total dataset, leaving 30 percent for testing.

```
train_data, test_data = loan_df_transformed.randomSplit([0.7, 0.3], seed=70)
```

Figure 18: split dataset for training and testing.

X. PREDICTIVE MODELING

The previous steps ensure that the data is ready to be fed into the machine learning algorithms to train and get the prediction regarding the Loan deafult of a certain customer given the details required significantly are provided. The question, **Is it possible to predict whether a client will default on their loan?**

It can be answered using regression. Two types of regression models have been used and the one with higher accuracy will be chosen at the end.

1. Logistic Regression [7]

Logistic regression is a statistical method used to predict binary outcomes from independent variables. It estimates the probability of the binary response based on input variables, aiming to find the most accurate coefficients that increase the probability of the predicted result being true.

Metrics	Values
Accuracy	0.56
Recall	0.9075288041319031
Precision	0.8236085303290821
F1 Score	0.8635345673890339

Table 3: Evaluation metrics of logistic regression.

The model outperforms in predicting class 0 (not in default) with a precision of 0.82 and a recall of 0.90, indicating that 82% of the samples predicted as not in default were actually not in default, and 90% of the samples correctly identified as not in default.

2. Random Forest Regression [8]

The random forest regression model can be used for both classification and regression, which uses multiple decision trees to learn and predict the required.

The data here shows that the model is reliable in classifying the Loan default.

Metrics	Values
Accuracy	0.5
Recall	0.9084661354581673
Precision	0.8253107192742972
F1 Score	0.8648942770746773

Table 4: Evaluation metrics of the random forest classifier.

The classification report reveals a model with high precision and F1 scores for the 0 class (non-default), but low scores for the 1 class (default), indicating it performs well in predicting non-default instances but not default instances.

XI. CONCLUSION

The logistic regression and random forest classifier models show high accuracy in identifying non-defaulting loans, with recall scores of approximately 0.91 and precision of around 0.82. They are effective at detecting the majority of non-default cases, with F1 scores of around 0.86, indicating a balance between precision and recall. However, the logistic regression model's accuracy is only 0.56, marginally better than a coin flip, and the random forest classifier's accuracy drops to 0.5, equivalent to random guessing. This raises concerns about the models' overall predictive power. The high precision and recall suggest that the models are good at classifying non-default loans but may not be as effective in correctly

predicting actual defaults. The models' performance in predicting non-default instances could be due to an imbalance in the dataset or an inherent bias towards the majority class. In conclusion, the models show promise in identifying non-default loans but their overall predictive accuracy is questionable. Further refinement and rebalancing of the dataset or model parameter adjustments could improve performance, particularly for default prediction, increasing their usefulness in practical banking scenarios.

References:

[1]
"Loan Default prediction dataset," *www.kaggle.com*. Available: https://www.kaggle.com/datasets/hemanthsai7/loandefault?resource=download.

[2]
"MongoDB Documentation," *https://github.com/mongodb/docs-bi-connector/blob/DOCSP-3279/source/index.txt*. Available: https://www.mongodb.com/docs/

[3]
"Apache Parquet," *Apache.org*, 2018. Available: https://parquet.apache.org/

[4]
"How To Perform Data Visualization with Pandas," *Analytics Vidhya*, Jul. 10, 2021. Available: https://www.analyticsvidhya.com/blog/2021/07/how-to-perform-data-visualization-with-pandas/

[5]
"Onehotencoder¶," OneHotEncoder - PySpark 3.5.0 documentation, https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.feature.OneHotEncoder.html
"VectorAssembler — PySpark 3.1.1 documentation." https://spark.apache.org/docs/3.1.1/api/python/reference/api/pyspark.ml.feature.VectorAssembler.html

[7]
G. Santos, "Introduction to Logistic Regression in PySpark," *Medium*, Nov. 04, 2023. Available: https://towardsdatascience.com/introduction-to-logistic-regression-in-pyspark-9f894299c32d.

[8]
"RandomForestClassifier — PySpark 3.5.0 documentation," *spark.apache.org*. Available: https://spark.apache.org/docs/latest/api/python/reference/api/pyspark.ml.classification.RandomForestClassifier.html.