Name: Perez, Junmar A. Lab No. 2

Git Repo/Colab Link: https://github.com/junmsr/Perez_Elec2.git Date: 03/20/2025

Objective

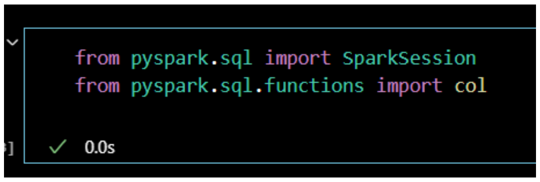
Load a dataset and divide it into smaller sections using two different methods. This Activity will apply basic data operations like summarizing, sorting, and filtering.

Introduction

This lab explores how partitioning improves data processing in Spark. We will apply two partitioning methods and perform operations like summarizing, sorting, and filtering on a dataset.

Methodology

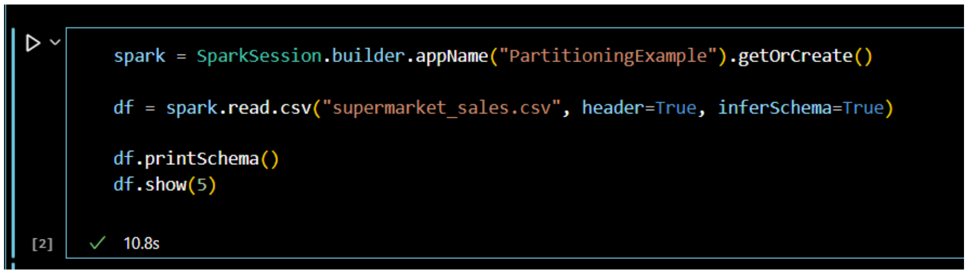
1. Initialize Spark Session



```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
```

```
✓ 0.0s
```

2. Load the dataset into Spark.



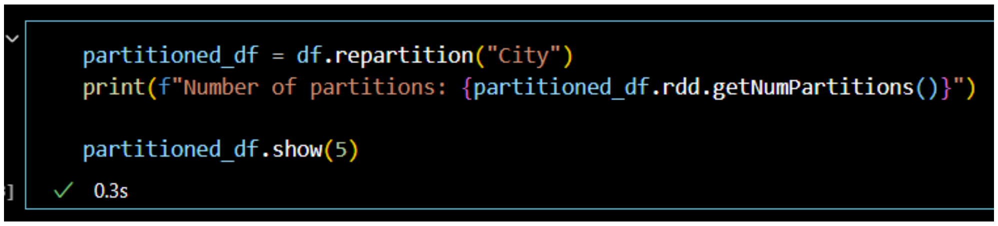
```
spark = SparkSession.builder.appName("PartitioningExample").getOrCreate()

df = spark.read.csv("supermarket_sales.csv", header=True, inferSchema=True)

df.printSchema()
df.show(5)
```

```
[2] ✓ 10.8s
```

3. Partitioning
   a. The dataset was divided into partitions based on different cities.

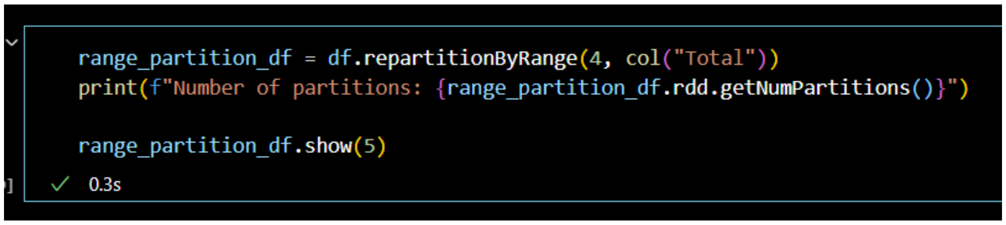


```
partitioned_df = df.repartition("City")
print(f"Number of partitions: {partitioned_df.rdd.getNumPartitions()}")

partitioned_df.show(5)
```

```
✓ 0.3s
```

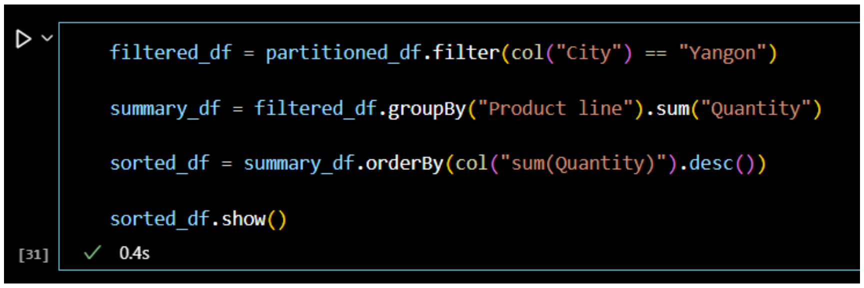
b. The dataset was repartitioned into 4 partitions based on transaction totals to balance the data distribution.



```
range_partition_df = df.repartitionByRange(4, col("Total"))
print(f"Number of partitions: {range_partition_df.rdd.getNumPartitions()}")

range_partition_df.show(5)
```
```
✓ 0.3s
```

4. Data Transformations
   a. Filter, summarize and sort.
      i. Extracted transactions specific to Yangon
      ii. Calculated total quantity sold per product line
      iii. Ordered the summarized data based on the highest quantity sold
      iv. Display the processed data



```
filtered_df = partitioned_df.filter(col("City") == "Yangon")

summary_df = filtered_df.groupBy("Product line").sum("Quantity")

sorted_df = summary_df.orderBy(col("sum(Quantity)").desc())

sorted_df.show()
```
```
[31] ✓ 0.4s
```

Results and Analysis

Partitioning by City made filtering faster, while Range Partitioning by Total Sales balanced the data, preventing overload. This improved query performance, making filtering, summarizing, and sorting more efficient.

Challenges and Solutions

Queries were slow because Spark scanned the entire dataset. City-based partitioning allowed Spark to focus only on relevant data, while Range Partitioning ensured even distribution. These strategies sped up sorting and summarizing.

Conclusion

Partitioning made Spark processing faster and more efficient. City-based partitioning improved filtering, and Range Partitioning balanced the workload, reducing query time and improving performance.