# ---

# ### \*\*Exercise: Working with Key-Value Pair RDDs in PySpark\*\*

# sales\_data = [

("ProductA", 100),

("ProductB", 150),

("ProductA", 200),

("ProductC", 300),

("ProductB", 250),

("ProductC", 100)

]

# You will also be working with an additional dataset for regional sales:

#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.**

from pyspark.sql import SparkSession

spark=SparkSession.builder \

.appName("RDD") \

.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.**

sales\_data = [

("ProductA", 100),

("ProductB", 150),

("ProductA", 200),

("ProductC", 300),

("ProductB", 250),

("ProductC", 100)

]

salesdata\_rdd=sc.parallelize(sales\_data)

**# - Print the first few elements of the RDD.**

print(salesdata\_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()`.**

groupbykey=salesdata\_rdd.groupByKey()

**# - Print the grouped data to understand its structure.**

groups = [(x, list(y)) for x, y in groupbykey.collect()]

print("Grouping data")

print(groups)

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

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

totalsale =salesdata\_rdd.reduceByKey(lambda x,y : x+y)

**# - Print the total sales for each product.**

print(totalsale.collect())

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

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

sorting\_by\_desc=totalsale.sortBy(lambda x: x[1], ascending=False)

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

print(sorting\_by\_desc.collect())

**# ### \*\*Step 4: Additional Transformations\*\***

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

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

higher\_sales= totalsale.filter(lambda x : x[1]>200)

**# - Print the products that meet this condition.**

print("Products having higher sales:",higher\_sales.collect())

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

**# - Create another RDD from the `regional\_sales\_data` list.**

regional\_sales\_data = [

("ProductA", 50),

("ProductC", 150)

]

regional\_rdd= sc.parallelize(regional\_sales\_data)

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

combining\_rdd = salesdata\_rdd.union(regional\_rdd)

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

new\_sale = combining\_rdd.reduceByKey(lambda x,y: x+y)

**# - Print the combined sales data.**

print(new\_sale.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.**

perform\_action= new\_sale.count()

**# - Print the count of distinct products.**

print(" The count is ",perform\_action)

**# 9. \*\*Task 8: Identify the Product with Maximum Sales\*\***

**# - Find the product with the maximum total sales using `reduce()`.**

max\_sale=new\_sale.reduce(lambda x, y: x if x[1] > y[1] else y)

**# - Print the product name and its total sales amount.**

print(max\_sale)

**# ### \*\*Challenge Task: Calculate the Average Sales per Product\*\***

**# 10. \*\*Challenge Task:\*\***

**# - Calculate the average sales amount per product using the key-value pair RDD.**

average\_sale = new\_sale.mapValues(lambda x: (x,1)).reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))

avg\_sale = average\_sale.mapValues(lambda x: x[0] / x[1])

# Print the average sales for each product

print(avg\_sale.collect())