AC41011/AC51047: Advanced Big Data Analysis
Big Data Processing Assignment-2

Yashash Thittamaranahalli Venkatesh(2546871)

Problems

First Question

Write a Scala program that reads in a file (it can be either given as an argument to the main method or hardcoded into it) that counts the number of occurrences of each letter a-z in the file and prints these counts on the screen.

```
import scala.io.Source

object LetterCount {

  def alphabetCounts(text: String): Map[Char, Int] = {

    text

      .toLowerCase

      .filter(_.isLetter)

      .groupBy(identity)

      .view

      .mapValues(_.length)

      .toMap

  }

  def totalCount(alphabetCounts: Map[Char, Int])= {

    println("Alphabet counts in a given file:")

        alphabetCounts.toList.sortBy(_._1).foreach {

      case (letter, count) => println(s"$letter:$count")      }

  }

  def main(args: Array[String])= {

    val fileLocation = "inputfile"

    val text = Source.fromFile(fileLocation).mkString

    val count = alphabetCounts(text)

    totalCount(count)

  }
}
```

import.scala.io.source

It is imports source class from 'scala.io' package which helps to read files and other input source.

object LetterCount{}

In scala program first we will create the object and with in this we will define the functions and main program is written inside this.

def alphabetCounts()

defining function 'alphabetCounts' which is taking a input of text file and it is also helping to count the occurrence of each alphabets.

With in this function we will convert the text to lowercase, filtering of non-alphabetic characters, grouping of letters, create a view, maps each group to its length and at last it converts to a map

def totalCount()

I am defining a function total count and inside this it is taking input from alphabetCounts it helps to print each letter with its count

def main()

The main function is defined and within this I am reading the file from its path and storing it to variable fileLocation, converting that file to the string and saving it to variable text, I am performing alphabetCounts function, saving it to count variable and finally calling totalcount on count

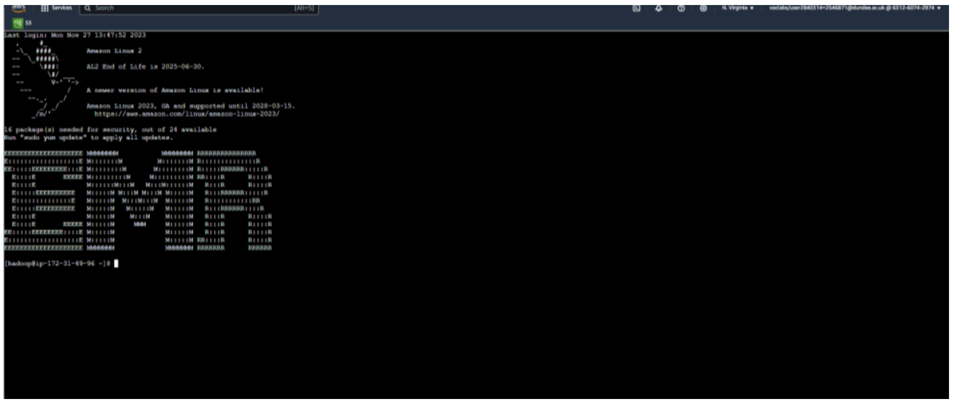
The Output will be in this form

```
Alphabet counts in a given file:

a:121063

b:23269

c:35849

d:67209

e:179574

f:30269

g:33181

h:92257

i:108300

j:2407
```

```
k:13745

l:59996

m:42935

n:106078

o:116949

p:25336

q:1638

r:88620

s:98493

t:131346

u:44438

v:13685

w:35208

x:2041

y:36931

z:351
Process finished with exit code 0
```

To execute Second and third question we need to perform some steps.

1. Creating S3 Bucket and uploading Csv files
2. Giving Security permission for SSHAccess in EC2 and setting it to Allow SSH access to developers
3. Creating EMR cluster by giving name of cluster ,Selecting Spark 3.4.1 and Hadoop 3.3.6 in Application bundle , Selecting m4.large for primary and core ,Amazon EC2 key pair for SSH to the cluster to **Vockey** , **EMR_defaultRole** in service role under Amazon EMR service role, EC2 instance profile for Amazon EMR under this selecting EMR_EC2_DefaultRole after all this we will click on create cluster.
4. We will go to EC2 instance in that we will try to connect if it is primary node than only it will connect or else it wont connect we will change root to Hadoop in this
5. Once EMR cluster is created and we r inside sparkshell we will type **pyspark** to code in python



```
Last login: Mon Nov 27 13:47:52 2023

                     Amazon Linux 2

                     AL2 End of Life is 2025-06-30.

                     A newer version of Amazon Linux is available!

                     Amazon Linux 2023, GA and supported until 2028-03-15.
                       https://aws.amazon.com/linux/amazon-linux-2023/

16 package(s) needed for security, out of 24 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-49-96 ~]$
```

EMR cluster

Second Question

Given is a dataset in the table Bakery.csv that contains transactions in the bakery. Each row in the transaction contains transaction identifier, item bought, date and time, daytime (morning/afternoon) and daytype (weekend, weekday).

Write Scala or Python code for Spark that will derive frequent itemsets of bakery items that are frequently bought together and association rules from this data set.

```python
from pyspark.ml.fpm import FPGrowth

dataFrame = spark.read.csv("s3://yashashass2-spark-bucket/Bakery.csv", header=True).rdd

keyValuePairs = dataFrame.map (lambda row : (row['TransactionNo'], {row['Items']}))

collectedKeyValuePairs = keyValuePairs.reduceByKey( lambda a,b : a.union(b))

itemsToList=collectedKeyValuePairs.map(lambda x : (x[0], list(x[1])))

transactionsFrame=spark.createDataFrame(itemsToList, ["id","items"])

model = FPGrowth(itemsCol="items", minSupport=0.025, minConfidence=0.25)

modelResults = model.fit(transactionsFrame)

modelResults.associationRules.show()

modelResults.freqItemsets.show()
```

importing FPGrowth from pyspark MLLib Library.

Reading CSV file named bakery.csv from S3 bucket into a spark dataframe named dataFrame. The .rdd at end converts dataFrame to Resilient Distributed Dataset.

Creating keyvaluepairs where the key is TransactionNo and the value is Items this helps in mapping each row of the data frame to a tuple like (TransactionNo,{Items}).

Reducing the keyvaluepair by merging the seta using union operation and also helps to create a unique set of items.

Converting to list with the format (TransactionNo,[Items])

Creating a new DataFrame from key-value pairs with columns **id** and **items.**

Creating an FP-growth model which is using a set of items with specified parameters itemsCol with minSupport and minConfidence

minSupport

Minimum support is the minimum proportion of transactions in which particular set of items which will appear. It will also helps to filter out infrequent itemsets and if it set too low, the algoriuthms finds a large number of itemsets, including a rare occurrence

minConfidence

In simple words I can say that it helps to find the consequent items out of antecedent items.

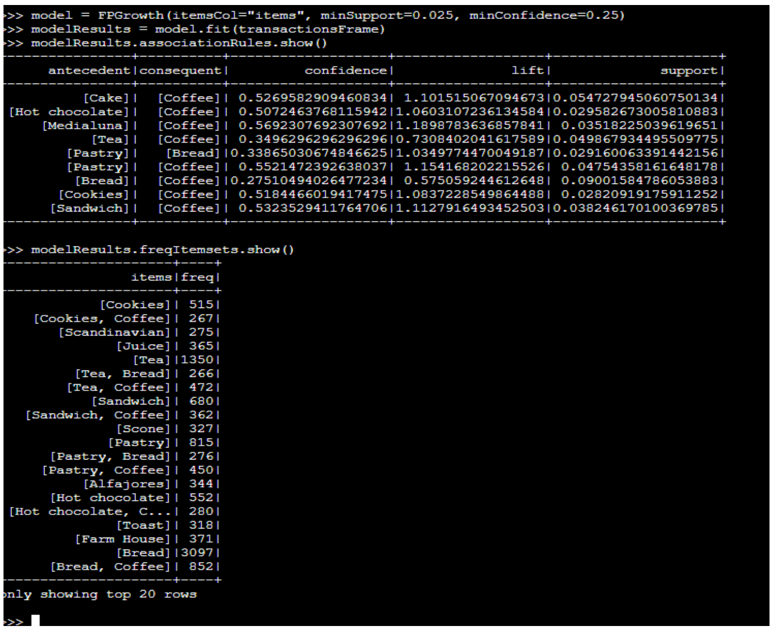
I can filter out items based on the value which I will give to minConfidence, by setting minimum I can filter out rules that do not meet a certain level of reliability , by setting higher confidence ensures that the discovered rules are more likely to be true and useful.

I have consider 4 different conditions to perform minSupport and minConfidence

1. **minSupport = Low and minConfidence= Low**

   model = FPGrowth(itemsCol="items", minSupport=0.025, minConfidence=0.25)

   In this case min support is set to 2.5% that means only itemset that appears in at least 2.5% of the transaction will be consider frequent. And min Confidence is set to 25% that means the rule is considered significant if it has at least 25% confidence which is suggesting a relatively weaker requirement for the rule to be consider valid.



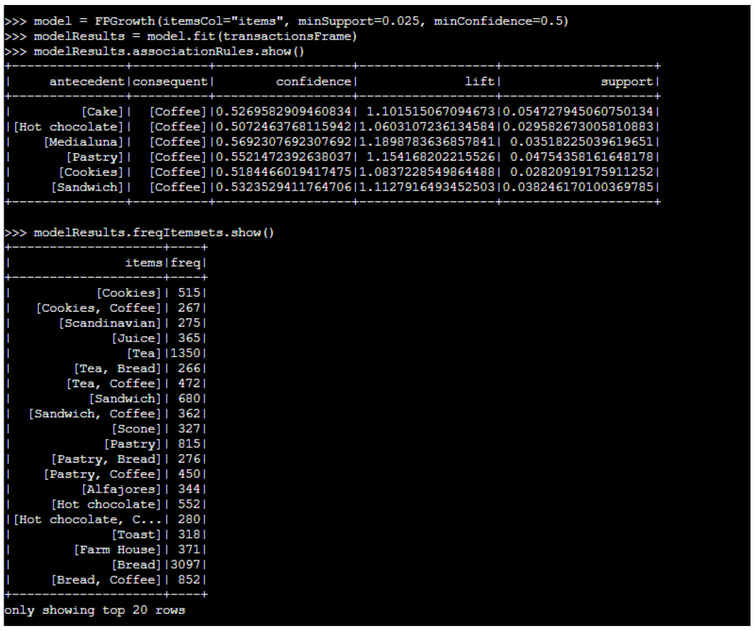
```
>>> model = FPGrowth(itemsCol="items", minSupport=0.025, minConfidence=0.25)
>>> modelResults = model.fit(transactionsFrame)
>>> modelResults.associationRules.show()
+---------------+----------+------------------+------------------+--------------------+
|     antecedent|consequent|        confidence|              lift|             support|
+---------------+----------+------------------+------------------+--------------------+
|         [Cake]|  [Coffee]| 0.526958290946083 4| 1.101515067094673|0.054727945060750134|
|[Hot chocolate]|  [Coffee]| 0.5072463768115942|1.0603107236134584|0.029582673005810883|
|    [Medialuna]|  [Coffee]| 0.5692307692307692|1.1898783636857841| 0.03518225039619651|
|          [Tea]|  [Coffee]| 0.3496296296296296|0.7308402041617589|0.049867934495509775|
|       [Pastry]|   [Bread]|0.3386503067484662 5|1.0349774470049187|0.029160063391442156|
|       [Pastry]|  [Coffee]| 0.5521472392638037| 1.154168202215526| 0.047543581616481 78|
|        [Bread]|  [Coffee]|0.27510494026477234| 0.575059244612648| 0.09001584786053883|
|      [Cookies]|  [Coffee]| 0.5184466019417475|1.0837228549864488| 0.028209191759112 52|
|     [Sandwich]|  [Coffee]| 0.5323529411764706|1.1127916493452503|0.038246170100369785|
+---------------+----------+------------------+------------------+--------------------+

>>> modelResults.freqItemsets.show()
+--------------------+----+
|               items|freq|
+--------------------+----+
|           [Cookies]| 515|
|   [Cookies, Coffee]| 267|
|      [Scandinavian]| 275|
|             [Juice]| 365|
|               [Tea]|1350|
|        [Tea, Bread]| 266|
|       [Tea, Coffee]| 472|
|          [Sandwich]| 680|
|  [Sandwich, Coffee]| 362|
|             [Scone]| 327|
|            [Pastry]| 815|
|     [Pastry, Bread]| 276|
|    [Pastry, Coffee]| 450|
|         [Alfajores]| 344|
|     [Hot chocolate]| 552|
|[Hot chocolate, C...| 280|
|             [Toast]| 318|
|        [Farm House]| 371|
|             [Bread]|3097|
|     [Bread, Coffee]| 852|
+--------------------+----+
only showing top 20 rows

>>> 
```

minSupport = low and minConfidence=Low

2. minSupport = Low and minConfidence= High

model = FPGrowth(itemsCol="items", minSupport=0.025, minConfidence=0.5)

In this case min support is set to 2.5% that means only itemset that appears in at least 2.5% of the transaction will be consider frequent. Min Confidence is set to 50% meaning that the rules will be more strongly associated to be considered valid.



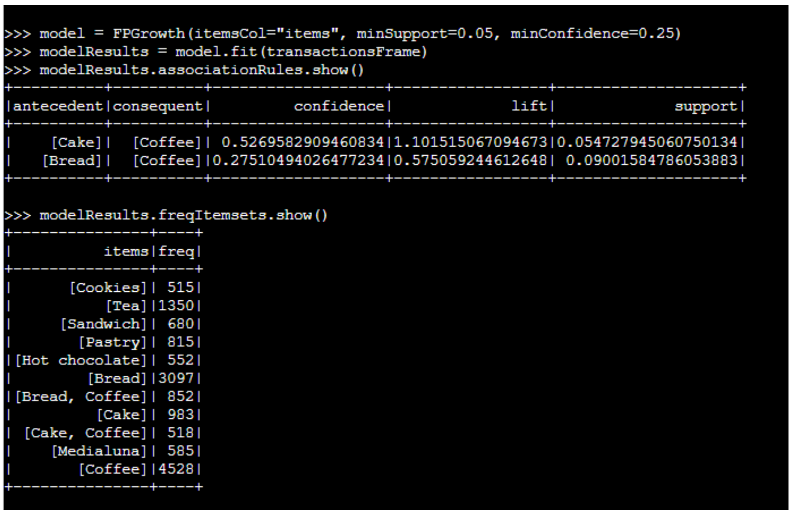
```
>>> model = FPGrowth(itemsCol="items", minSupport=0.025, minConfidence=0.5)
>>> modelResults = model.fit(transactionsFrame)
>>> modelResults.associationRules.show()
+---------------+----------+------------------+------------------+--------------------+
|     antecedent|consequent|        confidence|              lift|             support|
+---------------+----------+------------------+------------------+--------------------+
|         [Cake]|  [Coffee]|0.5269582909460834| 1.101515067094673|0.054727945060750134|
|[Hot chocolate]|  [Coffee]|0.5072463768115942|1.0603107236134584|0.029582673005810883|
|    [Medialuna]|  [Coffee]|0.5692307692307692|1.1898783636857841| 0.03518225039619651|
|       [Pastry]|  [Coffee]|0.5521472392638037| 1.154168202215526| 0.04754358161648178|
|      [Cookies]|  [Coffee]|0.5184466019417475|1.0837228549864488| 0.02820919175911252|
|     [Sandwich]|  [Coffee]|0.5323529411764706|1.1127916493452503|0.038246170100369785|
+---------------+----------+------------------+------------------+--------------------+

>>> modelResults.freqItemsets.show()
+--------------------+----+
|               items|freq|
+--------------------+----+
|           [Cookies]| 515|
|   [Cookies, Coffee]| 267|
|      [Scandinavian]| 275|
|             [Juice]| 365|
|               [Tea]|1350|
|        [Tea, Bread]| 266|
|       [Tea, Coffee]| 472|
|          [Sandwich]| 680|
|  [Sandwich, Coffee]| 362|
|             [Scone]| 327|
|            [Pastry]| 815|
|     [Pastry, Bread]| 276|
|    [Pastry, Coffee]| 450|
|         [Alfajores]| 344|
|     [Hot chocolate]| 552|
|[Hot chocolate, C...| 280|
|             [Toast]| 318|
|        [Farm House]| 371|
|             [Bread]|3097|
|     [Bread, Coffee]| 852|
+--------------------+----+
only showing top 20 rows
```

minSupport = low and minConfidence=high

3. minSupport = High and minConfidence=Low

model = FPGrowth(itemsCol="items", minSupport=0.05, minConfidence=0.25)

In this case min Support is set to 5% it means that only itemsets appearing in at least 5% of transactions are considered frequent. Min confidence of 25% is set and the rules don't need to be strongly associated



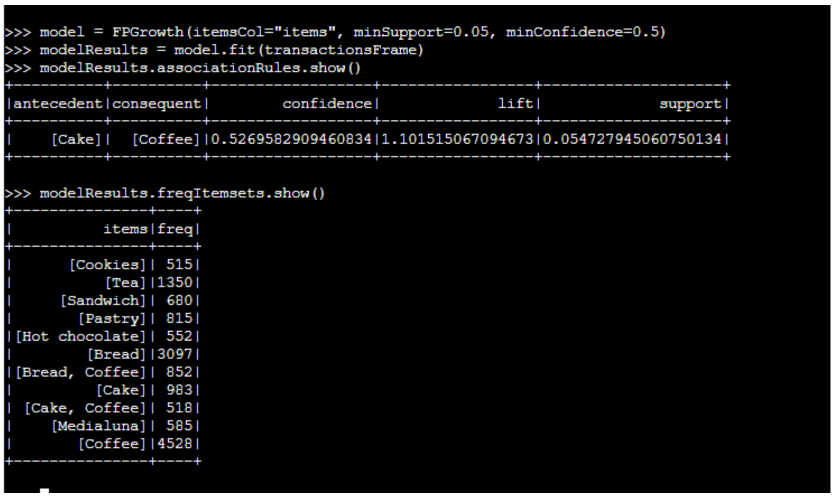
```
>>> model = FPGrowth(itemsCol="items", minSupport=0.05, minConfidence=0.25)
>>> modelResults = model.fit(transactionsFrame)
>>> modelResults.associationRules.show()
+----------+----------+-------------------+-----------------+--------------------+
|antecedent|consequent|         confidence|             lift|             support|
+----------+----------+-------------------+-----------------+--------------------+
|    [Cake]|  [Coffee]| 0.5269582909460834|1.101515067094673|0.054727945060750134|
|   [Bread]|  [Coffee]|0.27510494026477234|0.575059244612648| 0.09001584786053883|
+----------+----------+-------------------+-----------------+--------------------+

>>> modelResults.freqItemsets.show()
+---------------+----+
|          items|freq|
+---------------+----+
|      [Cookies]| 515|
|          [Tea]|1350|
|     [Sandwich]| 680|
|       [Pastry]| 815|
|[Hot chocolate]| 552|
|        [Bread]|3097|
|[Bread, Coffee]| 852|
|         [Cake]| 983|
| [Cake, Coffee]| 518|
|    [Medialuna]| 585|
|       [Coffee]|4528|
+---------------+----+
```

minSupport = High and minConfidence=Low

4. **minSupport =High and minConfidence=High**

   model = FPGrowth(itemsCol="items", minSupport=0.05, minConfidence=0.5)

   In this case min Support is set to 5% it means that only itemsets appearing in at least 5% of transactions are considered frequent. Min confidence is set to 50% which requires a stronger associations.



```
>>> model = FPGrowth(itemsCol="items", minSupport=0.05, minConfidence=0.5)
>>> modelResults = model.fit(transactionsFrame)
>>> modelResults.associationRules.show()
+----------+----------+------------------+-----------------+--------------------+
|antecedent|consequent|        confidence|             lift|             support|
+----------+----------+------------------+-----------------+--------------------+
|    [Cake]|  [Coffee]|0.5269582909460834|1.101515067094673|0.054727945060750134|
+----------+----------+------------------+-----------------+--------------------+

>>> modelResults.freqItemsets.show()
+---------------+----+
|          items|freq|
+---------------+----+
|      [Cookies]| 515|
|          [Tea]|1350|
|     [Sandwich]| 680|
|       [Pastry]| 815|
|[Hot chocolate]| 552|
|        [Bread]|3097|
|[Bread, Coffee]| 852|
|         [Cake]| 983|
| [Cake, Coffee]| 518|
|    [Medialuna]| 585|
|       [Coffee]|4528|
+---------------+----+
```

minSupport =High and minConfidence=High

In 4 different considerations we can observe the output of each based on minsupport and minconfidence. By adjusting minsupport and minconfidence parameter we can control sensitivity and specificity of the association rule mining process.

After this we will fit the model to the transaction data which will helps to generate frequent itemsets and association rules

At last, I am displaying the association rules and frequent itemset obtained from the fitted model which we can see in the screenshots which I have used in 4 conditions of minsupport and minconfidence.

Summary :- First import required libraries, read csv file and it will process it to create in a required format with the column which we will select , applies FPGrowth algorithm and display the association rule and frequent itemsets.

Third Question

Iris dataset (Iris.csv) is a dataset that contains measurement of iris flowers and their species classification. Write code in Scala or Python for Spark that will use Decision Tree Classification for predicting the flower species based on the four measurement of iris flowers.

```
from pyspark.mllib.tree import DecisionTree, DecisionTreeModel

from pyspark.mllib.regression import LabeledPoint

species_labels = ["Setosa", "Versicolor", "Virginica"]

def getFeatures(x):

    return [x["sepal.length"], x["sepal.width"], x["petal.length"], x["petal.width"]]

def getLabel(x):

    return species_labels.index(x["variety"])

df = spark.read.csv("s3://yashashass2-spark-bucket/iris.csv", header=True).rdd

(trainingData, testData) = df.randomSplit([0.7,0.3],seed=42)

labeledPoints = trainingData.map(lambda x : LabeledPoint( getLabel(x), getFeatures(x)))

model = DecisionTree.trainClassifier(labeledPoints, numClasses=3, categoricalFeaturesInfo={},
impurity='gini', maxDepth=5, maxBins=32)

predictions = model.predict(testData.map(lambda x : getFeatures(x)))

list(predictions.toLocalIterator())

labelsAndPredictions = testData.map(lambda x : getLabel(x)).zip(predictions)

testErr = labelsAndPredictions.filter(lambda lp : lp[0] != lp[1]).count() / float(testData.count())

print("Test Error is " + str(testErr))

accuracy = 1.0-testErr

print("Accuracy of the model:",accuracy)

def predictVariety(features):

    labeled_point = LabeledPoint(0, features)

    predicted_label = int(model.predict(labeled_point.features))
    predicted_variety = species_labels[predicted_label]
    return predicted_variety
input = [5.1, 3.5, 1.4, 0.2]
prediction = predictVariety(input)
print("Predicted Variety:", prediction)
.
```

Steps and Explanation for the above code

1. Importing libraries which will help to perform Decision tree-related functionality and regression-related functionalities.
   ```
   from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
   from pyspark.mllib.regression import LabeledPoint
   ```

2. Species labels is a list containing labels for three species and it is a independent index and I can also say target variable.
   ```
   species_labels = ["Setosa", "Versicolor", "Virginica"]
   ```

3. I am creating 2 functions in one it extracts the features from a data point and the second one will get the label index from species_lables.
   ```
   def getFeatures(x):
       return [x["sepal.length"], x["sepal.width"], x["petal.length"], x["petal.width"]]
   def getLabel(x):
       return species_labels.index(x["variety"])
   ```

4. Reading the csv file from s3 bucket which I have created and stored iris.csv file.
   ```
   df = spark.read.csv("s3://yashashass2-spark-bucket/iris.csv", header=True).rdd
   ```

5. I am splitting data into 70-30 ratio for testing and training where 70% is for training the model and 30% is for testing it. I have also used seed parameter and set it to 42 which will help me to ensure the reproductivity by using a fixed seed for random number generation during data splitting.
   ```
   (trainingData, testData) = df.randomSplit([0.7,0.3],seed=42)
   ```

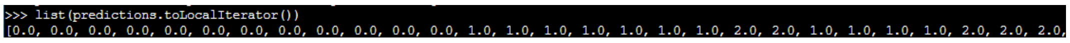
6. I am creating the label points which represents a labelled data point with features and label and maps each training data point to a labelled point using feature extraction functions.
   ```
   labeledPoints = trainingData.map(lambda x : LabeledPoint( getLabel(x), getFeatures(x)))
   ```

7. Here I am training decision tree module by using decision tree classifier model with parameters labeledpoints , number of classes, impurity measure ,tree depth and bins.
   ```
   model = DecisionTree.trainClassifier(labeledPoints, numClasses=3,
   categoricalFeaturesInfo={}, impurity='gini', maxDepth=5, maxBins=32)
   ```

8. We will use the trained model to predict labels for the test data.
   ```
   predictions = model.predict(testData.map(lambda x : getFeatures(x)))
   ```

list(predictions.toLocalIterator())

labelsAndPredictions = testData.map(lambda x : getLabel(x)).zip(predictions)

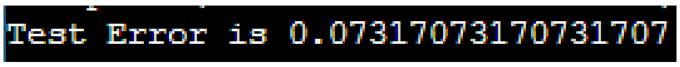


```
>>> list(predictions.toLocalIterator())
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0,
```

Helps to print the predictions the test data.

9. Model is evaluated by calculating the test error rate by comparing actual and predicted labels, it will also prints the trest Error and accuracy of the model . I am getting test error 0.073 (7.3%) and accuracy 0.92 (92.7%) for the model.
   testErr = labelsAndPredictions.filter(lambda lp : lp[0] != lp[1]).count() / float(testData.count())
   print("Test Error is " + str(testErr))

   accuracy = 1.0-testErr
   print("Accuracy of the model:",accuracy)

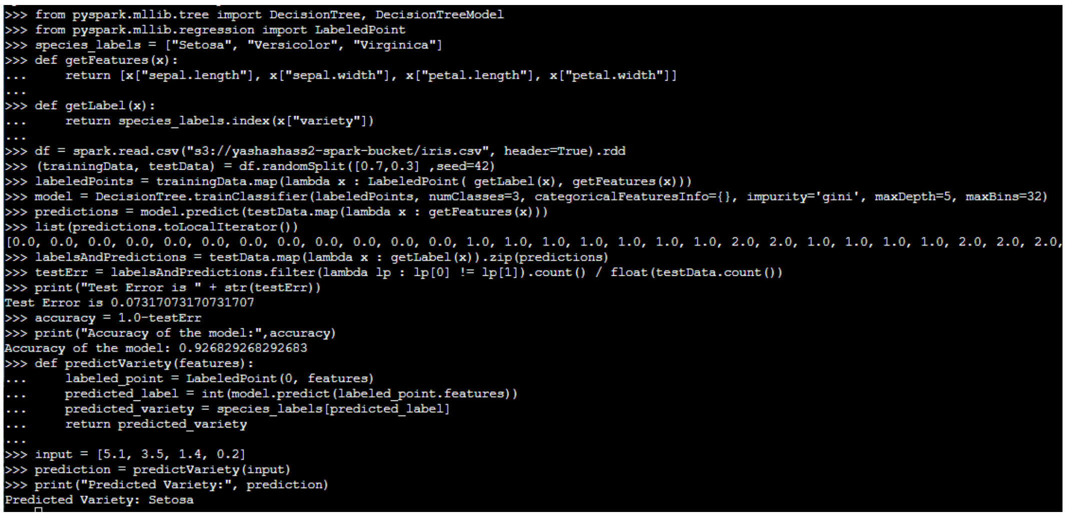


```
Test Error is 0.07317073170731707
```

Test Error

```
Accuracy of the model: 0.926829268292683
```

Accuracy

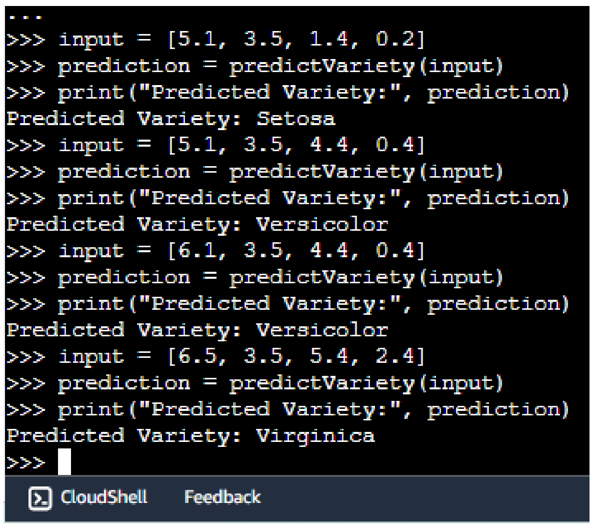


```
>>> from pyspark.mllib.tree import DecisionTree, DecisionTreeModel
>>> from pyspark.mllib.regression import LabeledPoint
>>> species_labels = ["Setosa", "Versicolor", "Virginica"]
>>> def getFeatures(x):
...     return [x["sepal.length"], x["sepal.width"], x["petal.length"], x["petal.width"]]
...
>>> def getLabel(x):
...     return species_labels.index(x["variety"])
...
>>> df = spark.read.csv("s3://yashashass2-spark-bucket/iris.csv", header=True).rdd
>>> (trainingData, testData) = df.randomSplit([0.7,0.3] ,seed=42)
>>> labeledPoints = trainingData.map(lambda x : LabeledPoint( getLabel(x), getFeatures(x)))
>>> model = DecisionTree.trainClassifier(labeledPoints, numClasses=3, categoricalFeaturesInfo={}, impurity='gini', maxDepth=5, maxBins=32)
>>> predictions = model.predict(testData.map(lambda x : getFeatures(x)))
>>> list(predictions.toLocalIterator())
[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, 2.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 2.0, 2.0,
>>> labelsAndPredictions = testData.map(lambda x : getLabel(x)).zip(predictions)
>>> testErr = labelsAndPredictions.filter(lambda lp : lp[0] != lp[1]).count() / float(testData.count())
>>> print("Test Error is " + str(testErr))
Test Error is 0.07317073170731707
>>> accuracy = 1.0-testErr
>>> print("Accuracy of the model:",accuracy)
Accuracy of the model: 0.926829268292683
>>> def predictVariety(features):
...     labeled_point = LabeledPoint(0, features)
...     predicted_label = int(model.predict(labeled_point.features))
...     predicted_variety = species_labels[predicted_label]
...     return predicted_variety
...
>>> input = [5.1, 3.5, 1.4, 0.2]
>>> prediction = predictVariety(input)
>>> print("Predicted Variety:", prediction)
Predicted Variety: Setosa
```

Executing code in spark using python

10. I am defining the function to predict the model based on four inputs which is given and the model will predict and return the output of the variety of iris flower. Below image shows the working of model
    def predictVariety(features):
        labeled_point = LabeledPoint(0, features)

```
   predicted_label = int(model.predict(labeled_point.features))
    predicted_variety = species_labels[predicted_label]
   return predicted_variety
input = [5.1, 3.5, 1.4, 0.2]
prediction = predictVariety(input)
print("Predicted Variety:", prediction)
```



```
...
>>> input = [5.1, 3.5, 1.4, 0.2]
>>> prediction = predictVariety(input)
>>> print("Predicted Variety:", prediction)
Predicted Variety: Setosa
>>> input = [5.1, 3.5, 4.4, 0.4]
>>> prediction = predictVariety(input)
>>> print("Predicted Variety:", prediction)
Predicted Variety: Versicolor
>>> input = [6.1, 3.5, 4.4, 0.4]
>>> prediction = predictVariety(input)
>>> print("Predicted Variety:", prediction)
Predicted Variety: Versicolor
>>> input = [6.5, 3.5, 5.4, 2.4]
>>> prediction = predictVariety(input)
>>> print("Predicted Variety:", prediction)
Predicted Variety: Virginica
>>>
```

Based on inputs it is giving the result