BDA Mini project CA2

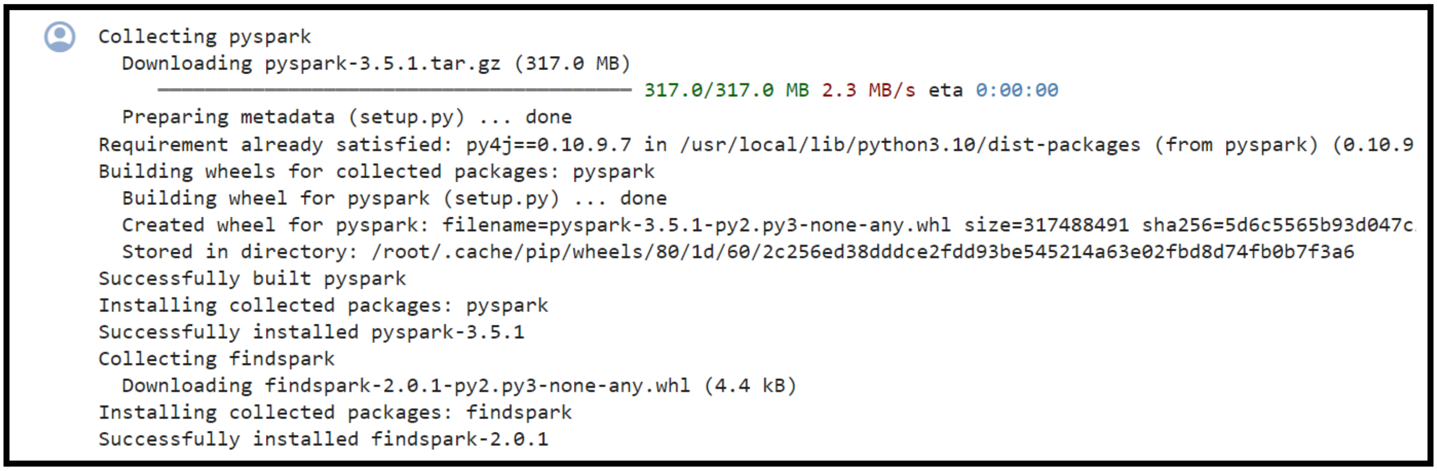
Write MapReduce/Spark Program to perform

1. **Matrix Vector Multiplication**

Code:

Locally installing Spark:

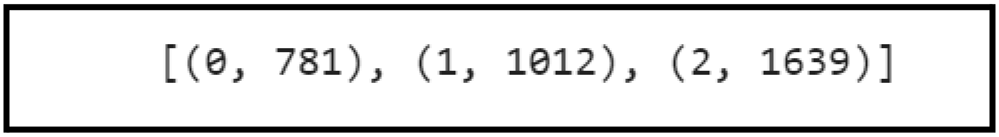
```
%pip install pyspark
%pip install findspark

import findspark
findspark.init()
from pyspark.sql import
SparkSession

spark = SparkSession.builder \
        .master('local[*]') \
        .appName('Basics') \
        .getOrCreate()
```



```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 2.3 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
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=317488491 sha256=5d6c5565b93d047c
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
Collecting findspark
  Downloading findspark-2.0.1-py2.py3-none-any.whl (4.4 kB)
Installing collected packages: findspark
Successfully installed findspark-2.0.1
```

```
from pyspark.sql import SparkSession

# Create SparkSession
spark = SparkSession.builder \
    .appName("MatrixVectorMultiplication") \
    .getOrCreate()

# Input matrix and vector
matrix = [
    [14, 15, 9],
    [24, 25, 6],
    [31, 41, 12]
]

vector = [11, 22, 33]

# Define the multiplication function
def multiply(row):
    matrix_row, values = row
    result = sum(value * vector[i] for i, value in enumerate(values))
    return (matrix_row, result)

# Parallelize the matrix
matrix_rdd = spark.sparkContext.parallelize(enumerate(matrix))

# Perform matrix-vector multiplication
result = matrix_rdd.map(multiply)

# Collect the result and print
print(result.collect())

# Stop the Spark Session
spark.stop()
```

Output:



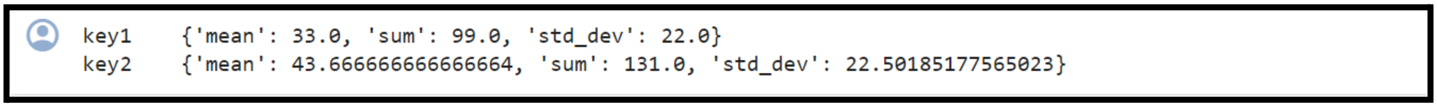
```
[(0, 781), (1, 1012), (2, 1639)]
```

2. Aggregations - Mean, Sum, Std Deviation

Code:

```
from pyspark import SparkContext
from math import sqrt

# Dummy input data
input_data = [
    'key1\t11',
    'key2\t21',
    'key1\t33',
    'key2\t44',
    'key1\t55',
    'key2\t66',
]

def map_func(line):
    key, value = line.split('\t')
    return key, float(value)

def reduce_func(data):
    values = list(data)  # Convert data to list for clarity
    mean_val = sum(values) / len(values)
    sum_val = sum(values)
    if len(values) > 1:  # Check if there are more than one value for calculation
        std_dev_val = sqrt(sum((x - mean_val) ** 2 for x in values) / (len(values) - 1))
    else:
        std_dev_val = 0
    return {
        'mean': mean_val,
        'sum': sum_val,
        'std_dev': std_dev_val
    }

if __name__ == '__main__':
    sc = SparkContext('local', 'AggregationSpark')
    try:
        lines = sc.parallelize(input_data)
        mapped = lines.map(map_func)
        grouped = mapped.groupByKey()
        result = grouped.mapValues(list).mapValues(reduce_func)
        output = result.collect()
        for key, value in output:
            print(f'{key}\t{value}')
    finally:
        sc.stop()
```

Output:



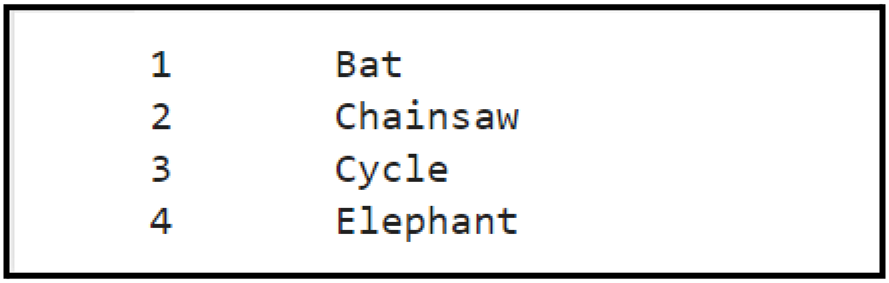
```
key1    {'mean': 33.0, 'sum': 99.0, 'std_dev': 22.0}
key2    {'mean': 43.666666666666664, 'sum': 131.0, 'std_dev': 22.50185177565023}
```

3. Sort the data

Code:

```
from pyspark.sql import SparkSession

# Create a Spark session
spark = SparkSession.builder \
    .appName("SortData") \
    .getOrCreate()

# Define dummy input data
dummy_data = [
    "3\tCycle",
    "1\tBat",
    "2\tChainsaw",
    "4\tElephant"
]

# Create RDD from dummy data
data_rdd = spark.sparkContext.parallelize(dummy_data)

# Sort the data based on the first column
sorted_data = data_rdd.sortBy(lambda x: x.split('\t')[0])

# Collect and print the sorted data
sorted_results = sorted_data.collect()
for result in sorted_results:
    print(result)

# Stop the Spark session
spark.stop()
```

Output:



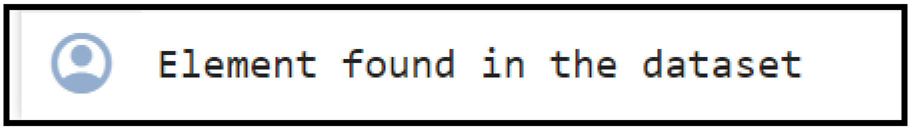
```
1       Bat
2       Chainsaw
3       Cycle
4       Elephant
```

4. Search a data element

Code:

```
from pyspark import SparkContext, SparkConf

# Create a Spark context
conf = SparkConf().setAppName("SearchElement").setMaster("local")
sc = SparkContext(conf=conf)

# Define the data to be searched
data = [210,310,456,588,329,514]

# Parallelize the data into RDD (Resilient Distributed Dataset)
rdd = sc.parallelize(data)

# Define the search function
def search_element(element):
   return element == 588  # Change the search element as needed

# Map function to search for the element in the dataset
result = rdd.map(search_element)

# Collect the results
search_result = result.collect()

# Print the search result
if True in search_result:
   print("Element found in the dataset")
else:
   print("Element not found in the dataset")

# Stop the Spark context
sc.stop()
```

Output:



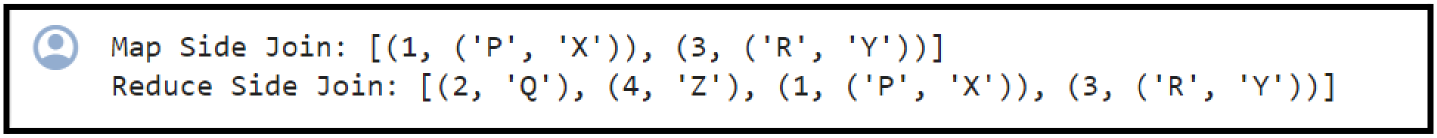
```
Element found in the dataset
```

5. Joins - Map Side and Reduce Side

Code:

```
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Joins")

# Create RDDs for left and right datasets
left_data = sc.parallelize([(1, "P"), (2, "Q"), (3, "R")])
right_data = sc.parallelize([(1, "X"), (3, "Y"), (4, "Z")])

# Perform map-side join
map_join = left_data.join(right_data)

# Perform reduce-side join
reduce_join = left_data.union(right_data).reduceByKey(lambda x, y: (x, y))

# Print the results
print("Map Side Join:", map_join.collect())
print("Reduce Side Join:", reduce_join.collect())

# Stop SparkContext
sc.stop()
```

Output:



```
Map Side Join: [(1, ('P', 'X')), (3, ('R', 'Y'))]
Reduce Side Join: [(2, 'Q'), (4, 'Z'), (1, ('P', 'X')), (3, ('R', 'Y'))]
```