EXPERIMENT – 6

AIM: To implement word count / frequency programs using MapReduce.

DESCRIPTION:
In MapReduce word count example, we find out the frequency of each word. Here, the role of Mapper is to map the keys to the existing values and the role of Reducer is to aggregate the keys of common values. So, everything is represented in the form of Key-value pair.

CODE:
mapper.py:

```
#!/usr/bin/python

# import sys because we need to read and write data to STDIN and STDOUT
import sys

# reading entire line from STDIN (standard input)
for line in sys.stdin:
    # to remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()

    # we are looping over the words array and printing the word
    # with the count of 1 to the STDOUT
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        print('%s\t%s' % (word, 1))
```

reducer.py:

```
#!/usr/bin/python
from operator import itemgetter
import sys
current_word = None
current_count = 0
word = None
# read the entire line from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # slpiting the data on the basis of tab we have provided in mapper.py
    word, count = line.split('\t', 1)
    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
```

```
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print('%s\t%s' % (current_word, current_count))
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print('%s\t%s' % (current_word, current_count))
```

OUTPUT:

```
cselab8@cselab8-OptiPlex-3060:~/Documents$ cat word_count_data.txt
hello cbit
welcome to cbit
cbit is the top private engineering college

cselab8@cselab8-OptiPlex-3060:~/Documents$ cat word_count_data.txt | python3 mapper.py | sort -k1,1 | python3 reducer.py
cbit    3
college 1
engineering     1
hello   1
is      1
private 1
the     1
to      1
top     1
welcome 1
cselab8@cselab8-OptiPlex-3060:~/Documents$ cd ..
cselab8@cselab8-OptiPlex-3060:~$ start-dfs.sh
start-dfs.sh: command not found
cselab8@cselab8-OptiPlex-3060:~$
```

 Signature of the Faculty………………………...

EXPERIMENT – 7

AIM: To implement a MapReduce program that processes a dataset.

DESCRIPTION:
This code simulates a MapReduce process to compute basic statistics (average, min, and max) for each feature in the Boston Housing dataset. It begins by loading the data into a pandas DataFrame, then applies a mapper function to generate key-value pairs for each feature in each row. A combiner function aggregates partial sums and counts locally, improving efficiency. Next, the data is sorted and grouped by feature names for the reduction phase, where the reducer function calculates final statistics for each feature. The final output provides a summary of average, minimum, and maximum values for each feature, allowing for a quick overview of the dataset's key metrics.

CODE:

```
import pandas as pd
from collections import defaultdict
import itertools

# Step 1: Load the dataset

dataset = pd.read_csv('/content/BostonHousing.csv')

# Step 2: MapReduce Functions

# Mapper function: For each row, emit (feature_name, value)
def mapper(record):
    mapped_values = []
    for column in record.index:
        mapped_values.append((column, record[column]))
    return mapped_values

# Combiner function: Calculate partial sum and count for each feature
def combiner(mapped_data):
    combined_data = defaultdict(lambda: [0, 0])  # Dictionary to store sum and count
    for feature, value in mapped_data:
        combined_data[feature][0] += value  # Sum of values
        combined_data[feature][1] += 1      # Count of values
    return combined_data.items()

# Reducer function: Calculate average, min, and max for each feature
def reducer(feature, values):
    sum_values, count = 0, 0
    min_value, max_value = float('inf'), float('-inf')
    for value, cnt in values:
        sum_values += value
        count += cnt
        min_value = min(min_value, value / cnt)
        max_value = max(max_value, value / cnt)
    average = sum_values / count
```

```
    return (feature, {'average': average, 'min': min_value, 'max': max_value})

# Step 3: MapReduce Process
def mapreduce(dataset):
    # Mapping phase
    mapped = []
    for _, record in dataset.iterrows():
        mapped.extend(mapper(record))

    # Combining phase
    combined = combiner(mapped)

    # Shuffle and Sort phase
    sorted_combined = sorted(combined, key=lambda x: x[0])
    grouped = itertools.groupby(sorted_combined, key=lambda x: x[0])

    # Reducing phase
    reduced = [reducer(feature, [value for _, value in group]) for feature, group in grouped]

    return reduced

# Run the MapReduce process
result = mapreduce(dataset)

# Output the results
for feature, stats in result:
    print(f"{feature} - Average: {stats['average']:.2f}, Min: {stats['min']:.2f}, Max: {stats['max']:.2f}")
```

OUTPUT:

```
age - Average: 68.57, Min: 68.57, Max: 68.57
b - Average: 356.67, Min: 356.67, Max: 356.67
chas - Average: 0.07, Min: 0.07, Max: 0.07
crim - Average: 3.61, Min: 3.61, Max: 3.61
dis - Average: 3.80, Min: 3.80, Max: 3.80
indus - Average: 11.14, Min: 11.14, Max: 11.14
lstat - Average: 12.65, Min: 12.65, Max: 12.65
medv - Average: 22.53, Min: 22.53, Max: 22.53
nox - Average: 0.55, Min: 0.55, Max: 0.55
ptratio - Average: 18.46, Min: 18.46, Max: 18.46
rad - Average: 9.55, Min: 9.55, Max: 9.55
rm - Average: 6.28, Min: 6.28, Max: 6.28
tax - Average: 408.24, Min: 408.24, Max: 408.24
zn - Average: 11.36, Min: 11.36, Max: 11.36
```

EXPERIMENT – 8

AIM: To implement Linear Regression using SPARK

DESCRIPTION:
Linear regression is also a type of machine-learning algorithm more specifically a supervised machine-learning algorithm that learns from the labeled datasets and maps the data points to the most optimized linear functions. which can be used for prediction on new datasets.

Problem Statement: Build a predictive Model for the shipping company, to find an estimate of how many Crew members a ship requires. The dataset contains 159 instances with 9 features.

The Description of dataset is as below:

- Ship Name
- Cruise Line
- Age (as of 2013)
- Tonnage (1000s of tons)
- passengers (100s)
- Length (100s of feet)
- Cabins (100s)
- Passenger Density
- Crew (100s)

CODE AND OUTPUT:
pip install pyspark

```
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.9/316.9 MB 4.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425344 sha256=a8c0a7209ff91c7fc546c80d68e45b217be8ce6d1dbe761fff44a8af89734f79
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName('housing_price_model').getOrCreate()
df=spark.read.csv('cruise_ship_info.csv',inferSchema=True,header=True)

df.show(10)

```
+-----------+-----------+---+------------------+----------+------+------+-----------------+----+
|  Ship_name|Cruise_line|Age|           Tonnage|passengers|length|cabins|passenger_density|crew|
+-----------+-----------+---+------------------+----------+------+------+-----------------+----+
|    Journey|    Azamara|  6|30.276999999999997|      6.94|  5.94|  3.55|            42.64|3.55|
|      Quest|    Azamara|  6|30.276999999999997|      6.94|  5.94|  3.55|            42.64|3.55|
|Celebration|   Carnival| 26|            47.262|     14.86|  7.22|  7.43|             31.8| 6.7|
|   Conquest|   Carnival| 11|             110.0|     29.74|  9.53| 14.88|            36.99|19.1|
|    Destiny|   Carnival| 17|           101.353|     26.42|  8.92| 13.21|            38.36|10.0|
|    Ecstasy|   Carnival| 22|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
|    Elation|   Carnival| 15|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
|    Fantasy|   Carnival| 23|            70.367|     20.56|  8.55| 10.22|            34.23| 9.2|
|Fascination|   Carnival| 19|            70.367|     20.52|  8.55|  10.2|            34.29| 9.2|
|    Freedom|   Carnival|  6|110.23899999999999|      37.0|  9.51| 14.87|            29.79|11.5|
+-----------+-----------+---+------------------+----------+------+------+-----------------+----+
only showing top 10 rows
```

df.printSchema()

```
root
 |-- Ship_name: string (nullable = true)
 |-- Cruise_line: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Tonnage: double (nullable = true)
 |-- passengers: double (nullable = true)
 |-- length: double (nullable = true)
 |-- cabins: double (nullable = true)
 |-- passenger_density: double (nullable = true)
 |-- crew: double (nullable = true)
```

df.columns

```
['Ship_name',
 'Cruise_line',
 'Age',
 'Tonnage',
 'passengers',
 'length',
 'cabins',
 'passenger_density',
 'crew']
```

```
from pyspark.ml.feature import StringIndexer
indexer=StringIndexer(inputCol='Cruise_line',outputCol='cruise_cat')
indexed=indexer.fit(df).transform(df)
for item in indexed.head(5):
    print(item)
    print('\n')
```

```
Row(Ship_name='Journey', Cruise_line='Azamara', Age=6, Tonnage=30.276999999999997, passengers=6.94, length=5.94, cabins=3.55, passenger_density=42.64, crew=3.55, cruise_cat=16.0)


Row(Ship_name='Quest', Cruise_line='Azamara', Age=6, Tonnage=30.276999999999997, passengers=6.94, length=5.94, cabins=3.55, passenger_density=42.64, crew=3.55, cruise_cat=16.0)


Row(Ship_name='Celebration', Cruise_line='Carnival', Age=26, Tonnage=47.262, passengers=14.86, length=7.22, cabins=7.43, passenger_density=31.8, crew=6.7, cruise_cat=1.0)


Row(Ship_name='Conquest', Cruise_line='Carnival', Age=11, Tonnage=110.0, passengers=29.74, length=9.53, cabins=14.88, passenger_density=36.99, crew=19.1, cruise_cat=1.0)


Row(Ship_name='Destiny', Cruise_line='Carnival', Age=17, Tonnage=101.353, passengers=26.42, length=8.92, cabins=13.21, passenger_density=38.36, crew=10.0, cruise_cat=1.0)
```

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
assembler=VectorAssembler(inputCols=['Age',
 'Tonnage',
 'passengers',
 'length',
 'cabins',
 'passenger_density',
 'cruise_cat'],outputCol='features')
output=assembler.transform(indexed)
output.select('features','crew').show(5)
```

```
+--------------------+----+
|            features|crew|
+--------------------+----+
|[6.0,30.276999999...|3.55|
|[6.0,30.276999999...|3.55|
|[26.0,47.262,14.8...| 6.7|
|[11.0,110.0,29.74...|19.1|
|[17.0,101.353,26....|10.0|
+--------------------+----+
only showing top 5 rows
```

```
final_data=output.select('features','crew')
train_data,test_data=final_data.randomSplit([0.7,0.3])
train_data.describe().show()
```

```
+-------+-----------------+
|summary|             crew|
+-------+-----------------+
|  count|              105|
|   mean|7.950761904761916|
| stddev|3.504040441250561|
|    min|             0.59|
|    max|             21.0|
+-------+-----------------+
```

```
test_data.describe().show()
```

```
+-------+-----------------+
|summary|             crew|
+-------+-----------------+
|  count|               53|
|   mean| 7.483962264150944|
| stddev|3.5149834990435767|
|    min|             0.59|
|    max|             19.1|
+-------+-----------------+
```

```
from pyspark.ml.regression import LinearRegression
ship_lr=LinearRegression(featuresCol='features',labelCol='crew')
trained_ship_model=ship_lr.fit(train_data)
```

ship_results=trained_ship_model.evaluate(train_data)
print('Rsquared Error :',ship_results.r2)

```
Rsquared Error : 0.9481847866542444
```

unlabeled_data=test_data.select('features')
unlabeled_data.show(5)

```
+--------------------+
|            features|
+--------------------+
|[5.0,86.0,21.04,9...|
|[5.0,115.0,35.74,...|
|[5.0,122.0,28.5,1...|
|[5.0,160.0,36.34,...|
|[6.0,30.276999999...|
+--------------------+
only showing top 5 rows
```

predictions=trained_ship_model.transform(unlabeled_data)

predictions.show()

```
+--------------------+------------------+
|            features|        prediction|
+--------------------+------------------+
|[5.0,86.0,21.04,9...| 9.278710202707135|
|[5.0,115.0,35.74,...| 11.76297202430732|
|[5.0,122.0,28.5,1...| 6.097929011890256|
|[5.0,160.0,36.34,...|14.999343148777454|
|[6.0,30.276999999...| 4.502605773133349|
|[6.0,90.0,20.0,9....| 10.25952060536163|
|[6.0,110.23899999...|10.872193239369155|
|[9.0,105.0,27.2,8...|11.238882153319183|
|[9.0,110.0,29.74,...|12.042083055746394|
|[10.0,46.0,7.0,6....|2.8333565182617333|
|[10.0,68.0,10.8,7...| 6.716896911843758|
|[10.0,90.09,25.01...| 8.775263533366306|
|[10.0,105.0,27.2,...|11.226614045747292|
|[11.0,91.0,20.32,...| 9.274239553427998|
|[11.0,110.0,29.74...| 12.03114238409525|
|[12.0,42.0,14.8,7...| 6.792692274315869|
|[12.0,88.5,21.24,...|  9.46980994116172|
|[12.0,88.5,21.24,...|10.407256080350553|
|[12.0,91.0,20.32,...| 9.261971445856107|
|[12.0,91.0,20.32,...| 9.261971445856107|
+--------------------+------------------+
only showing top 20 rows
```

EXPERIMENT – 9

AIM: To implement Naïve Bayes Classification techniques using SPARK

DESCRIPTION:
Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e., every pair of features being classified is independent of each other.

CODE AND OUTPUT:
pip install pyspark

```
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.9/316.9 MB 4.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425344 sha256=c9559224f356dc0116f6841bcd9f00a6afc1561be90879fe5c83e946b08e41d2
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

```
#naive bayes
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StringIndexer
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# Read data from the vehicle_stolen_dataset.csv
spark=SparkSession.builder.appName("bayesclass").getOrCreate()
data=spark.read.csv('vehicle_stolen_dataset.csv',inferSchema=True)
data.show()
```

```
+----+------+-----+-----+---+
| _c0|   _c1|  _c2|  _c3|_c4|
+----+------+-----+-----+---+
|N001|   BMW|black|night|yes|
|N002|  Audi|black|night| no|
|N003|NISSAN|black|night|yes|
|N004|  VEGA|  red|  day|yes|
|N005|   BMW| blue|  day| no|
|N006|  Audi|black|  day|yes|
|N007|  VEGA|  red|night| no|
|N008|  Audi| blue|  day|yes|
|N009|  VEGA|black|  day|yes|
|N010|NISSAN| blue|  day| no|
|N011|   BMW|black|night|yes|
|N012|NISSAN|  red|  day| no|
|N013|  VEGA|black|night|yes|
|N014|   BMW|  red|  day| no|
|N015|  Audi|black|  day|yes|
|N016|  Audi| blue|night|yes|
|N017|  Audi|  red|  day| no|
|N018|NISSAN|black|  day|yes|
|N019|   BMW| blue|  day|yes|
|N020|   BMW|  red|night|yes|
+----+------+-----+-----+---+
```

data.columns

```
['_c0', '_c1', '_c2', '_c3', '_c4']
```

vehicle_df = data.select(col("_c0").alias("number_plate"), col("_c1").alias("brand"), col("_c2").alias("color"),
col("_c3").alias("time"),
col("_c4").alias("stoled"))
indexers = [
StringIndexer(inputCol="brand", outputCol = "brand_index"),
StringIndexer(inputCol="color", outputCol = "color_index"), StringIndexer(inputCol="time", outputCol = "time_index"), StringIndexer(inputCol="stoled", outputCol = "label")]
pipeline = Pipeline(stages=indexers)
#Fitting a model to the input dataset.
indexed_vehicle_df = pipeline.fit(vehicle_df).transform(vehicle_df)
indexed_vehicle_df.show(5,False)

```
+------------+------+-----+-----+------+-----------+-----------+----------+-----+
|number_plate|brand |color|time |stoled|brand_index|color_index|time_index|label|
+------------+------+-----+-----+------+-----------+-----------+----------+-----+
|N001        |BMW   |black|night|yes   |1.0        |0.0        |1.0       |0.0  |
|N002        |Audi  |black|night|no    |0.0        |0.0        |1.0       |1.0  |
|N003        |NISSAN|black|night|yes   |2.0        |0.0        |1.0       |0.0  |
|N004        |VEGA  |red  |day  |yes   |3.0        |1.0        |0.0       |0.0  |
|N005        |BMW   |blue |day  |no    |1.0        |2.0        |0.0       |1.0  |
+------------+------+-----+-----+------+-----------+-----------+----------+-----+
only showing top 5 rows
```

vectorAssembler = VectorAssembler(inputCols = ["brand_index", "color_index", "time_index"],outputCol = "features")
vindexed_vehicle_df = vectorAssembler.transform(indexed_vehicle_df)
vindexed_vehicle_df.show(5, False)

```
+------------+------+-----+-----+------+-----------+-----------+----------+-----+------------+
+------------+------+-----+-----+------+-----------+-----------+----------+-----+
|number_plate| brand|color| time|stoled|brand_index|color_index|time_index|label|
+------------+------+-----+-----+------+-----------+-----------+----------+-----+
|        N001|   BMW|black|night|   yes|        1.0|        0.0|       1.0|  0.0|
|        N002|  Audi|black|night|    no|        0.0|        0.0|       1.0|  1.0|
|        N003|NISSAN|black|night|   yes|        2.0|        0.0|       1.0|  0.0|
+------------+------+-----+-----+------+-----------+-----------+----------+-----+
only showing top 3 rows
```

indexed_vehicle_df.show(3)
splits = vindexed_vehicle_df.randomSplit([0.6,0.4], 42)
 # optional value 42 is seed for sampling
train_df = splits[0]
test_df = splits[1]
nb = NaiveBayes(modelType="multinomial")
nbmodel = nb.fit(train_df)
predictions_df = nbmodel.transform(test_df)

predictions_df.show(5, True)

number_plate	brand	color	time	stoled	brand_index	color_index	time_index	label	features	rawPrediction	probability	prediction
N001	BMW	black	night	yes	1.0	0.0	1.0	0.0	[1.0,0.0,1.0]	[-2.8415815937267...	[0.70850202429149...	0.0
N003	NISSAN	black	night	yes	2.0	0.0	1.0	0.0	[2.0,0.0,1.0]	[-3.5347287742866...	[0.85868498527968...	0.0
N005	BMW	blue	day	no	1.0	2.0	0.0	1.0	[1.0,2.0,0.0]	[-3.2470467018348...	[0.80201649862511...	0.0
N007	VEGA	red	night	no	3.0	1.0	1.0	1.0	[3.0,1.0,1.0]	[-5.3264882435147...	[0.92678896750413...	0.0
N009	VEGA	black	day	yes	3.0	0.0	0.0	0.0	[3.0,0.0,0.0]	[-2.4361164856185...	[0.97330367074527...	0.0

only showing top 5 rows

evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
nbaccuracy = evaluator.evaluate(predictions_df)
print("Test accuracy = " + str(nbaccuracy))

```
Test accuracy = 0.5
```

EXPERIMENT – 10

AIM: To implement clustering techniques using Spark

DESCRIPTION:
Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups. It is basically a collection of objects on the basis of similarity and dissimilarity between them.

Here, we will be implementing two different types of clustering techniques:

1. **Density-Based Spatial Clustering Of Applications With Noise (DBSCAN):** Clusters are dense regions in the data space, separated by regions of the lower density of points. The *DBSCAN algorithm* is based on this intuitive notion of "clusters" and "noise". The key idea is that for each point of a cluster, the neighborhood of a given radius has to contain at least a minimum number of points.
2. K-means Clustering: K-means is a clustering algorithm that groups data points into K distinct clusters based on their similarity. It is an unsupervised learning technique that is widely used in data mining, machine learning, and pattern recognition. The algorithm works by iteratively assigning data points to a cluster based on their distance from the cluster's centroid and then recomputing the centroid of each cluster. The process continues until the clusters' centroids converge or a maximum number of iterations is reached.

CODE AND OUTPUT:

pip install pyspark

```
Collecting pyspark
  Downloading pyspark-3.5.0.tar.gz (316.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 316.9/316.9 MB 2.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.0-py2.py3-none-any.whl size=317425344 sha256=0d082275199321030f85ad035adfab0a04dd39900257fc84d75a2af64f514b28
  Stored in directory: /root/.cache/pip/wheels/41/4e/10/c2cf2467f71c678cfc8a6b9ac9241e5e44a01940da8fbb17fc
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.0
```

1. DBSCAN Clustering Technique:

#DBSCAN
import pyspark
from pyspark.sql import SparkSession
spark=SparkSession.builder.appName('testmodel').getOrCreate()
df=spark.read.csv('irisC.csv',inferSchema=True,header=True)
df.show(10)

```
+------------+-----------+-----------+
|Sepal Length|Sepal Width|    Species|
+------------+-----------+-----------+
|         5.1|        3.5|Iris-setosa|
|         4.9|        3.0|Iris-setosa|
|         4.7|        3.2|Iris-setosa|
|         4.6|        3.1|Iris-setosa|
|         5.0|        3.6|Iris-setosa|
|         5.4|        3.9|Iris-setosa|
|         4.6|        3.4|Iris-setosa|
|         5.0|        3.4|Iris-setosa|
|         4.4|        2.9|Iris-setosa|
|         4.9|        3.1|Iris-setosa|
+------------+-----------+-----------+
only showing top 10 rows
```

df.printSchema()

```
root
 |-- Sepal Length: double (nullable = true)
 |-- Sepal Width: double (nullable = true)
 |-- Species: string (nullable = true)
```

from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
assembler=VectorAssembler(inputCols=['Sepal Length','Sepal Width'],outputCol='features')
output=assembler.transform(df)
output.select('features').show(5)

```
+---------+
| features|
+---------+
|[5.1,3.5]|
|[4.9,3.0]|
|[4.7,3.2]|
|[4.6,3.1]|
|[5.0,3.6]|
+---------+
only showing top 5 rows
```

final_data=output.select('features','Species')
train_data,test_data=final_data.randomSplit([0.7,0.3])
train_data.describe().show()

```
+-------+--------------+
|summary|       Species|
+-------+--------------+
|  count|           104|
|   mean|          NULL|
| stddev|          NULL|
|    min|   Iris-setosa|
|    max|Iris-virginica|
+-------+--------------+
```

import numpy as np
np.array(final_data.select('features'))

```
array(DataFrame[features: vector], dtype=object)
```

pandas_df = final_data.toPandas()
pandas_df.head()

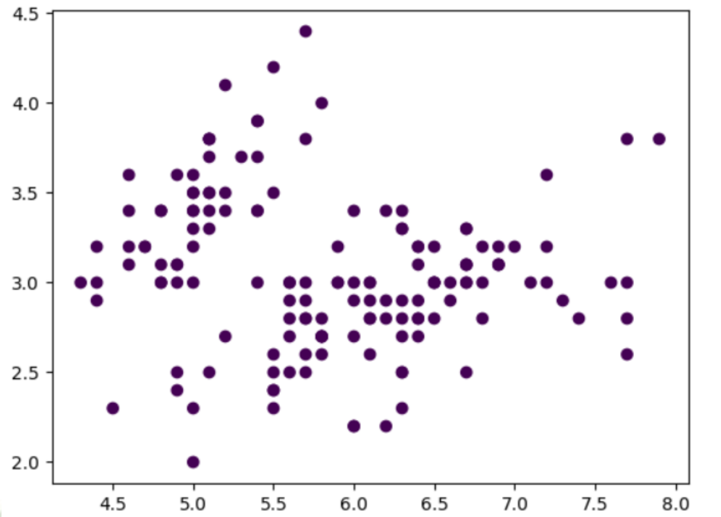
	features	Species
0	[5.1, 3.5]	Iris-setosa
1	[4.9, 3.0]	Iris-setosa
2	[4.7, 3.2]	Iris-setosa
3	[4.6, 3.1]	Iris-setosa
4	[5.0, 3.6]	Iris-setosa

from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.6, min_samples=3)
dbscan_labels = dbscan.fit_predict(pandas_df['features'].tolist())
print(dbscan_labels)

```
array([[5.1],
       [4.9],
       [4.7],
       [4.6],
       [5. ],
       [5.4],
       [4.6],
       [5. ],
       [4.4],
       [4.9],
       [5.4],
       [4.8],
       [4.8],
       [4.3],
       [5.8],
       [5.7],
       [5.4],
       [5.1],
       [5.7],
       [5.1],
       [5.4],
       [5.1],
       [4.6],
       [5.1],
       [4.8],
       [5. ],
       [5. ],
       [5.2],
       [5.2],
```

import matplotlib.pyplot as plt

plt.scatter(np.array(X),np.array(Y),c=dbscan_labels)
plt.show()



spark.stop()

1. K-Means Clustering Technique:
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('cluster').getOrCreate()
print('Spark Version: {}'.format(spark.version))

```
Spark Version: 3.5.0
```

#Loading the data
dataset = spark.read.csv("seeds_dataset.csv",header=True,inferSchema=True)
#show the data in the above file using the below command
dataset.show(5)

Area	Perimeter	Compactness	Length_of_kernel	Width_of_kernel	Asymmetry_coefficient	Length_of_kernel_grove
15.26	14.84	0.871	5.763	3.312	2.221	5.22
14.88	14.57	0.8811	5.554	3.333	1.018	4.956
14.29	14.09	0.905	5.291	3.337	2.699	4.825
13.84	13.94	0.8955	5.324	3.379	2.259	4.805
16.14	14.99	0.9034	5.658	3.562	1.355	5.175

only showing top 5 rows

#Print schema
dataset.printSchema()

```
root
 |-- Area: double (nullable = true)
 |-- Perimeter: double (nullable = true)
 |-- Compactness: double (nullable = true)
 |-- Length_of_kernel: double (nullable = true)
 |-- Width_of_kernel: double (nullable = true)
 |-- Asymmetry_coefficient: double (nullable = true)
 |-- Length_of_kernel_grove: double (nullable = true)
```

```
from pyspark.ml.feature import VectorAssembler
vec_assembler = VectorAssembler(inputCols = dataset.columns, outputCol='features')
final_data = vec_assembler.transform(dataset)
final_data.select('features').show(5)
```

```
+--------------------+
|            features|
+--------------------+
|[15.26,14.84,0.87...|
|[14.88,14.57,0.88...|
|[14.29,14.09,0.90...|
|[13.84,13.94,0.89...|
|[16.14,14.99,0.90...|
+--------------------+
only showing top 5 rows
```

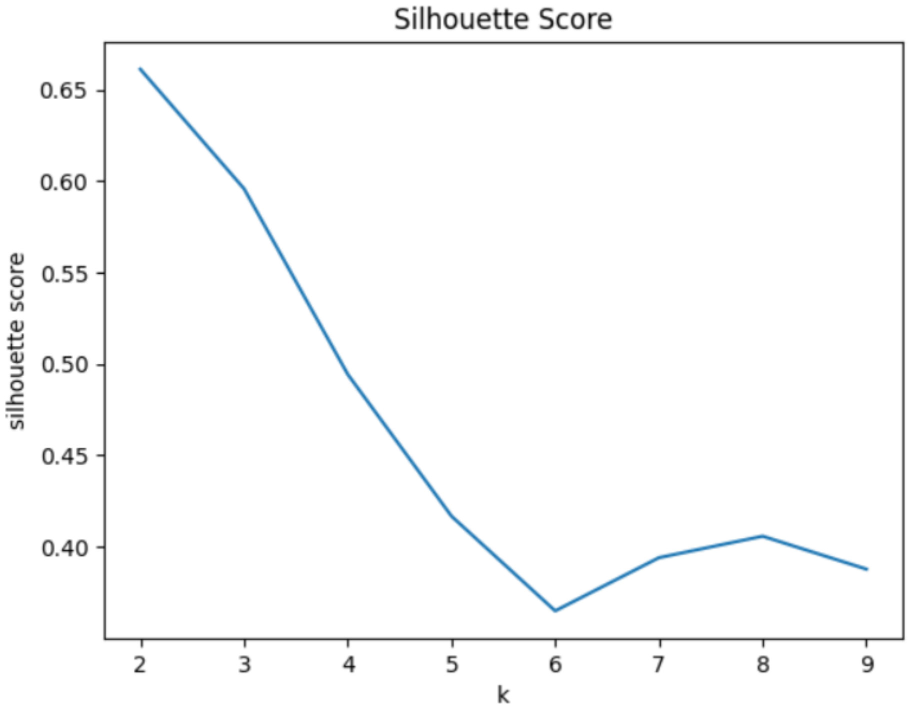
```
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=False)
# Compute summary statistics by fitting the StandardScaler
scalerModel = scaler.fit(final_data)
# Normalize each feature to have unit standard deviation.
final_data = scalerModel.transform(final_data)
final_data.select('scaledFeatures').show(5)
```

```
+--------------------+
|      scaledFeatures|
+--------------------+
|[5.24452795332028...|
|[5.11393027165175...|
|[4.91116018695588...|
|[4.75650503761158...|
|[5.54696468981581...|
+--------------------+
only showing top 5 rows
```

```
#Importing the model
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
silhouette_score=[]
evaluator = ClusteringEvaluator(predictionCol='prediction', featuresCol='scaledFeatures', metricName='silhouette', distanceMeasure='squaredEuclidean')
for i in range(2,10):
    kmeans=KMeans(featuresCol='scaledFeatures', k=i)
    model=kmeans.fit(final_data)
    predictions=model.transform(final_data)
    score=evaluator.evaluate(predictions)
    silhouette_score.append(score)
    print('Silhouette Score for k =',i,'is',score)
```

```
Silhouette Score for k = 2 is 0.6613125038335929
Silhouette Score for k = 3 is 0.5959078263451633
Silhouette Score for k = 4 is 0.4943210687863144
Silhouette Score for k = 5 is 0.4166976682907412
Silhouette Score for k = 6 is 0.3648649810130078
Silhouette Score for k = 7 is 0.39397743262544
Silhouette Score for k = 8 is 0.40573744412356627
Silhouette Score for k = 9 is 0.3877256432563701
```

#Visualizing the silhouette scores in a plot
import matplotlib.pyplot as plt
plt.plot(range(2,10),silhouette_score)
plt.xlabel('k')
plt.ylabel('silhouette score')
plt.title('Silhouette Score')
plt.show()



 # Trains a k-means model.
kmeans = KMeans(featuresCol='scaledFeatures',k=3)
model = kmeans.fit(final_data)
predictions = model.transform(final_data)
 # Printing cluster centers
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:

print(center)

```
Cluster Centers:
[ 4.91589737 10.9321157  37.2641905  12.39722305  8.58688868  1.77370551
 10.37323607]
[ 6.3407095  12.39263108 37.41143125 13.92892299  9.77251635  2.42396744
 12.28547936]
[ 4.06818854 10.13938448 35.87110297 11.81191124  7.52564313  3.24586152
 10.40780927]
```

predictions.select('prediction').show(5)

```
+----------+
|prediction|
+----------+
|         0|
|         0|
|         0|
|         0|
|         0|
+----------+
only showing top 5 rows
```

#End Session
spark.stop()