Installing PySpark

To install PySpark, you will need to follow these steps:

- First, you will need to install Java Development Kit (JDK) 8 or later on your machine. You can download it from the Oracle website.
- Next, you will need to download and install Apache Spark. You can download the latest version of Apache Spark from the official website.
- Once you have downloaded and installed Apache Spark, you will need to set the SPARK_HOME environment variable to the installation directory of Spark.
- After setting the environment variable, you can install PySpark using pip. Open a terminal or command prompt and type the following command:

```
pip install pyspark
```

- Once PySpark is installed, you can test the installation by running the following code in a Python script:

```
from pyspark.sql import SparkSession

    spark = SparkSession.builder \
            .appName("Test") \
            .getOrCreate()

    df = spark.createDataFrame([(1, "foo"), (2, "bar")], ("id", "value"))
    df.show()
```

In [1]:
```
# install java
# install apache spark with hadoop
# set environment variables
# !pip install pyspark
```

Import Libraries

In [3]:
```
import pyspark
from pyspark.sql import SparkSession
import pyspark.sql.functions as F

import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
# initialize the session
spark = SparkSession.builder.appName('loan_prediction').getOrCreate()
```

Load the Dataset

In [66]:
```
df = spark.read.csv('Loan Prediction Dataset.csv', header=True, sep=',', inferSchema=True)
df.show(5)
```

```
+--------+------+-------+----------+------------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+
| Loan_ID|Gender|Married|Dependents|   Education|Self_Employed|ApplicantIncome|CoapplicantIncome|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|       120|             360|             1|        Urban|          Y|
|LP001008|  Male|     No|         0|    Graduate|           No|           6000|              0.0|       141|             360|             1|        Urban|          Y|
+--------+------+-------+----------+------------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+
only showing top 5 rows
```

In [67]:
```
df.printSchema()
```

```
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)
```

In [68]:
```
df.dtypes
```

Out[68]:
```
[('Loan_ID', 'string'),
 ('Gender', 'string'),
 ('Married', 'string'),
 ('Dependents', 'string'),
 ('Education', 'string'),
 ('Self_Employed', 'string'),
 ('ApplicantIncome', 'int'),
 ('CoapplicantIncome', 'double'),
 ('LoanAmount', 'int'),
 ('Loan_Amount_Term', 'int'),
 ('Credit_History', 'int'),
 ('Property_Area', 'string'),
 ('Loan_Status', 'string')]
```

In [69]:
```
# convert spark dataframe to pandas
pandas_df = df.toPandas()
pandas_df.head()
```

Out[69]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban	Y
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural	N
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban	Y
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban	Y
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban	Y

Data Analysis

In [8]:
```
# display count based on loan status
df.groupBy('Loan_Status').count().show()
```

```
+-----------+-----+
|Loan_Status|count|
+-----------+-----+
|          Y|  422|
|          N|  192|
+-----------+-----+
```

In [9]:
```
df.select("Credit_History", "Loan_Status").groupBy('Loan_Status').agg(F.avg('Credit_History')).show()
```

```
+-----------+-------------------+
|Loan_Status|avg(Credit_History)|
+-----------+-------------------+
|          Y| 0.9818181818181818|
|          N| 0.5418994413407822|
+-----------+-------------------+
```

In [10]:
```
df.select('Gender', 'Loan_Status').groupBy('Loan_Status', 'Gender').count().show()
```

```
+-----------+------+-----+
|Loan_Status|Gender|count|
+-----------+------+-----+
|          N|Female|   37|
|          Y|  null|    8|
|          Y|Female|   75|
|          N|  null|    5|
|          Y|  Male|  339|
|          N|  Male|  150|
+-----------+------+-----+
```

Correlation Matrix

In [15]:
```
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('')
```

Out[15]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
ApplicantIncome	1.00	-0.12	0.54	-0.02	0.01
CoapplicantIncome	-0.12	1.00	0.19	-0.05	-0.06
LoanAmount	0.54	0.19	1.00	0.06	-0.03
Loan_Amount_Term	-0.02	-0.05	0.06	1.00	0.05
Credit_History	0.01	-0.06	-0.03	0.05	1.00

SQL Operations

In [16]:
```
import pyspark.sql as sparksql
```

In [17]:
```
df.createOrReplaceTempView('table')
```

In [19]:
```
# display top rows from the table
spark.sql("select * from table limit 5").show()
```

```
+--------+------+-------+----------+------------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+
| Loan_ID|Gender|Married|Dependents|   Education|Self_Employed|ApplicantIncome|CoapplicantIncome|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|       120|             360|             1|        Urban|          Y|
|LP001008|  Male|     No|         0|    Graduate|           No|           6000|              0.0|       141|             360|             1|        Urban|          Y|
+--------+------+-------+----------+------------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+
```

In [20]:
```
spark.sql('select Loan_ID from table where Credit_History=1').show()
```

```
+--------+
| Loan_ID|
+--------+
|LP001002|
|LP001003|
|LP001005|
|LP001006|
|LP001008|
|LP001011|
|LP001013|
|LP001018|
|LP001020|
|LP001024|
|LP001027|
|LP001028|
|LP001029|
|LP001030|
|LP001032|
|LP001038|
|LP001041|
|LP001046|
|LP001066|
|LP001068|
+--------+
only showing top 20 rows
```

Data Preprocessing

In [21]:
```
# display null values
df.select([F.count(F.when(F.col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

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

In [23]:
```
# get mean value of column
mean = df.select(F.mean(df['LoanAmount'])).collect()[0][0]
mean
```

Out[23]:
```
146.41216216216216
```

In [24]:
```
# fill null value
df = df.na.fill(mean, ['LoanAmount'])
```

In [26]:
```
# get mode value of column
df.groupby('Gender').count().orderBy("count", ascending=False).first()[0]
```

Out[26]:
```
'Male'
```

In [27]:
```
# fill null values for all the columns
numerical_cols = ['LoanAmount', 'Loan_Amount_Term']
categorical_cols = ['Gender', 'Married', 'Dependents', 'Self_Employed', 'Credit_History']
```

In [28]:
```
for col in numerical_cols:
    mean = df.select(F.mean(df[col])).collect()[0][0]
    df = df.na.fill(mean, [col])
```

In [29]:
```
for col in categorical_cols:
    mode = df.groupby(col).count().orderBy("count", ascending=False).first()[0]
    df = df.na.fill(mode, [col])
```

In [30]:
```
# display null values
df.select([F.count(F.when(F.col(c).isNull(), c)).alias(c) for c in df.columns]).show()
```

```
+-------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+
|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|
+-------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+
```

In [31]:
```
# create new feature column
df = df.withColumn('TotalIncome', F.col('ApplicantIncome') + F.col('CoapplicantIncome'))
df.show(2)
```

```
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+
| Loan_ID|Gender|Married|Dependents|Education|Self_Employed|ApplicantIncome|CoapplicantIncome|LoanAmount|Loan_Amount_Term|Credit_History|Property_Area|Loan_Status|TotalIncome|
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+
|LP001002|  Male|     No|         0| Graduate|           No|           5849|              0.0|       146|             360|             1|        Urban|          Y|     5849.0|
|LP001003|  Male|    Yes|         1| Graduate|           No|           4583|           1508.0|       128|             360|             1|        Rural|          N|     6091.0|
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+
only showing top 2 rows
```

In [32]:
```
# how to find and replace values
df = df.withColumn('Loan_Status', F.when(df['Loan_Status']=='Y', 1).otherwise(0))
df.show(2)
```

```
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+
| Loan_ID|Gender|Married|Dependents|Education|Self_Employed|ApplicantIncome|CoapplicantIncome|LoanAmount|Loan_Amount_Term|Credit_History|Property_Area|Loan_Status|TotalIncome|
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+
|LP001002|  Male|     No|         0| Graduate|           No|           5849|              0.0|       146|             360|             1|        Urban|          1|     5849.0|
|LP001003|  Male|    Yes|         1| Graduate|           No|           4583|           1508.0|       128|             360|             1|        Rural|          0|     6091.0|
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+
only showing top 2 rows
```

Feature Extraction

In [33]:
```
df.printSchema()
```

```
root
 |-- Loan_ID: string (nullable = true)
 |-- Gender: string (nullable = false)
 |-- Married: string (nullable = false)
 |-- Dependents: string (nullable = false)
 |-- Education: string (nullable = true)
 |-- Self_Employed: string (nullable = false)
 |-- 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: integer (nullable = false)
 |-- TotalIncome: double (nullable = true)
```

In [34]:
```
from pyspark.ml.feature import VectorAssembler, OneHotEncoder, StringIndexer
from pyspark.ml import Pipeline
```

In [49]:
```
categorical_columns = ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Property_Area', 'Credit_History']
numerical_columns = ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Loan_Amount_Term', 'TotalIncome']

# index the string columns
indexers = [StringIndexer(inputCol=col, outputCol="{0}_index".format(col)) for col in categorical_columns]

# encode the indexed values
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

# vectorize the encoded values
assembler = VectorAssembler(inputCols=input_columns, outputCol="feature")
```

In [50]:
```
# create the pipeline to transform the data
pipeline = Pipeline(stages = indexers + encoders + [assembler])
```

In [51]:
```
data_model = pipeline.fit(df)
```

In [52]:
```
transformed_df = data_model.transform(df)
```

In [53]:
```
transformed_df.show(1)
```

```
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+-----
-------+-------------+----------------+---------------+-------------------+-------------------+--------------------+---------------------+----------------------+---------------
---+----------------------+---------------------------+--------------------------+------------------------------+--------------------+
| Loan_ID|Gender|Married|Dependents|Education|Self_Employed|ApplicantIncome|CoapplicantIncome|LoanAmount|Loan_Amount_Term|Credit_History|Property_Area|Loan_Status|TotalIncome|Gende
r_index|Married_index|Dependents_index|Education_index|Self_Employed_index|Property_Area_index|Credit_History_index|Gender_index_encoded|Married_index_encoded|Dependents_index_enco
ded|Education_index_encoded|Self_Employed_index_encoded|Property_Area_index_encoded|Credit_History_index_encoded|             feature|
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+-----
-------+-------------+----------------+---------------+-------------------+-------------------+--------------------+---------------------+----------------------+---------------
---+----------------------+---------------------------+--------------------------+------------------------------+--------------------+
|LP001002|  Male|     No|         0| Graduate|           No|           5849|              0.0|       146|             360|             1|        Urban|          1|     5849.0|
0.0|          1.0|             0.0|            0.0|                0.0|                1.0|                 0.0|      (2,[0],[1.0])|        (2,[1],[1.0])|          (4,[0],[1.0])|
(2,[0],[1.0])|             (2,[0],[1.0])|             (3,[1],[1.0])|               (2,[0],[1.0])|(22,[0,3,4,8,10,1...|
+--------+------+-------+----------+---------+-------------+---------------+-----------------+----------+----------------+--------------+-------------+-----------+-----------+-----
-------+-------------+----------------+---------------+-------------------+-------------------+--------------------+---------------------+----------------------+---------------
---+----------------------+---------------------------+--------------------------+------------------------------+--------------------+
only showing top 1 row
```

In [54]:
```
# get input feature and output columns
transformed_df = transformed_df.select(['feature', 'Loan_Status'])
```

In [55]:
```
# split the data for train and test
train_data, test_data = transformed_df.randomSplit([0.8, 0.2], seed=42)
```

In [56]:
```
train_data.show(5)
```

```
+--------------------+-----------+
|             feature|Loan_Status|
+--------------------+-----------+
|(22,[0,2,4,8,10,1...|          1|
|(22,[0,2,4,8,10,1...|          1|
|(22,[0,2,4,8,10,1...|          0|
|(22,[0,2,4,8,10,1...|          1|
|(22,[0,2,4,8,10,1...|          1|
+--------------------+-----------+
only showing top 5 rows
```

Model Training & Testing

In [57]:
```
from pyspark.ml.classification import LogisticRegression, RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

In [58]:
```
lr = LogisticRegression(featuresCol='feature', labelCol='Loan_Status')
lr_model = lr.fit(train_data)
```

In [59]:
```
# predict on test data
predictions = lr_model.transform(test_data)
predictions.show(5)
```

```
+--------------------+-----------+--------------------+--------------------+----------+
|             feature|Loan_Status|       rawPrediction|         probability|prediction|
+--------------------+-----------+--------------------+--------------------+----------+
|(22,[0,2,4,8,10,1...|          1|[-2.1309019654107...|[0.10612939546952...|       1.0|
|(22,[0,2,4,8,10,1...|          1|[-2.1965880165663...|[0.10005730505861...|       1.0|
|(22,[0,2,4,8,10,1...|          1|[-2.1183870711017...|[0.10732249829704...|       1.0|
|(22,[0,2,4,8,10,1...|          1|[-2.1420673697694...|[0.10507482746799...|       1.0|
|(22,[0,2,4,8,10,1...|          1|[-2.1120272450757...|[0.10793332086955...|       1.0|
+--------------------+-----------+--------------------+--------------------+----------+
only showing top 5 rows
```

In [60]:
```
predictions = lr_model.transform(test_data)
auc = BinaryClassificationEvaluator().setLabelCol('Loan_Status')
print('AUC:', str(auc.evaluate(predictions)))
```

```
AUC: 0.782010582010582
```

In [61]:
```
rf = RandomForestClassifier(featuresCol='feature', labelCol='Loan_Status')
rf_model = rf.fit(train_data)
```

In [62]:
```
predictions = rf_model.transform(test_data)
auc = BinaryClassificationEvaluator().setLabelCol('Loan_Status')
print('AUC:', str(auc.evaluate(predictions)))
```

```
AUC: 0.8291005291005291
```

In [ ]: