ASSIGNMENT ON RDD

Dataset:

You will be working with the following sales data. Each entry in the dataset represents a product and its corresponding sales amount.

```
sales_data = [
    ("ProductA", 100),
    ("ProductB", 150),
    ("ProductA", 200),
    ("ProductC", 300),
    ("ProductB", 250),
    ("ProductC", 100)
]
regional_sales_data = [
    ("ProductA", 50),
    ("ProductC", 150)
]
```

Step 1: Initialize Spark Context

1. **Initialize SparkSession and SparkContext:**

   - Create a Spark session in PySpark and use the `spark.sparkContext` to create an RDD from the provided data.

Program:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
    .appName("Key-Value Pair RDD Example") \
    .getOrCreate()
sc = spark.sparkContext
```

Step 2: Create and Explore the RDD

2. **Task 1: Create an RDD from the Sales Data**

   - Create an RDD from the `sales_data` list provided above.

   - Print the first few elements of the RDD.

Program:

```
sales_data = [
    ("ProductA", 100),
    ("ProductB", 150),
    ("ProductA", 200),
    ("ProductC", 300),
    ("ProductB", 250),
    ("ProductC", 100)
]
sales_rdd = sc.parallelize(sales_data)
print("Sales RDD:", sales_rdd.collect())
```

Step 3: Grouping and Aggregating Data

3. **Task 2: Group Data by Product Name**

   - Group the sales data by product name using `groupByKey()`.

   - Print the grouped data to understand its structure.

Program:

```
grouped_sales_rdd = sales_rdd.groupByKey().mapValues(list)
print("Grouped Sales RDD:", grouped_sales_rdd.collect())
```

4. **Task 3: Calculate Total Sales by Product**

   - Use `reduceByKey()` to calculate the total sales for each product.

   - Print the total sales for each product.

Program:

total_sales_rdd = sales_rdd.reduceByKey(lambda x, y: x + y)

print("Total Sales by Product:", total_sales_rdd.collect())

5. **Task 4: Sort Products by Total Sales**

   - Sort the products by their total sales in descending order.

   - Print the sorted list of products along with their sales amounts.

Program:

sorted_sales_rdd = total_sales_rdd.sortBy(lambda x: x[1], ascending=False)

print("Products Sorted by Total Sales:", sorted_sales_rdd.collect())

Step 4: Additional Transformations

6. **Task 5: Filter Products with High Sales**

   - Filter the products that have total sales greater than 200.

   - Print the products that meet this condition.

Program:

high_sales_rdd = total_sales_rdd.filter(lambda x: x[1] > 200)

print("Products with Sales > 200:", high_sales_rdd.collect())

7. **Task 6: Combine Regional Sales Data**

   - Create another RDD from the `regional_sales_data` list.

   - Combine this RDD with the original sales RDD using `union()`.

   - Calculate the new total sales for each product after combining the datasets.

   - Print the combined sales data.

Program:

Regional sales data

regional_sales_data = [

    ("ProductA", 50),

```
    ("ProductC", 150)
]
# Create RDD for regional sales
regional_sales_rdd = sc.parallelize(regional_sales_data)

# Combine RDDs
combined_sales_rdd = sales_rdd.union(regional_sales_rdd)

# Recalculate total sales
new_total_sales_rdd = combined_sales_rdd.reduceByKey(lambda x, y: x + y)

# Print combined sales data
print("Combined Total Sales by Product:", new_total_sales_rdd.collect())
```

Step 5: Perform Actions on the RDD

8. **Task 7: Count the Number of Distinct Products**
   - Count the number of distinct products in the RDD.
   - Print the count of distinct products.

Program:

```
distinct_products_count = sales_rdd.keys().distinct().count()
print("Number of Distinct Products:", distinct_products_count)
```

9. **Task 8: Identify the Product with Maximum Sales**
   - Find the product with the maximum total sales using `reduce()`.
   - Print the product name and its total sales amount.

Program:

```
distinct_products_count = sales_rdd.keys().distinct().count()
```

print("Number of Distinct Products:", distinct_products_count)

Challenge Task: Calculate the Average Sales per Product

10. **Challenge Task:**
    - Calculate the average sales amount per product using the key-value pair RDD.
    - Print the average sales for each product.

Program:

```
# Calculate total sales and count for each product
sales_count_rdd = sales_rdd.mapValues(lambda x: (x, 1))
sales_sum_count_rdd = sales_count_rdd.reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))

# Calculate average sales
average_sales_rdd = sales_sum_count_rdd.mapValues(lambda x: x[0] / x[1])

# Print average sales for each product
print("Average Sales per Product:", average_sales_rdd.collect())
```