Hands-on 2 – (03/09/2024)

import pandas as pd

from datetime import datetime

# Sample sales data

data = {

       "TransactionID": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

       "CustomerID": [101, 102, 103, 101, 104, 102, 103, 104, 101, 105],

       "ProductID": [501, 502, 501, 503, 504, 502, 503, 504, 501, 505],

       "Quantity": [2, 1, 4, 3, 1, 2, 5, 1, 2, 1],

       "Price": [150.0, 250.0, 150.0, 300.0, 450.0, 250.0, 300.0, 450.0, 150.0, 550.0],

       "Date": [

           datetime(2024, 9, 1),

           datetime(2024, 9, 1),

           datetime(2024, 9, 2),

           datetime(2024, 9, 2),

           datetime(2024, 9, 3),

           datetime(2024, 9, 3),

           datetime(2024, 9, 4),

           datetime(2024, 9, 4),

           datetime(2024, 9, 5),

           datetime(2024, 9, 5)

       ]

   }

 # Create a DataFrame

df = pd.DataFrame(data)

# Save the DataFrame to a CSV file

df.to\_csv('sales\_data.csv', index=False)

print("Sample sales dataset has been created and saved as 'sales\_data.csv'.")

**output :**

Sample sales dataset has been created and saved as 'sales\_data.csv'.

### \*\*Part 2: Load and Analyze the Dataset Using PySpark\*\*

from pyspark.sql import SparkSession

# Initialize a Spark session

spark = SparkSession.builder \

    .appName("Sales Dataset Analysis") \

    .getOrCreate()

# Load the CSV file into a PySpark DataFrame

sales\_df = spark.read.csv('sales\_data.csv', header=True, inferSchema=True)

sales\_df.show(6)

output:

+-------------+----------+---------+--------+-----+----------+

|TransactionID|CustomerID|ProductID|Quantity|Price| Date|

+-------------+----------+---------+--------+-----+----------+

| 1| 101| 501| 2|150.0|2024-09-01|

| 2| 102| 502| 1|250.0|2024-09-01|

| 3| 103| 501| 4|150.0|2024-09-02|

| 4| 101| 503| 3|300.0|2024-09-02|

| 5| 104| 504| 1|450.0|2024-09-03|

| 6| 102| 502| 2|250.0|2024-09-03|

+-------------+----------+---------+--------+-----+----------+

#### \*\*Step 3: Explore the Data\*\*

# Display the schema of the DataFrame

sales\_df.printSchema()

output:

root

|-- TransactionID: integer (nullable = true)

|-- CustomerID: integer (nullable = true)

|-- ProductID: integer (nullable = true)

|-- Quantity: integer (nullable = true)

|-- Price: double (nullable = true)

|-- Date: date (nullable = true)

# Show the first 5 rows

sales\_df.show(5)

output:

+-------------+----------+---------+--------+-----+----------+

|TransactionID|CustomerID|ProductID|Quantity|Price| Date|

+-------------+----------+---------+--------+-----+----------+

| 1| 101| 501| 2|150.0|2024-09-01|

| 2| 102| 502| 1|250.0|2024-09-01|

| 3| 103| 501| 4|150.0|2024-09-02|

| 4| 101| 503| 3|300.0|2024-09-02|

| 5| 104| 504| 1|450.0|2024-09-03|

+-------------+----------+---------+--------+-----+----------+

# Get summary statistics for numeric columns

sales\_df.describe(['Quantity', 'Price']).show()

output:

+-------+-----------------+-----------------+

|summary| Quantity| Price|

+-------+-----------------+-----------------+

| count| 10| 10|

| mean| 2.2| 300.0|

| stddev|1.398411797560202|141.4213562373095|

| min| 1| 150.0|

| max| 5| 550.0|

+-------+-----------------+-----------------+

#### \*\*Step 4: Perform Data Transformations and Analysis\*\*

1. \*\*Calculate the Total Sales Value for Each Transaction:\*\*

- Add a new column called `TotalSales`, calculated by multiplying `Quantity` by `Price`.

from pyspark.sql.functions import col

# Add a new column 'TotalSales' by multiplying Quantity and Price

sales\_df = sales\_df.withColumn("TotalSales", col("Quantity") \* col("Price"))

sales\_df.show()

output:

+-------------+----------+---------+--------+-----+----------+----------+

|TransactionID|CustomerID|ProductID|Quantity|Price| Date|TotalSales|

+-------------+----------+---------+--------+-----+----------+----------+

| 1| 101| 501| 2|150.0|2024-09-01| 300.0|

| 2| 102| 502| 1|250.0|2024-09-01| 250.0|

| 3| 103| 501| 4|150.0|2024-09-02| 600.0|

| 4| 101| 503| 3|300.0|2024-09-02| 900.0|

| 5| 104| 504| 1|450.0|2024-09-03| 450.0|

| 6| 102| 502| 2|250.0|2024-09-03| 500.0|

| 7| 103| 503| 5|300.0|2024-09-04| 1500.0|

| 8| 104| 504| 1|450.0|2024-09-04| 450.0|

| 9| 101| 501| 2|150.0|2024-09-05| 300.0|

| 10| 105| 505| 1|550.0|2024-09-05| 550.0|

2. \*\*Group By ProductID and Calculate Total Sales Per Product:\*\*

- Group the data by `ProductID` and calculate the total sales for each product.

# Group by ProductID and calculate total sales per product

total\_sales\_per\_product = sales\_df.groupBy("ProductID").sum("TotalSales").alias("TotalSales")

total\_sales\_per\_product.show()

output:

+---------+---------------+

|ProductID|sum(TotalSales)|

+---------+---------------+

| 501| 1200.0|

| 504| 900.0|

| 502| 750.0|

| 505| 550.0|

| 503| 2400.0|

+---------+---------------+

3. \*\*Identify the Top-Selling Product:\*\*

- Find the product that generated the highest total sales.

# Find the product with the highest total sales

top\_selling\_product = total\_sales\_per\_product.orderBy(col("sum(TotalSales)").desc()).limit(1)

top\_selling\_product.show()

output:

+---------+---------------+

|ProductID|sum(TotalSales)|

+---------+---------------+

| 503| 2400.0|

+---------+---------------+

4. \*\*Calculate the Total Sales by Date:\*\*

- Group the data by `Date` and calculate the total sales for each day.

# Group by Date and calculate total sales for each day

total\_sales\_by\_date = sales\_df.groupBy("Date").sum("TotalSales").alias("TotalSales")

total\_sales\_by\_date.show()

output:

+----------+---------------+

| Date|sum(TotalSales)|

+----------+---------------+

|2024-09-03| 950.0|

|2024-09-01| 550.0|

|2024-09-02| 1500.0|

|2024-09-05| 850.0|

|2024-09-04| 1950.0|

+----------+---------------+

5. \*\*Filter High-Value Transactions:\*\*

- Filter the transactions to show only those where the total sales value is greater than ₹500.

# Filter transactions where the total sales value is greater than ₹500

high\_value\_transactions = sales\_df.filter(col("TotalSales") > 500)

high\_value\_transactions.show()

output:

+-------------+----------+---------+--------+-----+----------+----------+

|TransactionID|CustomerID|ProductID|Quantity|Price| Date|TotalSales|

+-------------+----------+---------+--------+-----+----------+----------+

| 3| 103| 501| 4|150.0|2024-09-02| 600.0|

| 4| 101| 503| 3|300.0|2024-09-02| 900.0|

| 7| 103| 503| 5|300.0|2024-09-04| 1500.0|

| 10| 105| 505| 1|550.0|2024-09-05| 550.0|

+-------------+----------+---------+--------+-----+----------+----------+

### \*\*Additional Challenge (Optional):

1. \*\*Identify Repeat Customers:\*\*

# Group by CustomerID and count the number of purchases for each customer

customer\_purchase\_count = sales\_df.groupBy("CustomerID").count()

# Filter customers who have made more than one purchase

repeat\_customers = customer\_purchase\_count.filter(col("count") > 1)

repeat\_customers.show()

output:

+----------+-----+

|CustomerID|count|

+----------+-----+

| 101| 3|

| 103| 2|

| 102| 2|

| 104| 2|

+----------+-----+

1. \*\*Calculate the Average Sale Price Per Product:\*\*

from pyspark.sql.functions import avg

# Group by ProductID and calculate the average price per unit

avg\_price\_per\_product = sales\_df.groupBy("ProductID").agg(avg("Price").alias("AvgPrice"))

avg\_price\_per\_product.show()

output:

|ProductID|AvgPrice|

+---------+--------+

| 501| 150.0|

| 504| 450.0|

| 502| 250.0|

| 505| 550.0|

| 503| 300.0|

+---------+--------+