ICP8 Report

1.&2.

```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Natural Numbers")

# Create a list of the first 15 natural numbers
numbers = list(range(1, 16))

# Parallelize the list to create an RDD
rdd = sc.parallelize(numbers)

#Produce RDD with List of first 15 natural numbers
print("Elements in the RDD:", rdd.collect())

# Show the number of partitions in the RDD
print("Number of partitions:", rdd.getNumPartitions())

# Stop the SparkContext
sc.stop()
```

```
Elements in the RDD: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
Number of partitions: 1
```

3.

```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Natural Numbers")

# Create a list of the first 15 natural numbers
numbers = list(range(1, 16))

# Parallelize the list to create an RDD
rdd = sc.parallelize(numbers)

# Get the first element of the RDD
first_element = rdd.first()

# Show the first element
print("First element in the RDD:", first_element)

# Stop the SparkContext
sc.stop()
```

```
First element in the RDD: 1
```

4.

```
[10] from pyspark import SparkContext

     # Initialize SparkContext
     sc = SparkContext("local", "Filter Even Numbers")

     # Create a list of the first 15 natural numbers
     numbers = list(range(1, 16))

     # Parallelize the list to create an RDD
     rdd = sc.parallelize(numbers)

     # Use the filter transformation to select only even numbers
     even_rdd = rdd.filter(lambda x: x % 2 == 0)

     # Collect and print the filtered RDD to see the result
     print("Even numbers in the RDD:", even_rdd.collect())

     # Stop the SparkContext
     sc.stop()
```

```
Even numbers in the RDD: [2, 4, 6, 8, 10, 12, 14]
```

5.

```
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Square Each Element")

# Create a list of the first 15 natural numbers
numbers = list(range(1, 16))

# Parallelize the list to create an RDD
rdd = sc.parallelize(numbers)

# Use the map transformation to square each element
squared_rdd = rdd.map(lambda x: x ** 2)

# Collect and print the transformed RDD
print("Squared numbers in the RDD:", squared_rdd.collect())

# Stop the SparkContext
sc.stop()
```

```
Squared numbers in the RDD: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225]
```

6.

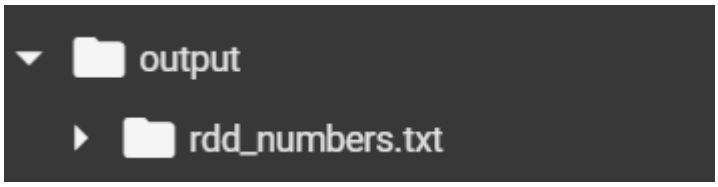
```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Reduce Example")

# Create a list of the first 15 natural numbers
numbers = list(range(1, 16))

# Parallelize the list to create an RDD
rdd = sc.parallelize(numbers)

# Use the reduce action to calculate the sum of all elements
sum_result = rdd.reduce(lambda x, y: x + y)

# Print the result
print("Sum of all elements in the RDD:", sum_result)

# Stop the SparkContext
sc.stop()
```

```
Sum of all elements in the RDD: 120
```

7.

```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Save RDD as Text File")

# Create a list of the first 15 natural numbers
numbers = list(range(1, 16))

# Parallelize the list to create an RDD
rdd = sc.parallelize(numbers)

# Save the RDD as a text file
rdd.saveAsTextFile("output/rdd_numbers.txt")

# Stop the SparkContext
sc.stop()
```

Output:-



8.

```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Union Example")

# Create two lists of numbers
list1 = [1, 2, 3, 4, 5]
list2 = [6, 7, 8, 9, 10]

# Parallelize the lists to create RDDs
rdd1 = sc.parallelize(list1)
rdd2 = sc.parallelize(list2)

# Use the union transformation to combine the two RDDs
combined_rdd = rdd1.union(rdd2)

# Collect and print the combined RDD
print("Combined RDD:", combined_rdd.collect())

# Stop the SparkContext
sc.stop()
```

```
Combined RDD: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

9.

```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Cartesian Example")

# Create two lists of numbers
list1 = [1, 2]
list2 = [3, 4]

# Parallelize the lists to create RDDs
rdd1 = sc.parallelize(list1)
rdd2 = sc.parallelize(list2)

# Use the cartesian transformation to get all ordered pairs
cartesian_rdd = rdd1.cartesian(rdd2)

# Collect and print the Cartesian product RDD
print("Cartesian product RDD:", cartesian_rdd.collect())

# Stop the SparkContext
sc.stop()
```

```
Cartesian product RDD: [(1, 3), (1, 4), (2, 3), (2, 4)]
```

10.

```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Dictionary RDD")

# Create a dictionary
data_dict = {"a": 1, "b": 2, "c": 3, "d": 4}

# Convert the dictionary to a list of tuples (key-value pairs)
data_list = list(data_dict.items())

# Parallelize the list to create an RDD
rdd = sc.parallelize(data_list)

# Collect and print the RDD
print("RDD with dictionary data:", rdd.collect())

# Stop the SparkContext
sc.stop()
```

```
RDD with dictionary data: [('a', 1), ('b', 2), ('c', 3), ('d', 4)]
```

11.

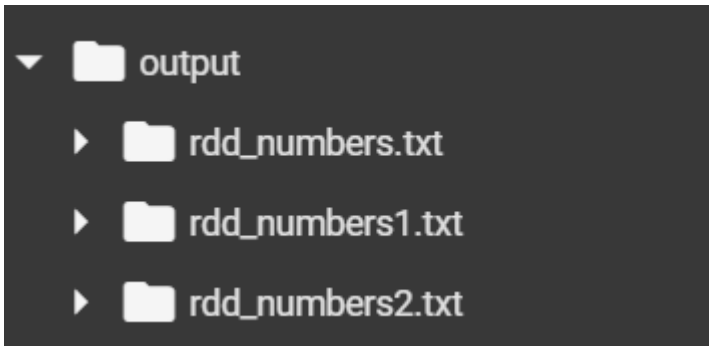
```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Count Unique Values")

# Create a list of numbers (with some repeated values)
numbers = [1, 2, 3, 2, 1, 1, 4, 5, 2, 3, 5]

# Parallelize the list to create an RDD
rdd = sc.parallelize(numbers)

# Use map transformation to create key-value pairs (value, 1)
rdd_pairs = rdd.map(lambda x: (x, 1))

# Use reduceByKey to aggregate counts for each unique value
counted_rdd = rdd_pairs.reduceByKey(lambda x, y: x + y)

# Collect and print the results
print("Unique values and their counts:", counted_rdd.collect())

# Stop the SparkContext
sc.stop()
```

```
Unique values and their counts: [(1, 3), (2, 3), (3, 2), (4, 1), (5, 2)]
```

12.

```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Save RDD Multiple Text Files")

# Create a list of the first 15 natural numbers
numbers1 = list(range(1, 16))
numbers2 = list(range(16, 31))

# Parallelize the list to create an RDD
rdd1 = sc.parallelize(numbers1)
rdd2 = sc.parallelize(numbers2)

# Save the RDD as a text file
rdd1.saveAsTextFile("output/rdd_numbers1.txt")
rdd2.saveAsTextFile("output/rdd_numbers2.txt")

# Stop the SparkContext
sc.stop()
```

Output:-




```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Combine Multiple Text Files")

# Load all .txt files from the "input_files" directory into a single RDD
rdd = sc.textFile("output/*.txt")

# Collect and print the RDD contents
print("Combined RDD from multiple files:")
for line in rdd.collect():
    print(line)

# Stop the SparkContext
sc.stop()
```

```
Combined RDD from multiple files:
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
```

13.

```python
from pyspark import SparkContext

# Initialize SparkContext
sc = SparkContext("local", "Inspect First 5 Lines of RDD")

# Create an example RDD (e.g., a list of lines)
rdd = sc.parallelize(["Line 1", "Line 2", "Line 3", "Line 4", "Line 5", "Line 6", "Line 7"])

# Use the take() action to get the first 5 lines
first_5_lines = rdd.take(5)

# Print the first 5 lines
print("First 5 lines of the RDD:")
for line in first_5_lines:
    print(line)

# Stop the SparkContext
sc.stop()
```

```
First 5 lines of the RDD:
Line 1
Line 2
Line 3
Line 4
Line 5
```

14.

```python
from pyspark.sql import SparkSession
from pyspark.sql import Row

# Create a Spark session
spark = SparkSession.builder.appName("Spark DataFrame Example").getOrCreate()

# Example data as a list of tuples
data = [("Alice", 25, "New York"),
        ("Bob", 30, "Los Angeles"),
        ("Charlie", 35, "Chicago")]

# Create a DataFrame from the data
df = spark.createDataFrame(data, ["Name", "Age", "City"])

# Show the DataFrame
df.show()
```

```
+-------+---+-----------+
|   Name|Age|       City|
+-------+---+-----------+
|  Alice| 25|   New York|
|    Bob| 30|Los Angeles|
|Charlie| 35|    Chicago|
+-------+---+-----------+
```

15.

```
#1 RDD
  #The most basic form of data in Spark
  #No structure, so you have full control, but it's harder to work with and slower
#2 DataFrame
  #A more structured form of data with column names and types
  #Faster than RDDs because it's optimized by Spark's engine
#3 Dataset
  #Like a DataFrame but with type safety (only in Scala/Java)
  #Both performance optimization and strong datatype
```

Github link:- https://github.com/Ksahitha/BDA.git