ACADEMIC TASK CA-2

Project Title:

Analyzing Sales Performance Using Spark SQL

Course Name:

Cluster Computing (INT315)

Submitted by:

Name: Md Imlak

Roll No: 29

Reg: 12104074

Section: K21AK

Submitted to:

Name: Dr. Saqib Ul Sabha

UID: 32521

Assistant Professor

LPU

Date of Submission: 21/04/2025

School of Computer Science Engineering

Lovely Professional University

Punjab

DECLEARATION

I am , Md Imlak a student of Bachelor of Technology under CSE discipline at Lovely Professional University, Punjab, hereby declare that all the information furnished in this project report is based on my own work and is genuine.

Your name : Md Imlak
Registration no : 12104074

Project: Analyzing Sales Performance Using Spark SQL

Context: A retail company wants to analyze its sales records to understand performance trends across products, regions, and time.
Tasks:

1. Load the sales dataset into Spark and convert it to a DataFrame.
2. Register the DataFrame as a temporary SQL table.
3. Write a SQL query to find total revenue generated by each product category.
4. Calculate monthly sales totals across different regions.
5. Identify the top 3 best-selling products based on quantity sold.
6. Visualize monthly sales performance using a line chart.
7. Calculate the average price of products in each category (using a subquery).
8. Find the region with the highest total revenue (using a subquery).
9. Visualize the total revenue by product category using a bar chart.
10. Find products with revenue greater than the average revenue.

Table Contents:

1. Objective:

The objective of this project is to leverage Apache Spark SQL to analyze retail sales data, focusing on key performance indicators such as product category revenue, regional sales trends, monthly sales variations, average product prices, and individual product performance relative to average revenue. This analysis aims to provide actionable insights into sales performance.

2. Tools and Technologies used:

- Apache Spark (version 3.5.4)
- Scala (version 2.12.18)
- Dataset used: project_sales_data.csv
- Environment: Windows PowerShell
- Python (3.11.5) with Pandas and Matplotlib libraries.

3. Dataset Description:

- Dataset type: CSV (Comma Separated Values)
- Number of rows: 161
- Number of columns: 8

- **Explanation of key columns:**
  - `OrderID`: Unique identifier for each order.
  - `Product`: Name of the product sold.
  - `Category`: Category of the product (e.g., Electronics, Clothing).
  - `Region`: Geographical region of the sale (e.g., North, South, East, West).
  - `Quantity`: Number of units sold.
  - `Price`: Price per unit.
  - `OrderDate`: Date of the order (DD-MM-YYYY).
  - `Revenue`: Total revenue generated from the order (Quantity * Price).

4. Implementation Steps:

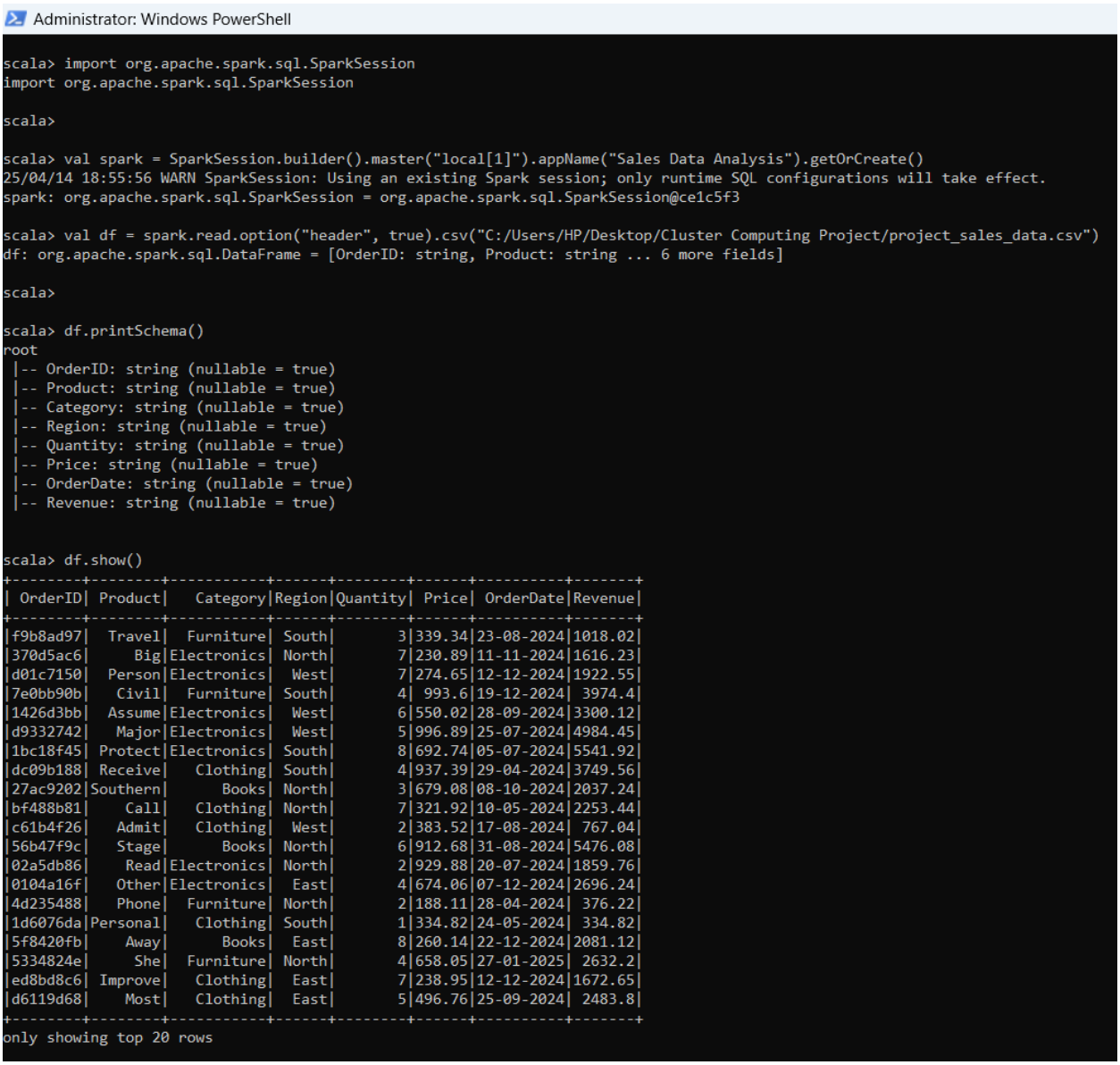
Task 1: Load the sales dataset into Spark and convert it to a DataFrame.

Explanation: This code initializes a SparkSession and loads the CSV file into a DataFrame. printSchema() displays the DataFrame's schema, and show() displays sample data.

Code:

```
import org.apache.spark.sql.SparkSession

val spark = SparkSession.builder().master("local[1]").appName("Sales Data Analysis").getOrCreate()
val df = spark.read.option("header", true).csv("C:/Users/HP/Desktop/Cluster Computing Project/project_sales_data.csv")

df.printSchema()
df.show()
```

Screenshot output:



```
Administrator: Windows PowerShell

scala> import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.SparkSession

scala>

scala> val spark = SparkSession.builder().master("local[1]").appName("Sales Data Analysis").getOrCreate()
25/04/14 18:55:56 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
spark: org.apache.spark.sql.SparkSession = org.apache.spark.sql.SparkSession@ce1c5f3

scala> val df = spark.read.option("header", true).csv("C:/Users/HP/Desktop/Cluster Computing Project/project_sales_data.csv")
df: org.apache.spark.sql.DataFrame = [OrderID: string, Product: string ... 6 more fields]

scala>

scala> df.printSchema()
root
 |-- OrderID: string (nullable = true)
 |-- Product: string (nullable = true)
 |-- Category: string (nullable = true)
 |-- Region: string (nullable = true)
 |-- Quantity: string (nullable = true)
 |-- Price: string (nullable = true)
 |-- OrderDate: string (nullable = true)
 |-- Revenue: string (nullable = true)


scala> df.show()
+--------+--------+-----------+------+--------+------+----------+-------+
| OrderID| Product|   Category|Region|Quantity| Price| OrderDate|Revenue|
+--------+--------+-----------+------+--------+------+----------+-------+
|f9b8ad97|  Travel|  Furniture| South|       3|339.34|23-08-2024|1018.02|
|370d5ac6|     Big|Electronics| North|       7|230.89|11-11-2024|1616.23|
|d01c7150|  Person|Electronics|  West|       7|274.65|12-12-2024|1922.55|
|7e0bb90b|   Civil|  Furniture| South|       4| 993.6|19-12-2024| 3974.4|
|1426d3bb|  Assume|Electronics|  West|       6|550.02|28-09-2024|3300.12|
|d9332742|   Major|Electronics|  West|       5|996.89|25-07-2024|4984.45|
|1bc18f45| Protect|Electronics| South|       8|692.74|05-07-2024|5541.92|
|dc09b188| Receive|   Clothing| South|       4|937.39|29-04-2024|3749.56|
|27ac9202|Southern|      Books| North|       3|679.08|08-10-2024|2037.24|
|bf488b81|    Call|   Clothing| North|       7|321.92|10-05-2024|2253.44|
|c61b4f26|   Admit|   Clothing|  West|       2|383.52|17-08-2024| 767.04|
|56b47f9c|   Stage|      Books| North|       6|912.68|31-08-2024|5476.08|
|02a5db86|    Read|Electronics| North|       2|929.88|20-07-2024|1859.76|
|0104a16f|   Other|Electronics|  East|       4|674.06|07-12-2024|2696.24|
|4d235488|   Phone|  Furniture| North|       2|188.11|28-04-2024| 376.22|
|1d6076da|Personal|   Clothing| South|       1|334.82|24-05-2024| 334.82|
|5f8420fb|    Away|      Books|  East|       8|260.14|22-12-2024|2081.12|
|5334824e|     She|  Furniture| North|       4|658.05|27-01-2025| 2632.2|
|ed8bd8c6| Improve|   Clothing|  East|       7|238.95|12-12-2024|1672.65|
|d6119d68|    Most|   Clothing|  East|       5|496.76|25-09-2024| 2483.8|
+--------+--------+-----------+------+--------+------+----------+-------+
only showing top 20 rows
```

Output: The schema and sample data were displayed.

Task 2: Register the DataFrame as a temporary SQL table.
Explanation: The DataFrame was registered as a temporary SQL table named "sales" for SQL querying.

Code:

```
df.createOrReplaceTempView("sales")
```

Output: No visible output, but the table "sales" was created.

Task 3: Write a SQL query to find total revenue generated by each product category.
Explanation: A SQL query was used to calculate the sum of revenue per category.

Code:

```
val revenueByCategory = spark.sql(""" SELECT Category, SUM(Revenue) AS TotalRevenue FROM sales GROUP BY Category ORDER BY TotalRevenue DESC """)
revenueByCategory.show()
```

Screenshot output:

```
scala> val revenueByCategory = spark.sql(""" SELECT Category, SUM(Revenue) AS TotalRevenue FROM sales GROUP BY Category ORDER BY TotalRevenue DESC """)
revenueByCategory: org.apache.spark.sql.DataFrame = [Category: string, TotalRevenue: double]

scala> revenueByCategory.show()
+-----------+------------------+
|   Category|      TotalRevenue|
+-----------+------------------+
|   Clothing|126834.02999999998|
|      Books|121005.02999999998|
|Electronics|         111527.88|
|  Furniture|107058.15000000001|
+-----------+------------------+
```

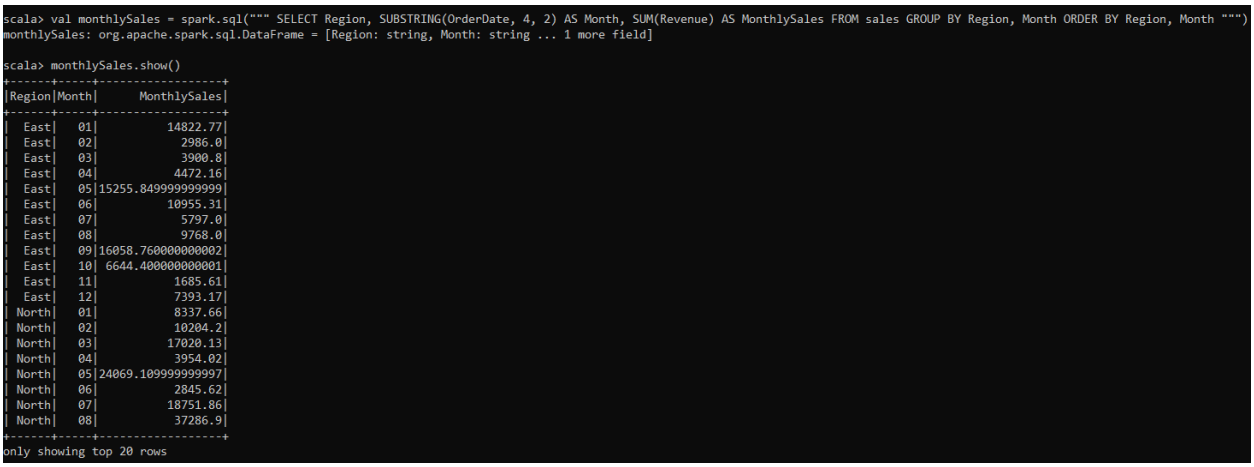
Output: Displayed total revenue by category.

Task 4: Calculate monthly sales totals across different regions.
Explanation: A SQL query was used to calculate monthly sales for each region.

Code:

```
val monthlySales = spark.sql(""" SELECT Region, SUBSTRING(OrderDate, 4, 2) AS Month, SUM(Revenue) AS MonthlySales FROM sales GROUP BY Region, Month ORDER BY Region, Month """)
monthlySales.show()
```

Screenshot output:



```
scala> val monthlySales = spark.sql(""" SELECT Region, SUBSTRING(OrderDate, 4, 2) AS Month, SUM(Revenue) AS MonthlySales FROM sales GROUP BY Region, Month ORDER BY Region, Month """)
monthlySales: org.apache.spark.sql.DataFrame = [Region: string, Month: string ... 1 more field]

scala> monthlySales.show()
+------+-----+------------------+
|Region|Month|      MonthlySales|
+------+-----+------------------+
|  East|   01|          14822.77|
|  East|   02|            2986.0|
|  East|   03|            3900.8|
|  East|   04|          4472.16|
|  East|   05|15255.849999999999|
|  East|   06|          10955.31|
|  East|   07|            5797.0|
|  East|   08|            9768.0|
|  East|   09|16058.760000000002|
|  East|   10| 6644.400000000001|
|  East|   11|           1685.61|
|  East|   12|           7393.17|
| North|   01|           8337.66|
| North|   02|           10204.2|
| North|   03|          17020.13|
| North|   04|           3954.02|
| North|   05|24069.109999999997|
| North|   06|           2845.62|
| North|   07|          18751.86|
| North|   08|           37286.9|
+------+-----+------------------+
only showing top 20 rows
```

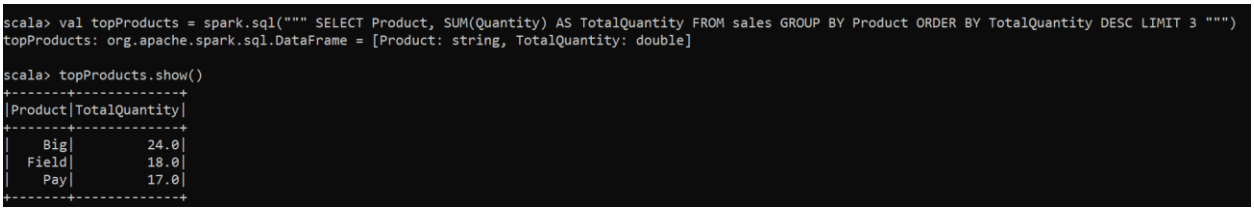
Output: Displayed monthly sales by region.

Task 5: Identify the top 3 best-selling products based on quantity sold.
Explanation: A SQL query was used to find the top 3 products with the highest quantity sold.

Code:

```
val topProducts = spark.sql(""" SELECT Product, SUM(Quantity) AS TotalQuantity
FROM sales GROUP BY Product ORDER BY TotalQuantity DESC LIMIT 3 """)
topProducts.show()
```

Snapshot output:



```
scala> val topProducts = spark.sql(""" SELECT Product, SUM(Quantity) AS TotalQuantity FROM sales GROUP BY Product ORDER BY TotalQuantity DESC LIMIT 3 """)
topProducts: org.apache.spark.sql.DataFrame = [Product: string, TotalQuantity: double]

scala> topProducts.show()
+-------+-------------+
|Product|TotalQuantity|
+-------+-------------+
|    Big|         24.0|
|  Field|         18.0|
|    Pay|         17.0|
+-------+-------------+
```

Output: Displayed the top 3 products.

Task 6: Visualize monthly sales performance using a line chart.
Explanation: Monthly sales data was visualized using a line chart in Python.

Code (Scala and Python):

- **Scala (Saving to CSV):**

```
monthlySales.coalesce(1).write.option("header",
"true").csv("C:/Users/HP/Desktop/Cluster Computing Project/monthly_sales.csv")
```

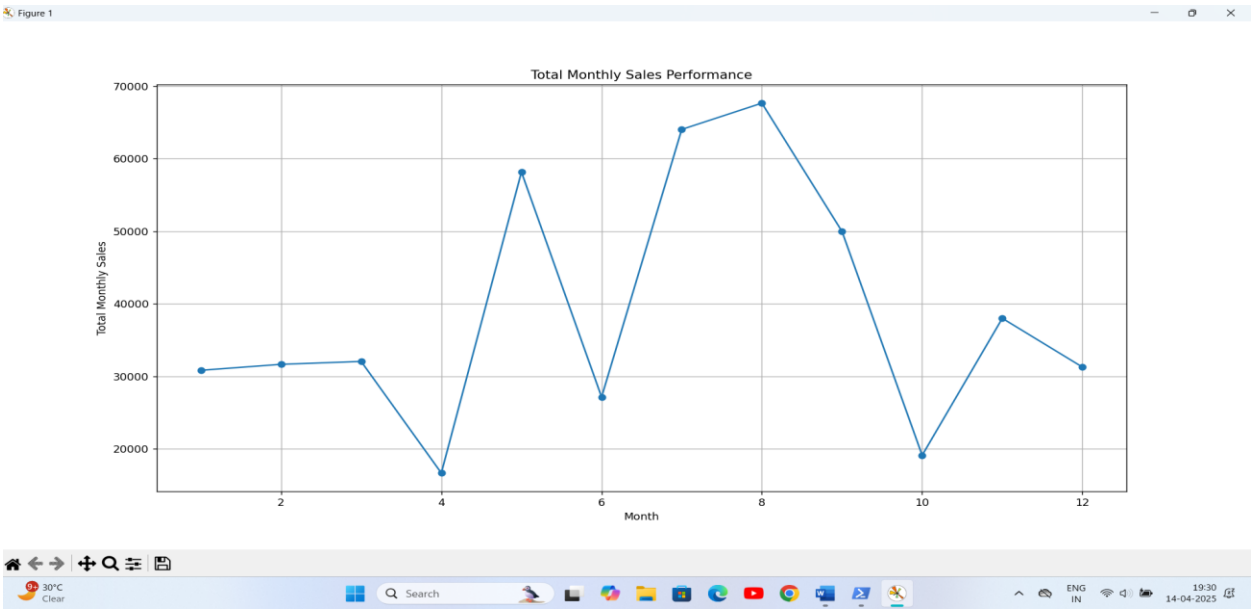
Python Code for Visualization:

```
import pandas as pd
import glob
import matplotlib.pyplot as plt

file_list = glob.glob("C:/Users/HP/Desktop/Cluster Computing
Project/monthly_sales.csv/part-*.csv")

if file_list:
    df = pd.read_csv(file_list[0])
    monthly_totals = df.groupby("Month")["MonthlySales"].sum().reset_index()
    plt.figure(figsize=(10, 6))
    plt.plot(monthly_totals["Month"], monthly_totals["MonthlySales"], marker="o")
    plt.title("Total Monthly Sales Performance")
    plt.xlabel("Month")
    plt.ylabel("Total Monthly Sales")
    plt.grid(True)
    plt.show()
else:
    print("No part files found!")
```

Explanation: This code saves the monthly sales data to a CSV and then uses Python's matplotlib to create a line chart.

Output: A line chart showing monthly sales Performance.

Screenshot output:



Task 7: Calculate the average price of products in each category (using a subquery).
Explanation: A subquery was used to select category and price, and then the average price per category was calculated.

Code:

```
val avgPriceByCategory = spark.sql(""" SELECT Category, AVG(Price) AS AveragePrice FROM (SELECT Category, Price FROM sales) AS product_prices GROUP BY Category ORDER BY AveragePrice DESC """)
avgPriceByCategory.show()
```

Snapshot output:

```
scala> val avgPriceByCategory = spark.sql(""" SELECT Category, AVG(Price) AS AveragePrice FROM (SELECT Category, Price FROM sales) AS product_prices GROUP BY Category ORDER BY AveragePrice DESC """)
avgPriceByCategory: org.apache.spark.sql.DataFrame = [Category: string, AveragePrice: double]

scala> avgPriceByCategory.show()
+-----------+-----------------+
|   Category|     AveragePrice|
+-----------+-----------------+
|  Furniture|621.8505714285715|
|Electronics| 569.821142857143|
|      Books|545.9354545454546|
|   Clothing|493.2893478260869|
+-----------+-----------------+
```

Output: Average price by category.

Task 8: Find the region with the highest total revenue (using a subquery).
Explanation: Subqueries were used to find the maximum revenue region.

Code:

```
val highestRevenueRegion = spark.sql(""" SELECT Region, TotalRevenue FROM (SELECT Region, SUM(Revenue) AS TotalRevenue FROM sales GROUP BY Region) AS region_revenue WHERE TotalRevenue = (SELECT MAX(TotalRevenue) FROM (SELECT SUM(Revenue) AS TotalRevenue FROM sales GROUP BY Region) AS max_revenue) """)
highestRevenueRegion.show()
```

Snapshot output:

```
scala> val highestRevenueRegion = spark.sql(""" SELECT Region, TotalRevenue FROM (SELECT Region, SUM(Revenue) AS TotalRevenue FROM sales GROUP BY Region) AS region_revenue WHERE TotalRevenue = (SELECT MAX(TotalR
evenue) FROM (SELECT SUM(Revenue) AS TotalRevenue FROM sales GROUP BY Region) AS max_revenue) """)
highestRevenueRegion: org.apache.spark.sql.DataFrame = [Region: string, TotalRevenue: double]

scala> highestRevenueRegion.show()
+------+-----------------+
|Region|     TotalRevenue|
+------+-----------------+
| North|161605.90999999995|
+------+-----------------+
```

Output: Region with highest revenue.

Task 9: Visualize the total revenue by product category using a bar chart.
Explanation: Revenue by category was visualized using a bar chart in Python.
Code (Scala and Python):

- **Scala (Saving to CSV):**

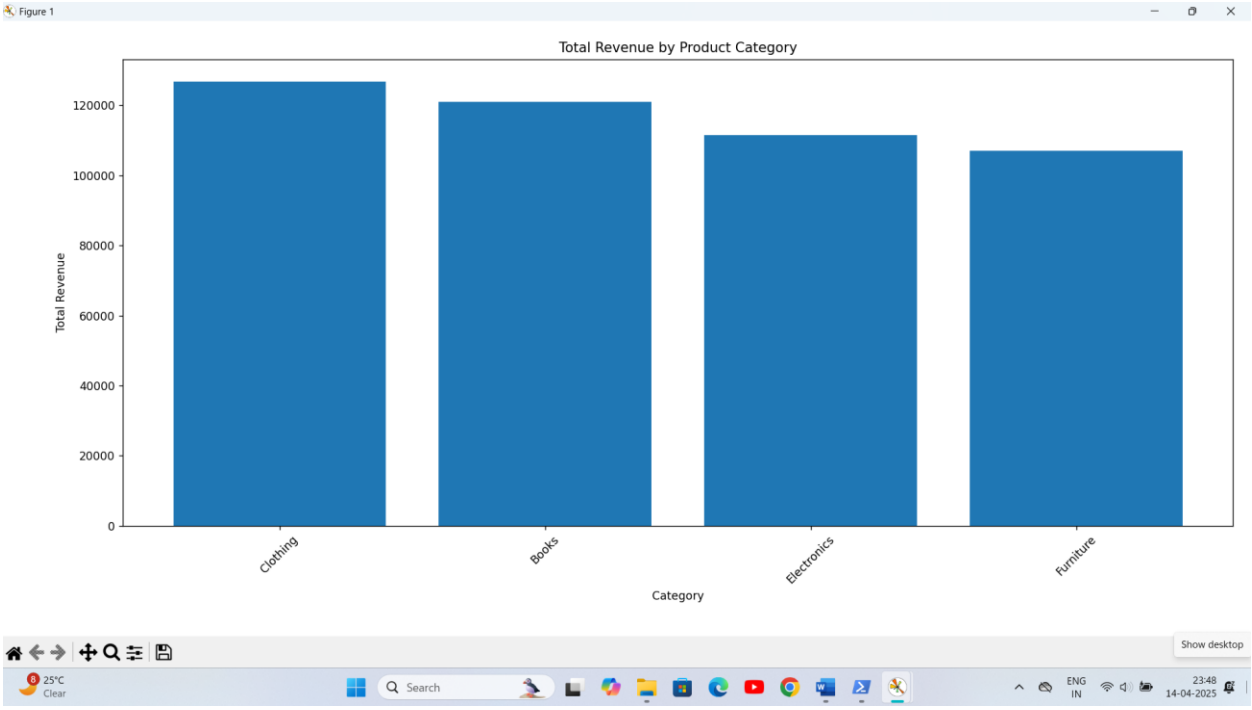
```
revenueByCategory.coalesce(1).write.option("header",
"true").csv("C:/Users/HP/Desktop/Cluster Computing Project/revenue_by_category.csv")
```

Python Code for Visualization:

```
import pandas as pd
import glob
import matplotlib.pyplot as plt

file_list = glob.glob("C:/Users/HP/Desktop/Cluster Computing
Project/revenue_by_category.csv/part-*.csv")

if file_list:
    df = pd.read_csv(file_list[0])
    plt.figure(figsize=(10, 6))
    plt.bar(df["Category"], df["TotalRevenue"])
    plt.title("Total Revenue by Product Category")
    plt.xlabel("Category")
    plt.ylabel("Total Revenue")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
else:
    print("No part files found!")
```

Explanation: This code generates a bar chart comparing total revenue across product categories.
Output: A bar chart showing revenue by category.

Snapshot :



Task 10: Find products with revenue greater than the average revenue.

Explanation: A subquery was used to find the average revenue, and then products with revenue greater than that were displayed.

Code:

```
val productsAboveAvgRevenue = spark.sql(""" SELECT Product, Revenue FROM sales WHERE Revenue > (SELECT AVG(Revenue) FROM sales) """)
productsAboveAvgRevenue.show()
```

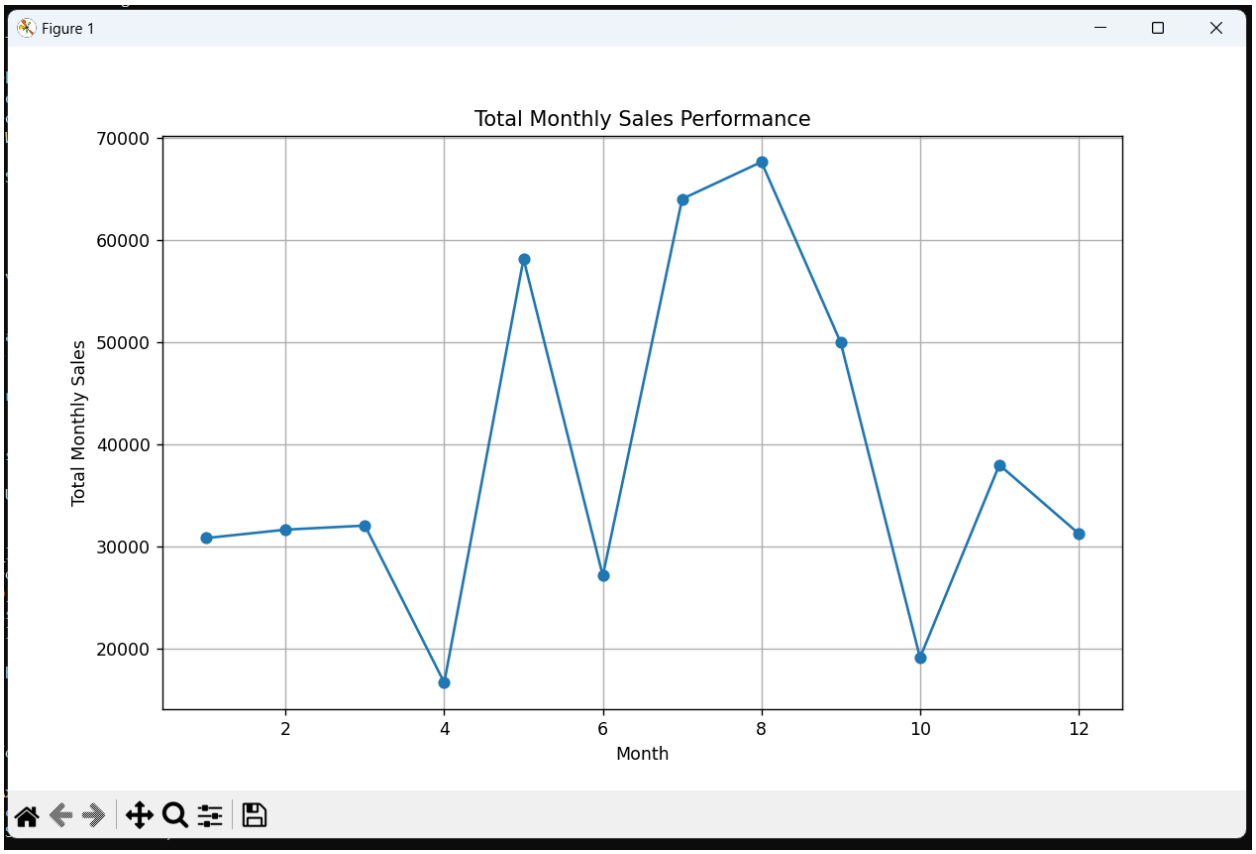
Screenshot output:

```
scala> val productsAboveAvgRevenue = spark.sql(""" SELECT Product, Revenue FROM sales WHERE Revenue > (SELECT AVG(Revenue) FROM sales) """)
productsAboveAvgRevenue: org.apache.spark.sql.DataFrame = [Product: string, Revenue: string]

scala> productsAboveAvgRevenue.show()
+-----------+-------+
|    Product|Revenue|
+-----------+-------+
|      Civil| 3974.4|
|     Assume|3300.12|
|      Major|4984.45|
|    Protect|5541.92|
|    Receive|3749.56|
|      Stage|5476.08|
|  Education|   2986|
|   Behavior|4536.75|
|     Anyone| 3050.6|
|       Wear|4706.91|
|Information|3183.18|
|  Determine|7038.48|
|   Behavior|5802.08|
|     Always|7698.64|
|     Myself|5047.26|
|   Indicate|8360.55|
|     Health|4895.45|
|     Before| 3860.1|
|Interesting|4472.16|
|         As|2945.07|
+-----------+-------+
only showing top 20 rows
```

Output: Products with revenue above average.

5. Final Visualizations:

Screenshot: Line chart

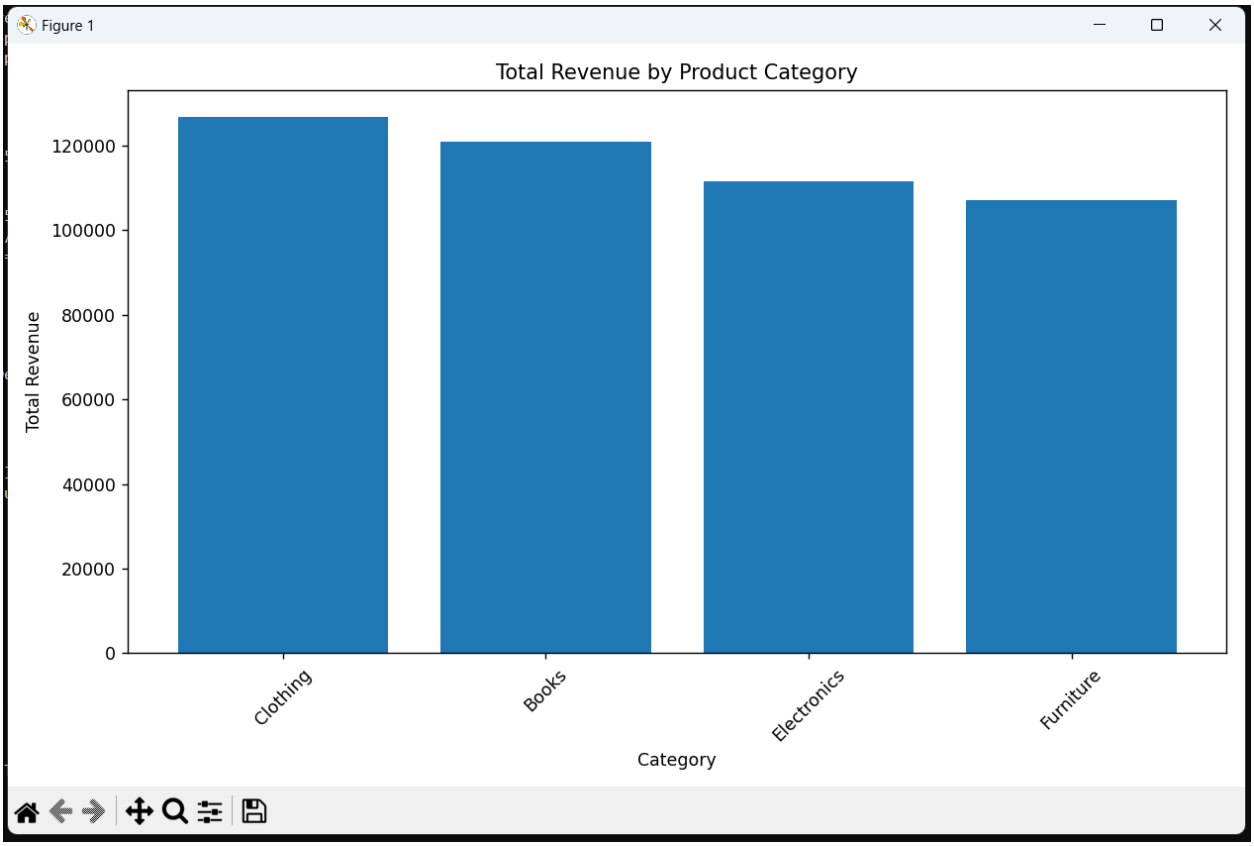


Explanation of Visualizations:

Line Chart: Total Monthly Sales Performance.

- This line chart visually represents the trend of total monthly sales over the analyzed period. The x-axis displays the months, and the y-axis represents the corresponding total sales revenue.
- The chart effectively illustrates the fluctuations in sales performance across different months. Peaks and troughs in the line indicate periods of high and low sales, respectively. This visualization helps to identify seasonal trends, monthly variations, and overall sales patterns, which are crucial for strategic business decisions.
- For example, from the provided data, we can see the sales for month 5 is very high. This type of information is very valuable to the business.

Screenshot: Bar chart



Explanation of Visualizations:

Bar Chart: Total Revenue by Product Category .

- This bar chart provides a comparative analysis of the total revenue generated by each product category. The x-axis represents the product categories (e.g., Clothing, Books, Electronics, Furniture), and the y-axis represents the total revenue for each category.
- The height of each bar corresponds to the total revenue earned from that particular category, allowing for a quick and easy comparison of the revenue performance of different product lines. This visualization helps to identify the most and least profitable product categories, enabling the company to focus on high-performing categories and address issues in underperforming ones.
- For example, we can see that the clothing category, and books category, creates the most revenue.

Key Takeaways from the Visualizations:

- **Trend Analysis:** The line chart helps in understanding the temporal patterns of sales, which can be correlated with marketing campaigns, seasonal changes, or economic factors.

- **Comparative Analysis:** The bar chart provides a direct comparison of category-wise revenue, which is essential for product portfolio management and resource allocation.
- **Decision Support:** Both charts offer valuable insights that can support data-driven decision-making in areas such as inventory management, sales forecasting, and marketing strategies.

6. Challenges Faced:

- **Challenge:** Issues with Python library installations in the PySpark environment.
- **Resolution:** Ensured libraries were installed correctly and accessible in the PySpark environment.
- **Challenge:** File path issues when reading and writing CSV files.
- **Resolution:** Used absolute paths and glob module to handle part file issues.

7. Conclusion:

This project demonstrated the use of Apache Spark SQL for advanced sales data analysis. Subqueries allowed for complex queries, and visualizations provided clear insights. The project enhanced understanding of Spark SQL and data visualization techniques.

8. References:

- Apache Spark Documentation: spark.apache.org/docs/latest/
- Pandas Documentation: pandas.pydata.org/docs/
- Matplotlib Documentation: matplotlib.org/stable/contents.html


THANK YOU!