Jay Shah/61
Noel Dason/11
Anish Masand/39

Group 61: 61,11,39

BDA Mini project CA2

Write MapReduce/Spark Program to perform

1. **Matrix Vector Multiplication**
   Code:

Locally installing Spark:

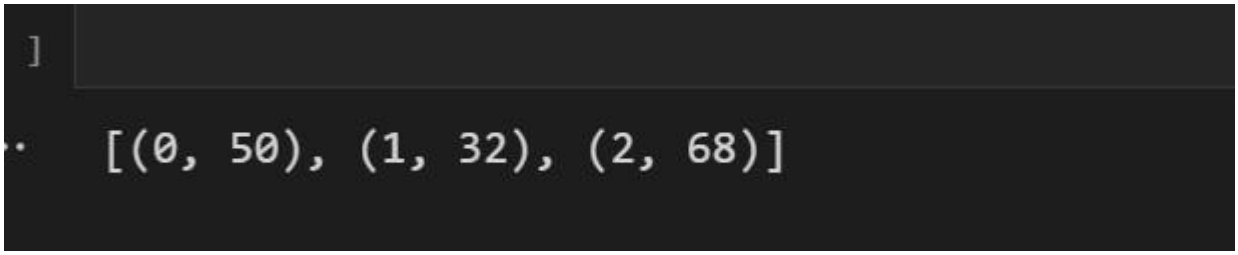
```
%pip install pyspark
%pip install findspark

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

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

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-packages (3.5.1)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Requirement already satisfied: findspark in /usr/local/lib/python3.10/dist-packages (2.0.1)
```

```
from pyspark.sql import
SparkSession

# Create SparkSession spark =
SparkSession.builder \

.appName("MatrixVectorMultiplicati
on") \
  .getOrCreate()

# Input matrix and vector
matrix = [
  [7, 8, 9],
  [4, 5, 6],
```

```
    [10, 11, 12]
]
vector = [1, 2, 3]

# 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(enum
erate(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:



2. **Aggregations - Mean, Sum, Std Deviation**

```
from pyspark import SparkContext
from math import sqrt
```

Code:

```
# Dummy input data
input_data = [
    'key1\t10',
    'key2\t20',
    'key1\t30',
    'key2\t40',
    'key1\t50',
    'key2\t60',
]

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': 30.0, 'sum': 90.0, 'std_dev': 20.0}
key2    {'mean': 40.0, 'sum': 120.0, 'std_dev': 20.0}
```

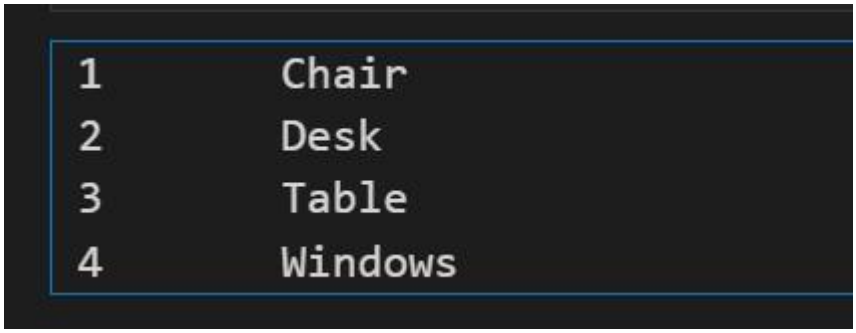
Code:

3. Sort the data

```
from pyspark.sql import
SparkSession

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

# Define dummy input data
dummy_data = [
    "3\tTable",
    "1\tChair",
    "2\tDesk",
    "4\tWindows"
]

# 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:



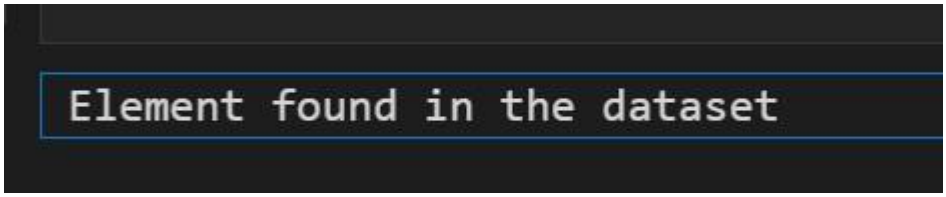
Code:

4. **Search a data element**

```
from pyspark import
SparkContext, SparkConf

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

# Define the data to be
searched data = [1, 2, 3, 4,
5, 6, 7, 8, 9, 10]

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

# Define the search
function def
search_element(element):
return element == 10 #
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:



5. **Joins - Map Side and Reduce Side**

```
from pyspark import
SparkContext

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

Code:

```
# Create RDDs for
left and right datasets
left_data =
sc.parallelize([(1,
"A"), (2, "B"), (3,
"C")]) right_data
=
sc.parallelize([(1,
"P"), (3, "Q"), (4,
"R")])

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

# Perform reduce-side
join reduce_join =
left_data.union(right_
data).reduceByKey(la
mbda 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, ('A', 'P')), (3, ('C', 'Q'))]
Reduce Side Join: [(2, 'B'), (4, 'R'), (1, ('A', 'P')), (3, ('C', 'Q'))]
```