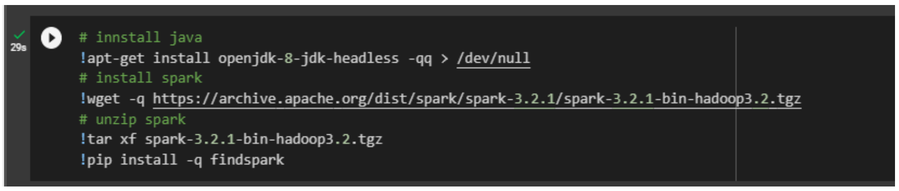
Introduction to BigData and Data Science

ICE 8

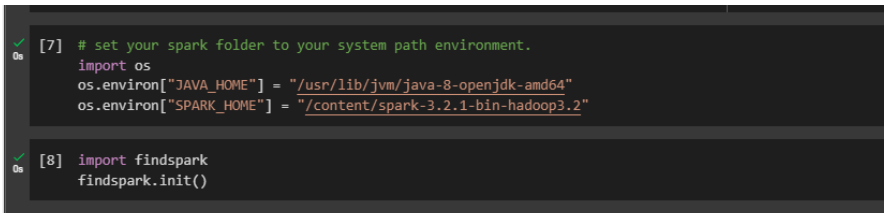
Task 1

Opened google colab and opened a new notebook. Installed java 8 and spark 3.2.1.



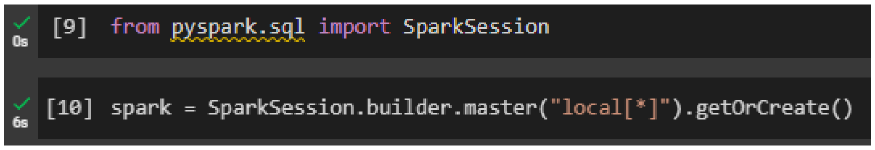
```
# innstall java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
# install spark
!wget -q https://archive.apache.org/dist/spark/spark-3.2.1/spark-3.2.1-bin-hadoop3.2.tgz
# unzip spark
!tar xf spark-3.2.1-bin-hadoop3.2.tgz
!pip install -q findspark
```

Referenced java and spark to environment variables and imported findspark to integrate python with spark



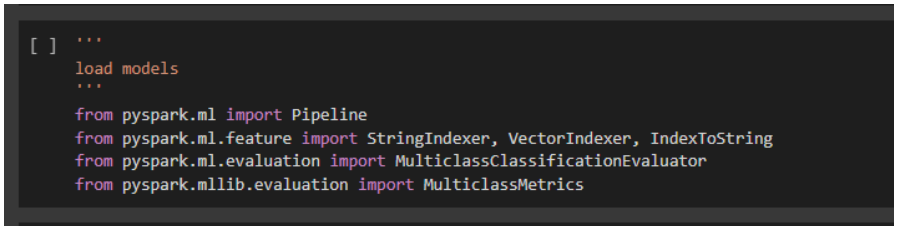
```
[7] # set your spark folder to your system path environment.
    import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark-3.2.1-bin-hadoop3.2"

[8] import findspark
    findspark.init()
```

Imported spark session from pyspark and created a spark session.



```
[9]  from pyspark.sql import SparkSession

[10] spark = SparkSession.builder.master("local[*]").getOrCreate()
```

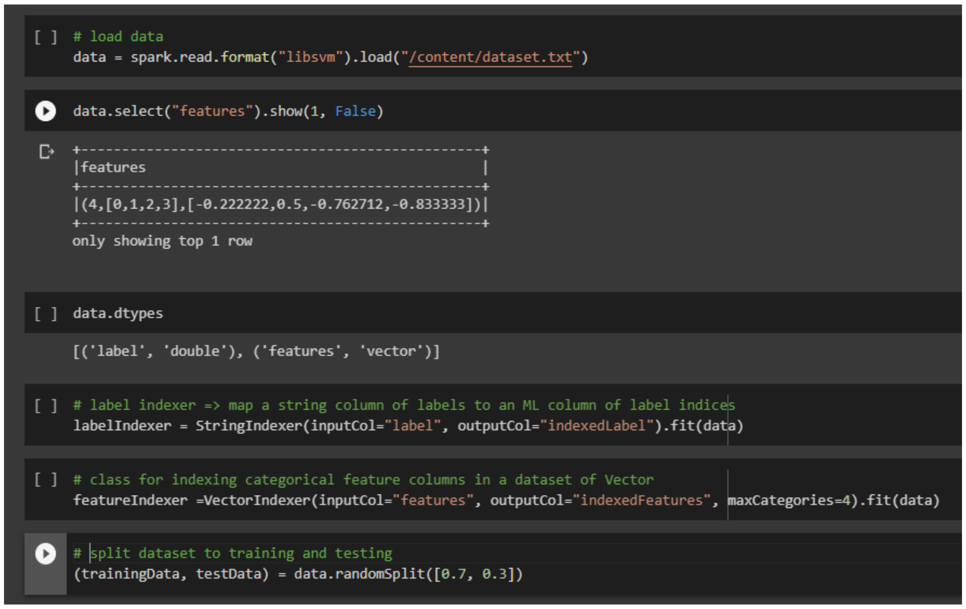
Loaded the necessary models from pyspark.ml package.



```
[ ] '''
    load models
    '''
    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, VectorIndexer, IndexToString
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    from pyspark.mllib.evaluation import MulticlassMetrics
```

Loaded the dataset and assigned it to the data variable and then viewed the data in features column for verification of data load. Checked for data types of the fields in the dataset. Indexed the columns of input with the output columns. The label column is indexed to indexedLabel column in the output. The features column is vector indexed to the column indexedFeatures in the output.

The data is then split into 2 for separating the testing and training data. The model is given with 70% of the data for training and the remaining 30% is kept for testing.



```
# load data
data = spark.read.format("libsvm").load("/content/dataset.txt")
```

```
data.select("features").show(1, False)
```

```
+-----------------------------------------------+
|features                                       |
+-----------------------------------------------+
|(4,[0,1,2,3],[-0.222222,0.5,-0.762712,-0.833333])|
+-----------------------------------------------+
only showing top 1 row
```

```
data.dtypes
```

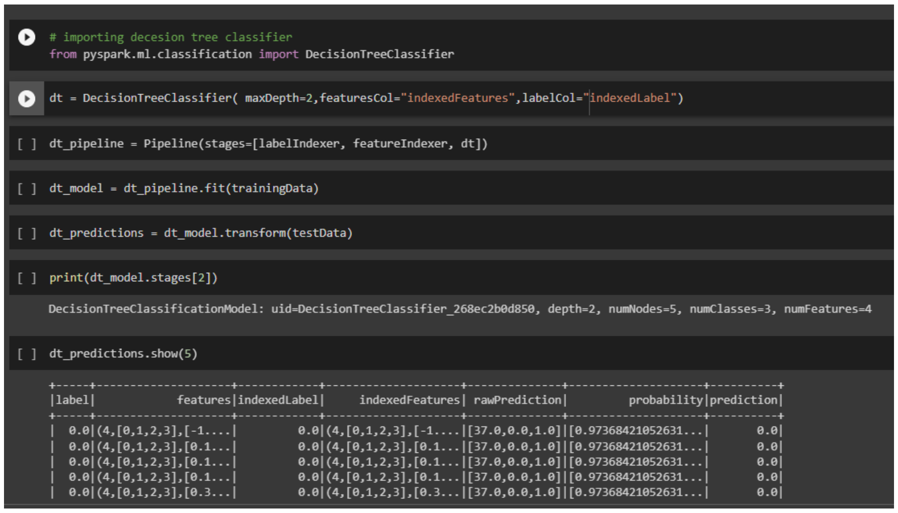
```
[('label', 'double'), ('features', 'vector')]
```

```
# label indexer => map a string column of labels to an ML column of label indices
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
```

```
# class for indexing categorical feature columns in a dataset of Vector
featureIndexer =VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
```

```
# split dataset to training and testing
(trainingData, testData) = data.randomSplit([0.7, 0.3])
```

Imported the Decision tree classifier model from pyspark ml classification to build an Decision tree classifier. The model is built with a depth of 2 and feature column as indexedFeatures and label column as indexedLabel. A pipeline has been created for the model with stages as label indexer and feature indexer for the data(dt). Then pipeline has been fitted with the training data which builds the model. Then predictions of the built model are obtained by providing the test data to the model. The prediction results of the model are viewed.



```
# importing decesion tree classifier
from pyspark.ml.classification import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier( maxDepth=2,featuresCol="indexedFeatures",labelCol="indexedLabel")
```

```
dt_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, dt])
```

```
dt_model = dt_pipeline.fit(trainingData)
```

```
dt_predictions = dt_model.transform(testData)
```

```
print(dt_model.stages[2])
```

```
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_268ec2b0d850, depth=2, numNodes=5, numClasses=3, numFeatures=4
```

```
dt_predictions.show(5)
```

```
+-----+------------------+-----------+------------------+-------------+------------------+----------+
|label|          features|indexedLabel|   indexedFeatures| rawPrediction|       probability|prediction|
+-----+------------------+-----------+------------------+-------------+------------------+----------+
|  0.0|(4,[0,1,2,3],[-1....|        0.0|(4,[0,1,2,3],[-1....|[37.0,0.0,1.0]|[0.97368421052631...|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|        0.0|(4,[0,1,2,3],[0.1...|[37.0,0.0,1.0]|[0.97368421052631...|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|        0.0|(4,[0,1,2,3],[0.1...|[37.0,0.0,1.0]|[0.97368421052631...|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|        0.0|(4,[0,1,2,3],[0.1...|[37.0,0.0,1.0]|[0.97368421052631...|       0.0|
|  0.0|(4,[0,1,2,3],[0.3...|        0.0|(4,[0,1,2,3],[0.3...|[37.0,0.0,1.0]|[0.97368421052631...|       0.0|
```

F)

The four performance metrics used for evaluating classifiers are precision, accuracy, f1 score and recall.

Precision: It is used to measure the effectiveness of the model which measures the proportion of true positives among the instances that the model predicted as positive. The metric name to calculate the precision using MultiClassificationEvaluator is "precisionByLabel". High precision indicates that the model is giving low proportion of false values.

Formula for calculating precision:
precision = true positives / (true positives + false positives)

Accuracy: It is the measure of ratio of the total number of forecasts to the number of right predictions. The metric name to calculate the precision using MultiClassificationEvaluator is "accuracy". High accuracy indicates that the model is giving more accurate pridictions.

Formula for calculating accuracy:

$$\text{accuracy} = \frac{(\text{true positives} + \text{true negatives})}{(\text{true positives} + \text{false positives} + \text{true negatives} + \text{false negatives})}$$

F1 Score: It is a harmonic mean of accuracy and recall, two additional metrics typically used to evaluate classification algorithms. A high F1 score indicates that the classification model is achieving a good balance between precision and recall.

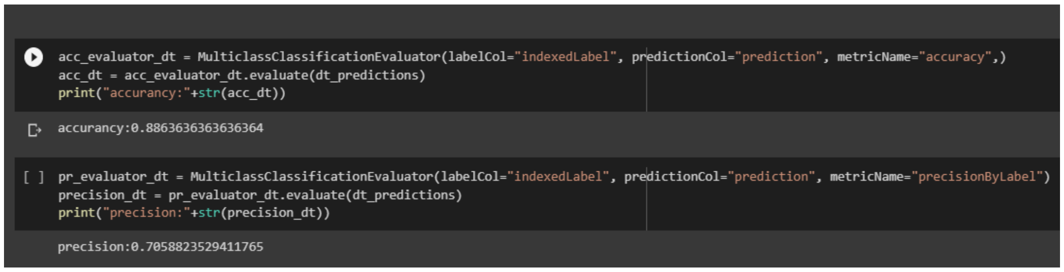
Formula for calculating F1 Score: F1 score = 2 * (precision * recall) / (precision + recall)

Recall: Recall measures how frequently the model correctly recognizes the positive class when it appears in the data. A high recall value shows that the model is effective at detecting all positive cases. A high recall indicates that the model is capturing a high proportion of actual positive instances in the data.

Formula for calculating recall: recall = true positives / (true positives + false negatives)

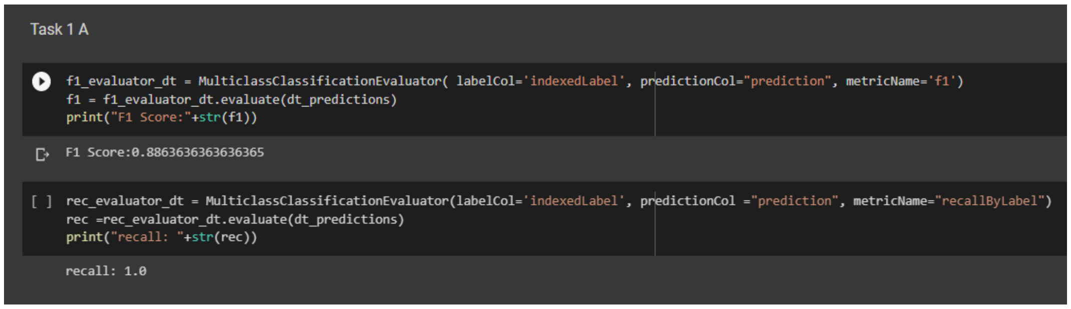
A)

Accuracy and precision of the decision tree is calculated using Multi-class-Classification-Evaluator. The accuracy of the model is 88.63% and the precision of the model is 70.58%.



```
acc_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
acc_dt = acc_evaluator_dt.evaluate(dt_predictions)
print("accurancy:"+str(acc_dt))

accurancy:0.8863636363636364

pr_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
precision_dt = pr_evaluator_dt.evaluate(dt_predictions)
print("precision:"+str(precision_dt))

precision:0.7058823529411765
```

The coding part has been completed by calculating the recall and f1 score for the decision tree model. The F1 score for the given dataset with the decision tree model is 0.8863 and the recall value is 1.0.

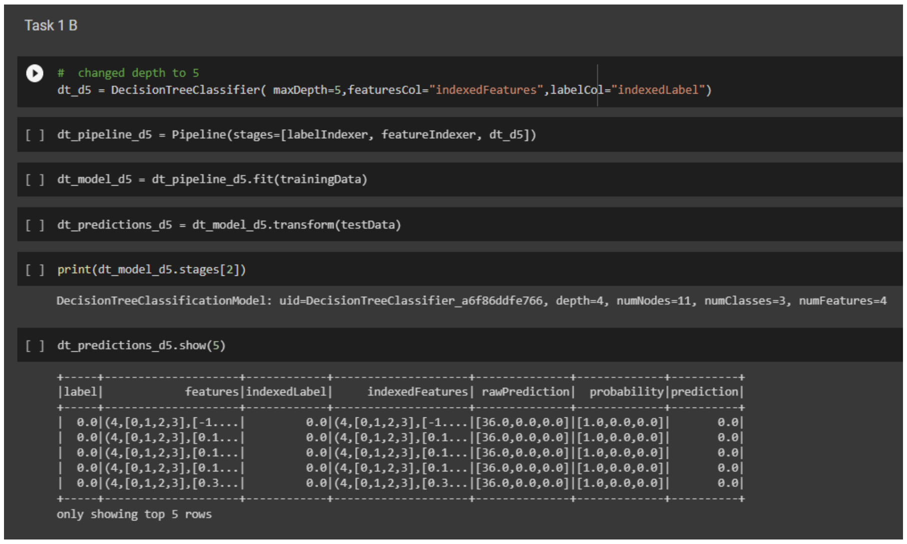


Task 1 A

```
f1_evaluator_dt = MulticlassClassificationEvaluator( labelCol='indexedLabel', predictionCol="prediction", metricName='f1')
f1 = f1_evaluator_dt.evaluate(dt_predictions)
print("F1 Score:"+str(f1))

F1 Score:0.8863636363636365

rec_evaluator_dt = MulticlassClassificationEvaluator(labelCol='indexedLabel', predictionCol ="prediction", metricName="recallByLabel")
rec =rec_evaluator_dt.evaluate(dt_predictions)
print("recall: "+str(rec))

recall: 1.0
```

B)

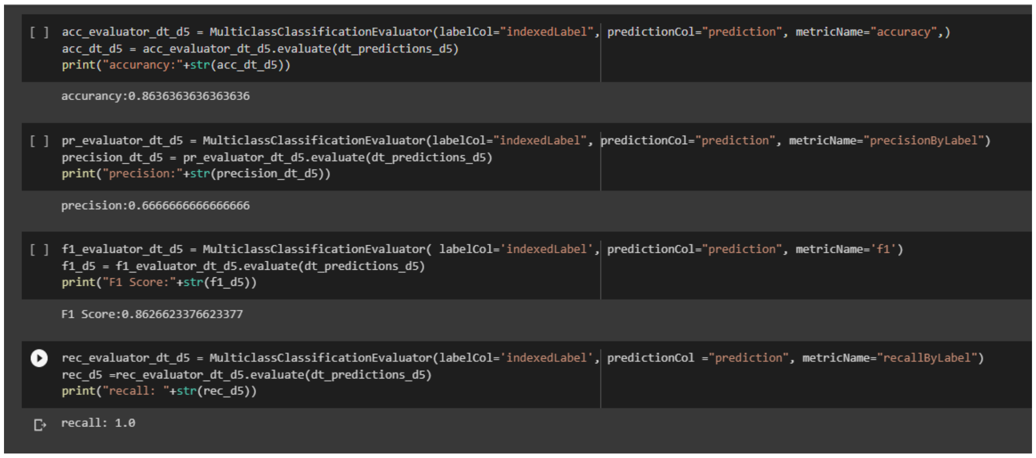
The Depth of the tree is changed to 5 for the above model and the predictions are made for the same training data and testing data. The predictions are displayed in the below picture.



Task 1 B

```
#  changed depth to 5
dt_d5 = DecisionTreeClassifier( maxDepth=5,featuresCol="indexedFeatures",labelCol="indexedLabel")

dt_pipeline_d5 = Pipeline(stages=[labelIndexer, featureIndexer, dt_d5])

dt_model_d5 = dt_pipeline_d5.fit(trainingData)

dt_predictions_d5 = dt_model_d5.transform(testData)

print(dt_model_d5.stages[2])

DecisionTreeClassificationModel: uid=DecisionTreeClassifier_a6f86ddfe766, depth=4, numNodes=11, numClasses=3, numFeatures=4

dt_predictions_d5.show(5)

+-----+------------------+------------+------------------+-------------+-------------+----------+
|label|          features|indexedLabel|   indexedFeatures| rawPrediction|  probability|prediction|
+-----+------------------+------------+------------------+-------------+-------------+----------+
|  0.0|(4,[0,1,2,3],[-1....|         0.0|(4,[0,1,2,3],[-1....|[36.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[36.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[36.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[36.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[0.3...|         0.0|(4,[0,1,2,3],[0.3...|[36.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
+-----+------------------+------------+------------------+-------------+-------------+----------+
only showing top 5 rows
```

The accuracy, precision, f1 score and recall values are calculated for the model build with depth 5. The accuracy of the model is 86.36%, precision of the model is 66.66%, f1 score is 0.86266 and recall value is 1.0.



```
acc_evaluator_dt_d5 = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
acc_dt_d5 = acc_evaluator_dt_d5.evaluate(dt_predictions_d5)
print("accurancy:"+str(acc_dt_d5))
```

```
accurancy:0.8636363636363636
```

```
pr_evaluator_dt_d5 = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
precision_dt_d5 = pr_evaluator_dt_d5.evaluate(dt_predictions_d5)
print("precision:"+str(precision_dt_d5))
```

```
precision:0.6666666666666666
```

```
f1_evaluator_dt_d5 = MulticlassClassificationEvaluator( labelCol='indexedLabel', predictionCol="prediction", metricName='f1')
f1_d5 = f1_evaluator_dt_d5.evaluate(dt_predictions_d5)
print("F1 Score:"+str(f1_d5))
```

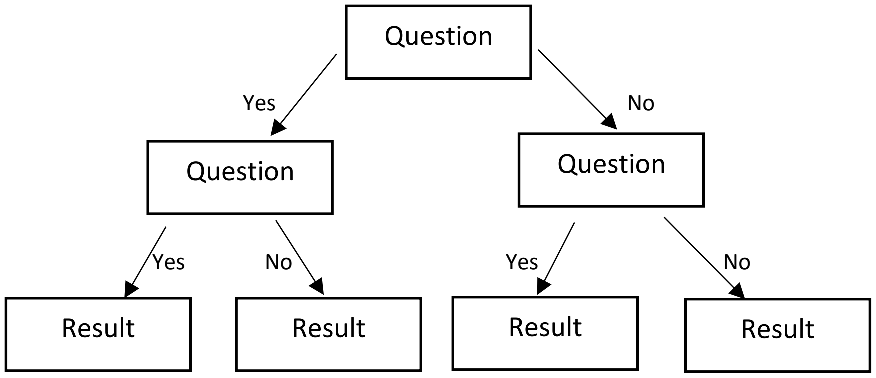
```
F1 Score:0.8626623376623377
```

```
rec_evaluator_dt_d5 = MulticlassClassificationEvaluator(labelCol='indexedLabel', predictionCol ="prediction", metricName="recallByLabel")
rec_d5 =rec_evaluator_dt_d5.evaluate(dt_predictions_d5)
print("recall: "+str(rec_d5))
```

```
recall: 1.0
```

C)

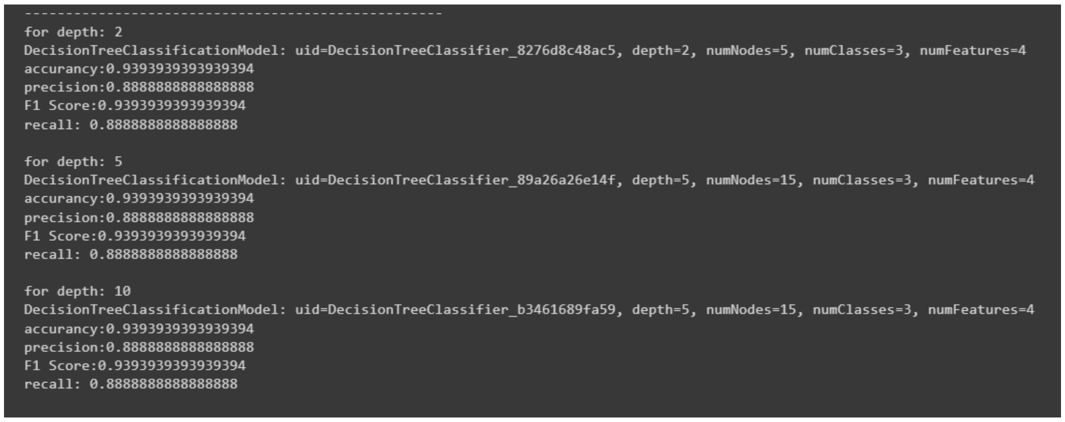
A decision tree is like a flowchart in that it assists a computer in making a choice by posing a series of questions about the data. Each question is based on a data attribute, such as color or size. The computer takes a different path through the tree based on the answers to each question until it reaches a final judgement or prediction. The tree is constructed by picking the best features to partition the data according to specific criteria, and the process is repeated until a stopping requirement is met. The resultant tree is easily seen and interpretable, assisting in understanding the links between the attributes and the target variable. Nevertheless, decision trees are susceptible to overfitting, hence it is critical to employ appropriate approaches.



D) The same model is built and tested with various ratios of training and testing data with various depths. The results of the tests are shown below.

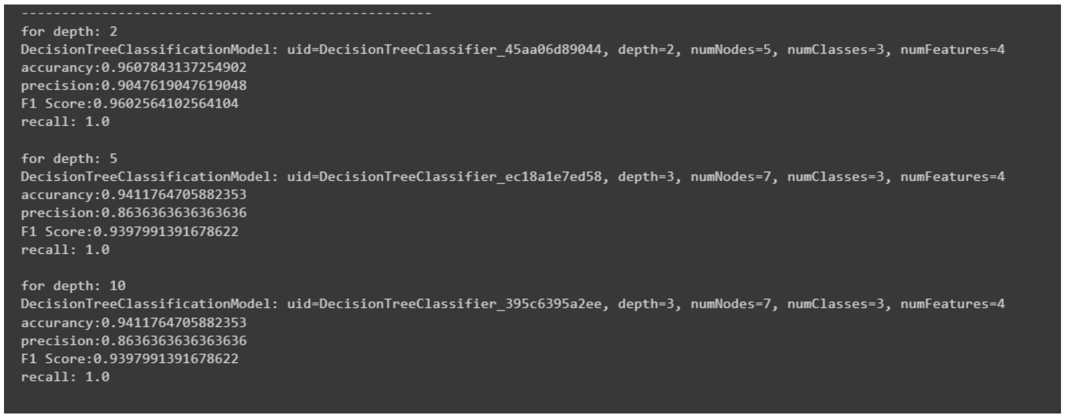
Train Test Split	Depth of Tree	Findings	Why this happened
Train: 60%, Test: 40%	2	accurancy:0.9857142857142858 precision:1.0 F1 Score:0.9856858422811634 recall: 0.9523809523809523	As the decision tree's depth rises, the model becomes more sophisticated and begins to overfit the training data, resulting in poor performance on test data. Nevertheless, if the tree's depth is too shallow, the model may not capture all of the key characteristics in the data, resulting in poor performance on both training and test data.
Train: 60%, Test: 40%	5	accurancy:0.9857142857142858 precision:0.9545454545454546 F1 Score:0.9857284229872058 recall: 1.0	
Train: 60%, Test: 40%	10	accurancy:0.9857142857142858 precision:0.9545454545454546 F1 Score:0.9857284229872058 recall: 1.0	
Train: 70%, Test: 30%	2	accurancy:0.9607843137254902 precision:0.9047619047619048 F1 Score:0.9602564102564104 recall: 1.0	The accuracy of the decision tree reduces somewhat from 2 to 5, but remains constant at 10. Precision, on the other hand, declines with depth, from 0.904 at depth 2 to 0.864 at depths 5 and 10. This means that as the decision tree becomes deeper, it makes more false positive predictions.
Train: 70%, Test: 30%	5	accurancy:0.9411764705882353 precision:0.8636363636363636 F1 Score:0.9397991391678622 recall: 1.0	
Train: 70%, Test: 30%	10	accurancy:0.9411764705882353 precision:0.8636363636363636 F1 Score:0.9397991391678622 recall: 1.0	
Train: 80%, Test: 20%	2	accurancy:0.9393939393939394 precision:0.8888888888888888 F1 Score:0.9393939393939394 recall: 0.8888888888888888	Based on the results, it appears that altering the value of max depth has no effect on the performance measures (accuracy, precision, recall, and F1 score) when the training and test set proportions are 80% and 20%, respectively. All three studies (with max depth values of 2, 5, and 10) produced comparable performance metrics, demonstrating that the choice of max depth had no effect on the model's performance in this case.
Train: 80%, Test: 20%	5	accurancy:0.9393939393939394 precision:0.8888888888888888 F1 Score:0.9393939393939394 recall: 0.8888888888888888	
Train: 80%, Test: 20%	10	accurancy:0.9393939393939394 precision:0.8888888888888888 F1 Score:0.9393939393939394 recall: 0.8888888888888888	

Results for training data 60% and test data 40%



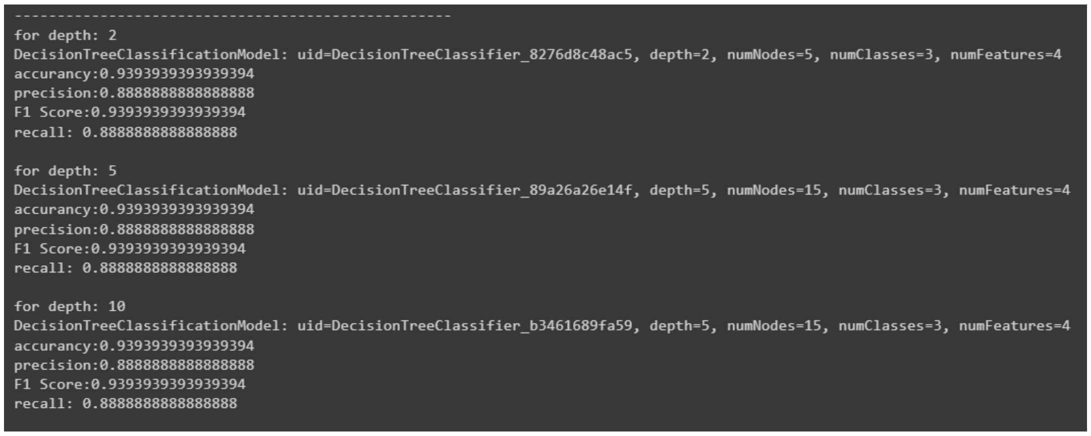
```
--------------------------------------------------
for depth: 2
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_8276d8c48ac5, depth=2, numNodes=5, numClasses=3, numFeatures=4
accurancy:0.9393939393939394
precision:0.8888888888888888
F1 Score:0.9393939393939394
recall: 0.8888888888888888

for depth: 5
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_89a26a26e14f, depth=5, numNodes=15, numClasses=3, numFeatures=4
accurancy:0.9393939393939394
precision:0.8888888888888888
F1 Score:0.9393939393939394
recall: 0.8888888888888888

for depth: 10
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_b3461689fa59, depth=5, numNodes=15, numClasses=3, numFeatures=4
accurancy:0.9393939393939394
precision:0.8888888888888888
F1 Score:0.9393939393939394
recall: 0.8888888888888888
```

Results for training data 70% and test data 30%



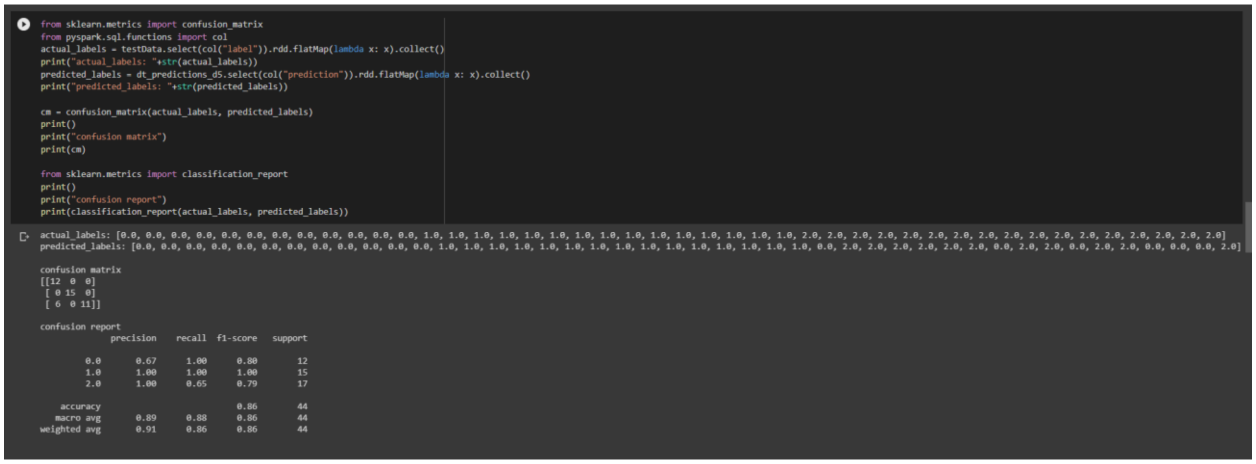
```
--------------------------------------------------
for depth: 2
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_45aa06d89044, depth=2, numNodes=5, numClasses=3, numFeatures=4
accurancy:0.9607843137254902
precision:0.9047619047619048
F1 Score:0.9602564102564104
recall: 1.0

for depth: 5
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_ec18a1e7ed58, depth=3, numNodes=7, numClasses=3, numFeatures=4
accurancy:0.9411764705882353
precision:0.8636363636363636
F1 Score:0.9397991391678622
recall: 1.0

for depth: 10
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_395c6395a2ee, depth=3, numNodes=7, numClasses=3, numFeatures=4
accurancy:0.9411764705882353
precision:0.8636363636363636
F1 Score:0.9397991391678622
recall: 1.0
```

Results for training data 80% and test data 20%



```
--------------------------------------------------
for depth: 2
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_8276d8c48ac5, depth=2, numNodes=5, numClasses=3, numFeatures=4
accurancy:0.9393939393939394
precision:0.8888888888888888
F1 Score:0.9393939393939394
recall: 0.8888888888888888

for depth: 5
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_89a26a26e14f, depth=5, numNodes=15, numClasses=3, numFeatures=4
accurancy:0.9393939393939394
precision:0.8888888888888888
F1 Score:0.9393939393939394
recall: 0.8888888888888888

for depth: 10
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_b3461689fa59, depth=5, numNodes=15, numClasses=3, numFeatures=4
accurancy:0.9393939393939394
precision:0.8888888888888888
F1 Score:0.9393939393939394
recall: 0.8888888888888888
```

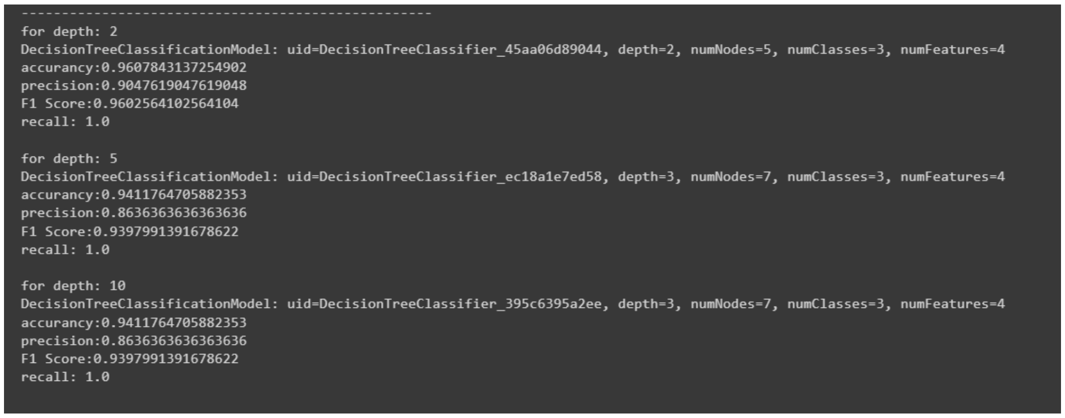
E) Confusion matrix has been created for test data 30% and train data 30%.



```
from sklearn.metrics import confusion_matrix
from pyspark.sql.functions import col
actual_labels = testData.select(col("label")).rdd.flatMap(lambda x: x).collect()
print("actual_labels: "+str(actual_labels))
predicted_labels = dt_predictions_d5.select(col("prediction")).rdd.flatMap(lambda x: x).collect()
print("predicted_labels: "+str(predicted_labels))

cm = confusion_matrix(actual_labels, predicted_labels)
print()
print("confusion matrix")
print(cm)

from sklearn.metrics import classification_report
print()
print("confusion report")
print(classification_report(actual_labels, predicted_labels))
```

```
actual_labels: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0]
predicted_labels: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 2.0, 2.0, 2.0, 2.0, 2.0, 2.0, 0.0, 2.0, 2.0, 0.0, 2.0, 2.0, 0.0, 0.0, 0.0, 2.0]

confusion matrix
[[12  0  0]
 [ 0 15  0]
 [ 6  0 11]]

confusion report
              precision    recall  f1-score   support

         0.0       0.67      1.00      0.80        12
         1.0       1.00      1.00      1.00        15
         2.0       1.00      0.65      0.79        17

    accuracy                           0.86        44
   macro avg       0.89      0.88      0.86        44
weighted avg       0.91      0.86      0.86        44
```

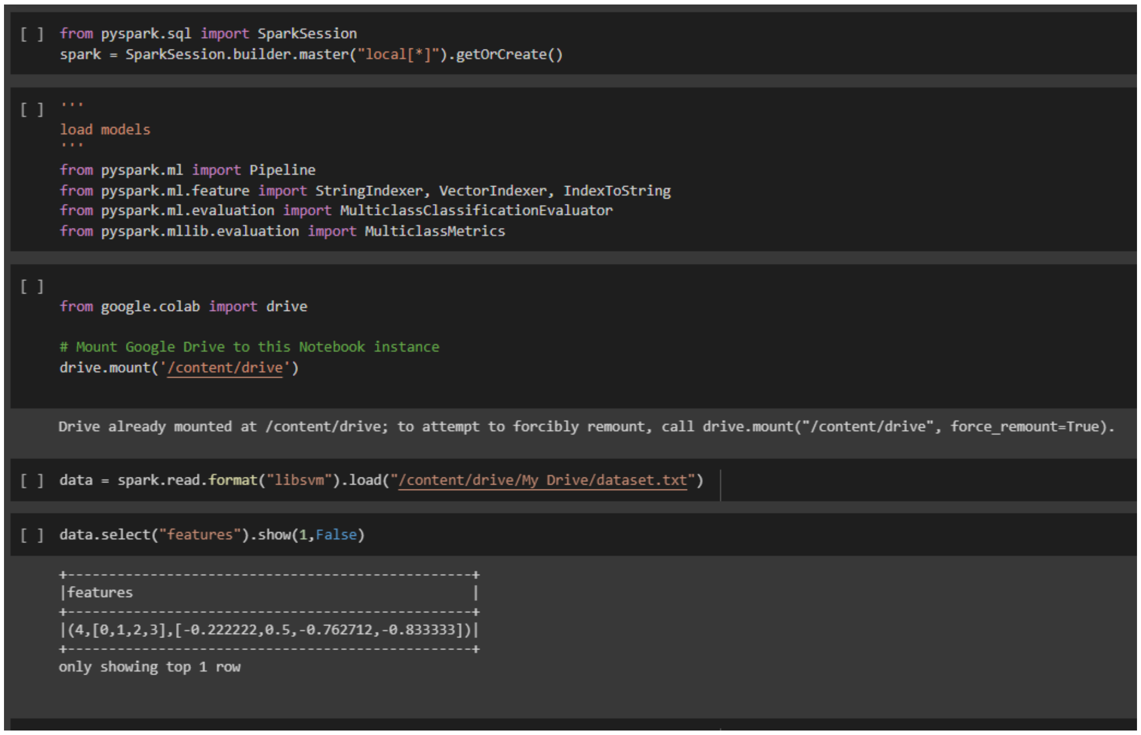
Accuracy, precision, recall and f1 scores for the test data 30% and train data 70% is show in the below figure.



```
---------------------------------------------------
for depth: 2
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_45aa06d89044, depth=2, numNodes=5, numClasses=3, numFeatures=4
accurancy:0.9607843137254902
precision:0.9047619047619048
F1 Score:0.9602564102564104
recall: 1.0

for depth: 5
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_ec18a1e7ed58, depth=3, numNodes=7, numClasses=3, numFeatures=4
accurancy:0.9411764705882353
precision:0.8636363636363636
F1 Score:0.9397991391678622
recall: 1.0

for depth: 10
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_395c6395a2ee, depth=3, numNodes=7, numClasses=3, numFeatures=4
accurancy:0.9411764705882353
precision:0.8636363636363636
F1 Score:0.9397991391678622
recall: 1.0
```

Task 2

Installed pyspark, loaded the necessary libraries and mounted the google drive to fetch the dataset. Loaded dataset and viewed a sample column using data.select.show.



```
[ ] from pyspark.sql import SparkSession
    spark = SparkSession.builder.master("local[*]").getOrCreate()
```

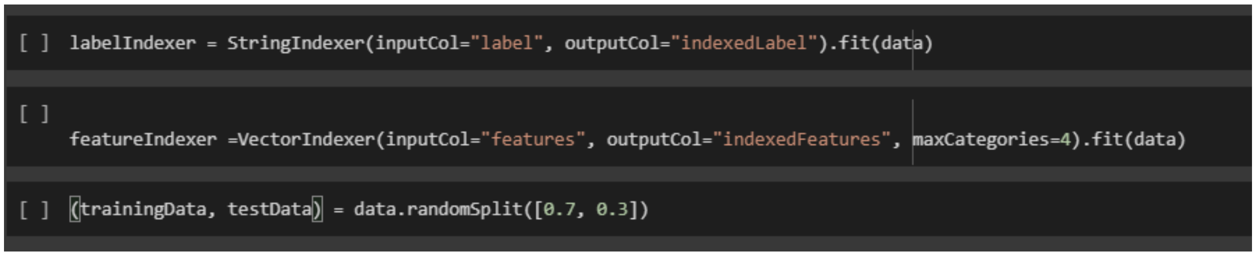
```
[ ] '''
    load models
    '''
    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, VectorIndexer, IndexToString
    from pyspark.ml.evaluation import MulticlassClassificationEvaluator
    from pyspark.mllib.evaluation import MulticlassMetrics
```

```
[ ]
    from google.colab import drive

    # Mount Google Drive to this Notebook instance
    drive.mount('/content/drive')

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
[ ] data = spark.read.format("libsvm").load("/content/drive/My Drive/dataset.txt")
```

```
[ ] data.select("features").show(1,False)

    +-----------------------------------------------+
    |features                                       |
    +-----------------------------------------------+
    |(4,[0,1,2,3],[-0.222222,0.5,-0.762712,-0.833333])|
    +-----------------------------------------------+
    only showing top 1 row
```

Indexed the label column of the dataset as the indexedlabel in the output. The features column of the data is indexed to the indexedFeatures in the output. The data is split into 70% and 30%. 70% is used for training purposes and 30% is used for testing purposes.



```
[ ] labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)
```

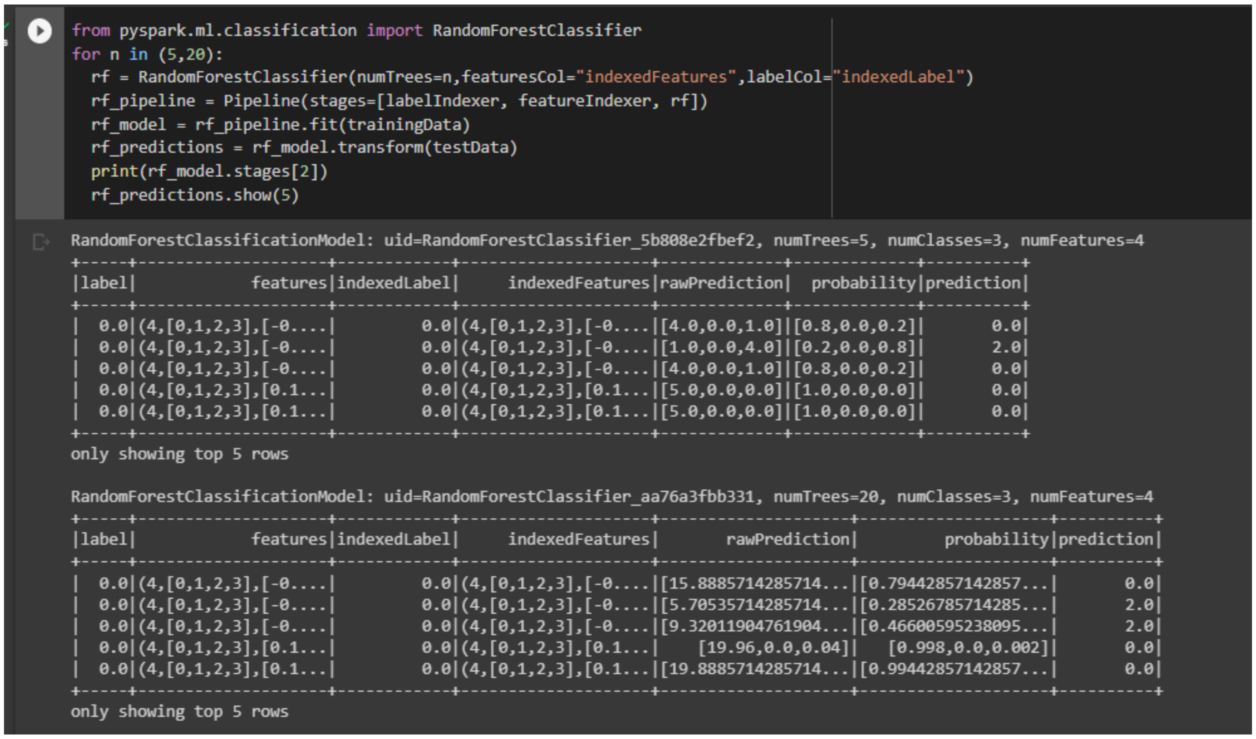
```
[ ]
    featureIndexer =VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
```

```
[ ] (trainingData, testData) = data.randomSplit([0.7, 0.3])
```

A & B)

Built the model with the training data and the predictions are obtained and for the tree values 5 and 20.

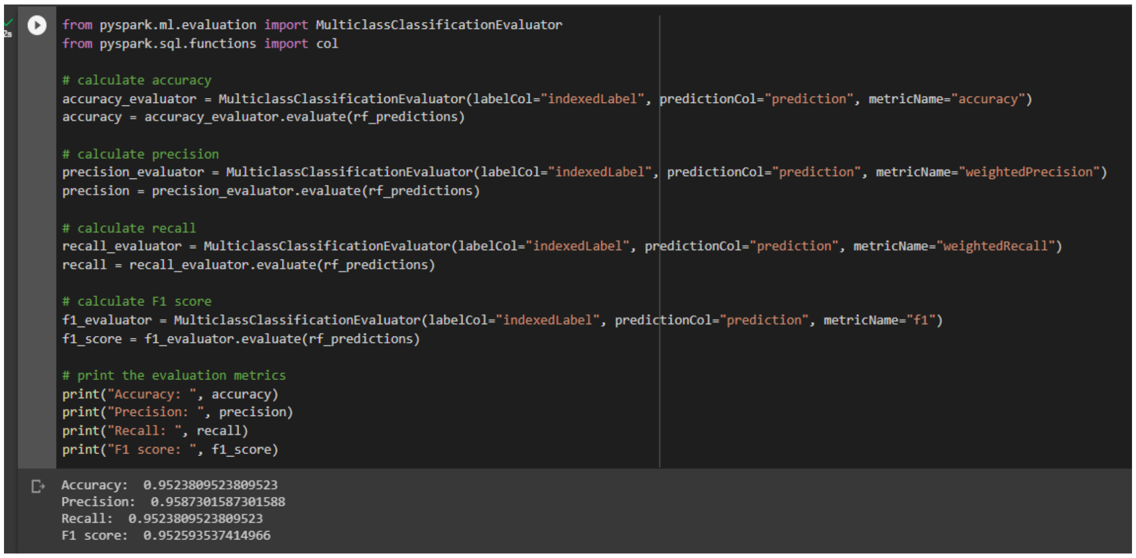
Increase in number of trees will not always improve the performance. After some critical value the increase in number of trees will reduce accordingly based on the data and the training and testing data ratios.



```
from pyspark.ml.classification import RandomForestClassifier
for n in (5,20):
  rf = RandomForestClassifier(numTrees=n,featuresCol="indexedFeatures",labelCol="indexedLabel")
  rf_pipeline = Pipeline(stages=[labelIndexer, featureIndexer, rf])
  rf_model = rf_pipeline.fit(trainingData)
  rf_predictions = rf_model.transform(testData)
  print(rf_model.stages[2])
  rf_predictions.show(5)
```

```
RandomForestClassificationModel: uid=RandomForestClassifier_5b808e2fbef2, numTrees=5, numClasses=3, numFeatures=4
+-----+--------------------+------------+--------------------+-------------+-------------+----------+
|label|            features|indexedLabel|     indexedFeatures|rawPrediction|  probability|prediction|
+-----+--------------------+------------+--------------------+-------------+-------------+----------+
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[4.0,0.0,1.0]|[0.8,0.0,0.2]|       0.0|
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[1.0,0.0,4.0]|[0.2,0.0,0.8]|       2.0|
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[4.0,0.0,1.0]|[0.8,0.0,0.2]|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[5.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[5.0,0.0,0.0]|[1.0,0.0,0.0]|       0.0|
+-----+--------------------+------------+--------------------+-------------+-------------+----------+
only showing top 5 rows

RandomForestClassificationModel: uid=RandomForestClassifier_aa76a3fbb331, numTrees=20, numClasses=3, numFeatures=4
+-----+--------------------+------------+--------------------+--------------------+--------------------+----------+
|label|            features|indexedLabel|     indexedFeatures|       rawPrediction|         probability|prediction|
+-----+--------------------+------------+--------------------+--------------------+--------------------+----------+
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[15.8885714285714...|[0.79442857142857...|       0.0|
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[5.70535714285714...|[0.28526785714285...|       2.0|
|  0.0|(4,[0,1,2,3],[-0....|         0.0|(4,[0,1,2,3],[-0....|[9.32011904761904...|[0.46600595238095...|       2.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|    [19.96,0.0,0.04]|   [0.998,0.0,0.002]|       0.0|
|  0.0|(4,[0,1,2,3],[0.1...|         0.0|(4,[0,1,2,3],[0.1...|[19.8885714285714...|[0.99442857142857...|       0.0|
+-----+--------------------+------------+--------------------+--------------------+--------------------+----------+
only showing top 5 rows
```

The evaluation methods for 20 trees are calculated below. The accuracy is 95.23%, precision is 0.9587, recall is 0.9523 and f1 score is 0.9525. The values are computed using MulticlassClassificationEvaluator.



```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.sql.functions import col

# calculate accuracy
accuracy_evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = accuracy_evaluator.evaluate(rf_predictions)

# calculate precision
precision_evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="weightedPrecision")
precision = precision_evaluator.evaluate(rf_predictions)

# calculate recall
recall_evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="weightedRecall")
recall = recall_evaluator.evaluate(rf_predictions)

# calculate F1 score
f1_evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
f1_score = f1_evaluator.evaluate(rf_predictions)

# print the evaluation metrics
print("Accuracy: ", accuracy)
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 score: ", f1_score)
```

```
Accuracy:  0.9523809523809523
Precision:  0.9587301587301588
Recall:  0.9523809523809523
F1 score:  0.952593537414966
```

C)

Random forest is a strategy for ensemble learning that mixes many decision trees to produce a more accurate and stable prediction model. Random forest works by training numerous decision trees on distinct subsets of training data and combining their results to get a efficient result.

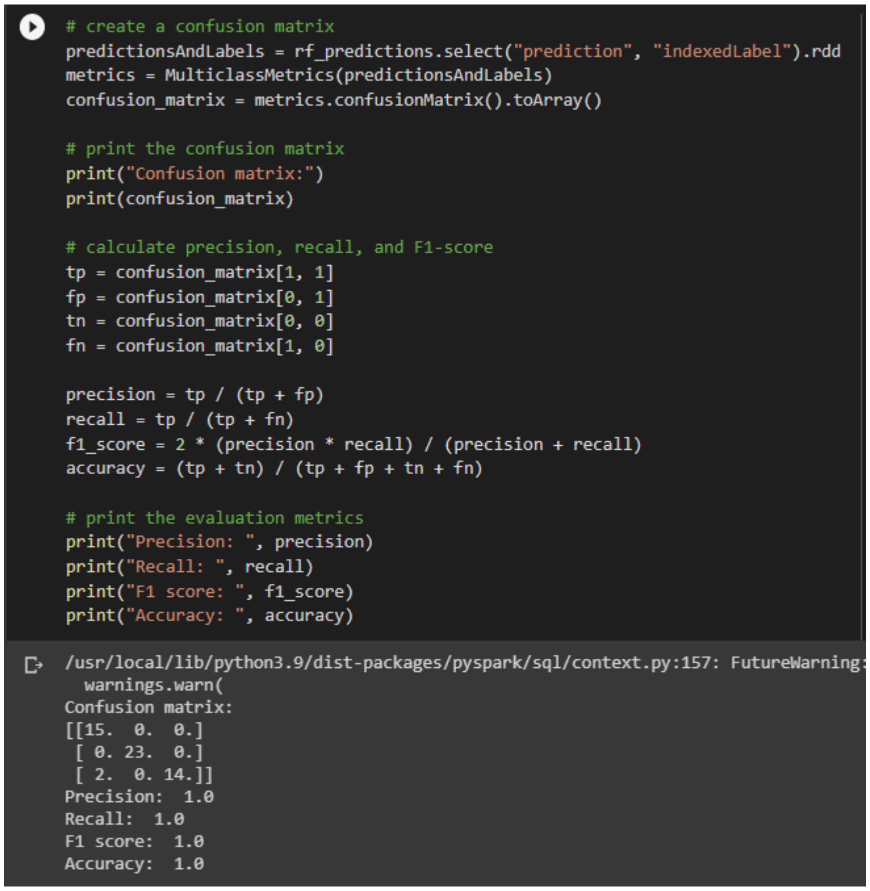
A random forest model choose a chunk of the training data at random and create a decision tree using the specified subset. This process is repeated multiple times by selecting a different subset of data each time. To get a final output, combine the forecasts of all the trees.

D)

Apart from increasing the number of trees, the performance of the random forest model can be increased by using other machine learning algorithms using ensemble methods such as bagging and boosting. The performance can also be improved by selecting only the most relevant data for training the model by removing the outliers from the data. Various other techniques can also be applied to improve the performance of the model. The above mentioned are a few of the techniques that can be used to improve the performance of the model.

E)

The confusion matrix is constructed for the predictions and test data and the precision, accuracy, f1 score and recall values are calculated using code and the result is show in the below figure.

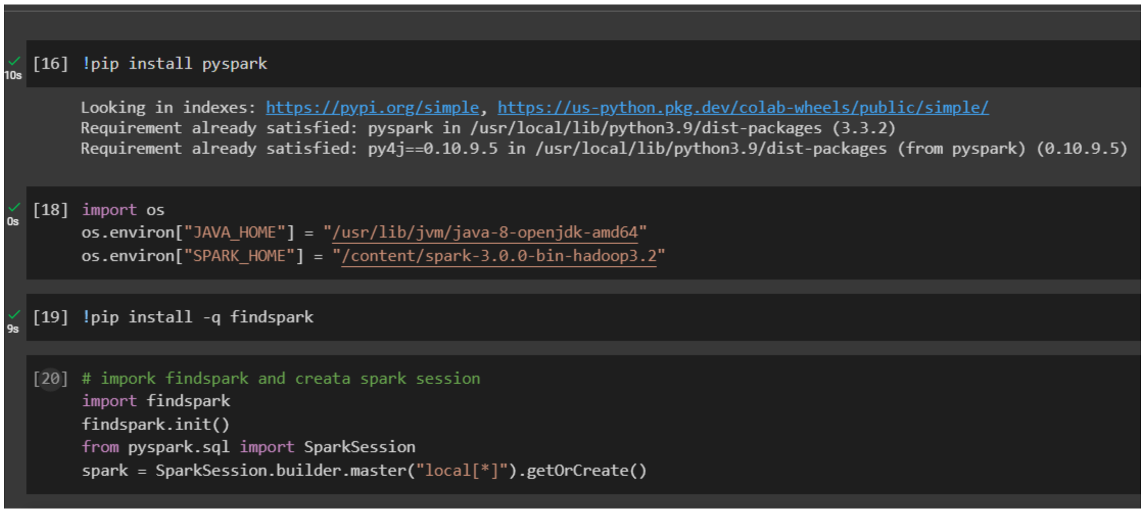


```
# create a confusion matrix
predictionsAndLabels = rf_predictions.select("prediction", "indexedLabel").rdd
metrics = MulticlassMetrics(predictionsAndLabels)
confusion_matrix = metrics.confusionMatrix().toArray()

# print the confusion matrix
print("Confusion matrix:")
print(confusion_matrix)

# calculate precision, recall, and F1-score
tp = confusion_matrix[1, 1]
fp = confusion_matrix[0, 1]
tn = confusion_matrix[0, 0]
fn = confusion_matrix[1, 0]

precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1_score = 2 * (precision * recall) / (precision + recall)
accuracy = (tp + tn) / (tp + fp + tn + fn)

# print the evaluation metrics
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 score: ", f1_score)
print("Accuracy: ", accuracy)
```

```
/usr/local/lib/python3.9/dist-packages/pyspark/sql/context.py:157: FutureWarning:
  warnings.warn(
Confusion matrix:
[[15.  0.  0.]
 [ 0. 23.  0.]
 [ 2.  0. 14.]]
Precision:  1.0
Recall:  1.0
F1 score:  1.0
Accuracy:  1.0
```

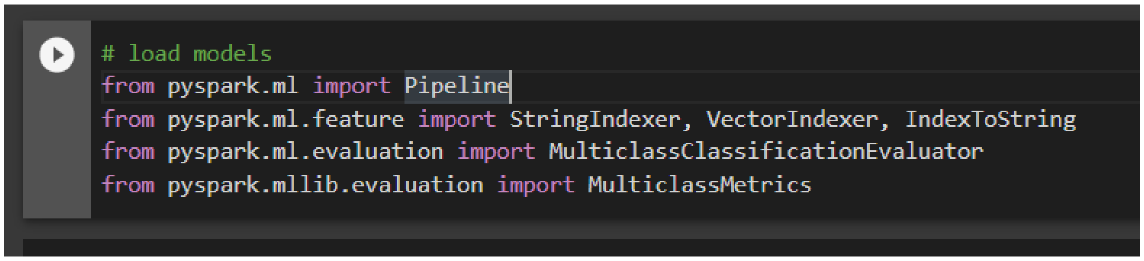
Task 3

A) Installed spark and configured spark folder with the environment variables. Installed findspark and created a sparksession.



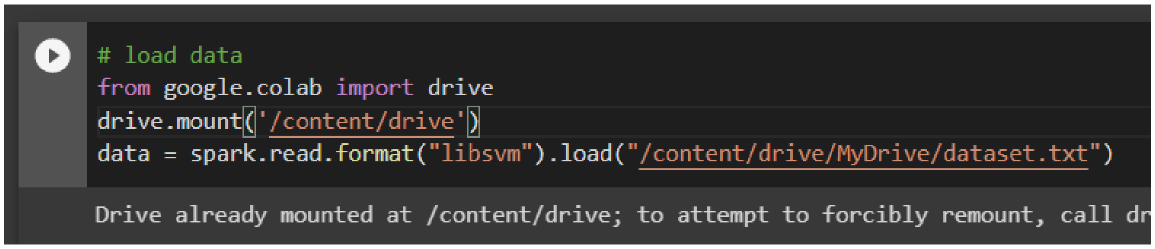
```
[16] !pip install pyspark

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Requirement already satisfied: pyspark in /usr/local/lib/python3.9/dist-packages (3.3.2)
     Requirement already satisfied: py4j==0.10.9.5 in /usr/local/lib/python3.9/dist-packages (from pyspark) (0.10.9.5)

[18] import os
     os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
     os.environ["SPARK_HOME"] = "/content/spark-3.0.0-bin-hadoop3.2"

[19] !pip install -q findspark

[20] # impork findspark and creata spark session
     import findspark
     findspark.init()
     from pyspark.sql import SparkSession
     spark = SparkSession.builder.master("local[*]").getOrCreate()
```

Imported all the models that are required for building and evaluating the model.



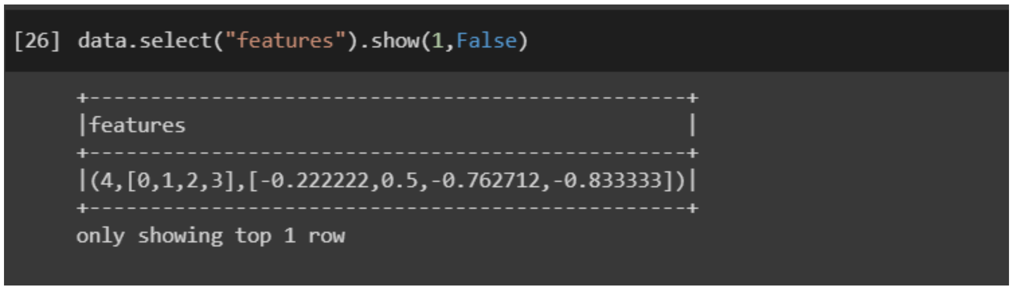
```
# load models
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorIndexer, IndexToString
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.mllib.evaluation import MulticlassMetrics
```

Mounted google drive and loaded the dataset to the file.



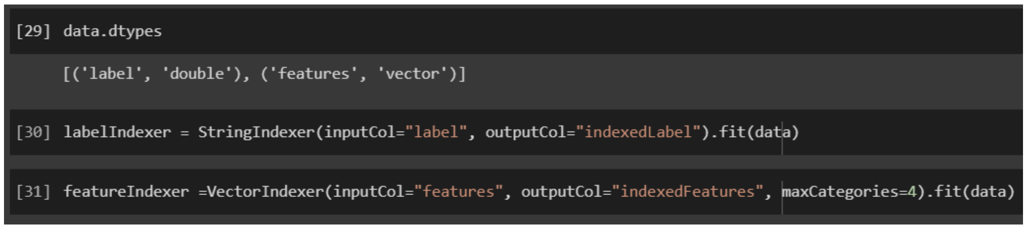
```
# load data
from google.colab import drive
drive.mount('/content/drive')
data = spark.read.format("libsvm").load("/content/drive/MyDrive/dataset.txt")

Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

Viewing the data after loading the data to the variable ‘data’ to verify if the data is loaded correctly or not.



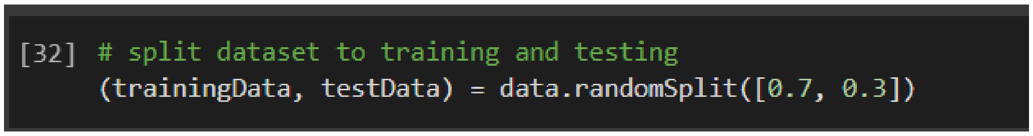
```
[26] data.select("features").show(1,False)

     +----------------------------------------------+
     |features                                      |
     +----------------------------------------------+
     |(4,[0,1,2,3],[-0.222222,0.5,-0.762712,-0.833333])|
     +----------------------------------------------+
     only showing top 1 row
```

Checking the datatypes of the columns in the data and indexing the columns. Input column label is indexed to the column ‘indexedLabel’ in the output. The input column features is indexed to the column ‘indexedFeatures’ in the output.



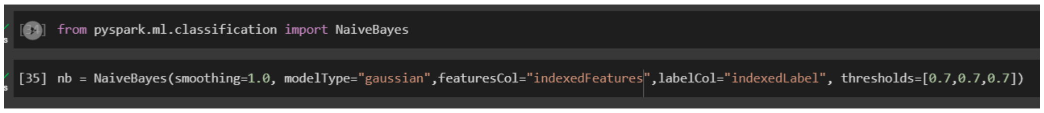
```
[29] data.dtypes

     [('label', 'double'), ('features', 'vector')]

[30] labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel").fit(data)

[31] featureIndexer =VectorIndexer(inputCol="features", outputCol="indexedFeatures", maxCategories=4).fit(data)
```

The data is split for the training and testing in 70% and 30% respectively.



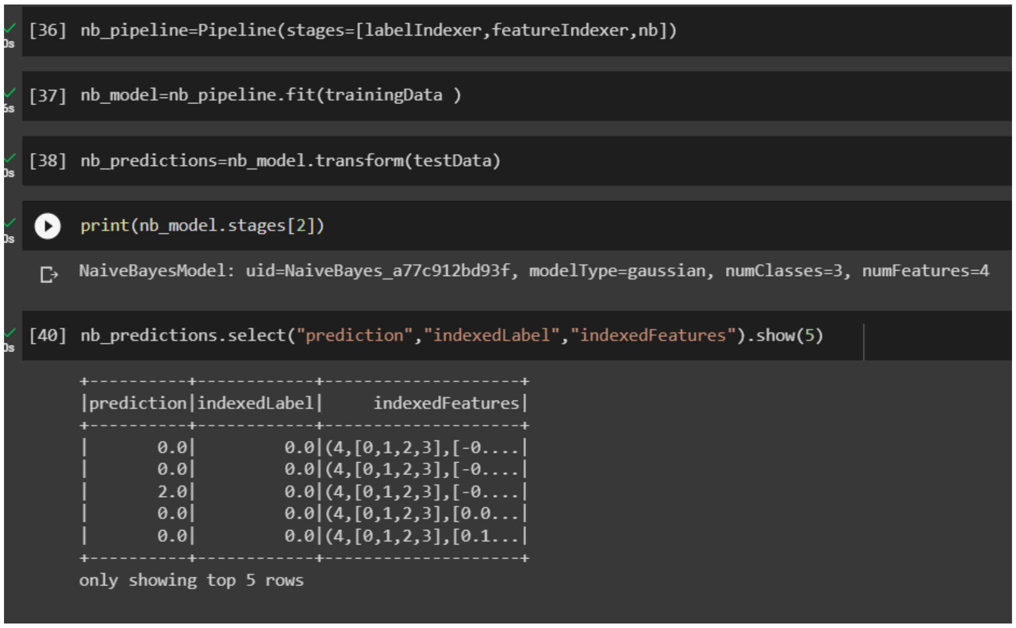
```
[32] # split dataset to training and testing
     (trainingData, testData) = data.randomSplit([0.7, 0.3])
```

Imported NaiveBayes from pyspark and created a model with model type ‘gaussian’ and linked the feature and label columns.



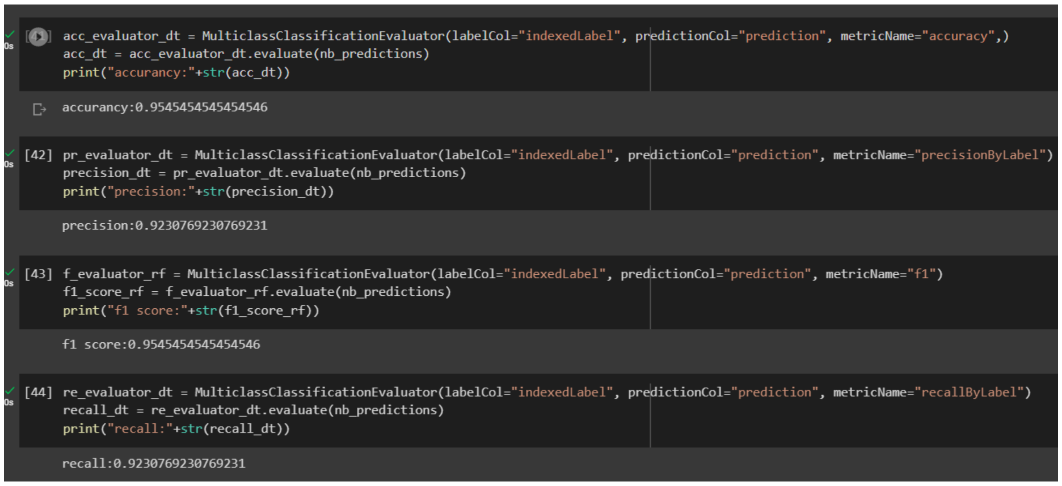
```
[34] from pyspark.ml.classification import NaiveBayes

[35] nb = NaiveBayes(smoothing=1.0, modelType="gaussian",featuresCol="indexedFeatures",labelCol="indexedLabel", thresholds=[0.7,0.7,0.7])
```

Created the pipeline for the model with indexed columns and the model itself. Then the model is fitted with the training data. It is fed with testing data to obtain the predictions. The results of the prediction are shown in the figure below.



```
[36] nb_pipeline=Pipeline(stages=[labelIndexer,featureIndexer,nb])

[37] nb_model=nb_pipeline.fit(trainingData )

[38] nb_predictions=nb_model.transform(testData)

     print(nb_model.stages[2])

     NaiveBayesModel: uid=NaiveBayes_a77c912bd93f, modelType=gaussian, numClasses=3, numFeatures=4

[40] nb_predictions.select("prediction","indexedLabel","indexedFeatures").show(5)

     +----------+------------+-------------------+
     |prediction|indexedLabel|    indexedFeatures|
     +----------+------------+-------------------+
     |       0.0|         0.0|(4,[0,1,2,3],[-0....|
     |       0.0|         0.0|(4,[0,1,2,3],[-0....|
     |       2.0|         0.0|(4,[0,1,2,3],[-0....|
     |       0.0|         0.0|(4,[0,1,2,3],[0.0...|
     |       0.0|         0.0|(4,[0,1,2,3],[0.1...|
     +----------+------------+-------------------+
     only showing top 5 rows
```

The performance metrics of the model are calculated using the MulticlassClassificationEvaluator. The accuracy of the model is obtained as 95.45% and the precision is 0.9230, recall is 0.9230 and the f1 score of the model is 0.9545



```
[41] acc_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy",)
     acc_dt = acc_evaluator_dt.evaluate(nb_predictions)
     print("accurancy:"+str(acc_dt))

     accurancy:0.9545454545454546

[42] pr_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="precisionByLabel")
     precision_dt = pr_evaluator_dt.evaluate(nb_predictions)
     print("precision:"+str(precision_dt))

     precision:0.9230769230769231

[43] f_evaluator_rf = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="f1")
     f1_score_rf = f_evaluator_rf.evaluate(nb_predictions)
     print("f1 score:"+str(f1_score_rf))

     f1 score:0.9545454545454546

[44] re_evaluator_dt = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="recallByLabel")
     recall_dt = re_evaluator_dt.evaluate(nb_predictions)
     print("recall:"+str(recall_dt))

     recall:0.9230769230769231
```

B)

Naive Bayes is a machine learning technique that uses the probability technique of Bayes theorem. The classification algorithm is referred to as “naïve” because the characteristics of the classification are independent of one another. This feature of the model is the reason for efficient computing of the data, and they will play an important role in achieving efficient results which improves the performance of the model.

The Naive Bayes classifier first examines the training data set to find the probability distribution of each class. Whenever there is new data available, the model analyzes the input and assigns a class to it based on the data characteristics. Finally, the classifier allocates the input with the highest probability to the class.

The main reason for Naive Bayes models’ exceptional performance is, it can handle high dimensional data though there are less samples and the model can categorize the new data very quickly and efficiently that is suitable for real time applications.

C)

Apart from the techniques we have used in this ice, The performance of Naive Bayes model can be increased by handling the imbalance data. Removing more irrelevant data from the dataset has the high potential to enhance the performance of the model. Techniques, under-sampling for the majority classes and over-sampling for the minority classes can help to balance the data which helps in increasing the performance of the model.

Bayesian optimization is a method for tuning hyperparameters that entails building a probabilistic model of the goal function (such as the model's accuracy) and using it to efficiently explore the space of potential hyperparameters. This can aid in the discovery of better hyperparameters for Naive Bayes models than typical grid or random search approaches.

E) Supervised Learning:

Fundamentally, supervised learning is when we train the machine using label columns of the data. This means that some data has already been labeled with the right answer. When a machine is given a new set of data, the model will analyze the training data which usually is a portion of the dataset to create a proper result from the data.

Supervised learning is concerned with or learns from "labeled" data. Types of supervised learning are as follows.

Regression: it is a machine learning technique that is used to predict series of values. The aim of the regression is to plot a best curve between the data.

Logistic Regression: It is used to calculate or predict the probability of a binary (yes/no) event occurring.

Naive Bayes Classifiers: Naive Bayes classifiers are based on probabilistic classifiers which are taken from Bayes' theorem. They are used for classification tasks and are particularly useful when working with high-dimensional data.

Decision Trees: They are used for both classification and regression tasks. They involve breaking the data into small parts based on the values of the input and building a tree-like model that predicts the output based on the input features.

Unsupervised learning:

It is a machine learning approach in which models are created without the use of a training dataset. Instead, models discover unidentified patterns in the provided data. It is comparable to the learning that occurs in the person’s brain while learning new things.

Clustering: It is a way of breaking down things into clusters. The highest similarity data is broken down and further grouped based on the similarity of the data.

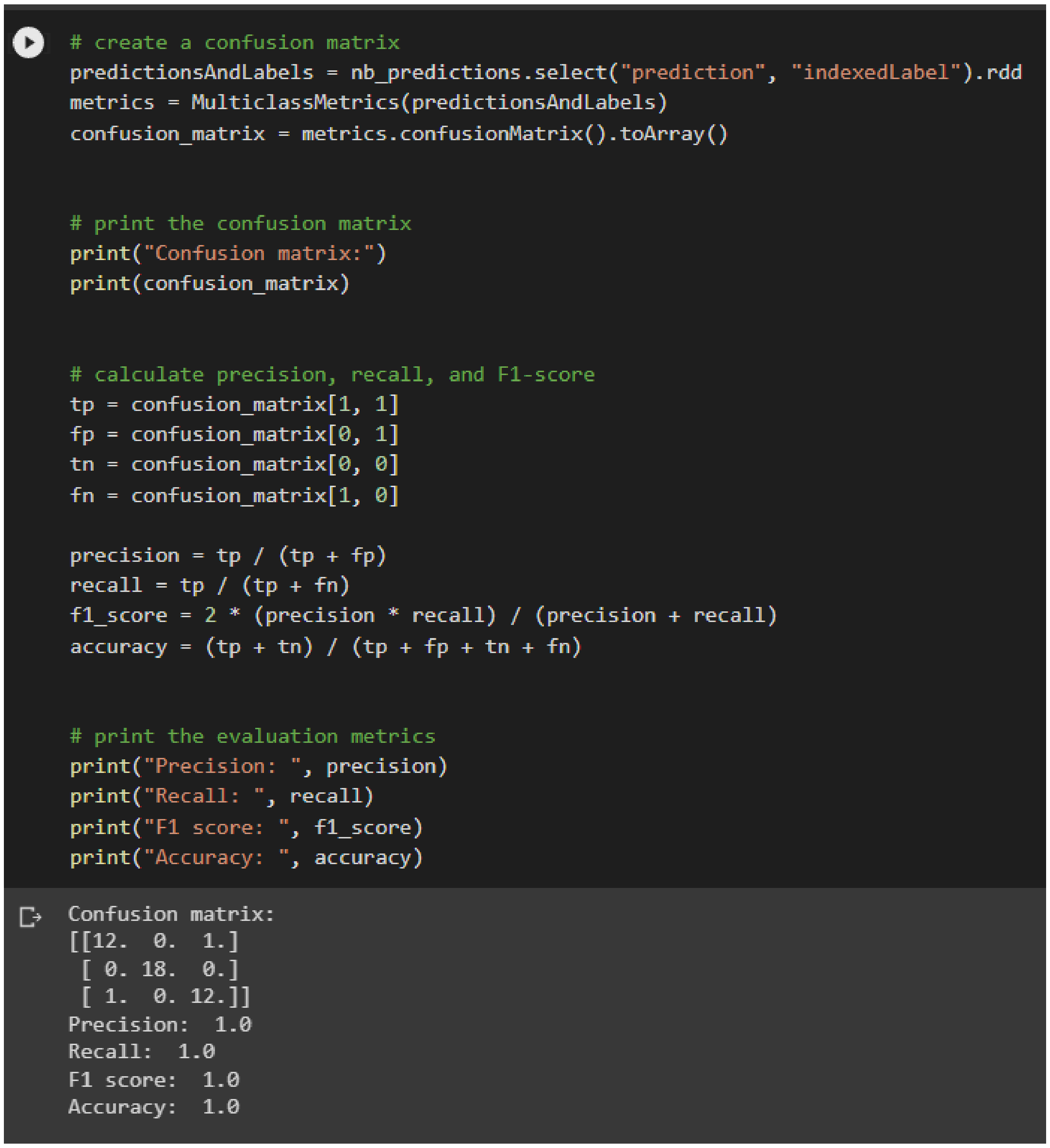
Dimensionality reduction: This involves reducing the number of inputs while preserving important information in data.

Anomaly detection: This is about identifying data points that are different from the rest of the data.

Association rule learning: This is about finding patterns in the data, such as "people who buy bread often also buy milk."

D)

The confusion matrix of the model is calculated, and the performance metrics are calculated again after calculating the confusion matrix. The results of the performance metrics and the confusion matrix are present in the below figure.



```python
# create a confusion matrix
predictionsAndLabels = nb_predictions.select("prediction", "indexedLabel").rdd
metrics = MulticlassMetrics(predictionsAndLabels)
confusion_matrix = metrics.confusionMatrix().toArray()


# print the confusion matrix
print("Confusion matrix:")
print(confusion_matrix)


# calculate precision, recall, and F1-score
tp = confusion_matrix[1, 1]
fp = confusion_matrix[0, 1]
tn = confusion_matrix[0, 0]
fn = confusion_matrix[1, 0]

precision = tp / (tp + fp)
recall = tp / (tp + fn)
f1_score = 2 * (precision * recall) / (precision + recall)
accuracy = (tp + tn) / (tp + fp + tn + fn)


# print the evaluation metrics
print("Precision: ", precision)
print("Recall: ", recall)
print("F1 score: ", f1_score)
print("Accuracy: ", accuracy)
```

```
Confusion matrix:
[[12.  0.  1.]
 [ 0. 18.  0.]
 [ 1.  0. 12.]]
Precision:  1.0
Recall:  1.0
F1 score:  1.0
Accuracy:  1.0
```