Loan Eligibility Prediction using PySpark

Suvajyoti Chakraborty
Roll No.: A21034
Praxis Business School

INTRODUCTION

Loan Eligibility is defined as set of Criteria basis which a financial institution evaluates to decide the eligibility of a customer for a particular loan.

PROBLEM STATEMENT

Loan eligibility is decided after a long an intensive process of verification of documents and validation of set of criteria's which takes up huge amount of time .

The most needed loan applicants have to wait for long amount of time.

PREFERRED SOLUTION

Automation of the process reduce significant amount of time

Identifying the real segment of customers those are eligible for loan

This time consuming process can be solved by developing a system decided a person could take a loan or not using Machine Learning model.

TOOLS AND TECHNIQUES USED

- GOOGLE COLAB
- SPARK
- PYTHON
- MATPLOTLIB

STEPWISE DOINGS

- Collect Dataset (Uploaded on GitHub Repository)
- Setting up Spark Ecosystem
- Import dataset
- Null value imputation
- Exploratory Data Analysis
- Feature Engineering
- Data Preprocessing
- Model Preparation
- Model Building
- Conclusion

SETTING UP SPARK ENVIRONMENT

- Using google colab for this assignment.
- Using !wget command first installed the Spark
- !tar is the unzip programme of linux.
- Then install pyspark which allows to access spark with the help of python.
- Created local spark instance to check and run the spark.

```
!wget -q http://apache.osuosl.org/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz
!tar xf spark-3.1.2-bin-hadoop3.2.tgz
!pip install -q pyspark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-3.1.2-bin-hadoop3.2"
```

```
     |████████████████████████████████| 281.4 MB 37 kB/s
     |████████████████████████████████| 198 kB 54.8 MB/s
  Building wheel for pyspark (setup.py) ... done
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").config('spark.ui.port', '4050').getOrCreate()
```

```
sc = spark.sparkContext
```

```
sc
```

SparkContext

Spark UI

Version
    v3.1.2
Master
    local[*]
AppName
    pyspark-shell

IMPORTING THE DATA SET

- The dataset is already uploaded in my github repository
- Using !wget command loaded the data in the spark ecosystem.
- Read the data
- Count the number of rows and columns for cross checking.

```
[ ] !wget -q https://raw.githubusercontent.com/Suvajyoti/Loan_eligibility_prediction_with_pyspark/main/loan_eligibility_data.csv
```

```
[ ] ls

drive/                   spark-3.1.2-bin-hadoop3.2/
loan_eligibility_data.csv  spark-3.1.2-bin-hadoop3.2.tgz
sample_data/
```

Reading the CSV file with spark read csv command and stored it into a variable called df.

```
[ ] df=spark.read.csv('/content/loan_eligibility_data.csv',inferSchema=True,header=True)
```

Counting the number of rows and columns.There are total 614 rows and 13 columns.

```
[ ] print((df.count(),len(df.columns)))

(614, 13)
```

SCHEMA OF THE DATASET

- Loan_ID is the unique id against each customer who have applied for the loan.
- Gender, Married, Dependents, Education, Self Employed, Property Area, Loan Status are categorical data.
- Applicant Income, Co-applicant Income, Loan Amount, Loan Amount Term, Credit History are numerical data.
- In total 614 rows and 13 columns.

```
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)
```

SUMMARY OF THE DATASET

Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
601	611	599	614	582	614	614	592	600	564	614	614
null	null	0.5547445255474452	null	null	5403.459283387622	1621.245798027101	146.41216216216216	342.0	0.8421985815602837	null	null
null	null	0.7853289861674311	null	null	6109.041673387181	2926.2483692241894	85.58732523570545	65.12040985461255	0.3648783192364052	null	null
Female	No	0	Graduate	No	150	0.0	9	12	0	Rural	N
Male	Yes	3+	Not Graduate	Yes	81000	41667.0	700	480	1	Urban	Y

- Checking Five Summary Statistics

  Count, Mean, Standard Deviation, Minimum Range and Maximum Range

- Dropped Loan Id column

NULL VALUE HANDLING 1

Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
13	3	15	0	32	0	0	22	14	50	0	0

- Null/Missing values are present in the dataset.
- No null value is present in the target column.
- For numerical column mean and median and for categorical column mode will be used to impute the null/missing values.

NULL VALUE HANDLING 2

- First calculated the average loan amount of the column LoanAmount. The Average value is 146.412.
- Then imputed the average value, new average turns out to 146.39.

```
from pyspark.sql.functions import mean
mean_val = df.select(mean(df.LoanAmount)).collect()
print('Average value', mean_val[0][0])
```

```
Average value 146.41216216216216
```

```
mean_loan_amount = mean_val[0][0]
df = df.na.fill(mean_loan_amount, subset=['LoanAmount'])
mean_val = df.select(mean(df.LoanAmount)).collect()
print('New Average value', mean_val[0][0])
```

```
New Average value 146.3973941368078
```

NULL VALUE HANDLING 3

- First Calculated Medium of the Loan Amount Term. The median value is 360.
- Imputed the median value.

```
median_loan_term = df.approxQuantile("Loan_Amount_Term", [0.5], 0.25)
median_loan_term = int(median_loan_term[0])
print('Median value', median_loan_term)
```

```
Median value 360
```

```
df = df.na.fill(median_loan_term, subset=['Loan_Amount_Term'])
```

NULL VALUE HANDLING 4

- At first counted the null, female and male. Male count is much higher than the Female.Using Mode imputed the null values with Male.
- Same way imputed the null value for variable Married, dependents, self employed and credit history.

```
df.groupBy("Gender").count().show()
```

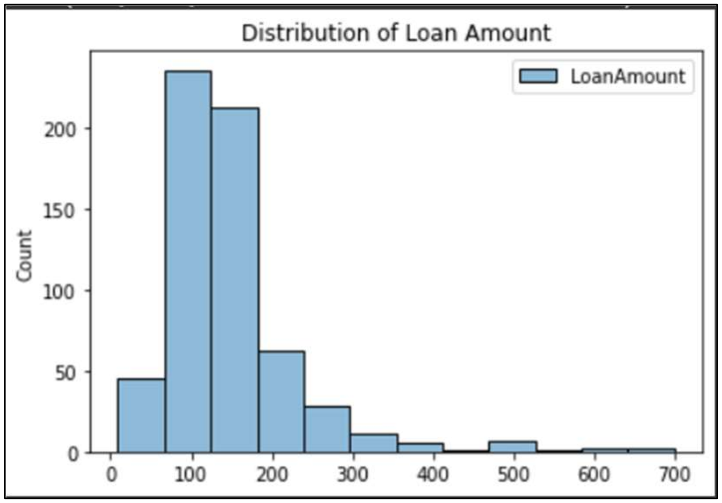
```
+------+-----+
|Gender|count|
+------+-----+
|  null|   13|
|Female|  112|
|  Male|  489|
+------+-----+
```

```
df = df.na.fill('Male',subset=['Gender'])
df.groupBy("Gender").count().show()
```

```
+------+-----+
|Gender|count|
+------+-----+
|Female|  112|
|  Male|  502|
+------+-----+
```

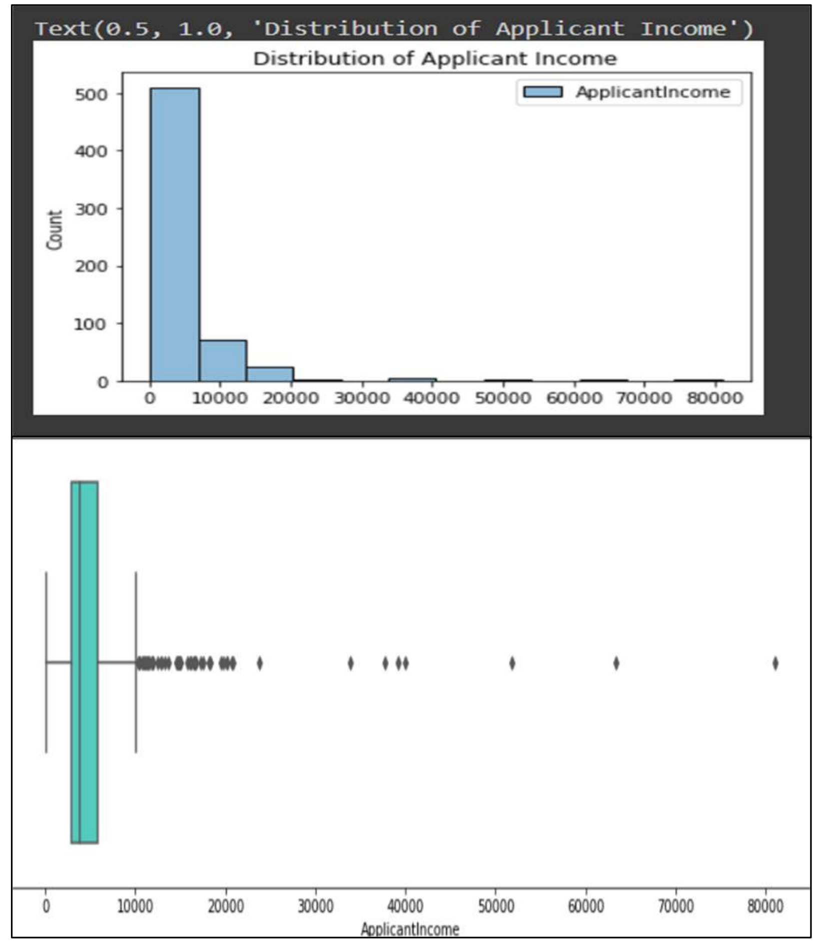
EXPLORATORY DATA ANALYSIS 1

- Histogram of Loan Amount clearly depicts that it is rightly skewed
- Mean value is greater than median value.



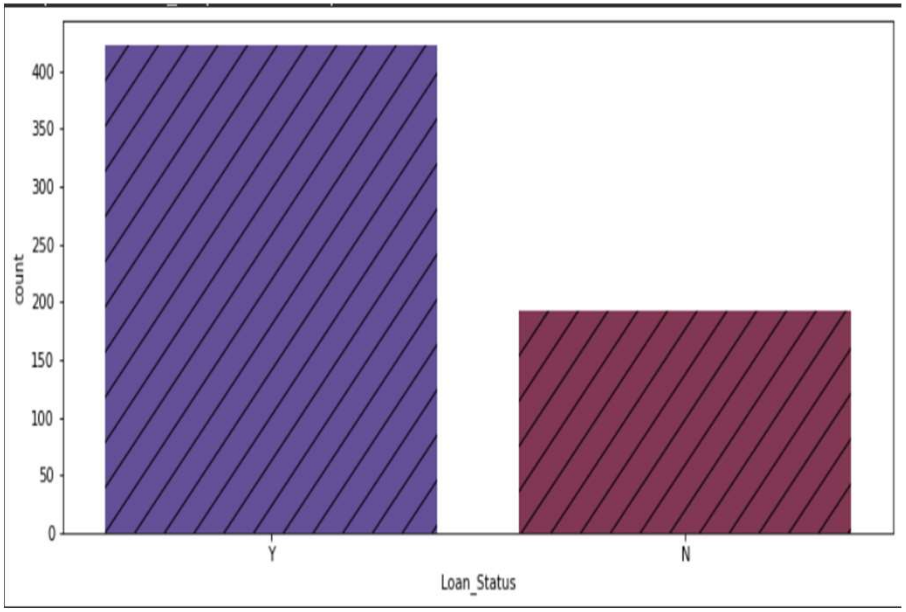
EXPLORATORY DATA ANALYSIS 2

- Histogram of Applicant Income clearly depicts that it is rightly skewed
- Mean value is greater than median value.
- Presence of outliers has been observed.



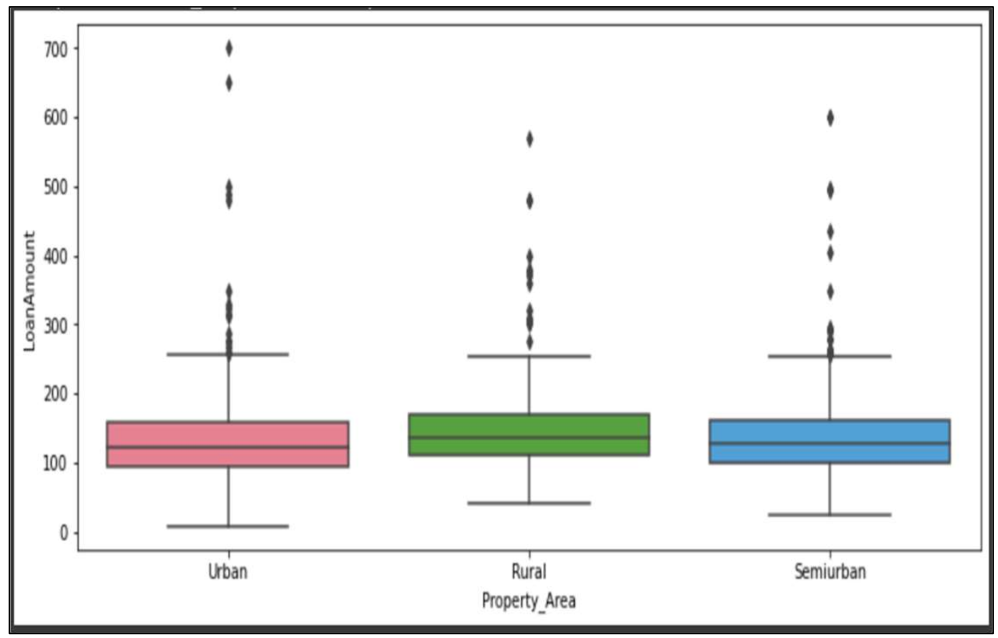
EXPLORATORY DATA ANALYSIS 3

- Target column Loan Status has high number of approval mentioned as yes.
- For modelling stratified sampling of train and test data will be required for good prediction.



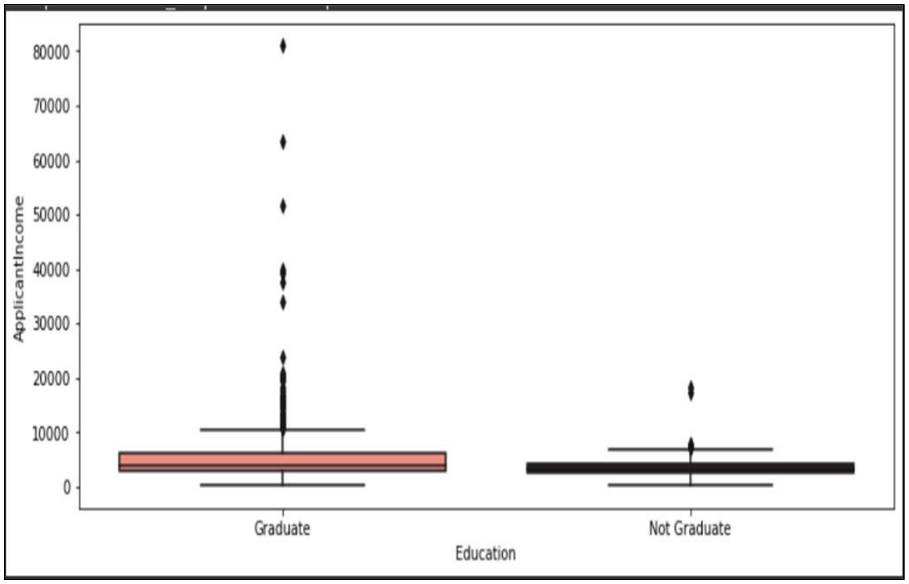
EXPLORATORY DATA ANALYSIS 4

- Median Loan Amount provided to Rural Area is slightly higher than Urban.
- There are some people in Urban who has got higher loan amounts compared to Rural or Semi urban.

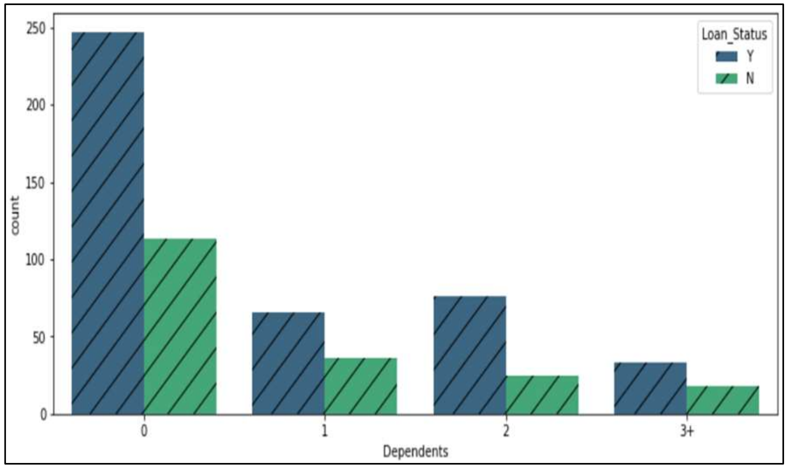


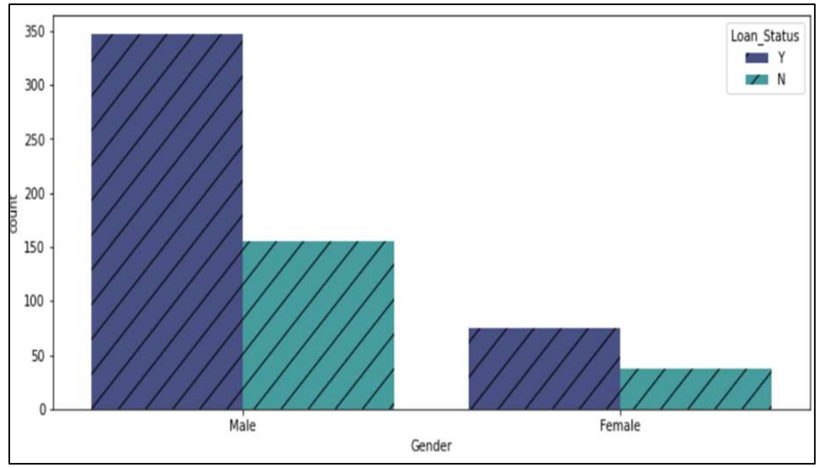
EXPLORATORY DATA ANALYSIS 5

- Some of the graduates are generally have more income than the not graduates.
- There is no huge difference between their mean income.



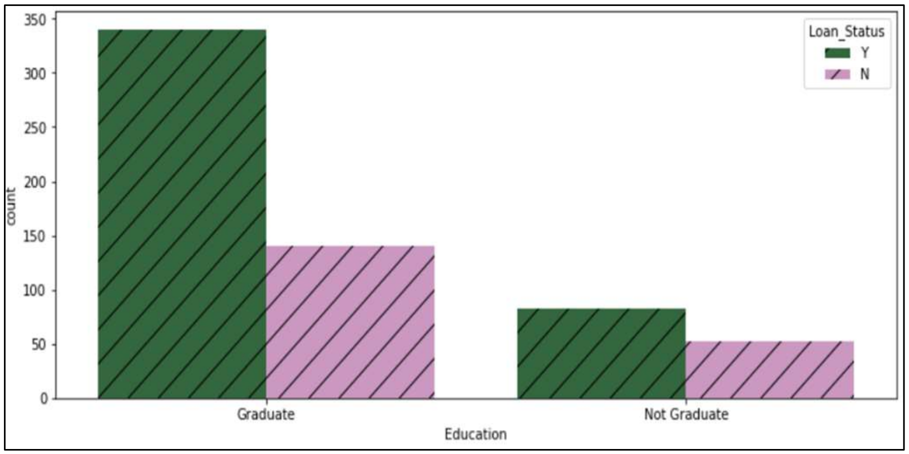
EXPLORATORY DATA ANALYSIS 6

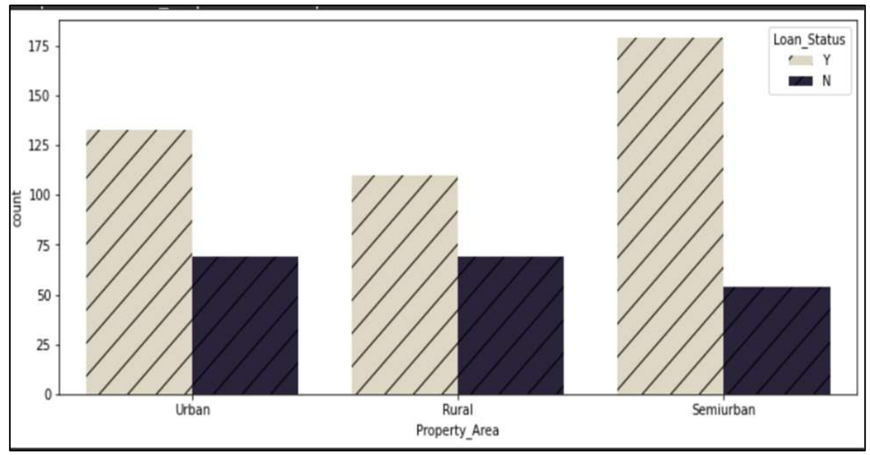




- Dependent 2 received more chance of getting loan.
- Loan Applicants are more in Male than the female but the chances of getting loan to the each category is almost equal.

EXPLORATORY DATA ANALYSIS 7





- Graduates have more chance of getting loan approved.
- Semi urban areas have much higher chance of getting loan than the other areas.

FEATURE ENGINEERING 1

Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	TotalIncome
Male	No	0	Graduate	No	5849	0.0	146	360	1	Urban	Y	5849.0
Male	Yes	1	Graduate	No	4583	1508.0	128	360	1	Rural	N	6091.0
Male	Yes	0	Graduate	Yes	3000	0.0	66	360	1	Urban	Y	3000.0
Male	Yes	0	Not Graduate	No	2583	2358.0	120	360	1	Urban	Y	4941.0
Male	No	0	Graduate	No	6000	0.0	141	360	1	Urban	Y	6000.0

only showing top 5 rows

- Creating a Total Income column to sum the amount of income of Applicant Income and co applicant income.

FEATURE ENGINEERING 2

Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status	TotalIncome	EMI
No	0	Graduate	No	5849	0.0	146	360	1	Urban	Y	5849.0	0.40555555555555556
Yes	1	Graduate	No	4583	1508.0	128	360	1	Rural	N	6091.0	0.35555555555555557
Yes	0	Graduate	Yes	3000	0.0	66	360	1	Urban	Y	3000.0	0.18333333333333332
Yes	0	Not Graduate	No	2583	2358.0	120	360	1	Urban	Y	4941.0	0.3333333333333333
No	0	Graduate	No	6000	0.0	141	360	1	Urban	Y	6000.0	0.39166666666666666

owing top 5 rows

- Creating EMI column by dividing the loan amount by the loan amount term.

DATA PREPROCESSING – STRING INDEXER & OHE

- StringIndexer encodes a string column of labels to a column of label indices. If the input column is numeric, we cast it to string and index the string values. The indices are in [0, numLabels).
- One Hot Encoding is used for converting categorical attributes into a numeric vector that machine learning models can understand.

```python
from pyspark.ml.feature import StringIndexer, OneHotEncoder
SI_gender = StringIndexer(inputCol='Gender',outputCol='gender_Index')
SI_married = StringIndexer(inputCol='Married',outputCol='married_Index')
SI_dependents = StringIndexer(inputCol='Dependents',outputCol='dependents_Index')
SI_education = StringIndexer(inputCol='Education',outputCol='education_Index')
SI_selfemp = StringIndexer(inputCol='Self_Employed',outputCol='selfemp_Index')
SI_credit = StringIndexer(inputCol='Credit_History',outputCol='credit_Index')
SI_property = StringIndexer(inputCol='Property_Area',outputCol='property_Index')
SI_loanstatus = StringIndexer(inputCol='Loan_Status',outputCol='loanstatus_Index')

df = SI_gender.fit(df).transform(df)
df = SI_married.fit(df).transform(df)
df = SI_dependents.fit(df).transform(df)
df = SI_education.fit(df).transform(df)
df = SI_selfemp.fit(df).transform(df)
df = SI_credit.fit(df).transform(df)
df = SI_property.fit(df).transform(df)
df = SI_loanstatus.fit(df).transform(df)
```

```python
OHE = OneHotEncoder(inputCols=['gender_Index', 'married_Index','dependents_Index','education_Index',
                               'selfemp_Index','credit_Index','property_Index','loanstatus_Index'],
                    outputCols=['gender_OHE', 'married_OHE','dependents_OHE','education_OHE',
                                'selfemp_OHE','credit_OHE','property_OHE','loanstatus_OHE'])
df = OHE.fit(df).transform(df)
df.select('gender_Index', 'gender_OHE', 'education_Index','education_OHE','credit_Index',
          'credit_OHE','property_Index','property_OHE').show(10)
```

DATA PREPROCESSING – CHECKING CORRELATION

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	TotalIncome	EMI	gender_Index	married_Index	dependents_Index	education_Index	selfemp_Index	credit_Index	property_Index	loanstatus_Index
ApplicantIncome	1.000000	-0.116605	0.565621	-0.046531	-0.018615	0.893037	0.320525	-0.058809	-0.051708	0.118202	-0.140760	0.127180	0.018615	0.017321	0.004710
CoapplicantIncome	-0.116605	1.000000	0.187863	-0.059383	0.011134	0.342781	0.135695	-0.082912	-0.075948	0.030430	-0.062290	-0.016100	-0.011134	0.019087	0.059187
LoanAmount	0.565621	0.187863	1.000000	0.036486	-0.001412	0.620118	0.491286	-0.107909	-0.147131	0.163108	-0.167041	0.115259	0.001412	0.028995	0.036345
Loan_Amount_Term	-0.046531	-0.059383	0.036486	1.000000	-0.004705	-0.070917	-0.501359	0.074030	0.100912	-0.103864	-0.073928	-0.033739	0.004705	-0.016086	0.022549
Credit_History	-0.018615	0.011134	-0.001412	-0.004705	1.000000	-0.012563	0.015005	-0.009170	-0.010938	-0.040160	-0.073658	-0.001550	-1.000000	-0.033102	-0.540556
TotalIncome	0.893037	0.342781	0.620118	-0.070917	-0.012563	1.000000	0.364654	-0.093191	-0.083319	0.125590	-0.161362	0.113000	0.012563	0.025032	0.031271
EMI	0.320525	0.135695	0.491286	-0.501359	0.015005	0.364654	1.000000	-0.060169	-0.094347	0.103414	-0.075777	0.051647	-0.015005	0.005380	0.013595
gender_Index	-0.058809	-0.082912	-0.107909	0.074030	-0.009170	-0.093191	-0.060169	1.000000	0.364569	-0.172914	-0.045364	0.000525	0.009170	-0.109521	0.017987
married_Index	-0.051708	-0.075948	-0.147131	0.100912	-0.010938	-0.083319	-0.094347	0.364569	1.000000	-0.334216	-0.012304	-0.004489	0.010938	0.007281	0.091478
dependents_Index	0.118202	0.030430	0.163108	-0.103864	-0.040160	0.125590	0.103414	-0.172914	-0.334216	1.000000	0.055752	0.056798	0.040160	-0.001601	-0.010118
education_Index	-0.140760	-0.062290	-0.167041	-0.073928	-0.073658	-0.161362	-0.075777	-0.045364	-0.012304	0.055752	1.000000	-0.010383	0.073658	0.066740	0.085884
selfemp_Index	0.127180	-0.016100	0.115259	-0.033739	-0.001550	0.113000	0.051647	0.000525	-0.004489	0.056798	-0.010383	1.000000	0.001550	0.007124	0.003700
credit_Index	0.018615	-0.011134	0.001412	0.004705	-1.000000	0.012563	-0.015005	0.009170	0.010938	0.040160	0.073658	0.001550	1.000000	0.033102	0.540556
property_Index	0.017321	0.019087	0.028995	-0.016086	-0.033102	0.025032	0.005380	-0.109521	0.007281	-0.001601	0.066740	0.007124	0.033102	1.000000	0.137545
loanstatus_Index	0.004710	0.059187	0.036345	0.022549	-0.540556	0.031271	0.013595	0.017987	0.091478	-0.010118	0.085884	0.003700	0.540556	0.137545	1.000000

- Applicant Income and Total Income are highly Correlated with each other.
- Loan Amount Term and Credit History are negatively correlated.

DATA PREPROCESSING – VECTOR ASSEMBLER

- Using Vector assembler to merge multiple column into a vector column and taken output as a single feature column. Inputs are given for all the necessary columns.
- Showing the transformed vector as features column with the loan status index column.

```
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=['gender_Index','married_Index','dependents_Index',
                                       'education_Index','selfemp_Index','ApplicantIncome',
                                       'CoapplicantIncome','EMI','LoanAmount','Loan_Amount_Term',
                                       'credit_Index','property_Index','gender_OHE',
                                       'married_OHE','dependents_OHE','education_OHE',
                                       'selfemp_OHE','credit_OHE','property_OHE'],
                            outputCol='features')
df2 = assembler.transform(df)
```

```
df2.select('features','loanstatus_Index').show(5)
```

```
+--------------------+----------------+
|            features|loanstatus_Index|
+--------------------+----------------+
|(22,[1,5,7,8,9,11...|             0.0|
|(22,[2,5,6,7,8,9,...|             1.0|
|(22,[4,5,7,8,9,11...|             0.0|
|(22,[3,5,6,7,8,9,...|             0.0|
|(22,[1,5,7,8,9,11...|             0.0|
+--------------------+----------------+
only showing top 5 rows
```

```
df2.select('features','loanstatus_OHE').show(5)
```

```
+--------------------+--------------+
|            features|loanstatus_OHE|
+--------------------+--------------+
|(22,[1,5,7,8,9,11...| (1,[0],[1.0])|
|(22,[2,5,6,7,8,9,...|     (1,[],[])|
|(22,[4,5,7,8,9,11...| (1,[0],[1.0])|
|(22,[3,5,6,7,8,9,...| (1,[0],[1.0])|
|(22,[1,5,7,8,9,11...| (1,[0],[1.0])|
+--------------------+--------------+
only showing top 5 rows
```

MODEL PREPARATION

- Renaming the loan status index column as label column.
- Splitting the data into training dataset and test dataset to perform Machine learning model building and verifying it. Choose random state 20.

```
model_df = df2.select(['features','loanstatus_Index'])
model_df = model_df.withColumnRenamed('loanstatus_Index','label')
model_df.printSchema()
```

```
root
 |-- features: vector (nullable = true)
 |-- label: double (nullable = false)
```

```
(train_data, test_data) = model_df.randomSplit([0.7, 0.3], 20) # random state - 20

print("Records for training: " + str(train_data.count()))
print("Records for evaluation: " + str(test_data.count()))
```

```
Records for training: 432
Records for evaluation: 182
```

MODEL BUILDING – STEP 1

- **Naive Bayes classification** is simply based on probabilistic classification with the assumption of independence between the feature variables. It is a conditional probability model.

- For implementing the model from pyspark ml classification model imported the naive bayes classifier. Created a instance of that with mentioning the feature columns , label columns and model type. Fit the model into the train data and performed prediction with test data.

- For evaluation of the performance of the model from pyspark ml evaluation imported the Binary Classification Evaluator and measured the AUC(Area under the curve).More the AUC better the prediction.
- AUC is **0.52**.

```
from pyspark.ml.classification import NaiveBayes

nb = NaiveBayes(featuresCol = 'features',
                             labelCol = 'label',modelType='multinomial')
nbModel = nb.fit(train_data)
predictions = nbModel.transform(test_data)
predictions.show(10)
```

```
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|(22,[0,1,2,3,5,7,...|  1.0|[-2862.0256478600...|[1.0,2.2940272603...|       0.0|
|(22,[0,1,2,5,7,8,...|  1.0|[-2749.6537598875...|[1.0,1.0647420900...|       0.0|
|(22,[0,1,2,5,7,8,...|  1.0|[-2886.7763942280...|[1.0,4.9258940609...|       0.0|
|(22,[0,1,3,4,5,7,...|  0.0|[-7017.2782940283...|           [1.0,0.0]|       0.0|
|(22,[0,1,3,5,7,8,...|  0.0|[-2081.8085985339...|[1.0,1.2764559524...|       0.0|
|(22,[0,1,3,5,7,8,...|  0.0|[-2939.8180470506...|[1.0,8.3413257530...|       0.0|
|(22,[0,1,4,5,7,8,...|  0.0|[-2679.3679550083...|[1.0,2.5469213055...|       0.0|
|(22,[0,1,5,6,7,8,...|  0.0|[-4716.2677853899...|[2.95096840479742...|       1.0|
|(22,[0,1,5,6,7,8,...|  0.0|[-5018.2315279663...|[1.76912468006447...|       1.0|
|(22,[0,1,5,7,8,9,...|  1.0|[-2395.7246968486...|[1.0,4.7304920413...|       0.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 10 rows
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator()
print("Naive Bayes - Test set AUC: " + str(evaluator.evaluate
                                  (predictions,
                                   {evaluator.metricName: "areaUnderROC"})))
```

```
Naive Bayes - Test set AUC: 0.5232202447163516
```

MODEL BUILDING – STEP 2

- **Decision Tree** is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- AUC is **0.31**.

```
from pyspark.ml.classification import DecisionTreeClassifier

dt = DecisionTreeClassifier(featuresCol = 'features',
                            labelCol = 'label', maxDepth = 3)
dtModel = dt.fit(train_data)
predictions = dtModel.transform(test_data)
predictions.show(10)
```

```
+--------------------+-----+-------------+--------------------+----------+
|            features|label|rawPrediction|         probability|prediction|
+--------------------+-----+-------------+--------------------+----------+
|(22,[0,1,2,3,5,7,...|  1.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,2,5,7,8,...|  1.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,2,5,7,8,...|  1.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,3,4,5,7,...|  0.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,3,5,7,8,...|  0.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,3,5,7,8,...|  0.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,4,5,7,8,...|  0.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,5,6,7,8,...|  0.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,5,6,7,8,...|  0.0| [287.0,72.0]|[0.79944289693593...|       0.0|
|(22,[0,1,5,7,8,9,...|  1.0|   [6.0,56.0]|[0.09677419354838...|       1.0|
+--------------------+-----+-------------+--------------------+----------+
only showing top 10 rows
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator()
print("Decision Tree - Test set AUC: " + str(evaluator.evaluate
                                        (predictions,
                                         {evaluator.metricName: "areaUnderROC"})))
```

```
Decision Tree - Test set AUC: 0.3090934371523915
```

MODEL BUILDING – STEP 3

- **Random Forest** is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.
- AUC is **0.75**

```
from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol = 'features', labelCol = 'label')
rfModel = rf.fit(train_data)
predictions = rfModel.transform(test_data)
predictions.show(10)
```

```
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|(22,[0,1,2,3,5,7,...|  1.0|[15.2022381049376...|[0.76011190524688...|       0.0|
|(22,[0,1,2,5,7,8,...|  1.0|[15.5211954150533...|[0.77605977075266...|       0.0|
|(22,[0,1,2,5,7,8,...|  1.0|[17.0180388144164...|[0.85090194072082...|       0.0|
|(22,[0,1,3,4,5,7,...|  0.0|[16.0389948622396...|[0.80194974311198...|       0.0|
|(22,[0,1,3,5,7,8,...|  0.0|[14.9811865576138...|[0.74905932788069...|       0.0|
|(22,[0,1,3,5,7,8,...|  0.0|[16.5592377067984...|[0.82796188533992...|       0.0|
|(22,[0,1,4,5,7,8,...|  0.0|[15.8371635155710...|[0.79185817577855...|       0.0|
|(22,[0,1,5,6,7,8,...|  0.0|[14.6383539600041...|[0.73191769800020...|       0.0|
|(22,[0,1,5,6,7,8,...|  0.0|[15.5542018678766...|[0.77771009339383...|       0.0|
|(22,[0,1,5,7,8,9,...|  1.0|[3.04990549351583...|[0.15249527467579...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 10 rows
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator()
print("Random Forest - Test set AUC: " + str(evaluator.evaluate
                                        (predictions,
                                         {evaluator.metricName: "areaUnderROC"})))
```

```
Random Forest - Test set AUC: 0.7577864293659624
```

MODEL BUILDING – STEP 4

- **Gradient Boosting** algorithm is one of the most powerful algorithms in the field of machine learning. As we know that the errors in machine learning algorithms are broadly classified into two categories i.e. Bias Error and Variance Error. As gradient boosting is one of the boosting algorithms it is used to minimize bias error of the model.
- AUC is **0.71**

```
from pyspark.ml.classification import GBTClassifier
gbm = GBTClassifier(featuresCol='features', labelCol='label')
gbm_model = gbm.fit(train_data)
predictions = gbm_model.transform(test_data)
predictions.show(10)
```

```
+--------------------+-----+--------------------+--------------------+----------+
|            features|label|       rawPrediction|         probability|prediction|
+--------------------+-----+--------------------+--------------------+----------+
|(22,[0,1,2,3,5,7,...|  1.0|[0.07796957572121...|[0.53890598000683...|       0.0|
|(22,[0,1,2,5,7,8,...|  1.0|[0.71262844996282...|[0.80616120988108...|       0.0|
|(22,[0,1,2,5,7,8,...|  1.0|[0.73006972095076...|[0.81155400113496...|       0.0|
|(22,[0,1,3,4,5,7,...|  0.0|[0.83564067112004...|[0.84174658257578...|       0.0|
|(22,[0,1,3,5,7,8,...|  0.0|[1.06491833399942...|[0.89376950167066...|       0.0|
|(22,[0,1,3,5,7,8,...|  0.0|[0.77588183756646...|[0.82516831536389...|       0.0|
|(22,[0,1,4,5,7,8,...|  0.0|[0.41620601645343...|[0.69686469981676...|       0.0|
|(22,[0,1,5,6,7,8,...|  0.0|[1.08009944989005...|[0.89661798684676...|       0.0|
|(22,[0,1,5,6,7,8,...|  0.0|[0.79580054414847...|[0.83084124574319...|       0.0|
|(22,[0,1,5,7,8,9,...|  1.0|[-1.5363277580638...|[0.04424938613636...|       1.0|
+--------------------+-----+--------------------+--------------------+----------+
only showing top 10 rows
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator()
print("Gradient Boost - Test set AUC: " + str(evaluator.evaluate
                                        (predictions,
                                         {evaluator.metricName: "areaUnderROC"})))
```

```
Gradient Boost - Test set AUC: 0.7134315906562851
```

CONCLUSION

- Among the four models Random Forest preforms best. So, will use Random Forest for model building.
- Build a model which can automate the loan eligible process and will reduce time significantly.

THANK YOU!