Name: Barquilla, Meshe Mae N. Lab No. 1

Git Repo/ Colab Link: https://github.com/meshemae25/Barquilla.git Date: 26/01/2025

Objective

To create and execute a Spark RDD pipeline with at least five transformations. The goal is to understand how Spark RDD transformations work, including mapping, filtering, reducing, sorting, and collecting data.

Introduction Apache Spark is a distributed computing framework that enables efficient processing of large datasets. Resilient Distributed Datasets (RDDs) form the backbone of Spark's data structures, allowing transformations and actions to be performed in a fault-tolerant manner. In this lab, we will create an RDD pipeline and apply multiple transformations to process data efficiently.

Methodology

1. **Initialize SparkSession and SparkContext**: Create a SparkSession and obtain the SparkContext for handling RDD operations.
2. **Create an RDD**: Load a small dataset of names and ages into an RDD.
3. **Apply Transformations**:
   - **Map**: Multiply the age by 2.
   - **Filter**: Retain only records where the modified age is greater than 20.
   - **ReduceByKey**: Sum the ages grouped by name.
   - **SortByKey**: Sort the data by name.
   - **Collect**: Gather the final results.
4. **Execute and Analyze**: Run the Spark job and observe the output.
5. **Stop Spark Session**: Properly terminate the session to release resources.

Results and Analysis

The final output after executing the transformations is a sorted list of names with their aggregated ages. The transformations successfully processed the data by doubling the ages, filtering out small values, grouping by name, and sorting the final results. The collected output confirms that the transformations were applied correctly.

Example Output:

[('Albert', 24), ('Alfred', 46), ('Amber', 62)]

Key observations:

- The original dataset contained duplicate names, which were summed up after applying reduceByKey.
- Filtering removed records with adjusted ages below 20.

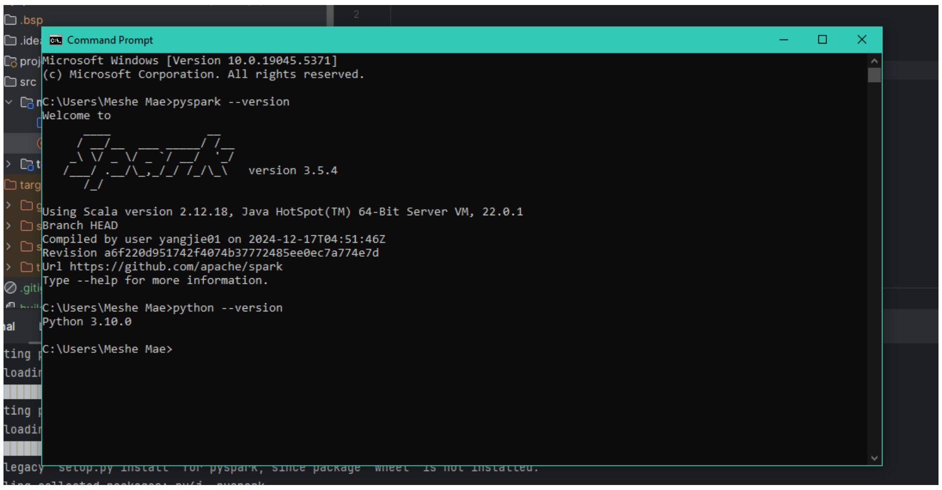
- Sorting ensured a structured presentation of results.

Challenges and Solutions

- **Memory Allocation Issue**: Initial Spark configuration required tuning to handle data efficiently. Solution: Increased driver and executor memory allocation.
- **Debugging Transformations**: Ensuring transformations are applied correctly requires using intermediate collect() statements for debugging.

Conclusion

The lab successfully demonstrated how to construct an RDD pipeline in Spark using multiple transformations. Through this process, we gained insights into key Spark operations, including mapping, filtering, reducing, and sorting. Understanding these transformations is crucial for efficient big data processing and optimization in distributed environments.



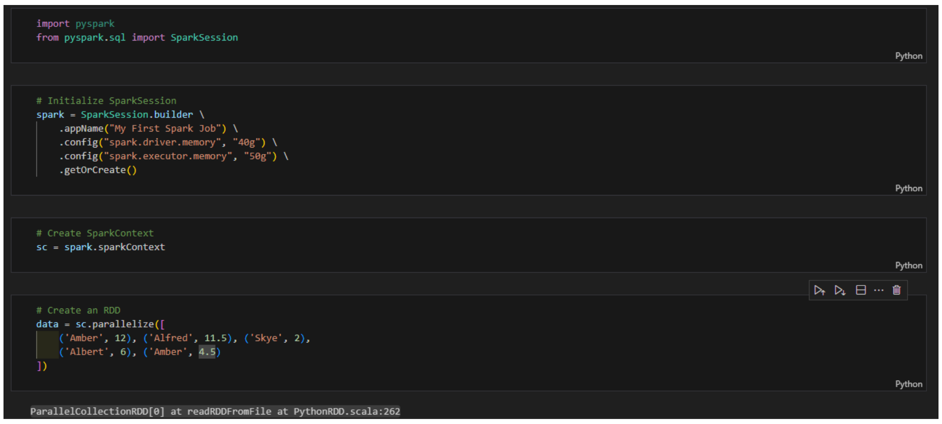
```
Microsoft Windows [Version 10.0.19045.5371]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Meshe Mae>pyspark --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.4
      /_/

Using Scala version 2.12.18, Java HotSpot(TM) 64-Bit Server VM, 22.0.1
Branch HEAD
Compiled by user yangjie01 on 2024-12-17T04:51:46Z
Revision a6f220d951742f4074b37772485ee0ec7a774e7d
Url https://github.com/apache/spark
Type --help for more information.

C:\Users\Meshe Mae>python --version
Python 3.10.0

C:\Users\Meshe Mae>
```



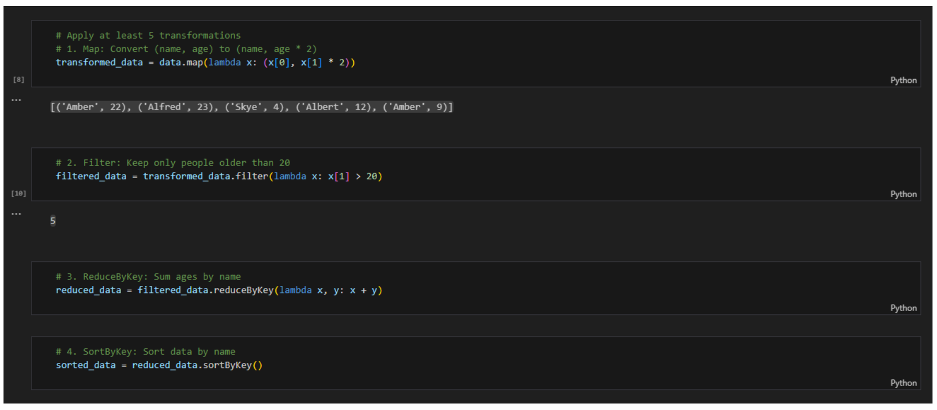
```python
import pyspark
from pyspark.sql import SparkSession
```

```python
# Initialize SparkSession
spark = SparkSession.builder \
    .appName("My First Spark Job") \
    .config("spark.driver.memory", "40g") \
    .config("spark.executor.memory", "50g") \
    .getOrCreate()
```

```python
# Create SparkContext
sc = spark.sparkContext
```

```python
# Create an RDD
data = sc.parallelize([
    ('Amber', 12), ('Alfred', 11.5), ('Skye', 2),
    ('Albert', 6), ('Amber', 4.5)
])
```

```
ParallelCollectionRDD[0] at readRDDFromFile at PythonRDD.scala:262
```



```python
# Apply at least 5 transformations
# 1. Map: Convert (name, age) to (name, age * 2)
transformed_data = data.map(lambda x: (x[0], x[1] * 2))
```

```
[('Amber', 22), ('Alfred', 23), ('Skye', 4), ('Albert', 12), ('Amber', 9)]
```

```python
# 2. Filter: Keep only people older than 20
filtered_data = transformed_data.filter(lambda x: x[1] > 20)
```

```
5
```

```python
# 3. ReduceByKey: Sum ages by name
reduced_data = filtered_data.reduceByKey(lambda x, y: x + y)
```

```python
# 4. SortByKey: Sort data by name
sorted_data = reduced_data.sortByKey()
```