BDA Assignment 9

Name Devansh Thakkar
Srn no: 202201187
Roll no: 25
TY-A

Problem Statement: Group and perform aggregate functions on columns in a Spark Dataframe

```
// Import Spark session and functions
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

val spark = SparkSession.builder().appName("GroupByExample").master("local[*]").getOrCreate()
import spark.implicits._

// Create the DataFrame
val df = Seq(
  ("HR", "Alice", 5000),
  ("HR", "Bob", 4500),
  ("IT", "Charlie", 6000),
  ("IT", "David", 6500),
  ("Sales", "Eve", 7000),
  ("Sales", "Frank", 7200)
).toDF("department", "employee", "salary")

// Group by department and calculate total and average salary
```

```
val aggregatedDf = df.groupBy("department")
  .agg(
    sum("salary").as("total_salary"),
    avg("salary").as("avg_salary")
  )

// Show the result
aggregatedDf.show()

:quit
```

CODE & OUTPUT:

1) **Step 1: Import Required Libraries**

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import col, sum, avg, max, min, count
```

- :

2) **Step 2: Create a SparkSession**

```
>>> # Initialize Spark session
>>> spark = SparkSession.builder.master("local").appName("Aggregation Example").
getOrCreate()
24/11/14 10:21:22 WARN SparkSession: Using an existing Spark session; only runti
me SQL configurations will take effect.
```

- SparkSession.builder: Starts the process of building a SparkSession. We specify the master node as "local" (meaning it runs on a local machine) and give the session an app name ("Aggregation Example").
- getOrCreate(): Creates the SparkSession if it doesn't exist, or retrieves the existing one if it does.

3) **Step 3: Create a Sample DataFrame**

```
>>> data = [
...     ("Alice", "HR", 3000),
...     ("Bob", "Finance", 4000),
...     ("Alice", "HR", 3500),
...     ("Bob", "Finance", 4200),
...     ("Charlie", "IT", 5000),
...     ("Charlie", "IT", 5200)
... ]
>>> columns = ["Name", "Department", "Salary"]
>>> df = spark.createDataFrame(data, columns)
```

- **data**: A list of tuples, where each tuple contains data for a row. Each row has three values: Name, Department, and Salary.
- **columns**: A list of column names, which correspond to the data in the rows.
- **spark.createDataFrame(data, columns)**: This method converts the list of tuples data into a DataFrame with the specified column names.

4) **Step 4: Perform GroupBy and Aggregate Operations**

```
>>> # Perform aggregation
>>> result = df.groupBy("Department").agg(
...     sum("Salary").alias("Total_Salary"),
...     avg("Salary").alias("Average_Salary"),
...     max("Salary").alias("Max_Salary"),
...     min("Salary").alias("Min_Salary"),
...     count("Salary").alias("Count")
... )
```

- **groupBy("Department")**: This groups the DataFrame by the "Department" column. It will create separate groups for each unique department value (HR, Finance, IT).
- **agg()**: The aggregation function. Inside it, we define what kind of aggregation we want for each column:
    - **sum("Salary")**: Adds up the salary values for each group (department).
    - **avg("Salary")**: Calculates the average salary for each department.
    - **max("Salary")**: Finds the highest salary in each department.
    - **min("Salary")**: Finds the lowest salary in each department.
    - **count("Salary")**: Counts the number of rows (salaries) in each department.
- **alias()**: Renames the results of the aggregation for better readability. For example, sum("Salary") is renamed to Total_Salary, avg("Salary") is renamed to Average_Salary, and so on.

5) **Step 5: Show the Results**

```
>>> result.show()
+----------+------------+--------------+----------+----------+-----+
|Department|Total_Salary|Average_Salary|Max_Salary|Min_Salary|Count|
+----------+------------+--------------+----------+----------+-----+
|        HR|        6500|        3250.0|      3500|      3000|    2|
|   Finance|        8200|        4100.0|      4200|      4000|    2|
|        IT|       10200|        5100.0|      5200|      5000|    2|
+----------+------------+--------------+----------+----------+-----+
```

- **show()**: This command displays the resulting DataFrame, which contains the aggregated values grouped by the "Department".

pip3 install pyspark

nano spark_script.py

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import sum, avg, max, min, count

# Create a SparkSession
spark = SparkSession.builder \
    .appName("Aggregation Example") \
    .master("local[*]") \
    .getOrCreate()

# Create a DataFrame
data = [
    ("Alice", "HR", 5000),
    ("Bob", "HR", 4500),
    ("Charlie", "Finance", 6000),
    ("David", "Finance", 5500),
```

```
    ("Eve", "IT", 7000),
    ("Frank", "IT", 7200)
]
columns = ["Name", "Department", "Salary"]

df = spark.createDataFrame(data, columns)
df.show()

# Perform Aggregations
aggregated_df = df.groupBy("Department").agg(
    sum("Salary").alias("Total Salary"),
    avg("Salary").alias("Average Salary"),
    max("Salary").alias("Max Salary"),
    min("Salary").alias("Min Salary"),
    count("Salary").alias("Number of Employees")
)
aggregated_df.show()

python3 spark_script.py
```