Loan Approval Prediction Using Spark and ML

By:

Pardhesh Maddala (BT22CSE175)

Keerthan Anumalsetty (BT22CSE172)

Rohit Magham (BT22CSE176)

1. Introduction

1.1 Problem Statement

The loan approval process is a critical function for financial institutions that requires careful assessment of applicant creditworthiness. Manual evaluation of loan applications is time-consuming, potentially inconsistent, and may not efficiently identify all relevant patterns in applicant data. This project aims to automate the loan eligibility process in real-time when large amount of applications received, based on customer details provided during online application submission.

Financial institutions can leverage this predictive model to:

- Streamline the loan approval workflow
- Ensure consistency in decision-making
- Reduce processing time
- Minimize the risk of loan defaults
- Make data-driven lending decisions

1.2 Dataset Description

The dataset used in this project is the "Loan Prediction Problem Dataset" from Kaggle (https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset). It contains information about loan applicants and whether their applications were approved or rejected.

The dataset includes the following features:

Variable	Description
Loan_ID	Unique Loan ID
Gender	Male/ Female
Married	Applicant married (Y/N)
Dependents	Number of dependents
Education	Applicant Education (Graduate/ Under Graduate)
Self_Employed	Self employed (Y/N)
ApplicantIncome	Applicant income
CoapplicantIncome	Coapplicant income
LoanAmount	Loan amount in thousands
Loan_Amount_Term	Term of loan in months
Credit_History	Credit history meets guidelines
Property_Area	Urban/ Semi Urban/ Rural
Loan_Status	Loan approved (Y/N) - Target variable

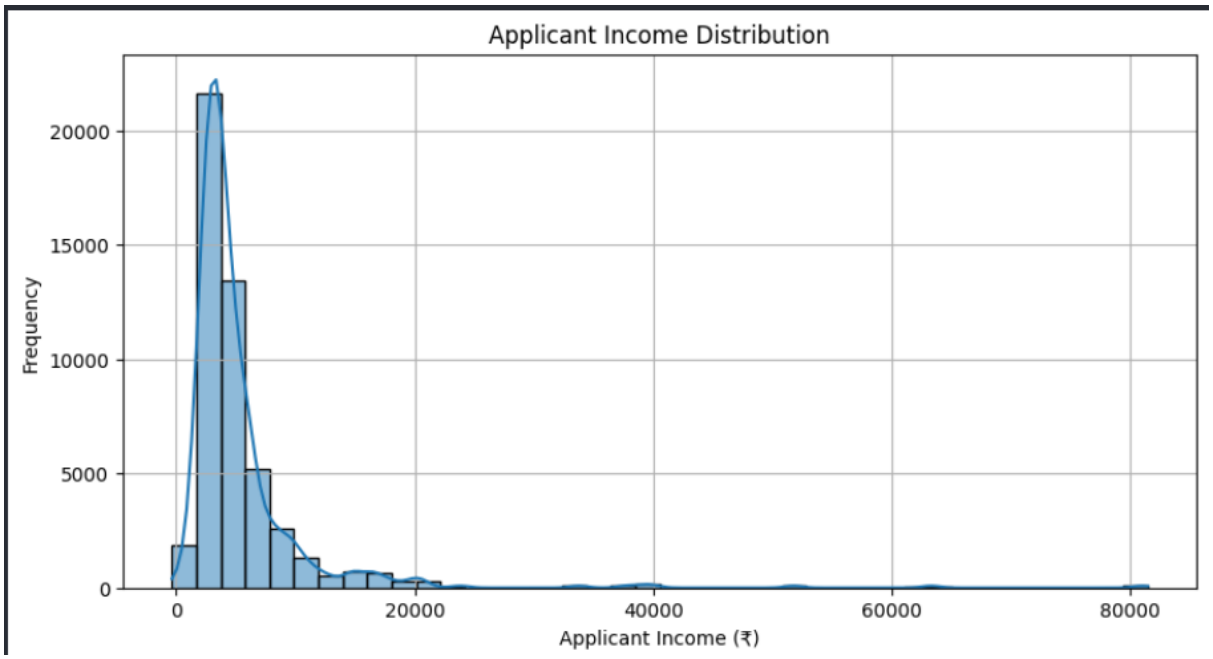
This is a standard supervised classification task where we predict whether a loan application will be approved (**Y**) or rejected (**N**) based on the applicant's characteristics.
A few data augmentation techniques are performed on the dataset before passing through the

model to make it suitable for the spark system so that the spark capabilities won't be overkill. The final dataset contains **49,120 rows** with **13 features**.

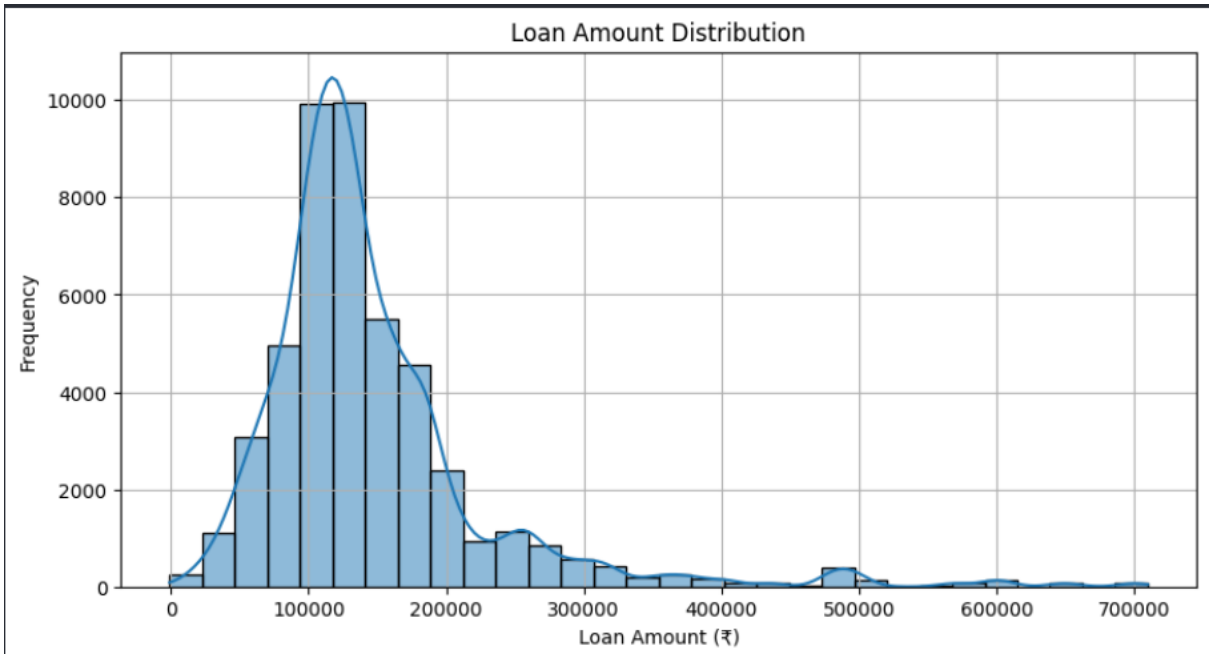
To download the dataset here is the link:
https://drive.google.com/file/d/1M0IgCB81X9ZN0DCAbhZPfNG7VkKMTTEk/view?usp=sharing

Here are some of the representations describing the dataset:

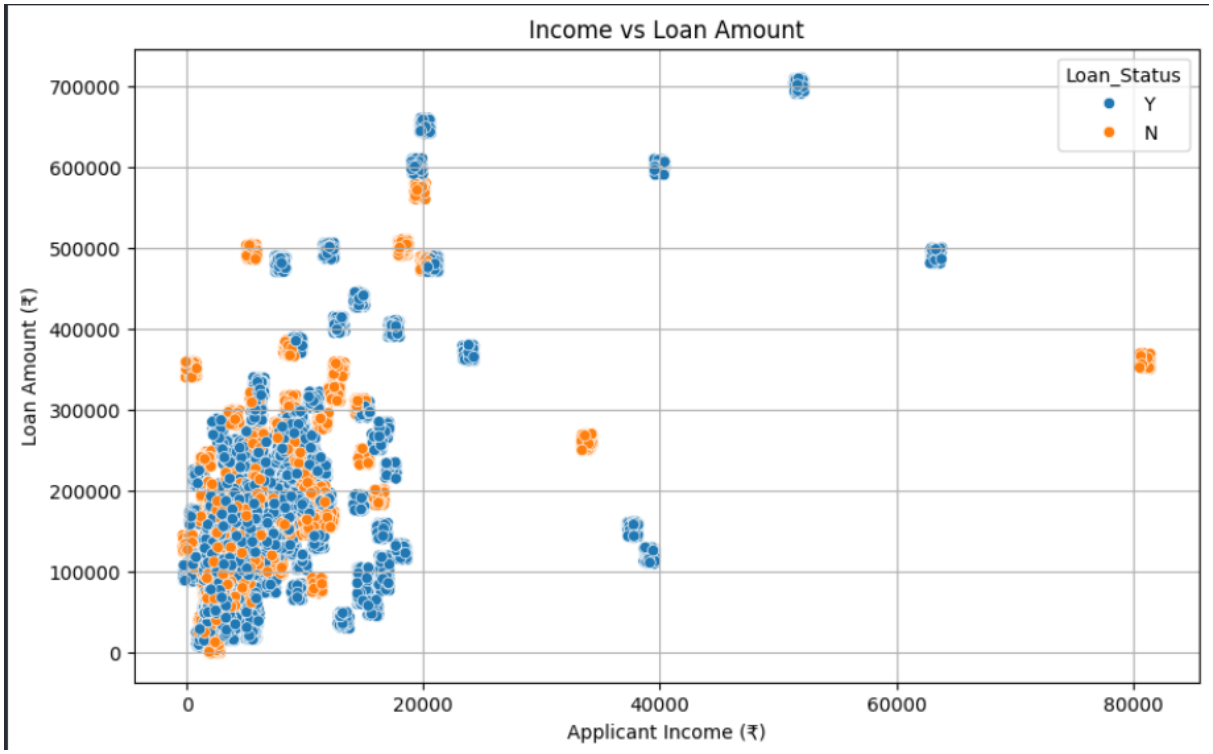
(i). Distribution of Income of the applicants throughout the dataset



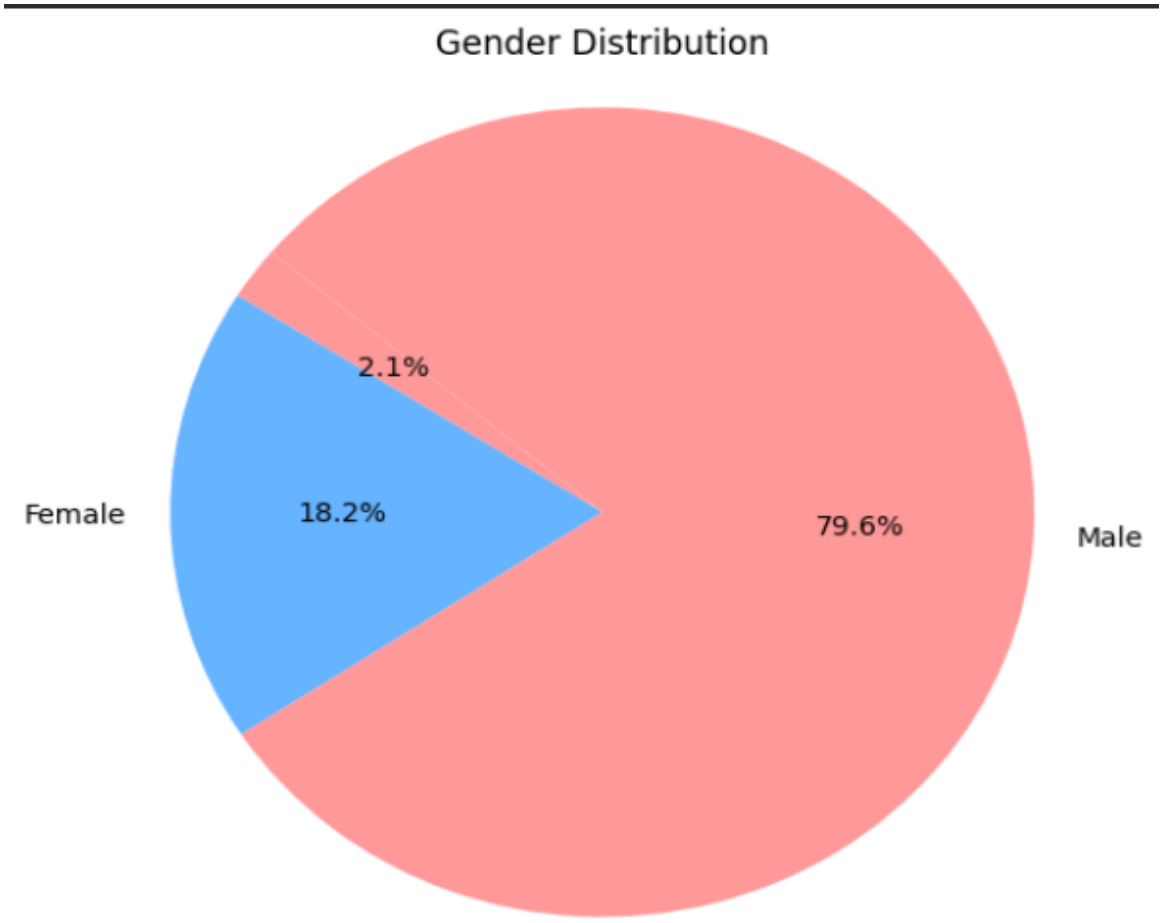
(ii). Distribution of applied Loan amounts



(iii). Scatter plot between Applicant income and Loan Amount



(iv). Gender Distribution in dataset



2. Method to solve the problem

2.1 Overview

Our loan prediction system employs several key methods and approaches:

1. **Distributed Data Processing Using Spark** - Apache Spark's distributed computing framework to efficiently process large loan application datasets
2. **Exploratory Data Analysis (EDA)** - Visualizing and analyzing data patterns, distributions, and relationships to gain insights
3. **Data Cleaning and Preprocessing** - Handling missing values and transforming categorical variables
4. **Feature Engineering** - Creating derived features using spark features to improve model performance
5. **Machine Learning Models** - Implementing multiple classification algorithms:
    - **Logistic Regression**
    - **Linear Support Vector Classification (SVC)**
    - **Decision Tree Classification**
    - **Random Forest Classification**
6. **Model Evaluation** - Assessing model performance using accuracy metrics

2.2 Apache Spark and PySpark

We used Apache Spark through its Python API (PySpark) as our primary data processing and machine learning framework. Spark was chosen for its distributed computing capabilities, which make it ideal for processing large datasets efficiently. For financial institutions that handle thousands of loan applications, this scalability is crucial.

PySpark offers several advantages for this application:

- Parallel processing of data
- In-memory computation for faster processing
- Built-in machine learning library (MLlib)
- SQL-like interface for data manipulation
- Seamless integration with Python's data science ecosystem

We initialized a Spark session for our loan prediction application:

```
import pyspark
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
```

```
#Spark Session
spark = SparkSession.builder.appName('loan_prediction').getOrCreate()
```

Python

2.3 Dataset Loading and Exploration

We loaded the loan dataset from a CSV file using Spark's data reading capabilities, which automatically infer schema types. This approach eliminates the need for manual schema definition and accelerates the initial data preparation phase.

```
df = spark.read.csv('/content/Loan_Dataset.csv', header=True, sep=',', inferSchema=True)
df.show(5)
```

Python

Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
LP001000	Male	No	0	Graduate	No	6155	0.0	NULL	360.0	1.0	Urban	Y
LP001001	Male	Yes	1	Graduate	No	4442	1508.0	136.0	360.0	1.0	Rural	N
LP001002	Male	Yes	0	Graduate	Yes	2998	0.0	60.0	360.0	1.0	Urban	Y
LP001003	Male	Yes	0	Not Graduate	No	2138	2358.0	124.0	360.0	1.0	Urban	Y
LP001004	Male	No	0	Graduate	No	6124	0.0	137.0	360.0	1.0	Urban	Y

only showing top 5 rows

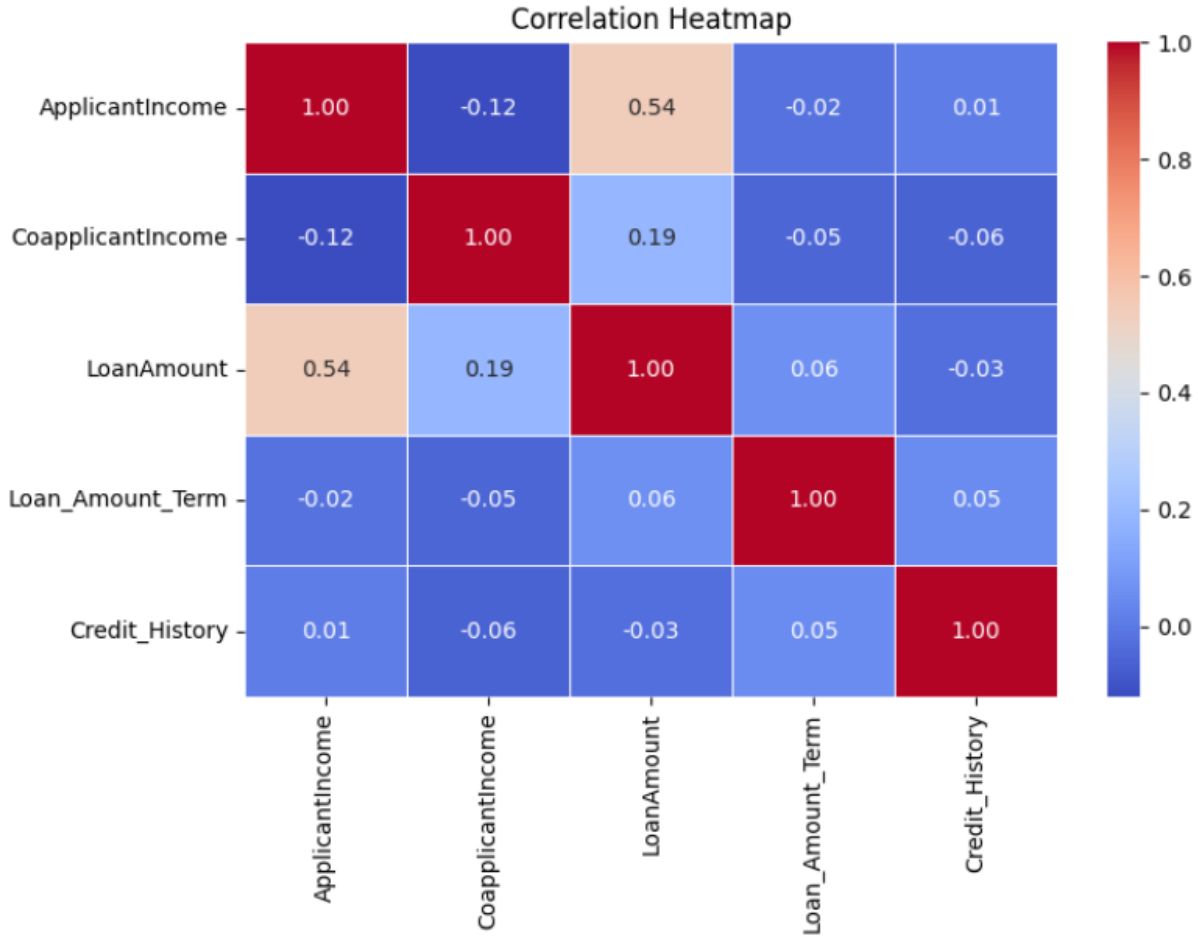
The dataset exploration phase was critical to understanding the underlying patterns in loan applications and identifying potential factors that influence loan approval decisions.

We conducted exploratory data analysis to understand patterns and relationships.

We created visualizations to better understand the data.

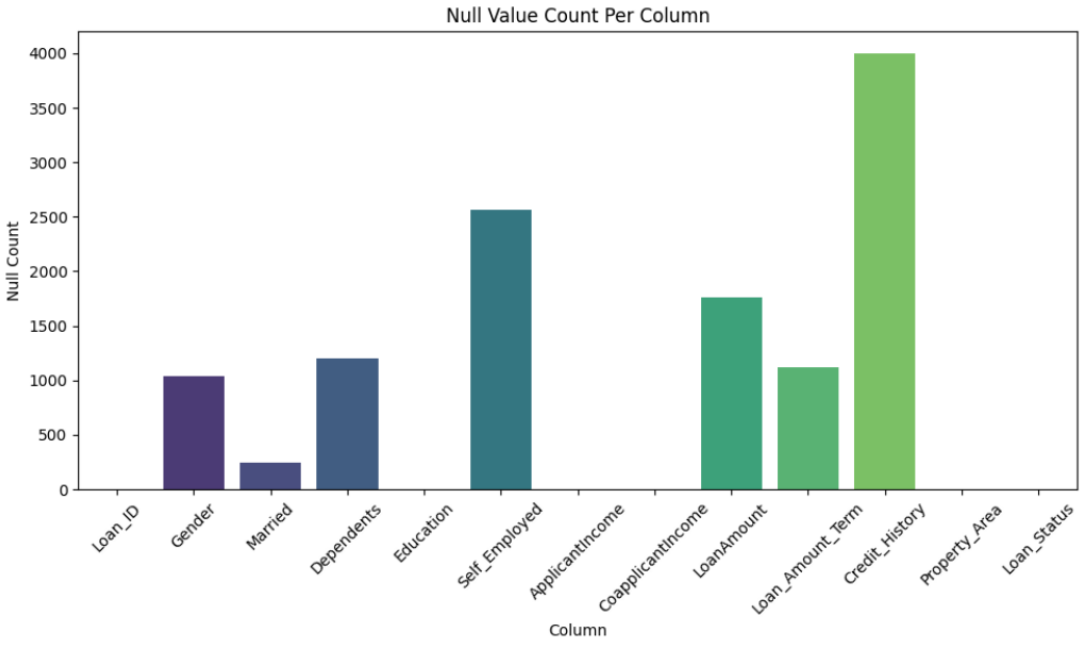
We also created a correlation matrix to understand relationships between numerical variables.

```
columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History']
corr_df = pd.DataFrame()
for i in columns:
    corr = []
    for j in columns:
        corr.append(round(df.stat.corr(i, j), 2))
    corr_df = pd.concat([corr_df, pd.Series(corr)], axis=1)
corr_df.columns = columns
corr_df.insert(0, '', columns)
corr_df.set_index('')
```



2.4 Data Cleaning and Preprocessing

Our analysis found missing values in several columns, including gender, marital status, loan amount, and credit history. Missing data in financial applications can significantly impact model performance, so proper handling was essential.



For cleaning the data, we handled missing values using appropriate imputation strategies:

- For numerical columns (Loan Amount, Loan Amount Term), we used mean imputation to preserve the average distribution of values
- For categorical columns (Gender, Marital Status, Dependents, etc.), we used mode imputation (most frequent value) to maintain the most common category

These approaches are standard in financial modeling where maintaining data distribution is important.

```
numerical_cols = ['LoanAmount', 'Loan_Amount_Term']
categorical_cols = ['Gender', 'Married', 'Dependents', 'Self_Employed', 'Credit_History']


for col in numerical_cols:
    mean = df.select(F.mean(df[col])).collect()[0][0]
    df = df.na.fill(mean, [col])


for col in categorical_cols:
    mode = df.groupby(col).count().orderBy("count", ascending=False).first()[0]
    df = df.na.fill(mode, [col])
```

Feature engineering played a crucial role in improving model performance:

- We created a TotalIncome feature to capture the combined income of the applicant and co-applicant, which is a stronger indicator of repayment capacity than individual incomes alone
- We converted the loan status to binary values (1/0) for compatibility with classification algorithms

For data preparation, we implemented a pipeline approach that included:

- StringIndexer to convert categorical strings to indices
- OneHotEncoder to create binary vectors for categorical features
- VectorAssembler to combine all features into a single vector column required by Spark's machine learning algorithms

```
from pyspark.ml.feature import VectorAssembler, OneHotEncoder, StringIndexer
from pyspark.ml import Pipeline


categorical_columns = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area', 'Credit_History']
numerical_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'TotalIncome']

indexers = [StringIndexer(inputCol=col, outputCol="{0}_index".format(col)) for col in categorical_columns]
encoders = [OneHotEncoder(dropLast=False, inputCol=indexer.getOutputCol(), outputCol="{0}_encoded".format(indexer.getOutputCol()))
            for indexer in indexers]
input_columns = [encoder.getOutputCol() for encoder in encoders] + numerical_columns

assembler = VectorAssembler(inputCols=input_columns, outputCol="feature")
```

This pipeline approach ensures consistency between training and prediction phases, which is critical for model reliability in production environments.

2.5 Model Training

We trained four different classification models to predict loan approval, each with distinct advantages:

```
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier, DecisionTreeClassifier, LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
```

2.5.1 Logistic Regression

Logistic regression's primary advantage in loan prediction is its ability to provide probability scores and feature importance, which can help explain why certain applicants were approved or rejected. The model works by estimating the log-odds of loan approval as a linear combination of the features, then converting these log-odds to probabilities.

```
lr = LogisticRegression(featuresCol='feature', labelCol='Loan_Status')
lr_model = lr.fit(train_data)


predictions = lr_model.transform(test_data)
predictions.show(5)
```

2.5.2 Linear Support Vector Classifier (SVC)

Linear SVC focuses on maximizing the margin between loan approval and rejection classes, which can lead to better generalization on new data. This model is less sensitive to outliers in the feature space compared to logistic regression, which is beneficial when dealing with financial data that often contains anomalies.

```
svc = LinearSVC(labelCol="Loan_Status", featuresCol="feature")
svc_model = svc.fit(train_data)
svc_predictions = svc_model.transform(test_data)
evaluator = MulticlassClassificationEvaluator(labelCol="Loan_Status", predictionCol="prediction", metricName="accuracy")
svc_accuracy = evaluator.evaluate(svc_predictions)
print(f"Linear SVC Accuracy: {svc_accuracy:.4f}")
```

```
Linear SVC Accuracy: 0.8128
```

2.5.3 Decision Tree

Decision trees mimic human decision-making processes by creating a series of if-then-else decision rules. This makes them particularly valuable in loan assessment contexts where transparency in decision-making is important. The tree structure can be visualized and explained to stakeholders, showing which features (e.g., credit history, income level) were most influential in classification decisions.

```
dt = DecisionTreeClassifier(labelCol="Loan_Status", featuresCol="feature")
dt_model = dt.fit(train_data)
dt_predictions = dt_model.transform(test_data)

evaluator = MulticlassClassificationEvaluator(labelCol="Loan_Status", predictionCol="prediction", metricName="accuracy")
dt_accuracy = evaluator.evaluate(dt_predictions)
print(f"Decision Tree Accuracy: {dt_accuracy:.4f}")
```

```
Decision Tree Accuracy: 0.8402
```

2.5.4 Random Forest

Random Forest combines multiple decision trees to create a more robust model. By averaging predictions across many trees trained on different subsets of data, it reduces variance and improves generalization. This ensemble approach often yields higher accuracy than single models, making it a strong candidate for loan prediction where both precision and recall are important.

2.6 Model Evaluation

We evaluated all models using accuracy as the primary metric, which measures the proportion of correct predictions among the total number of cases examined.

The accuracy results for each model are:

Model	Accuracy
Logistic Regression	81.62%
Linear SVC	81.28%
Decision Tree	84.02%
Random Forest	82.57%

3. Conclusion

This project successfully implemented an end-to-end machine learning pipeline using PySpark to predict loan approvals. The solution addresses a critical business need in the financial sector by automating the loan eligibility assessment process.

The project followed a comprehensive approach including:

1. **Data Exploration**: The dataset was thoroughly analyzed to understand the distribution of variables, their relationships, and patterns that might influence loan approval decisions.
2. **Data Preprocessing**: Missing values were handled appropriately, with means used for numerical features and modes used for categorical features. The data was transformed into a format suitable for machine learning algorithms.
3. **Feature Engineering**: New features like TotalIncome were created to enhance model performance, and categorical variables were properly encoded using techniques like one-hot encoding.
4. **Model Training**: Four different classification algorithms were implemented and compared:
    - **Logistic Regression**
    - **Linear Support Vector Classifier (SVC)**
    - **Decision Tree**
    - **Random Forest**

5. **Evaluation**: All models were evaluated using accuracy as the performance metric. While specific accuracy values were not displayed in the provided output, the implementation allows for effective comparison between models.

The use of PySpark for this implementation provides scalability benefits, making this solution suitable for processing large volumes of loan applications that financial institutions typically handle. The entire pipeline, from data loading to prediction, is automated and can be deployed in production environments for real-time loan approval prediction.

4. Future Work

Several promising directions could be explored to enhance this loan prediction system:

1. Ensemble Methods, as well as Advanced Models:

- o Implement stacking or blending of several models to improve performance.
- o Explore different deep learning approaches in order to achieve feature extraction.
- o Implement Gradient Increased Trees, which do often perform well enough on tabular data.

2. Business-Oriented Evaluation:

- o Develop tailored evaluation metrics which align to business costs (e.g., cost for false approvals against false rejections).
- o Implement profit-based model selection that specifically considers predictions' financial impact.
- o Generate individual segment-specific models for various applicant profiles.

3. Explainable AI Implementation:

- o Develop a system for applicants that provides several explanations. These explanations will concern decisions regarding their loan.
- o Implement counterfactual explanations, revealing changes yielding approval to applicants.
- o Create of a web interface via database connection for most efficient workflow.

4. Real-time Processing-related Enhancement:

- o Somewhat optimize the Spark pipeline for lower latency in real-time predictions.
- o Implement incremental learning in order to adapt to patterns that are changing without full retraining.
- o Develop streaming capabilities so as to process applications when they arrive.

5. References

1. Loan default prediction using spark machine learning algorithms – by Aiman Muhammad Uwais, Hamidreza Khaleghzadeh
2. Loan Approval Prediction with Logistic Regression in PySpark – by Gonzalo Surribas Sayago
3. Kaggle. (n.d.). Loan Prediction Problem Dataset: https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset