



L O V E L Y  
P R O F E S S I O N A L  
U N I V E R S I T Y

## Continuous Assessment II

**Name:** Mohit Bali

**Subject:** Cluster Computing (INT-315)

**Registration Number:** 12214189

**Roll Number:** RK22YPA18

**Project Number:** 18

**Project:** Loan Status Prediction with Logistic Regression

**GitHub Link:** <http://bit.ly/4p11FV1>

# Predicting Loan Status with PySpark

## Introduction:

The goal of this project is to build a machine learning model that predicts whether a loan application will be approved or rejected based on applicant income, employment details, credit history, dependents, and property area.

The project uses PySpark MLlib, demonstrating distributed data processing, feature engineering, Logistic Regression modeling, evaluation, and visualization.

## Dataset Details:

1. **Dataset:** Loan Prediction Dataset
2. **Dataset Source:** Kaggle  
<https://www.kaggle.com/datasets/ninzaami/loan-predication>
3. **Rows:** 614
4. **Target Variable:** Loan\_Status
5. **Key Features:** Gender, Married, Dependents, Education, Self\_Employed, ApplicantIncome, CoapplicantIncome, LoanAmount, Credit\_History, Property\_Area

## Source Code:

Data was loaded into Spark, cleaned, and encoded using StringIndexer and OneHotEncoder. Features were combined with VectorAssembler, a Logistic Regression model was trained and tested, and results were evaluated with Accuracy, F1-score, and visualized using bar charts and a confusion matrix.

## I. Loading the libraries:

```
# Data Manipulation and Visualization
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# PySpark Basic
from pyspark.sql import SparkSession
from pyspark.sql.functions import mean, col

# Pyspark Machine Learning
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml import Pipeline
import pyspark.sql.functions as F

import warnings
warnings.filterwarnings("ignore")
```

## II. Loading Dataset into Spark DataFrame:

```
# Initializing Spark Session
spark = SparkSession.builder.appName("Loan_Prediction").getOrCreate()

# Importing the dataset
data = spark.read.csv("Loan_Prediction_Data.csv", header=True,
inferSchema=True)

data.show(5)
```

---

## Output:

Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoaapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
LP001002	Male	No	0	Graduate	No	5849	0.0	NULL	360	1	Urban	Y
LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128	360	1	Rural	N
LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66	360	1	Urban	Y
LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	128	360	1	Urban	Y
LP001008	Male	No	0	Graduate	No	6000	0.0	141	360	1	Urban	Y

### III. Data Exploration:

```
data.printSchema()
print("-" * 100)
print(data.columns)
print("-" * 100)
print(data.dtypes)
```

#### Output:

```
root
|-- Loan_ID: string (nullable = true)
|-- Gender: string (nullable = true)
|-- Married: string (nullable = true)
|-- Dependents: string (nullable = true)
|-- Education: string (nullable = true)
|-- Self_Employed: string (nullable = true)
|-- ApplicantIncome: integer (nullable = true)
|-- CoapplicantIncome: double (nullable = true)
|-- LoanAmount: integer (nullable = true)
|-- Loan_Amount_Term: integer (nullable = true)
|-- Credit_History: integer (nullable = true)
|-- Property_Area: string (nullable = true)
|-- Loan_Status: string (nullable = true)

-----
['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'Credit_History', 'Property_Area']

[('Loan_ID', 'string'), ('Gender', 'string'), ('Married', 'string'), ('Dependents', 'string'), ('Education', 'string'), ('Self_Employed', 'string'), ('ApplicantIncome', 'int'), ('Coapp
```

### Value Count for Target Category:

```
print(data.groupBy('Loan_Status').count().show())
```

#### Output:

Loan_Status	count
Y	422
N	192

### Gender wise Value Count:

```
data.select("Gender", "Loan_Status").groupBy("Loan_Status",
"Gender").count().show()

# More male applicants and more male approval
```

#### Output:

Loan_Status	Gender	count
N Female	37	
Y NULL	8	
Y Female	75	
N NULL	5	
Y Male	339	
N Male	150	

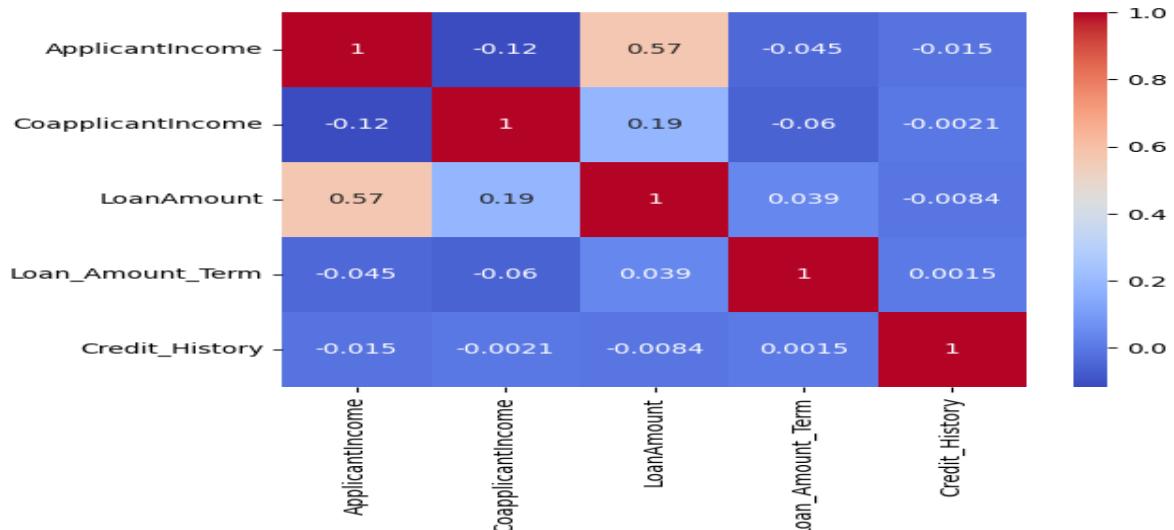
## Correlation Matrix:

```

numeric_cols =
["ApplicantIncome", "CoapplicantIncome", "LoanAmount", "Loan_Amount_Term", "Credit_History"]
pdf = data.select(numeric_cols).toPandas()
corr = pdf.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()

# As the applicant's income increases, the amount of the loan they apply for
also tends to increase.

```



#### **IV. Data Cleaning and Pre-processing:**

# Checking for Null Values

```
data.select([F.count(F.when(F.col(c).isNull(), c)).alias(c) for c in data.columns]).show()
```

## Output:

## Filling Null Values:

```
numeric_cols_null = ["LoanAmount","Loan_Amount_Term"]
categorical_cols_null =
["Gender","Married","Dependents","Self_Employed","Credit_History"]

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

for col in categorical_cols_null:
    mode = data.groupBy(col).count().orderBy('count',
ascending=False).first()[0]
    data = data.na.fill(mode, [col])

# All Null values filled
data.select([F.count(F.when(F.col(c).isNull(), c)).alias(c) for c in
data.columns]).show()
```

---

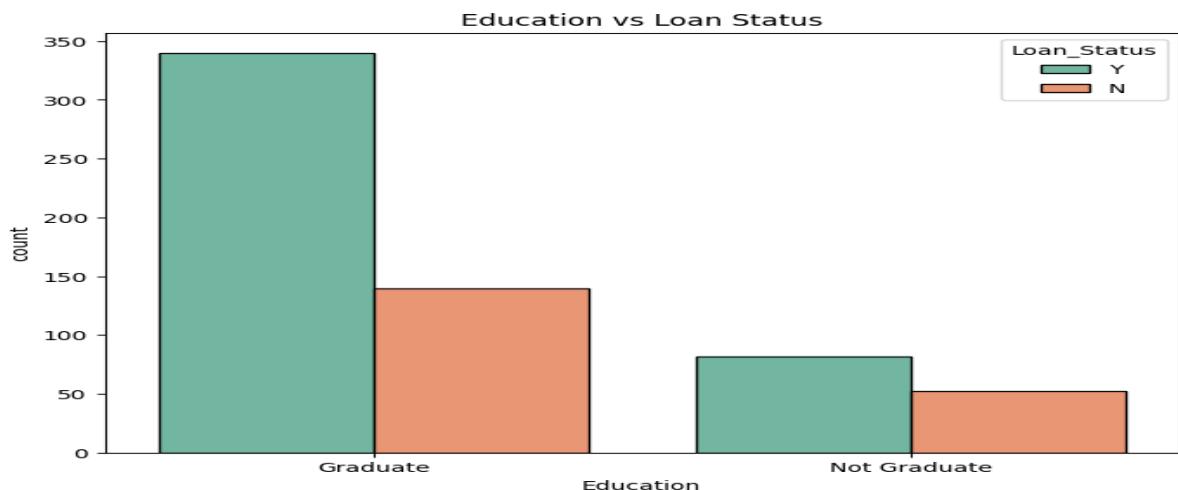
## Output:

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|Loan_ID|Gender|Married|Dependents|Education|Self_Employed|ApplicantIncome|CoapplicantIncome|LoanAmount|Loan_Amount_Term|Credit_History|Property_Area|Loan_Status|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|    0|     0|     0|      0|     0|      0|      0|      0|      0|      0|      0|     0|     0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

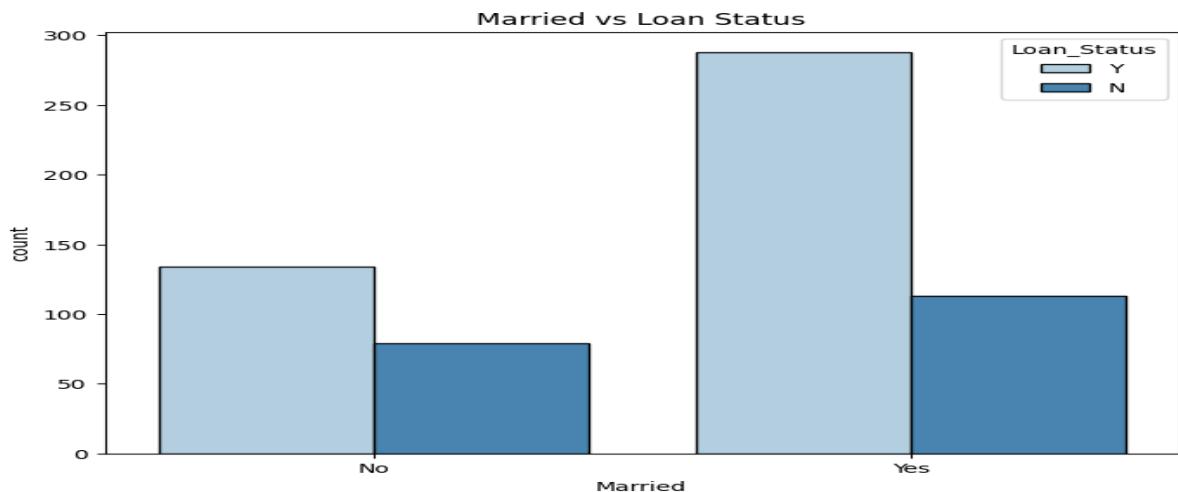
# Data Visualizations:

```
pdf = data.toPandas()

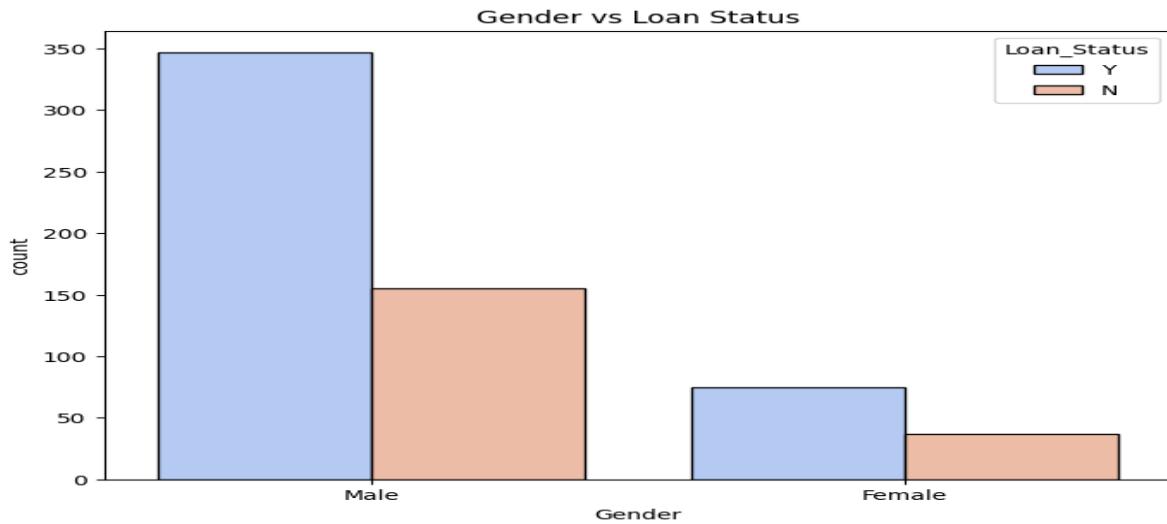
# Education vs Loan Status
plt.figure(figsize=(8, 6))
sns.countplot(x="Education", hue="Loan_Status", data=pdf, edgecolor="black",
palette= "Set2")
plt.title("Education vs Loan Status")
plt.show()
```



```
# Married vs Loan Status
plt.figure(figsize=(8, 6))
sns.countplot(x="Married", hue="Loan_Status", data=pdf, edgecolor="black",
palette= "Blues")
plt.title("Married vs Loan Status")
plt.show()
```



```
# Gender vs Loan Status
plt.figure(figsize=(8, 6))
sns.countplot(x="Gender", hue="Loan_Status", data=pdf, edgecolor="black",
palette="coolwarm")
plt.title("Gender vs Loan Status")
plt.show()
```



## V. Feature Engineering:

```
# Label indexer
label_indexer = StringIndexer( inputCol="Loan_Status",
outputCol="Loan_Status_index")

categorical_cols =
["Gender","Married","Dependents","Education","Self_Employed","Property_Area"]
indexers = [ StringIndexer(inputCol=c, outputCol=f"{c}_index") for c in
categorical_cols ]

encoders = [OneHotEncoder( inputCols=[f"{c}_index"],
outputCols=[f"{c}_encoded"], dropLast=False) for c in categorical_cols]

numeric_cols =
["ApplicantIncome", "CoapplicantIncome", "LoanAmount", "Loan_Amount_Term", "Credit_History"]
feature_cols = [f"{c}_encoded" for c in categorical_cols] + numeric_cols

assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
preprocess_pipeline = Pipeline(stages=[label_indexer] + indexers + encoders +
[assembler])

processed_data = preprocess_pipeline.fit(data).transform(data)
processed_data.show(5)
```

## Output:

Loan_Status_index	Gender_index	Married_index	Dependents_index	Education_index	Self_Employed_index	Property_Area_index	Gender_encoded	Married_encoded	Dependents_encoded	Education_encoded
0.0	0.0	1.0	0.0	0.0	0.0	1.0	(2,[0],[1.0])	(2,[1],[1.0])	(4,[0],[1.0])	(2,[0],[1.0])
1.0	0.0	0.0	1.0	0.0	0.0	2.0	(2,[0],[1.0])	(2,[0],[1.0])	(4,[1],[1.0])	(2,[0],[1.0])
0.0	0.0	0.0	0.0	0.0	1.0	1.0	(2,[0],[1.0])	(2,[0],[1.0])	(4,[0],[1.0])	(2,[0],[1.0])
0.0	0.0	0.0	0.0	1.0	0.0	1.0	(2,[0],[1.0])	(2,[0],[1.0])	(4,[0],[1.0])	(2,[1],[1.0])
0.0	0.0	1.0	0.0	0.0	0.0	1.0	(2,[0],[1.0])	(2,[1],[1.0])	(4,[0],[1.0])	(2,[0],[1.0])

## VI. Model Training and Predictions:

```
# Train-test split
train_data, test_data = processed_data.randomSplit([0.8, 0.2], seed=42)

# Logistic Regression model
lr = LogisticRegression( featuresCol="features", labelCol="Loan_Status_index")

# Fit LR model
lr_model = lr.fit(train_data)

# Predictions
pred = lr_model.transform(test_data)
pred.select("Loan_Status", "Loan_Status_index", "prediction", "probability").show(5)
```

## Output:

Loan_Status	Loan_Status_index	prediction	probability
Y	0.0	0.0	[0.83253216505644...]
Y	0.0	0.0	[0.74788254915643...]
Y	0.0	0.0	[0.83202464179686...]
N	1.0	0.0	[0.74008156842011...]
Y	0.0	0.0	[0.80913146662578...]

only showing top 5 rows

## VII. Model Evaluation:

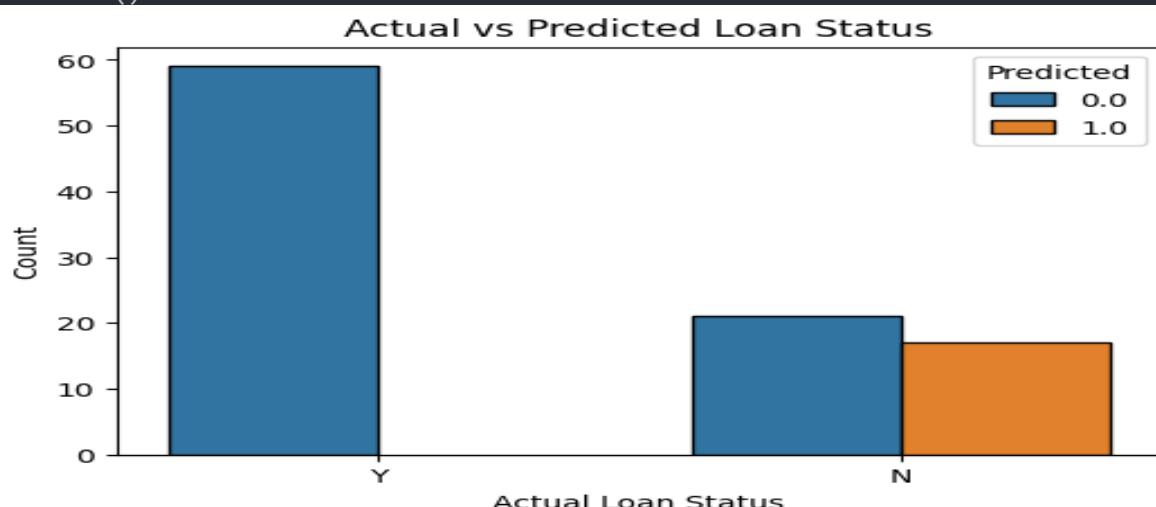
```
acc = MulticlassClassificationEvaluator( LabelCol="Loan_Status_index",  
predictionCol="prediction", metricName="accuracy").evaluate(pred)  
f1 = MulticlassClassificationEvaluator( LabelCol="Loan_Status_index",  
predictionCol="prediction", metricName="f1").evaluate(pred)  
recall = MulticlassClassificationEvaluator( LabelCol="Loan_Status_index",  
predictionCol="prediction", metricName="weightedRecall").evaluate(pred)  
precision = MulticlassClassificationEvaluator( LabelCol="Loan_Status_index",  
predictionCol="prediction", metricName="weightedPrecision").evaluate(pred)  
  
print(f'Recall: {recall*100:.2f}%')  
print(f'Precision: {precision*100:.2f}%')  
print(f'Accuracy: {acc*100:.2f}%')  
print(f'F1 Score: {f1*100:.2f}%')
```

### Output :

```
Recall: 78.35%  
Precision: 84.03%  
Accuracy: 78.35%  
F1 Score: 75.85%
```

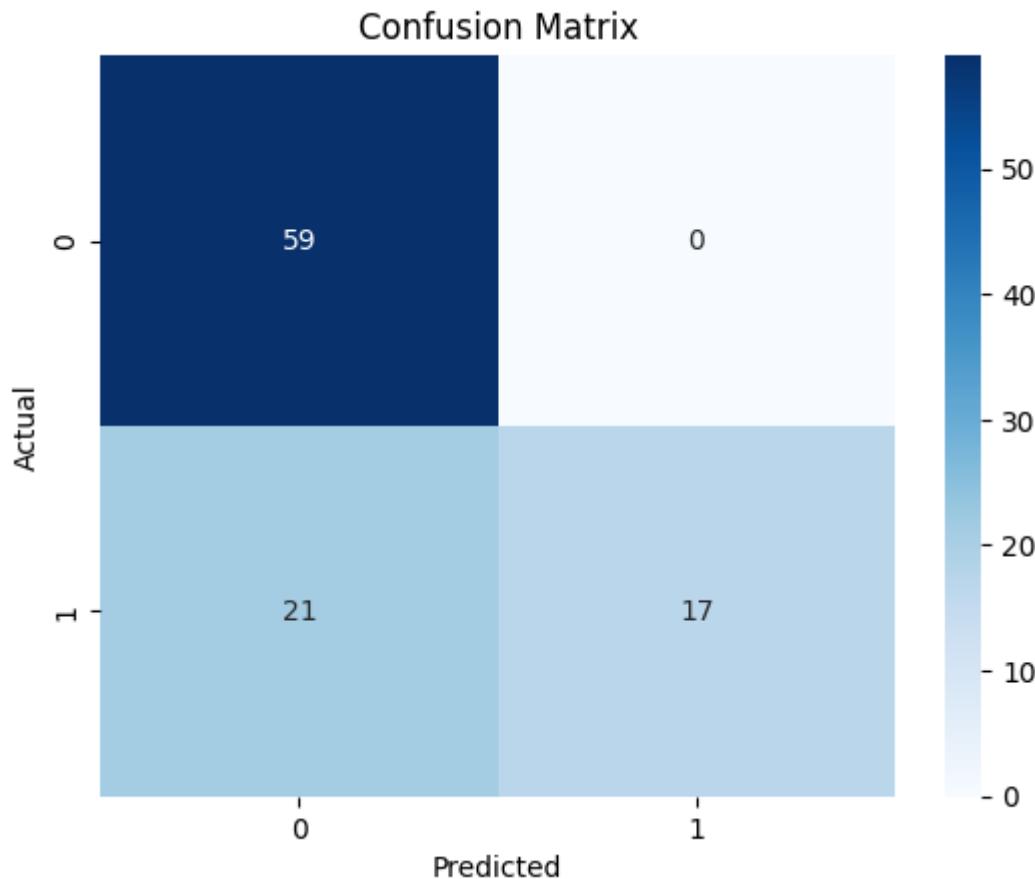
## Bar Chart for Actual vs Predicted Category

```
pdf = pred.select("Loan_Status", "prediction").toPandas()  
plt.figure(figsize=(6,4))  
sns.countplot(x="Loan_Status", hue="prediction", data=pdf, edgecolor="black")  
plt.title("Actual vs Predicted Loan Status")  
plt.xlabel("Actual Loan Status")  
plt.ylabel("Count")  
plt.legend(title="Predicted")  
plt.show()
```



## Confusion Matrix:

```
pdf = pred.select("Loan_Status_index", "prediction").toPandas()
cm = confusion_matrix(pdf["Loan_Status_index"], pdf["prediction"])
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```



## VIII. Results and Interpretation:

The Logistic Regression model achieved an Accuracy of **78.35%** and an F1-score of **75.85%**, with Precision at **84.03%** and Recall at **78.35%**.

The confusion matrix shows that the model correctly predicted most approved loans (59 correct approvals and 0

misclassified), but misclassified 21 rejected applications as approved.

Overall, the model is strong at detecting approved loans but less effective at identifying rejected ones. Despite this, performance is reasonable for a basic linear model on this dataset.

Future improvement can include balancing the dataset, using stronger models such as Random Forest or Gradient Boosting, adding more relevant features, and tuning hyper parameters to improve the model's ability to detect rejected loans.