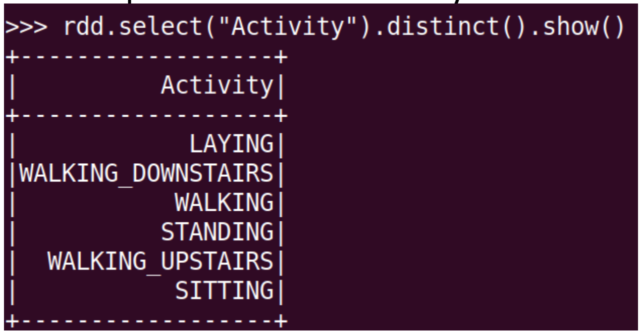
BIG Data Assignment

By : Hatem , Mojeeb

Students numbers :2132159,2141987

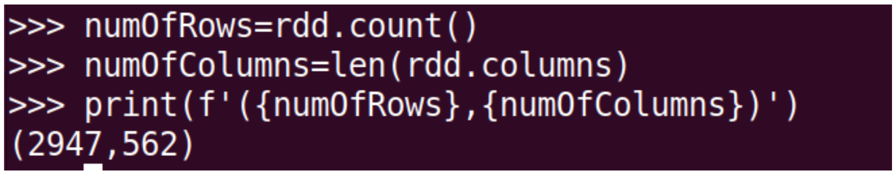
Problem 1:

1. Use PySpark to read the data into a DataFrame and print the number of unique classes present in the 'Activity' column.



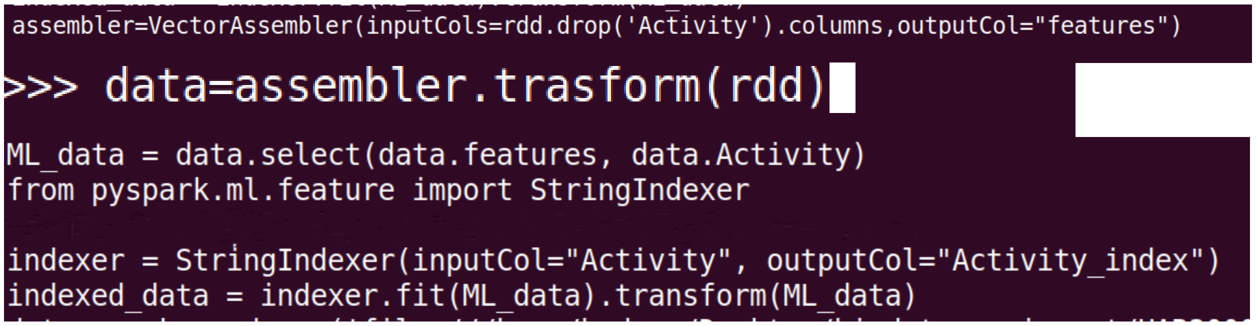
```
>>> rdd.select("Activity").distinct().show()
+------------------+
|          Activity|
+------------------+
|            LAYING|
|WALKING_DOWNSTAIRS|
|           WALKING|
|          STANDING|
|  WALKING_UPSTAIRS|
|           SITTING|
+------------------+
```

2. What are the dimensions of this dataset?



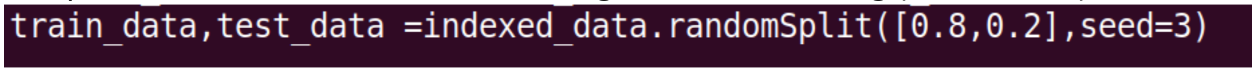
```
>>> numOfRows=rdd.count()
>>> numOfColumns=len(rdd.columns)
>>> print(f'({numOfRows},{numOfColumns})')
(2947,562)
```

3. Prepare the dataset for the logistic regression classification algorithm



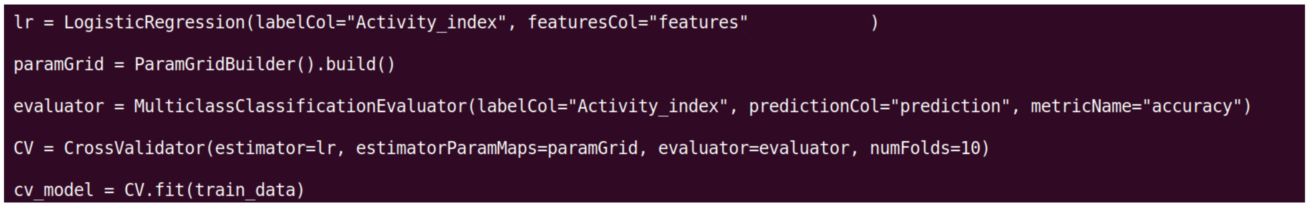
```
assembler=VectorAssembler(inputCols=rdd.drop('Activity').columns,outputCol="features")
>>> data=assembler.trasform(rdd)
ML_data = data.select(data.features, data.Activity)
from pyspark.ml.feature import StringIndexer

indexer = StringIndexer(inputCol="Activity", outputCol="Activity_index")
indexed_data = indexer.fit(ML_data).transform(ML_data)
```

4. Split the dataset into 80% for training and 20 for testing (use seed=3).



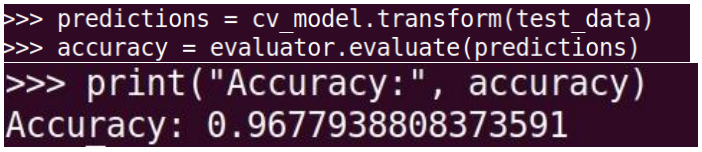
```
train_data,test_data =indexed_data.randomSplit([0.8,0.2],seed=3)
```

5. On the training dataset, apply logistic regression through cross-validation with a 10-fold.



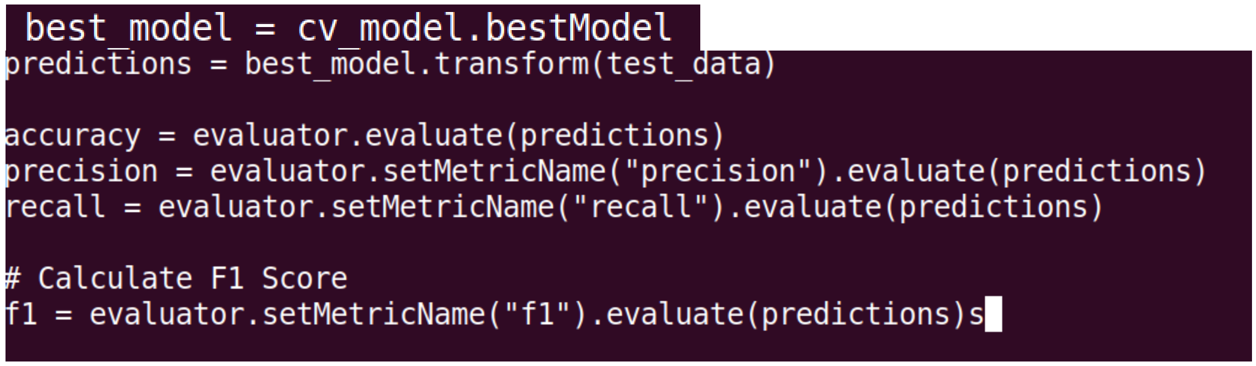
```
lr = LogisticRegression(labelCol="Activity_index", featuresCol="features"            )

paramGrid = ParamGridBuilder().build()

evaluator = MulticlassClassificationEvaluator(labelCol="Activity_index", predictionCol="prediction", metricName="accuracy")

CV = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=10)

cv_model = CV.fit(train_data)
```

6. Evaluate the best model on the unseen testing dataset generated above



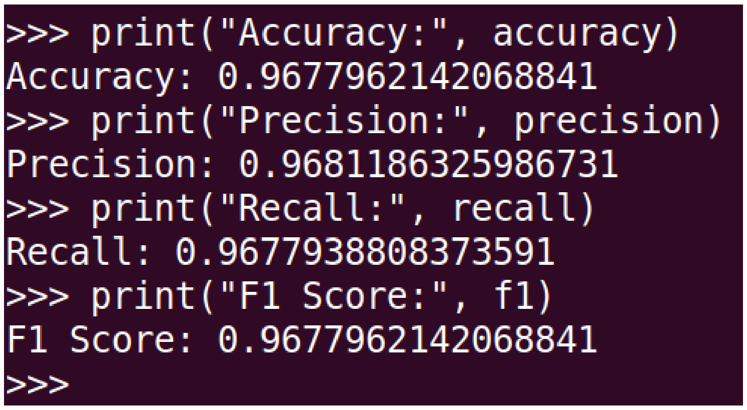
```
>>> predictions = cv_model.transform(test_data)
>>> accuracy = evaluator.evaluate(predictions)
>>> print("Accuracy:", accuracy)
Accuracy: 0.9677938808373591
```

7. Use multiclass classification evaluation and print out the accuracy, precision, recall, and F1- score.



```
 best_model = cv_model.bestModel
predictions = best_model.transform(test_data)

accuracy = evaluator.evaluate(predictions)
precision = evaluator.setMetricName("precision").evaluate(predictions)
recall = evaluator.setMetricName("recall").evaluate(predictions)

# Calculate F1 Score
f1 = evaluator.setMetricName("f1").evaluate(predictions)s
```

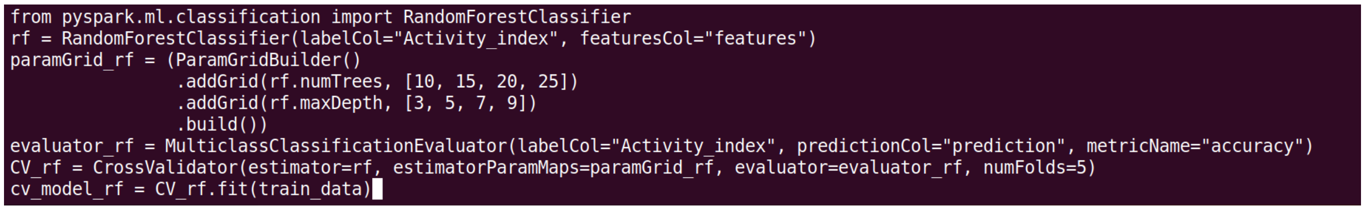
Here is the results :



```
>>> print("Accuracy:", accuracy)
Accuracy: 0.9677962142068841
>>> print("Precision:", precision)
Precision: 0.9681186325986731
>>> print("Recall:", recall)
Recall: 0.9677938808373591
>>> print("F1 Score:", f1)
F1 Score: 0.9677962142068841
>>>
```

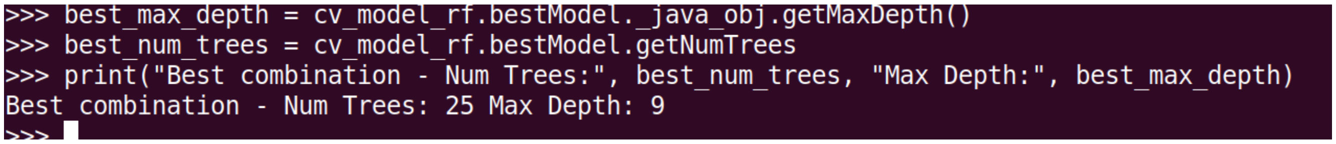
8. On the same training dataset and testing dataset apply the RandomForest algorithm:

8.1 Use the grid search method to set the number of trees where the searching space is [10, 15, 20, 25] and the max depth where the searching space is [3, 5, 7, 9] that maximize the accuracy.



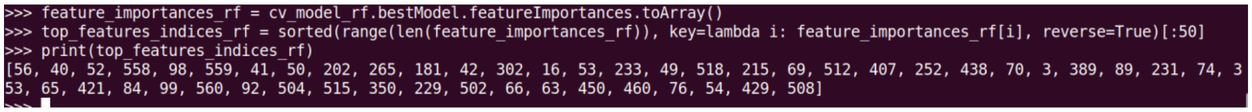
```
from pyspark.ml.classification import RandomForestClassifier
rf = RandomForestClassifier(labelCol="Activity_index", featuresCol="features")
paramGrid_rf = (ParamGridBuilder()
                .addGrid(rf.numTrees, [10, 15, 20, 25])
                .addGrid(rf.maxDepth, [3, 5, 7, 9])
                .build())
evaluator_rf = MulticlassClassificationEvaluator(labelCol="Activity_index", predictionCol="prediction", metricName="accuracy")
CV_rf = CrossValidator(estimator=rf, estimatorParamMaps=paramGrid_rf, evaluator=evaluator_rf, numFolds=5)
cv_model_rf = CV_rf.fit(train_data)
```

8.2 Which combination shows the maximum accuracy?



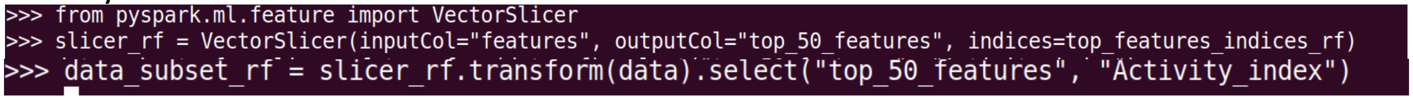
```
>>> best_max_depth = cv_model_rf.bestModel._java_obj.getMaxDepth()
>>> best_num_trees = cv_model_rf.bestModel.getNumTrees
>>> print("Best combination - Num Trees:", best_num_trees, "Max Depth:", best_max_depth)
Best combination - Num Trees: 25 Max Depth: 9
>>>
```

8.3 Extract the top 50 important features according to the RandomForest model you developed.



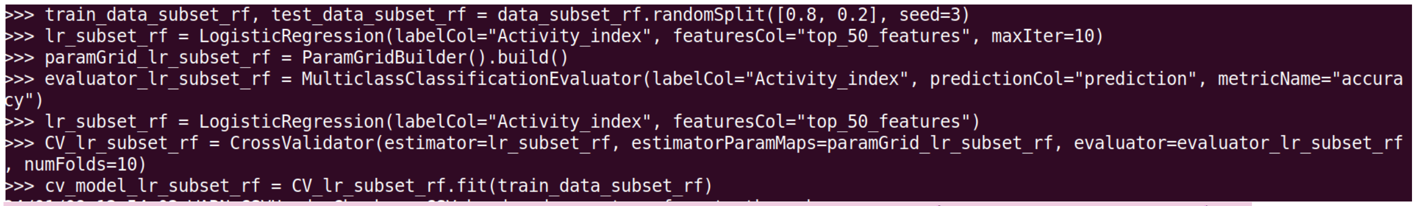
```
>>> feature_importances_rf = cv_model_rf.bestModel.featureImportances.toArray()
>>> top_features_indices_rf = sorted(range(len(feature_importances_rf)), key=lambda i: feature_importances_rf[i], reverse=True)[:50]
>>> print(top_features_indices_rf)
[56, 40, 52, 558, 98, 559, 41, 50, 202, 265, 181, 42, 302, 16, 53, 233, 49, 518, 215, 69, 512, 407, 252, 438, 70, 3, 389, 89, 231, 74, 3
53, 65, 421, 84, 99, 560, 92, 504, 515, 350, 229, 502, 66, 63, 450, 460, 76, 54, 429, 508]
>>>
```

8.4 Subset the original dataset and keep only the top 50 features selected above + the Activity column.

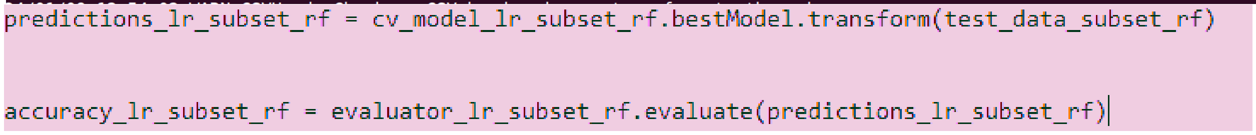


```
>>> from pyspark.ml.feature import VectorSlicer
>>> slicer_rf = VectorSlicer(inputCol="features", outputCol="top_50_features", indices=top_features_indices_rf)
>>> data_subset_rf = slicer_rf.transform(data).select("top_50_features", "Activity_index")
```

9. Prepare the new subset dataset (50 features + Activity label), apply logistic regression again, and evaluate the model on the unseen testing dataset.

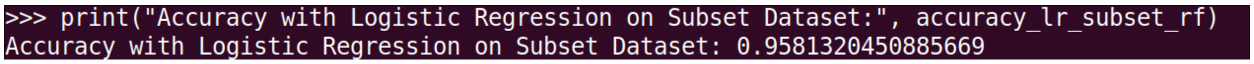


```
>>> train_data_subset_rf, test_data_subset_rf = data_subset_rf.randomSplit([0.8, 0.2], seed=3)
>>> lr_subset_rf = LogisticRegression(labelCol="Activity_index", featuresCol="top_50_features", maxIter=10)
>>> paramGrid_lr_subset_rf = ParamGridBuilder().build()
>>> evaluator_lr_subset_rf = MulticlassClassificationEvaluator(labelCol="Activity_index", predictionCol="prediction", metricName="accura
cy")
>>> lr_subset_rf = LogisticRegression(labelCol="Activity_index", featuresCol="top_50_features")
>>> CV_lr_subset_rf = CrossValidator(estimator=lr_subset_rf, estimatorParamMaps=paramGrid_lr_subset_rf, evaluator=evaluator_lr_subset_rf
, numFolds=10)
>>> cv_model_lr_subset_rf = CV_lr_subset_rf.fit(train_data_subset_rf)
```



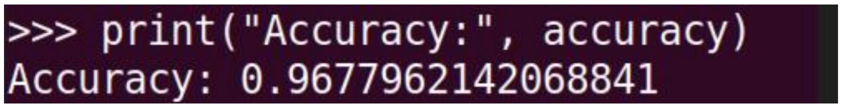
```
predictions_lr_subset_rf = cv_model_lr_subset_rf.bestModel.transform(test_data_subset_rf)


accuracy_lr_subset_rf = evaluator_lr_subset_rf.evaluate(predictions_lr_subset_rf)
```

10. Compare the accuracy you got in (9) and the one you got in (7) above. What do you think? We can get similar results by using less features and bulding a less complex model



```
>>> print("Accuracy with Logistic Regression on Subset Dataset:", accuracy_lr_subset_rf)
Accuracy with Logistic Regression on Subset Dataset: 0.9581320450885669
```

The accuracy with all the features:



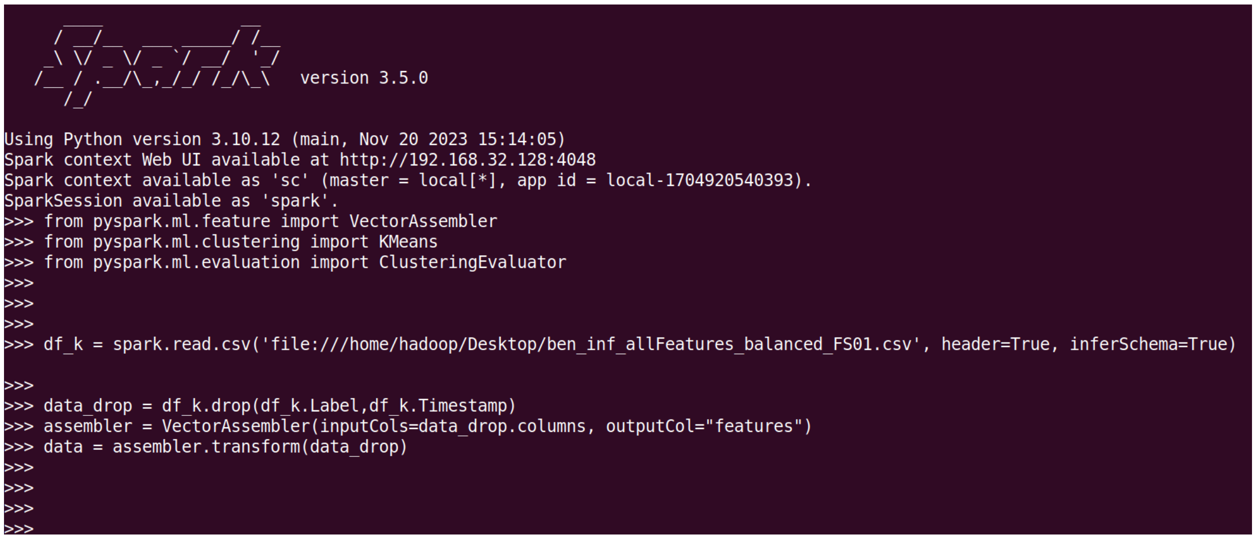
```
>>> print("Accuracy:", accuracy)
Accuracy: 0.9677962142068841
```

Problem 2:

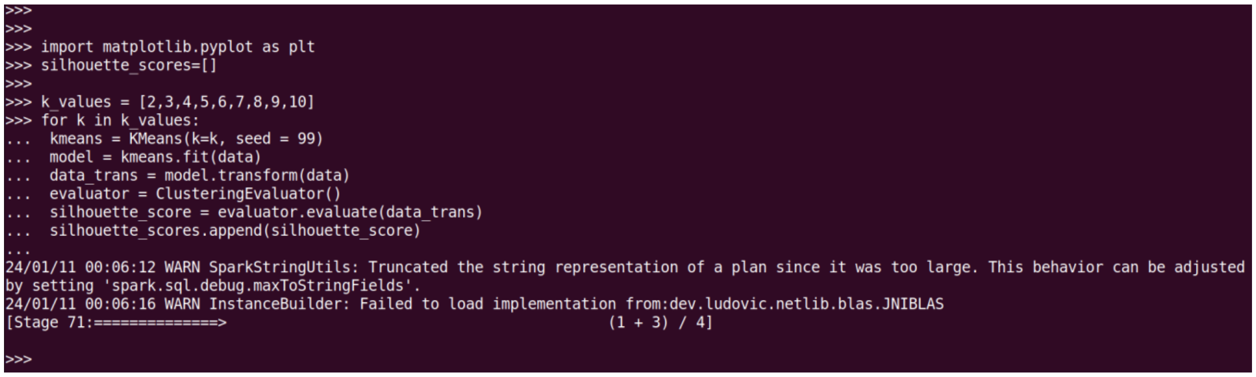
Consider the attached “ben_inf_allFeatures_balanced_FS01.csv” which represents a sample of a dataset related to a Cybersecurity project. This collection of data captures various network activities and potential threats. It may include data such as log files, network traffic patterns, system vulnerabilities, and other relevant information that security analysts use to monitor, analyze, and respond to cybersecurity events. The dataset provides valuable insights into the cybersecurity landscape, aiding in the development of machine learning models and algorithms for threat detection, anomaly detection, and overall cybersecurity defense strategies

1. Read the dataset, create a data frame, and prepare the dataset for clustering using the PySpark k-mean algorithm. Please use seed= 99.

2. What is the value of K that maximizes the overall Silhouette Score? You may need to try values between 2 and 10. Remember, you need to show the evidence. A good one could be the figure in Ch05-Slide number 31.

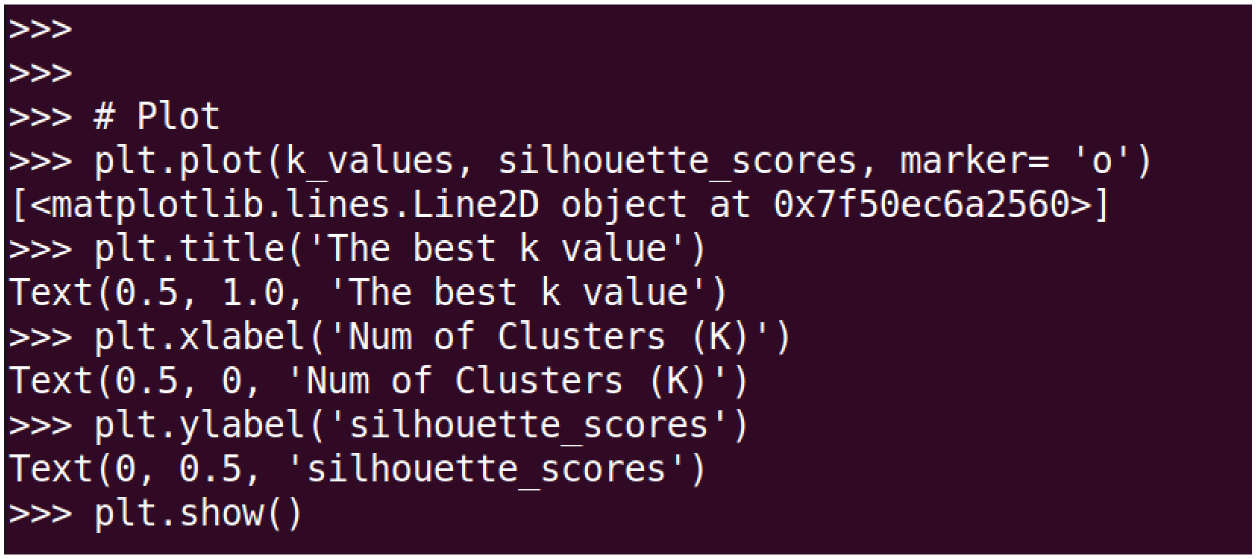


```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Python version 3.10.12 (main, Nov 20 2023 15:14:05)
Spark context Web UI available at http://192.168.32.128:4048
Spark context available as 'sc' (master = local[*], app id = local-1704920540393).
SparkSession available as 'spark'.
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.clustering import KMeans
>>> from pyspark.ml.evaluation import ClusteringEvaluator
>>>
>>>
>>>
>>> df_k = spark.read.csv('file:///home/hadoop/Desktop/ben_inf_allFeatures_balanced_FS01.csv', header=True, inferSchema=True)

>>>
>>> data_drop = df_k.drop(df_k.Label,df_k.Timestamp)
>>> assembler = VectorAssembler(inputCols=data_drop.columns, outputCol="features")
>>> data = assembler.transform(data_drop)
>>>
>>>
>>>
>>>
```



```
>>>
>>>
>>> import matplotlib.pyplot as plt
>>> silhouette_scores=[]
>>>
>>> k_values = [2,3,4,5,6,7,8,9,10]
>>> for k in k_values:
...  kmeans = KMeans(k=k, seed = 99)
...  model = kmeans.fit(data)
...  data_trans = model.transform(data)
...  evaluator = ClusteringEvaluator()
...  silhouette_score = evaluator.evaluate(data_trans)
...  silhouette_scores.append(silhouette_score)
...
24/01/11 00:06:12 WARN SparkStringUtils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted
by setting 'spark.sql.debug.maxToStringFields'.
24/01/11 00:06:16 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
[Stage 71:==============>                                          (1 + 3) / 4]

>>>
```

PLOT:



```
>>>
>>>
>>> # Plot
>>> plt.plot(k_values, silhouette_scores, marker= 'o')
[<matplotlib.lines.Line2D object at 0x7f50ec6a2560>]
>>> plt.title('The best k value')
Text(0.5, 1.0, 'The best k value')
>>> plt.xlabel('Num of Clusters (K)')
Text(0.5, 0, 'Num of Clusters (K)')
>>> plt.ylabel('silhouette_scores')
Text(0, 0.5, 'silhouette_scores')
>>> plt.show()
```

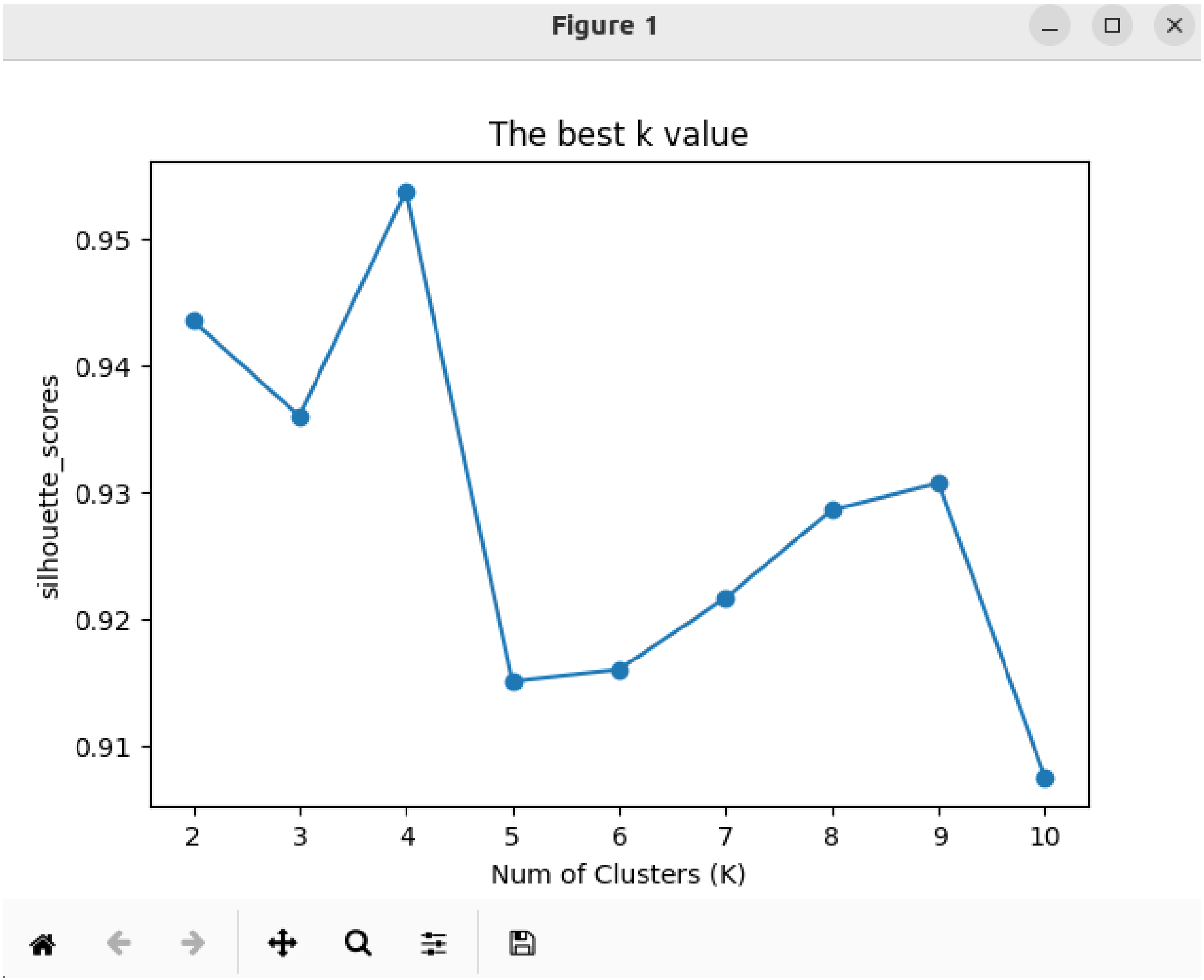


Figure 1
The best k value
silhouette_scores
Num of Clusters (K)