CAPSTONE PROJECT - 4

Computer Hardware Software Workshop
COCSC19

Submitted by:-
Kanishk Kumar (2021UCS1627)
Shubham Yadav (2021UCS1630)
Kashish Goel (2021UCS1633)
CSE Section - 2

Objective:-

Task 1: Explore RDD in spark
Task 2: In PySpark, create a program that reads a CSV file containing sales data, performs data cleaning by handling missing values and removing duplicates, calculates the total sales amount for each product, and finally, outputs the results to a new CSV file. Ensure to use transformations and actions in your PySpark script

1. Explore RDD in spark

Apache Spark is built around a distributed collection of immutable Java Virtual Machine (JVM) objects called Resilient Distributed Datasets (RDDs for short). As we are working with Python, it is important to note that the Python data is stored within these JVM objects. RDDs are calculated against, cached, and stored in-memory: a scheme that results in orders of magnitude faster computations compared to other traditional distributed frameworks like Apache Hadoop.Resilient Distributed Datasets), keeping the flexibility and extensibility of the Hadoop platform to perform a wide variety of calculations. RDDs apply and log transformations to the data in parallel, resulting in both increased speed and fault-tolerance. By Registering the transformations, RDDs provide data lineage - a form of an ancestry tree for each intermediate step in the form of a graph. This effect, guards the RDDs against data loss - if a partition of an RDD is lost it still has enough information to recreate that partition instead of simply depending on replication. RDDs represent an immutable, distributed collection of objects that can be processed in parallel across a cluster. Here's an exploration of RDDs in Spark:

- Immutable: RDDs are immutable, meaning once created, they cannot be changed. If you want to modify an RDD, you have to transform it into a new RDD.

- Distributed: RDDs are distributed across multiple nodes in a cluster, allowing parallel processing. Spark automatically distributes the data across the cluster, handling partitioning and distribution transparently to the user.

- Resilient: The "R" in RDD stands for "Resilient," which means that RDDs are fault-tolerant. Spark keeps track of the lineage of transformations applied to the base dataset, allowing it to reconstruct any lost partition due to node failures.

- Transformation and Action: RDDs support two types of operations: transformations and actions. Transformations create a new RDD from an existing one (e.g., map, filter, reduceByKey), while actions perform computation on the RDD and return results to the driver program (e.g., collect, count, saveAsTextFile).

- Lazy Evaluation: Transformations on RDDs are lazily evaluated. This means that Spark doesn't compute the result right away when you apply a transformation. Instead, it builds up a Directed Acyclic Graph (DAG) of the transformations and waits until an action is called to execute them. Lazy evaluation allows Spark to optimize the execution plan and minimize unnecessary computation.

- Fault Tolerance: RDDs achieve fault tolerance through lineage information. When a partition of an RDD is lost due to a node failure, Spark can recompute the lost partition by applying the transformations from the original dataset. This resilience to failures is one of the key features of Spark, making it suitable for large-scale data processing.

- Persistence: RDDs can be persisted in memory to avoid recomputation, especially for iterative algorithms or when the same dataset is reused multiple times. Spark provides various levels of persistence, allowing you to choose between storing RDDs in memory, on disk, or both.

- Supported Data Sources: RDDs can be created from various data sources including HDFS, HBase, Cassandra, JSON, CSV, etc. Spark provides built-in support for reading and writing data from/to these sources, making it easy to integrate with existing data ecosystems.

Two common ways to create RDDs (Resilient Distributed Datasets) in Apache Spark:

1. Parallelizing a Collection:This method involves creating an RDD from an existing collection in your driver program by parallelizing it. It's suitable when you have a relatively small dataset that fits comfortably in memory.

   ```
   from pyspark import SparkContext
   # Create a SparkContext
   sc = SparkContext("local", "RDD Example")

   # Create a Python list
   ```

```
data = [1, 2, 3, 4, 5]
# Parallelize the list to create an RDD
rdd = sc.parallelize(data)
```

sc.parallelize() is used to distribute the elements of the Python list data across the Spark cluster, creating an RDD named rdd.

2. Loading from External Data Sources: Spark provides methods to create RDDs from various external data sources such as text files, sequence files, JSON files, HDFS, HBase, Cassandra, etc. This method is commonly used when dealing with large datasets stored in distributed file systems or databases.

```
from pyspark import SparkContext

# Create a SparkContext
sc = SparkContext("local", "RDD Example")

# Create an RDD from a text file
text_rdd = sc.textFile("hdfs://path/to/file.txt")

# Create an RDD from a CSV file using Spark CSV
from pyspark.sql import SparkSession

# Create a SparkSession
spark = SparkSession.builder \
    .appName("CSV to RDD Example") \
    .getOrCreate()

# Read a CSV file and convert it to an RDD
csv_rdd = spark.read.csv("hdfs://path/to/file.csv", header=True,
inferSchema=True).rdd
```

sc.textFile() is used to create an RDD named text_rdd from a text file stored in HDFS, while Spark's DataFrame API is utilized to read a CSV file and convert it to an RDD named csv_rdd.

2. In PySpark, create a program that reads a CSV file containing sales data, performs data cleaning by handling missing values and removing duplicates, calculates the total sales amount for each product, and finally, outputs the results to a new CSV file.

Sales Dataset:

The sales data contains attributes Order_ID, Date of Order, Customer_ID, Product_ID, Product Name, Price and Quantity.
Order_ID is unique for each product.
Each Customer might buy multiple products and each product might be bought by multiple customers.

Order_ID	Date	Customer_I	Product_ID	Product_Na	Price	Quantity
1	########	1001	P1001	Widget A		2
2	########	1002	P1002	Widget B	29.99	1
3	########	1003	P1003	Widget C	19.99	3
4	########	1004	P1004	Widget D	99.99	1
5	########	1005	P1005	Widget E	149.99	2
6	########		P1006	Widget F	39.99	1
7	########	1002	P1007	Widget G	79.99	2
8	########	1003	P1008	Widget H	49.99	1
9	########	1004	P1009	Widget I	199.99	1
10	########	1005	P1010	Widget J		3
11	########	1001	P1001	Widget A	49.99	2
12	########	1002	P1002	Widget B	29.99	2
13	########	1003	P1003	Widget C	19.99	1
14	########	1004	P1004	Widget D	99.99	3
15	########	1005	P1005	Widget E	149.99	1
16	########		P1006	Widget F	39.99	2
17	########	1007	P1007	Widget G	79.99	1
18	########		P1008	Widget H	49.99	2
19	########	1009	P1009	Widget I	199.99	1
20	########	1010	P1010	Widget J	99.99	4
21	########	1006	P1001	Widget A	49.99	1
22	########	1007	P1002	Widget B	29.99	1
23	########	1008	P1004	Widget D	99.99	2
24	########	1009	P1005	Widget E	149.99	3
25	########		P1006	Widget F	39.99	2
26	########	1001	P1007	Widget G	79.99	1

1)Installing and importing required libraries i.e PySpark

```
pip install pyspark
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 4.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=317488491 sha256=f4eeff76f33519a85c738ab478fde81aeb6b69781dea26d874256194f03864e4
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
```

```
[ ] import pyspark
```

2) Creating a Spark Session and reading the raw dataset.

```
[ ] pyspark
```

```
<module 'pyspark' from '/usr/local/lib/python3.10/dist-packages/pyspark/__init__.py'>
```

```
from pyspark.sql import SparkSession
```

```
[ ] spark=SparkSession.builder.appName('Practise').getOrCreate()
```

```
[ ] df_pyspark=spark.read.csv('/content/Sales_Dataset.csv',header=True,inferSchema=True)
```

```
[ ] df_pyspark
```

```
DataFrame[Order_ID: int, Date: string, Customer_ID: int, Product_ID: string, Product_Name: string, Price: double, Quantity: int]
```

```
df_pyspark.show()
```

```
+--------+----------+-----------+----------+------------+------+--------+
|Order_ID|      Date|Customer_ID|Product_ID|Product_Name| Price|Quantity|
+--------+----------+-----------+----------+------------+------+--------+
|       1|01-01-2024|       1001|     P1001|    Widget A|  NULL|       2|
|       2|02-01-2024|       1002|     P1002|    Widget B| 29.99|       1|
|       3|03-01-2024|       1003|     P1003|    Widget C| 19.99|       3|
|       4|04-01-2024|       1004|     P1004|    Widget D| 99.99|       1|
|       5|05-01-2024|       1005|     P1005|    Widget E|149.99|       2|
|       6|06-01-2024|       NULL|     P1006|    Widget F| 39.99|       1|
|       7|07-01-2024|       1002|     P1007|    Widget G| 79.99|       2|
|       8|08-01-2024|       1003|     P1008|    Widget H| 49.99|       1|
|       9|09-01-2024|       1004|     P1009|    Widget I|199.99|       1|
|      10|10-01-2024|       1005|     P1010|    Widget J|  NULL|       3|
|      11|11-01-2024|       1001|     P1001|    Widget A| 49.99|       2|
|      12|12-01-2024|       1002|     P1002|    Widget B| 29.99|       2|
|      13|13-01-2024|       1003|     P1003|    Widget C| 19.99|       1|
|      14|14-01-2024|       1004|     P1004|    Widget D| 99.99|       3|
|      15|15-01-2024|       1005|     P1005|    Widget E|149.99|       1|
|      16|16-01-2024|       NULL|     P1006|    Widget F| 39.99|       2|
|      17|17-01-2024|       1007|     P1007|    Widget G| 79.99|       1|
|      18|18-01-2024|       NULL|     P1008|    Widget H| 49.99|       2|
|      19|19-01-2024|       1009|     P1009|    Widget I|199.99|       1|
|      20|20-01-2024|       1010|     P1010|    Widget J| 99.99|       4|
+--------+----------+-----------+----------+------------+------+--------+
only showing top 20 rows
```

3) Dropping columns unnecessary to the required problem i.e date of order

```
[ ]  df_pyspark=df_pyspark.drop('Date')
```

```
df_pyspark.show()
```

```
+--------+-----------+----------+------------+------+--------+
|Order_ID|Customer_ID|Product_ID|Product_Name| Price|Quantity|
+--------+-----------+----------+------------+------+--------+
|       1|       1001|     P1001|    Widget A|  NULL|       2|
|       2|       1002|     P1002|    Widget B| 29.99|       1|
|       3|       1003|     P1003|    Widget C| 19.99|       3|
|       4|       1004|     P1004|    Widget D| 99.99|       1|
|       5|       1005|     P1005|    Widget E|149.99|       2|
|       6|       NULL|     P1006|    Widget F| 39.99|       1|
|       7|       1002|     P1007|    Widget G| 79.99|       2|
|       8|       1003|     P1008|    Widget H| 49.99|       1|
|       9|       1004|     P1009|    Widget I|199.99|       1|
|      10|       1005|     P1010|    Widget J|  NULL|       3|
|      11|       1001|     P1001|    Widget A| 49.99|       2|
|      12|       1002|     P1002|    Widget B| 29.99|       2|
|      13|       1003|     P1003|    Widget C| 19.99|       1|
|      14|       1004|     P1004|    Widget D| 99.99|       3|
|      15|       1005|     P1005|    Widget E|149.99|       1|
|      16|       NULL|     P1006|    Widget F| 39.99|       2|
|      17|       1007|     P1007|    Widget G| 79.99|       1|
|      18|       NULL|     P1008|    Widget H| 49.99|       2|
|      19|       1009|     P1009|    Widget I|199.99|       1|
|      20|       1010|     P1010|    Widget J| 99.99|       4|
+--------+-----------+----------+------------+------+--------+
only showing top 20 rows
```

4) Cleaning the dataset by removing

- records with any missing values of Customer_ID or Product_ID
- Duplicate records

```
df_pyspark=df_pyspark.na.drop(how="any",subset=['Customer_ID','Product_ID'])
df_pyspark=df_pyspark.dropDuplicates()
```

```
[ ] df_pyspark.show()
```

```
+--------+-----------+----------+------------+------+--------+
|Order_ID|Customer_ID|Product_ID|Product_Name| Price|Quantity|
+--------+-----------+----------+------------+------+--------+
|       3|       1003|     P1003|    Widget C| 19.99|       3|
|      43|       1008|     P1004|    Widget D| 99.99|       2|
|       4|       1004|     P1004|    Widget D| 99.99|       1|
|       7|       1002|     P1007|    Widget G| 79.99|       2|
|      37|       1002|     P1008|    Widget H| 49.99|       1|
|      40|       1005|     P1001|    Widget A| 49.99|       4|
|       9|       1004|     P1009|    Widget I|199.99|       1|
|      22|       1007|     P1002|    Widget B| 29.99|       1|
|       8|       1003|     P1008|    Widget H| 49.99|       1|
|      38|       1003|     P1009|    Widget I|199.99|       2|
|       2|       1002|     P1002|    Widget B| 29.99|       1|
|      41|       1006|     P1002|    Widget B| 29.99|       1|
|      27|       1002|     P1008|    Widget H| 49.99|       1|
|      29|       1004|     P1010|    Widget J| 99.99|       1|
|      17|       1007|     P1007|    Widget G| 79.99|       1|
|      15|       1005|     P1005|    Widget E|149.99|       1|
|      47|       1002|     P1008|    Widget H| 49.99|       1|
|      33|       1008|     P1004|    Widget D| 99.99|       1|
|      36|       1001|     P1007|    Widget G| 79.99|       2|
|       5|       1005|     P1005|    Widget E|149.99|       2|
+--------+-----------+----------+------------+------+--------+
only showing top 20 rows
```

5) Substituting the missing values of Price with the mode i.e the most common value of product prices by making use of the Imputer method.

```
[ ] from pyspark.ml.feature import Imputer

    imputer = Imputer(
        inputCols=['Price'],
        outputCols=["{}_imputed".format(c) for c in ['Price']]
        ).setStrategy("mode")
```

```
[ ] imputed_df = imputer.fit(df_pyspark).transform(df_pyspark)
    imputed_df.show()
```

```
+--------+-----------+----------+------------+------+--------+-------------+
|Order_ID|Customer_ID|Product_ID|Product_Name| Price|Quantity|Price_imputed|
+--------+-----------+----------+------------+------+--------+-------------+
|       3|       1003|     P1003|    Widget C| 19.99|       3|        19.99|
|      43|       1008|     P1004|    Widget D| 99.99|       2|        99.99|
|       4|       1004|     P1004|    Widget D| 99.99|       1|        99.99|
|       7|       1002|     P1007|    Widget G| 79.99|       2|        79.99|
|      37|       1002|     P1008|    Widget H| 49.99|       1|        49.99|
|      40|       1005|     P1001|    Widget A| 49.99|       4|        49.99|
|       9|       1004|     P1009|    Widget I|199.99|       1|       199.99|
|      22|       1007|     P1002|    Widget B| 29.99|       1|        29.99|
|       8|       1003|     P1008|    Widget H| 49.99|       1|        49.99|
|      38|       1003|     P1009|    Widget I|199.99|       2|       199.99|
|       2|       1002|     P1002|    Widget B| 29.99|       1|        29.99|
|      41|       1006|     P1002|    Widget B| 29.99|       1|        29.99|
|      27|       1002|     P1008|    Widget H| 49.99|       1|        49.99|
|      29|       1004|     P1010|    Widget J| 99.99|       1|        99.99|
|      17|       1007|     P1007|    Widget G| 79.99|       1|        79.99|
|      15|       1005|     P1005|    Widget E|149.99|       1|       149.99|
|      47|       1002|     P1008|    Widget H| 49.99|       1|        49.99|
|      33|       1008|     P1004|    Widget D| 99.99|       1|        99.99|
|      36|       1001|     P1007|    Widget G| 79.99|       2|        79.99|
|       5|       1005|     P1005|    Widget E|149.99|       2|       149.99|
+--------+-----------+----------+------------+------+--------+-------------+
only showing top 20 rows
```

```
[ ] imputed_df=imputed_df.drop('Price')
    imputed_df.show()
```

```
+--------+-----------+----------+------------+--------+-------------+
|Order_ID|Customer_ID|Product_ID|Product_Name|Quantity|Price_imputed|
+--------+-----------+----------+------------+--------+-------------+
|       3|       1003|     P1003|    Widget C|       3|        19.99|
|      43|       1008|     P1004|    Widget D|       2|        99.99|
|       4|       1004|     P1004|    Widget D|       1|        99.99|
|       7|       1002|     P1007|    Widget G|       2|        79.99|
|      37|       1002|     P1008|    Widget H|       1|        49.99|
|      40|       1005|     P1001|    Widget A|       4|        49.99|
|       9|       1004|     P1009|    Widget I|       1|       199.99|
|      22|       1007|     P1002|    Widget B|       1|        29.99|
|       8|       1003|     P1008|    Widget H|       1|        49.99|
|      38|       1003|     P1009|    Widget I|       2|       199.99|
|       2|       1002|     P1002|    Widget B|       1|        29.99|
|      41|       1006|     P1002|    Widget B|       1|        29.99|
|      27|       1002|     P1008|    Widget H|       1|        49.99|
|      29|       1004|     P1010|    Widget J|       1|        99.99|
|      17|       1007|     P1007|    Widget G|       1|        79.99|
|      15|       1005|     P1005|    Widget E|       1|       149.99|
|      47|       1002|     P1008|    Widget H|       1|        49.99|
|      33|       1008|     P1004|    Widget D|       1|        99.99|
|      36|       1001|     P1007|    Widget G|       2|        79.99|
|       5|       1005|     P1005|    Widget E|       2|       149.99|
+--------+-----------+----------+------------+--------+-------------+
only showing top 20 rows
```

6) Calculating the total price of each order by multiplying the Quantity of the product with the price.

```
[ ]  from pyspark.sql.functions import col, format_number
     df = imputed_df.withColumn("Total_Price", col("Quantity") * col("Price_imputed"))
     df = df.withColumn("Total_Price", format_number(col("Total_Price"), 2))
```

```
[ ]  df.show()
```

```
+--------+-----------+----------+------------+--------+-------------+-----------+
|Order_ID|Customer_ID|Product_ID|Product_Name|Quantity|Price_imputed|Total_Price|
+--------+-----------+----------+------------+--------+-------------+-----------+
|       3|       1003|     P1003|    Widget C|       3|        19.99|      59.97|
|      43|       1008|     P1004|    Widget D|       2|        99.99|     199.98|
|       4|       1004|     P1004|    Widget D|       1|        99.99|      99.99|
|       7|       1002|     P1007|    Widget G|       2|        79.99|     159.98|
|      37|       1002|     P1008|    Widget H|       1|        49.99|      49.99|
|      40|       1005|     P1001|    Widget A|       4|        49.99|     199.96|
|       9|       1004|     P1009|    Widget I|       1|       199.99|     199.99|
|      22|       1007|     P1002|    Widget B|       1|        29.99|      29.99|
|       8|       1003|     P1008|    Widget H|       1|        49.99|      49.99|
|      38|       1003|     P1009|    Widget I|       2|       199.99|     399.98|
|       2|       1002|     P1002|    Widget B|       1|        29.99|      29.99|
|      41|       1006|     P1002|    Widget B|       1|        29.99|      29.99|
|      27|       1002|     P1008|    Widget H|       1|        49.99|      49.99|
|      29|       1004|     P1010|    Widget J|       1|        99.99|      99.99|
|      17|       1007|     P1007|    Widget G|       1|        79.99|      79.99|
|      15|       1005|     P1005|    Widget E|       1|       149.99|     149.99|
|      47|       1002|     P1008|    Widget H|       1|        49.99|      49.99|
|      33|       1008|     P1004|    Widget D|       1|        99.99|      99.99|
|      36|       1001|     P1007|    Widget G|       2|        79.99|     159.98|
|       5|       1005|     P1005|    Widget E|       2|       149.99|     299.98|
+--------+-----------+----------+------------+--------+-------------+-----------+
only showing top 20 rows
```

7) Writing the cleaned data to a new file

```
[ ]  df.write.csv("cleaned_data.csv", header=True)
```

8) Calculating Total sales amount of each product

```
[ ]  total_sales_by_product = df.groupBy("Product_ID") \
         .agg(format_number(sum(col("Total_Price")), 2).alias("Total_Sales_Amount")) \
         .orderBy("Product_ID")

     # Show the result
     total_sales_by_product.show()
```

```
+----------+------------------+
|Product_ID|Total_Sales_Amount|
+----------+------------------+
|      P1001|            649.87|
|      P1002|            209.93|
|      P1003|             99.95|
|      P1004|            899.91|
|      P1005|          1,799.88|
|      P1006|            119.97|
|      P1007|            559.93|
|      P1008|            199.96|
|      P1009|          1,599.92|
|      P1010|            749.91|
+----------+------------------+
```

Cleaned Dataset

Order_ID	Customer_I	Product_ID	Product_Na	Quantity	Price_imput	Total_Price
3	1003	P1003	Widget C	3	19.99	59.97
43	1008	P1004	Widget D	2	99.99	199.98
4	1004	P1004	Widget D	1	99.99	99.99
7	1002	P1007	Widget G	2	79.99	159.98
37	1002	P1008	Widget H	1	49.99	49.99
40	1005	P1001	Widget A	4	49.99	199.96
9	1004	P1009	Widget I	1	199.99	199.99
22	1007	P1002	Widget B	1	29.99	29.99
8	1003	P1008	Widget H	1	49.99	49.99
38	1003	P1009	Widget I	2	199.99	399.98
2	1002	P1002	Widget B	1	29.99	29.99
41	1006	P1002	Widget B	1	29.99	29.99
27	1002	P1008	Widget H	1	49.99	49.99
29	1004	P1010	Widget J	1	99.99	99.99
17	1007	P1007	Widget G	1	79.99	79.99
15	1005	P1005	Widget E	1	149.99	149.99
47	1002	P1008	Widget H	1	49.99	49.99
33	1008	P1004	Widget D	1	99.99	99.99
36	1001	P1007	Widget G	2	79.99	159.98
5	1005	P1005	Widget E	2	149.99	299.98
34	1009	P1005	Widget E	3	149.99	449.97
21	1006	P1001	Widget A	1	49.99	49.99
48	1003	P1009	Widget I	2	199.99	399.98
35	1010	P1006	Widget F	1	39.99	39.99
19	1009	P1009	Widget I	1	199.99	199.99
26	1001	P1007	Widget G	1	79.99	79.99