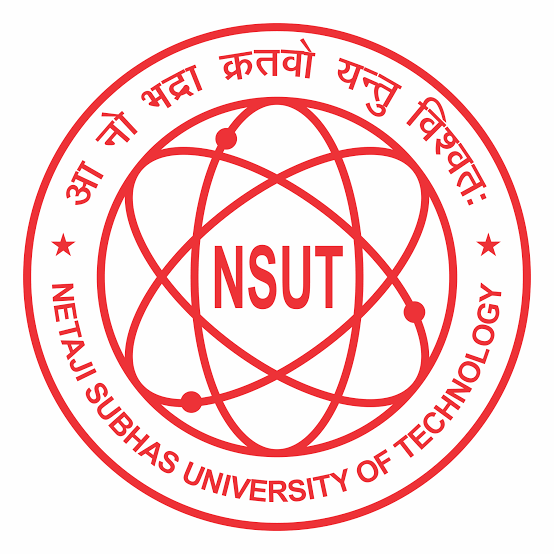
**COMPUTER HARDWARE SOFTWARE WORKSHOP ( COCSC19 )**

**Title: Sparks;**

**UNIT -4**



Submitted by :

Amar Kumar Tarei: 2021UCS1614

Shreyansh Agrawal: 2021UCS1612

Ayushi Singh: 2021UCS1616

**PRACTICAL 04: -**

**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.

**TASK 1: EXPLORING RDD IN SPARKS**

In Apache Spark, RDD stands for Resilient Distributed Dataset. RDD is the fundamental abstraction in Spark, representing an immutable, distributed collection of objects that can be operated on in parallel across a cluster. Here's an exploration of RDDs in Spark:

1. Resilient: RDDs are resilient because they can be rebuilt if part of the dataset is lost due to node failure. This resilience is achieved through lineage, which records the sequence of transformations applied to the base dataset to build the RDD. If any partition of an RDD is lost due to a worker failure, Spark can use this lineage information to recompute the lost partition.

2. Distributed: RDDs are distributed across multiple nodes in a cluster, enabling parallel processing. Each RDD is split into multiple partitions, with each partition being processed independently on different nodes. This distributed nature allows Spark to scale horizontally and handle large datasets.

3. Dataset: RDD represents a dataset of elements, which could be anything from basic types like integers or strings to complex types like tuples, objects, or even other RDDs.

4. Immutable: Once created, RDDs are immutable, meaning their contents cannot be changed. However, you can apply transformations to create new RDDs derived from existing ones. This immutability simplifies concurrency control and makes RDDs easier to reason about in distributed environments.

5. Transformations and Actions: RDDs support two types of operations: transformations and actions. Transformations create a new RDD by applying some operation to the elements of the parent RDD(s), while actions compute a result based on an RDD and return it to the driver program or perform some side effects, such as writing data to storage.

6. Lazy Evaluation: Transformations in Spark are lazy, meaning they are not executed immediately when called. Instead, Spark remembers the sequence of transformations applied to the RDD and only computes them when an action is called. This lazy evaluation allows Spark to optimize the execution plan and minimize unnecessary computation.

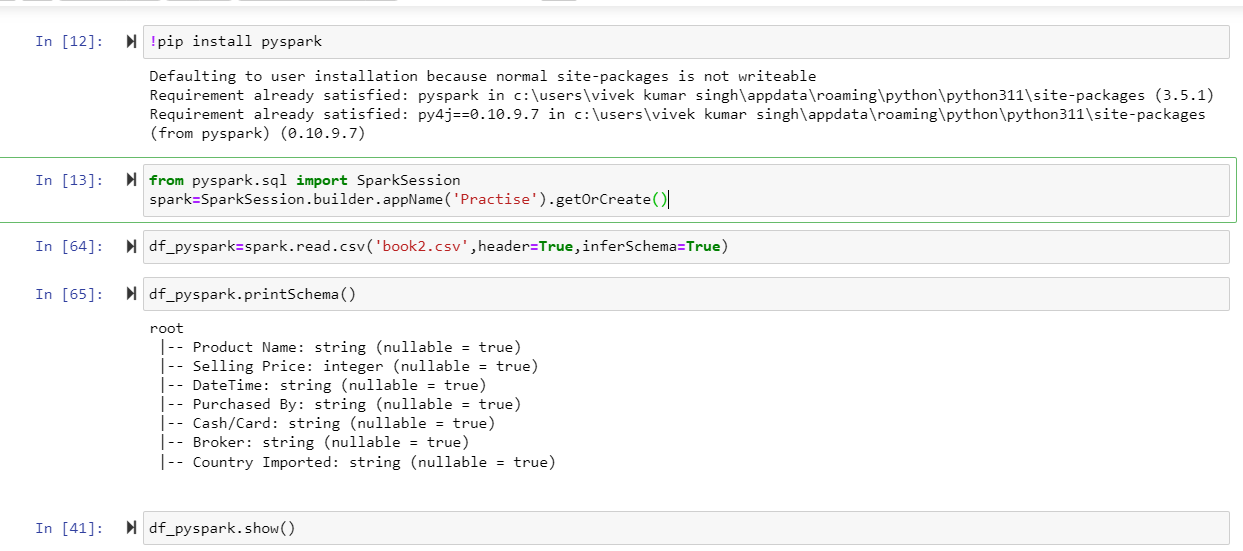
7. Fault Tolerance: As mentioned earlier, RDDs are fault-tolerant due to their lineage information, which allows Spark to recompute lost partitions in case of node failures. This fault tolerance is crucial for ensuring the reliability of computations on large-scale distributed datasets.

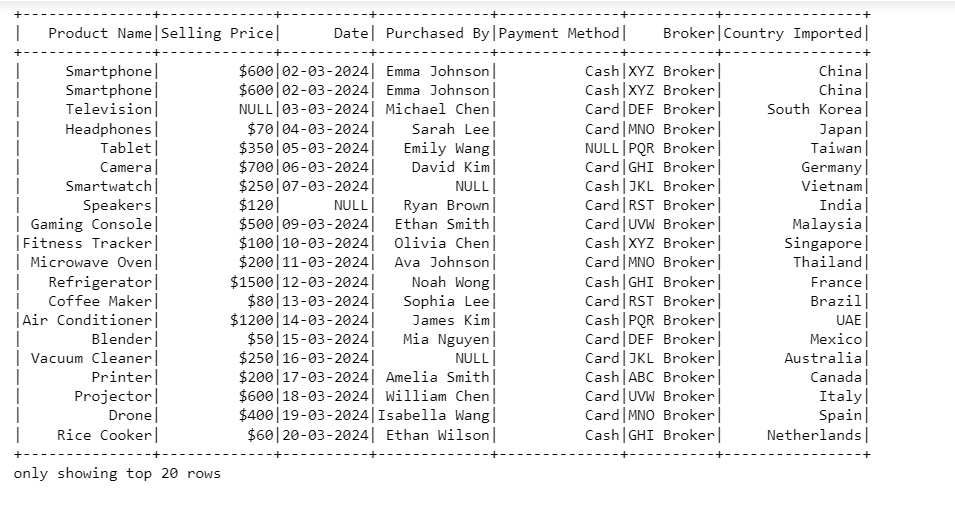
Overall, RDDs provide a powerful and flexible abstraction for distributed data processing in Apache Spark, enabling developers to build complex data processing workflows and applications with ease. However, with the introduction of Data Frames and Datasets in Spark, RDDs are less commonly used directly in Favor of these higher-level abstractions, which offer better performance optimizations and a more user-friendly API for structured data processing**.**

| **Operation** | **Syntax** | **Description** |
| --- | --- | --- |
| **Creating RDDs** |  |  |
| Parallelizing | **data = [1, 2, 3, 4, 5]**<br>**rdd = sc.parallelize(data)** | Create an RDD by parallelizing a Python collection. |
| Reading from storage | **rdd = sc.textFile("file.txt")** | Create an RDD by reading data from an external storage, such as a text file. |
| **Transformations** |  |  |
| Map | **rdd\_mapped = rdd.map(lambda x: x \* 2)** | Apply a function to each element of the RDD. |
| Filter | **rdd\_filtered = rdd.filter(lambda x: x % 2 == 0)** | Filter elements based on a predicate function. |
| FlatMap | **rdd\_flat\_mapped = rdd.flatMap(lambda x: (x, x\*2))** | Similar to map, but each input item can be mapped to 0 or more output items. |
| ReduceByKey | **rdd\_key\_values = sc.parallelize([("a", 1), ("b", 2), ("a", 3)])**<br>**rdd\_reduced = rdd\_key\_values.reduceByKey(lambda x, y: x + y)** | Combine values with the same key. |
| **Actions** |  |  |
| Collect | **result = rdd.collect()** | Retrieve all elements of the RDD as an array (use with caution on large datasets). |
| Count | **count = rdd.count()** | Count the number of elements in the RDD. |
| Take | **elements = rdd.take(5)** | Retrieve the first n elements of the RDD. |
| SaveAsTextFile | **rdd.saveAsTextFile("output\_folder")** | Save the RDD as a text file. |
| **Other Operations** |  |  |
| GetNumPartitions | **num\_partitions = rdd.getNumPartitions()** | Get the number of partitions in the RDD. |
| Cache | **rdd.cache()** | Cache the RDD in memory. |
| Unpersist | **rdd.unpersist()** | Remove RDD from memory. |
| Sampling | **sampled\_rdd = rdd.sample(withReplacement=True, fraction=0.5)** | Sample the RDD with or without replacement. |

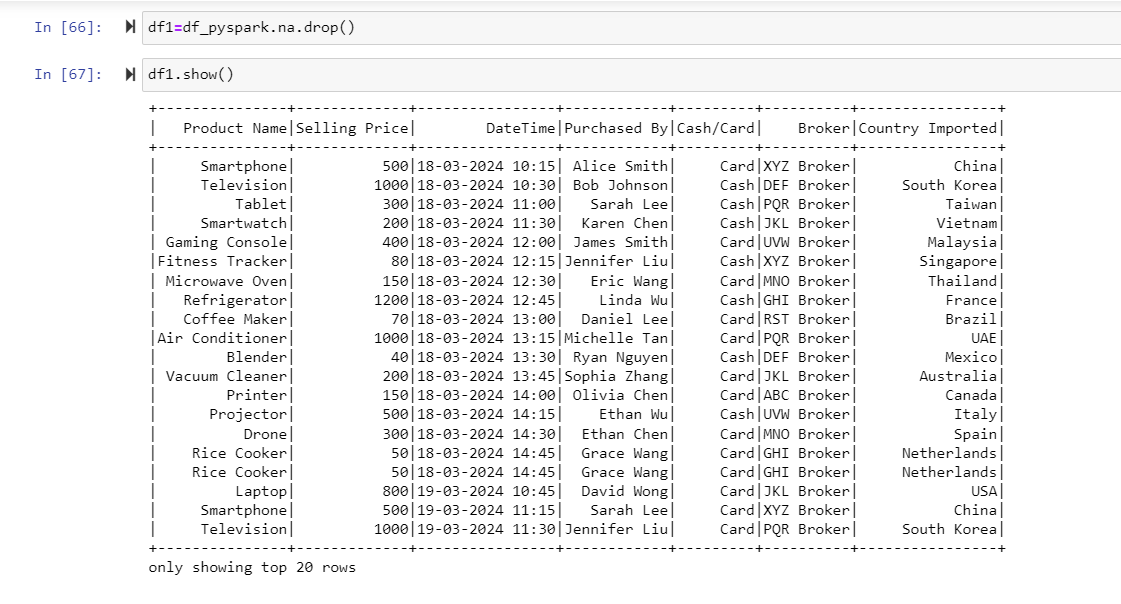
**TASK 2**

**Step 1: Read the csv into data frames.**

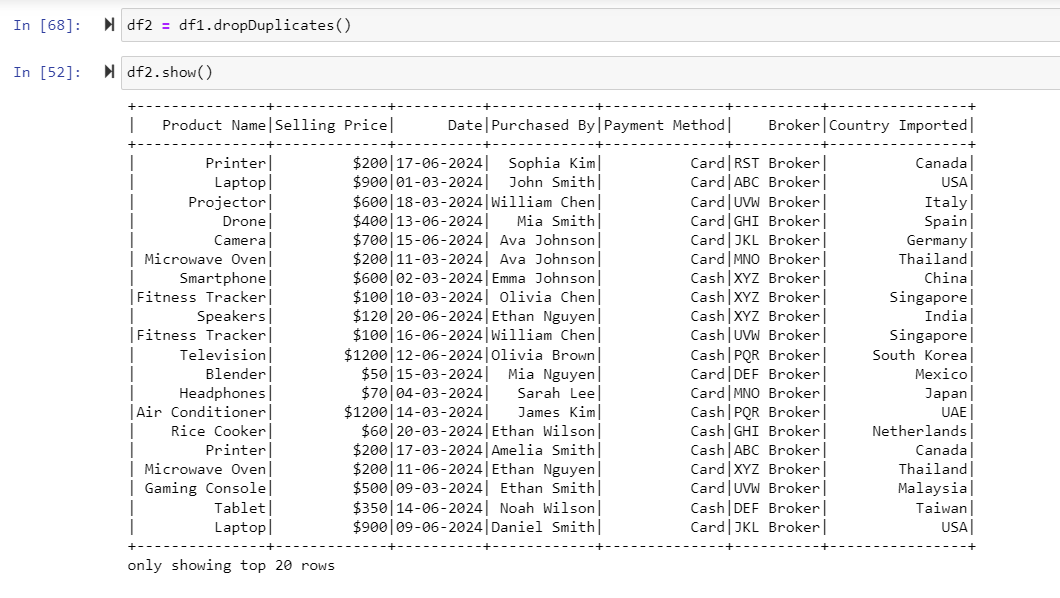
****

****

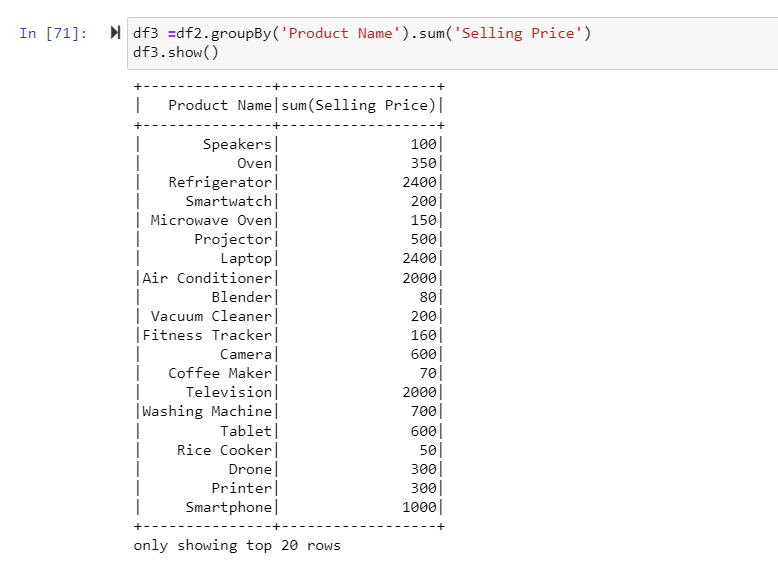
**Step 2: Remove the missing values column of tuples**

****

**Step 3: Remove the duplicates**

****

**Step 4: find the aggregate sum of sales of each product**

****

**Step 5: writing the dataframe into the csv file**

****