**Lab 3 — Aggregation Operations **

GroupBy, Count, Sum & Basic Aggregations in Databricks

Author: Sandeep Kumar Sharma

🎯Learning Objective

This lab teaches absolute beginners how to perform **very simple aggregation operations** in Databricks using **pure Python with PySpark DataFrames — without importing the SQL functions library (**from pyspark.sql import functions as F**)**.

We will only use:

- `groupBy()`
- Basic built-in aggregation strings like `"sum"`, `"count"`, `"avg"`, `"max"`, `"min"`

No SQL, no `F.sum()`, no `F.avg()` — completely clean and beginner friendly.

What You Will Learn (Learning Outcomes)

By the end of this lab, you will be able to:

- Create a small dataset inside Databricks
- Understand the idea of grouping and summarizing
- Perform simple aggregations without importing any SQL modules
- Display and interpret the summarized output

Dataset Used (Created Directly in Notebook)

We will create a tiny dataset representing simple purchases made by customers.

Dataset — purchases

customer_id	product	amount
1	Laptop	50000
1	Mouse	800
2	Keyboard	1500

customer_id	product	amount
2	Mouse	700
3	Laptop	52000
3	Laptop Bag	2500
3	Mouse	900

Step-by-Step Hands-On Lab

Step 1 — Create the Sample DataFrame

```
from pyspark.sql import Row

purchase_data = [
    Row(customer_id=1, product="Laptop", amount=50000),
    Row(customer_id=1, product="Mouse", amount=800),
    Row(customer_id=2, product="Keyboard", amount=1500),
    Row(customer_id=2, product="Mouse", amount=700),
    Row(customer_id=3, product="Laptop", amount=52000),
    Row(customer_id=3, product="Laptop Bag", amount=2500),
    Row(customer_id=3, product="Mouse", amount=900)
]

df_purchase = spark.createDataFrame(purchase_data)

display(df_purchase)
```

🧩Quick Explanation — What Is Aggregation?

Aggregation means **summarizing** data.

Example in real life:

- If a student has 5 marks, you can calculate their *total marks*, *average marks*, etc.

In data engineering:

- Total money spent by customer

- Total sales of each product
- Number of orders made per customer

This is exactly what we are going to do.

🎴Step 2 — Count Purchases per Customer (No SQL Functions)

```
count_df = df_purchase.groupBy("customer_id").agg({"amount": "count"})
display(count_df)
```

Meaning: How many purchases each customer made.

Step 3 — Total Amount Spent by Each Customer (No SQL Functions)

```
total_spend_df = df_purchase.groupBy("customer_id").agg({"amount": "sum"})
display(total_spend_df)
```

This returns a simple column with the total spend per customer.

Step 4 — Total Revenue Generated by Each Product

```
product_spend_df = df_purchase.groupBy("product").agg({"amount": "sum"})
display(product_spend_df)
```

This shows how much total revenue each product generated.

📷Step 5 — Multiple Aggregations Together (Still No SQL Functions)

We can perform multiple aggregations using a dictionary:

```
multi_agg_df = df_purchase.groupBy("customer_id").agg({
    "amount": "sum",   # total spent
    "amount": "avg",   # average spend
```

```
    # Spark applies aggregations in an ordered manner
})
```

However, Spark only accepts unique keys in a dictionary.

So we will run them separately:

Total amount per customer

```
total_df = df_purchase.groupBy("customer_id").agg({"amount": "sum"})
display(total_df)
```

Average amount per customer

```
avg_df = df_purchase.groupBy("customer_id").agg({"amount": "avg"})
display(avg_df)
```

Maximum amount per customer

```
max_df = df_purchase.groupBy("customer_id").agg({"amount": "max"})
display(max_df)
```

Minimum amount per customer

```
min_df = df_purchase.groupBy("customer_id").agg({"amount": "min"})
display(min_df)
```

This keeps everything **super simple** and avoids SQL functions completely.

🎉Lab Completed Successfully!

In this simplified aggregation lab, you learned how to:

- Create a PySpark DataFrame
- Use groupBy
- Perform count, sum, average, max, min using basic Python dictionary syntax
- Avoid importing SQL libraries entirely

This is the cleanest and simplest way to teach beginners aggregation.
