# UNIT - 4

# SPARK

*SHRISHIR SRIVATSA – 2021UCS1590*
*TEJAS MISHRA – 2021UCS1587*
*PRAJWAL BHAT – 2021UCS1609*

# Task 1:

## Explore RDD in spark

RDD (Resilient Distributed Dataset) is the fundamental data structure in Apache Spark. It represents an immutable, partitioned collection of records that can be operated on in parallel. RDDs can be created from various data sources or by transforming other RDDs.

1. Creation: RDDs can be created in several ways, such as by parallelizing an existing collection in your driver program, by loading external datasets (e.g., from HDFS, S3, or a relational database), or by transforming an existing RDD.

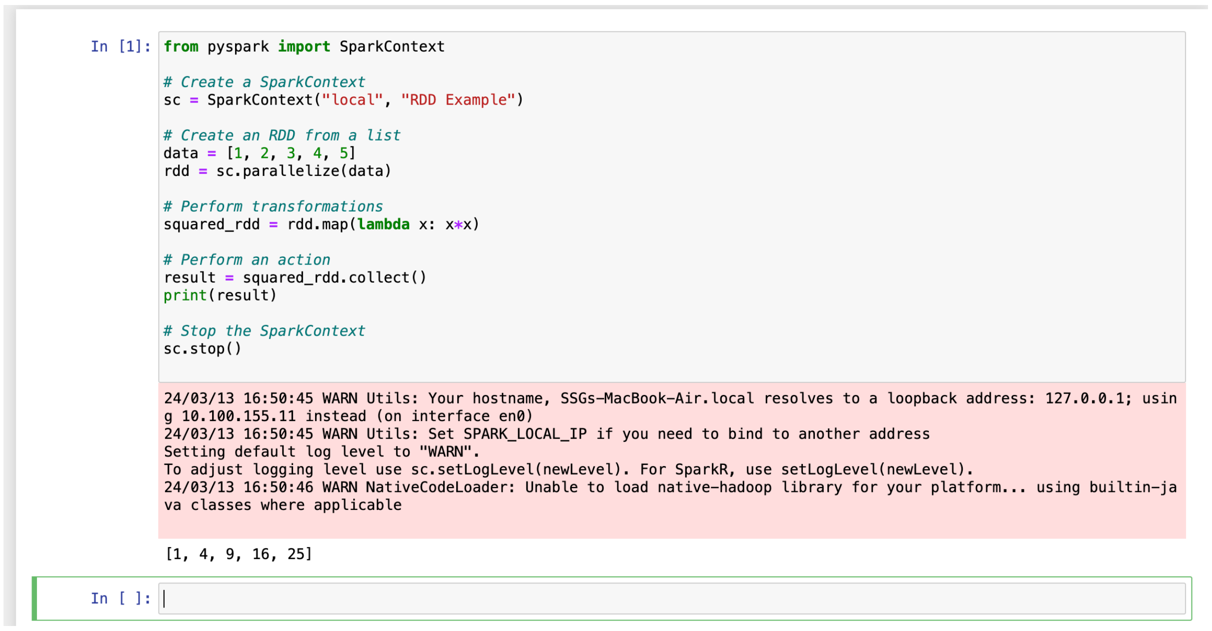
2. Transformation: RDDs support two types of operations: transformations and actions. Transformations create a new RDD from an existing one, such as `map()`, `filter()`, `flatMap()`, etc. These operations are lazily evaluated, meaning Spark delays executing them until an action is invoked.

3. Action: Actions compute a result based on an RDD, triggering the actual execution of the RDD transformations. Examples of actions include `collect()`, `count()`, `reduce()`, `take()`, `saveAsTextFile()`, etc.

4. Immutability: RDDs are immutable, meaning once created, they cannot be changed. If you need to modify an RDD, you create a new RDD based on the original.

5. Fault tolerance: RDDs are fault-tolerant by nature. Spark tracks the lineage of each RDD (i.e., the sequence of transformations used to build it) so that it can recompute lost data partitions due to node failures.

6. Partitioning: RDDs are partitioned collections of records. Spark automatically distributes the data across the cluster, allowing parallel processing of the data.



```
In [1]: from pyspark import SparkContext

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

        # Create an RDD from a list
        data = [1, 2, 3, 4, 5]
        rdd = sc.parallelize(data)

        # Perform transformations
        squared_rdd = rdd.map(lambda x: x*x)

        # Perform an action
        result = squared_rdd.collect()
        print(result)

        # Stop the SparkContext
        sc.stop()
```

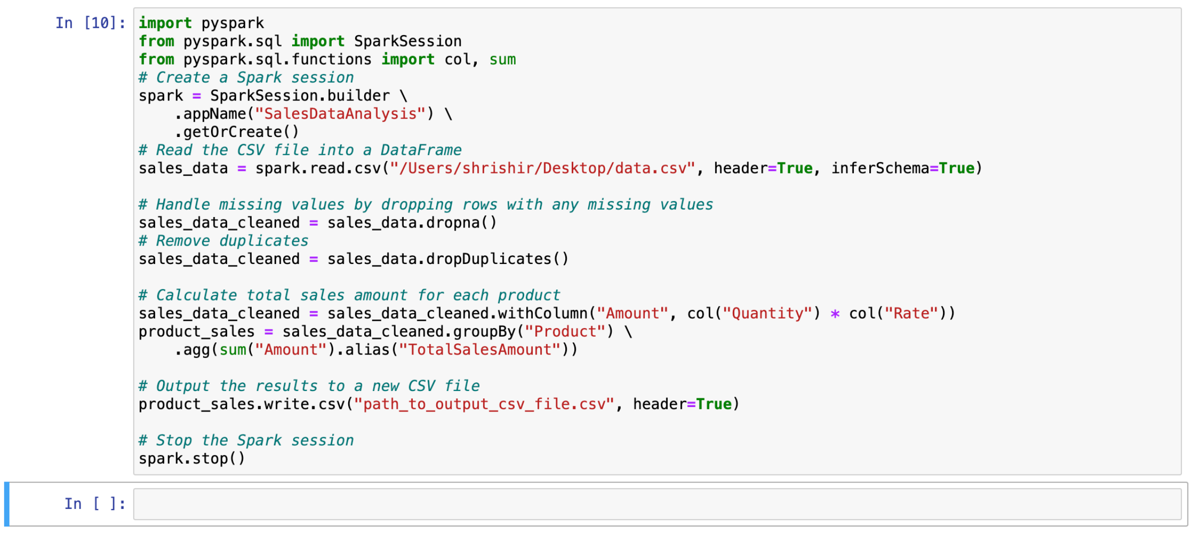
```
24/03/13 16:50:45 WARN Utils: Your hostname, SSGs-MacBook-Air.local resolves to a loopback address: 127.0.0.1; usin
g 10.100.155.11 instead (on interface en0)
24/03/13 16:50:45 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
24/03/13 16:50:46 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-ja
va classes where applicable

[1, 4, 9, 16, 25]
```

```
In [ ]:
```

In this example, we create an RDD from a list, square each element using `map()`, and then collect the results using `collect()`.

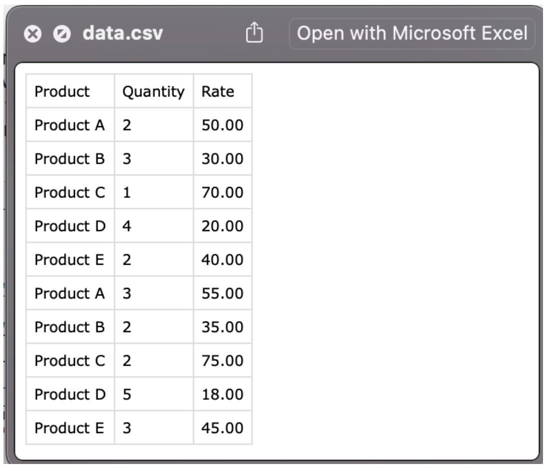
## Task 2:

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



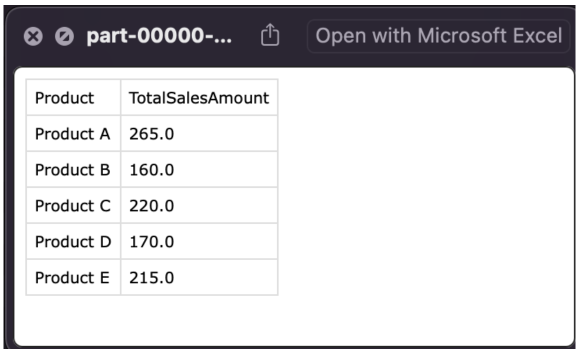
```
In [10]: import pyspark
         from pyspark.sql import SparkSession
         from pyspark.sql.functions import col, sum
         # Create a Spark session
         spark = SparkSession.builder \
             .appName("SalesDataAnalysis") \
             .getOrCreate()
         # Read the CSV file into a DataFrame
         sales_data = spark.read.csv("/Users/shrishir/Desktop/data.csv", header=True, inferSchema=True)

         # Handle missing values by dropping rows with any missing values
         sales_data_cleaned = sales_data.dropna()
         # Remove duplicates
         sales_data_cleaned = sales_data.dropDuplicates()

         # Calculate total sales amount for each product
         sales_data_cleaned = sales_data_cleaned.withColumn("Amount", col("Quantity") * col("Rate"))
         product_sales = sales_data_cleaned.groupBy("Product") \
             .agg(sum("Amount").alias("TotalSalesAmount"))

         # Output the results to a new CSV file
         product_sales.write.csv("path_to_output_csv_file.csv", header=True)

         # Stop the Spark session
         spark.stop()

In [ ]:
```

Sample Data:


data.csv
Open with Microsoft Excel

| Product | Quantity | Rate |
|---|---|---|
| Product A | 2 | 50.00 |
| Product B | 3 | 30.00 |
| Product C | 1 | 70.00 |
| Product D | 4 | 20.00 |
| Product E | 2 | 40.00 |
| Product A | 3 | 55.00 |
| Product B | 2 | 35.00 |
| Product C | 2 | 75.00 |
| Product D | 5 | 18.00 |
| Product E | 3 | 45.00 |

New Output CSV file:



| Product | TotalSalesAmount |
|---|---|
| Product A | 265.0 |
| Product B | 160.0 |
| Product C | 220.0 |
| Product D | 170.0 |
| Product E | 215.0 |

<u>Transformations and actions used</u>

Transformations:

sales_data = spark.read.csv("path_to_your_csv_file.csv", header=True, inferSchema=True):
This line reads the CSV file into a DataFrame, which is a transformation.

sales_data_cleaned = sales_data.dropDuplicates():
This line removes duplicates from the DataFrame, which is another transformation.

sales_data_cleaned = sales_data_cleaned.withColumn("Amount", col("Quantity") * col("Rate")):
This line adds a new column "Amount" to the DataFrame, calculated as the product of "Quantity" and "Rate". This is also a transformation.

Action:

product_sales.write.csv("path_to_output_csv_file.csv", header=True):
This line writes the final DataFrame to a CSV file, which is an action. It triggers the execution of the preceding transformations.