Name: **Gñilo, Jela Mae N.** Lab No. **1**

Date: **January 27, 2025**

Objective

The objective of this lab is to understand and implement data transformations using Apache Spark RDDs. Specifically, the goal is to create a pipeline that performs at least five transformations on a dataset, showcasing the flexibility and efficiency of Spark for big data processing.

Introduction

Apache Spark is an open-source distributed computing framework designed for processing large-scale data efficiently. Its core data structure, the Resilient Distributed Dataset (RDD), supports fault-tolerant, in-memory computations of distributed data.

This lab focuses on building a Spark RDD pipeline to process a sample dataset. The pipeline includes five transformations, such as filtering, mapping, grouping, sorting, and formatting detailed output. These operations demonstrate how Spark handles data transformations in parallel, leveraging cluster computing for scalability and speed.

Methodology

1. Load the Dataset
   A sample dataset of transactions, including details (e.g., Product, Category, Amount) is represented as a list and converted into RDD using **sc.parallelize().**
2. Filter Transactions
   The dataset is filtered to include only transactions with an amount greater than 500 using the **filter** transformation.
3. Map Data
   The filtered records are transformed into key-value pairs format **(Category, Amount)** using the **map** transformation.
4. Aggregate Data
   The total amount spent per category is calculated using the **reduceByKey** transformation.
5. Sort Results
   The categories are sorted by total spending in descending order using the **sortBy** transformation.
6. Format Detailed Output
   The sorted results are formatted into detailed output strings (e.g., “Category: Electronics, Total: 3500.00”) using the **flatMap** transformation.
7. Collect and Display Results
   The final processed data is collected using **collect()** and displayed as detailed output.

```python
from pyspark import SparkContext

# Initialize Spark Context
sc = SparkContext("local", "RDD Pipeline Lab")

# Sample Dataset (List of Transactions)
# Format: (TransactionID, Product, Category, Amount, Date)
data = [
    (1, "Laptop", "Electronics", 1200.00, "2025-01-01"),
    (2, "Smartphone", "Electronics", 800.00, "2025-01-02"),
    (3, "Washing Machine", "Home Appliances", 1500.00, "2025-01-03"),
    (4, "Shirt", "Clothing", 300.00, "2025-01-04"),
    (5, "Refrigerator", "Home Appliances", 1000.00, "2025-01-05"),
    (6, "Jeans", "Clothing", 700.00, "2025-01-06"),
    (7, "Headphones", "Electronics", 200.00, "2025-01-07"),
    (8, "Television", "Electronics", 1500.00, "2025-01-08"),
]

# Create an RDD from the list
rdd = sc.parallelize(data)

# Step 1: Filter transactions where the amount > 500
filtered_rdd = rdd.filter(lambda record: record[3] > 500)

# Step 2: Map data to extract (Category, Amount)
mapped_rdd = filtered_rdd.map(lambda record: (record[2], record[3]))

# Step 3: Reduce by key to calculate total amount spent per category
reduced_rdd = mapped_rdd.reduceByKey(lambda x, y: x + y)

# Step 4: Sort categories by total amount spent in descending order
sorted_rdd = reduced_rdd.sortBy(lambda record: record[1], ascending=False)

# Step 5: FlatMap to create detailed output strings
detailed_rdd = sorted_rdd.flatMap(lambda record: [f"Category: {record[0]}, Total: {record[1]}"])
```

```python
# Collect and Print Results
results = detailed_rdd.collect()
for result in results:
    print(result)

# Stop Spark Context
sc.stop()
```

Results and Analysis

The Spark pipeline processed the dataset and returned the total spending per category, sorted by the highest expenditure:

```
Category: Electronics, Total: 3500.00
Category: Home Appliances, Total: 2500.00
Category: Clothing, Total: 700.00
```

Key Findings:

- The Electronics category had the highest total spending at 3500.00.
- The Home Appliances category followed with 2500.00.
- The Clothing category had the lowest total spending at 700.00

Challenges and Solutions

Challenges:

- Setting up Apache Spark was challenging, with multiple errors encountered during the process.
- Apache Spark is not supported by the latest version of Python, which caused compatibility issues.
- It took several days to troubleshoot and resolve these issues.

Solutions:

- Downgraded Python to a compatible version to ensure Apache Spark worked properly.
- Used Visual Studio Code as the preferred coding environment, which provided familiarity and ease of use.

Conclusion

This lab successfully demonstrated the use of Apache Spark's RDD API to implement a data processing pipeline. The objectives of filtering, transforming, aggregating, and sorting data were achieved.