Business Optimization Analytics with PySpark - Tech Titans

Chandu Nelavelli, Abdul Raheem Syed, Venkata Keerthi Kamani, Shruthi Chinthakayala

December 2, 2023

Project Idea

This project aims to integrate and analyze sales, customer, and product data through PySpark within Jupyter Lab. The objective is to uncover meaningful insights, patterns, and trends that inform data-driven decision-making. The analysis includes exploratory data analysis (EDA), feature engineering, and visualization, contributing to a holistic understanding of the business environment.

Tools and Technologies

- PySpark for distributed data processing
- Jupyter Lab for interactive data exploration and visualization
- Matplotlib for data visualization

Architecture Diagram

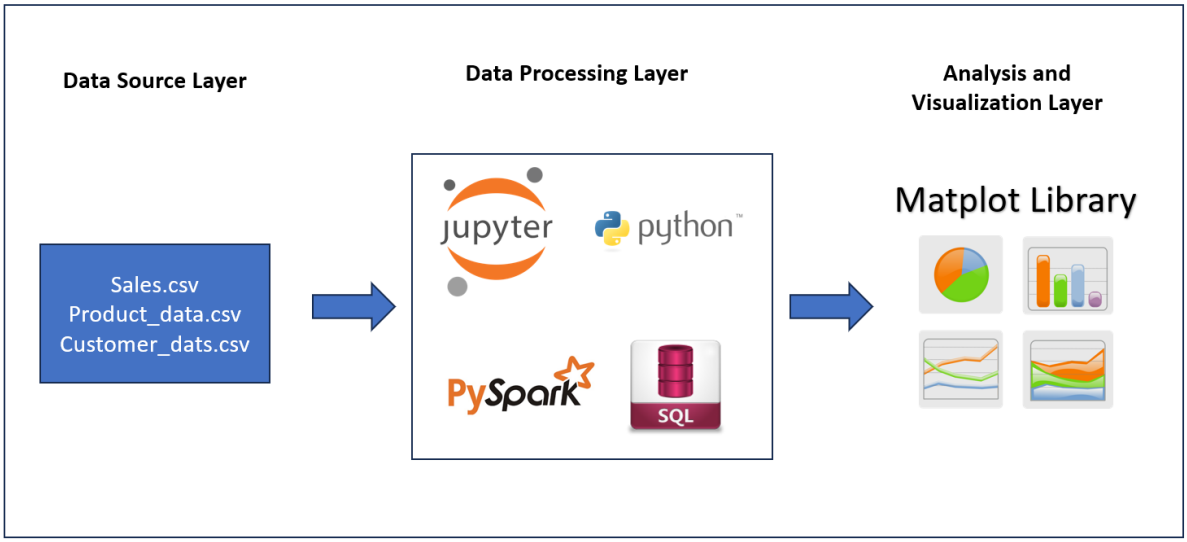


Figure 1: high-level architecture or methodology

Explanation of Diagram

Data Source Layer

- Information comes from three CSV files: sales transactions (`sales.csv`), product details (`product_data.csv`), and customer information (`customer_data.csv`).

Data Processing Layer

- PySpark handles large-scale data processing, making it efficient for analysis.
- Data cleaning techniques ensure accurate and reliable data.
- Spark SQL is used for querying and analyzing data, treating it like a database.

Analysis and Visualization Layer

- Jupyter Lab provides an interactive environment for data exploration and analysis.
- Matplotlib and SQL are used within Jupyter Lab for effective data visualization and analysis.

Goals

1. Calculate total sales amount by product
2. Calculate Average Sale Amount by City
3. Identify Top 5 Products by Total Sales
4. Count the Number of Sales per Customer
5. Explore Product Categories with Lowest Sales
6. Identify High-Value Customers
7. Calculate Total Sales Amount by Category
8. Calculate Total Sales Amount by Customer

PySpark Analytics Implementation

Step 1: Setup and Environment

```
from pyspark.sql import SparkSession
import matplotlib.pyplot as plt

spark = SparkSession.builder.getOrCreate()
```

Explanation:

1. **Import Libraries:**
   - pyspark.sql.SparkSession: Used to create a Spark session, the entry point to Spark functionality.
   - matplotlib.pyplot: Used for data visualization.
2. **Initialize Spark Session:**
   - SparkSession.builder.getOrCreate(): Initializes a Spark session. If a session already exists, it is reused.

Step 2: Load Data

```
sales_data = spark.read.format('csv').option('header', 'true').load('sales_data.csv')
customer_data = spark.read.format('csv').option('header', 'true').load('customer_data.csv')
product_data = spark.read.format('csv').option('header', 'true').load('product_data.csv')
```

Explanation:

1. **Read CSV Files:**
   - `spark.read.format('csv')`: Specifies CSV format.
   - `.option('header', 'true')`: Specifies that the first row contains headers.
   - `.load('sales_data.csv')`: Loads CSV into Spark DataFrame (`sales_data`). Similar steps for `customer_data` and `product_data`.

Step 3: Merge Datasets

```
# Merge datasets based on common columns
merged_data = sales_data.join(customer_data, 'CustomerID', 'inner') \
                        .join(product_data, 'ProductID', 'inner')
merged_data.createOrReplaceTempView("combined_data")
```

Explanation:

1. **Inner Joins:**
   - `.join(customer_data, 'CustomerID', 'inner')`: Inner join between `sales_data` and `customer_data` on 'CustomerID'.
   - `.join(product_data, 'ProductID', 'inner')`: Joins the result with `product_data` on 'ProductID'.
   - Result is `merged_data` DataFrame.
2. **Create Temporary View:**
   - `merged_data.createOrReplaceTempView("combined_data")`: Creates a temporary SQL view "combined_data" for SQL queries.

Step 4: Start generating goals

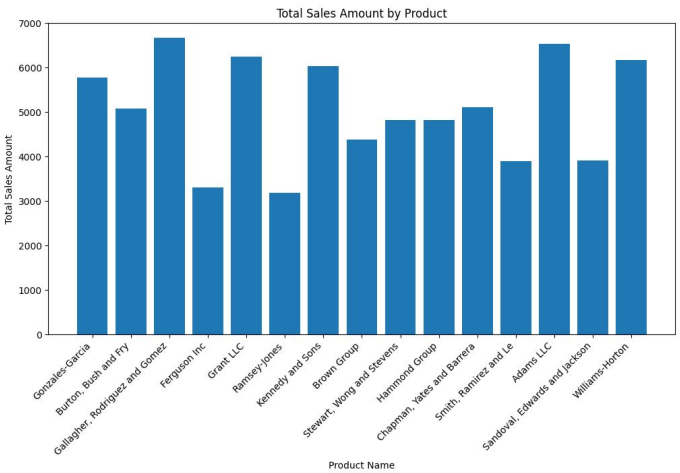
Goal 1: Calculate Total Sales Amount by Product

Results: The code calculates the total sales amount for each product and visualizes the top 15 products with the highest total sales.

```
# Goal 1: Calculate total sales amount by product
total_sales_by_product = spark.sql("""
    SELECT ProductName, SUM(SaleAmount) AS TotalSales
    FROM combined_data
    GROUP BY ProductName limit 15
""")
total_sales_by_product.show()


total_sales_pd = total_sales_by_product.toPandas()
plt.figure(figsize=(12, 6))
plt.bar(total_sales_pd['ProductName'], total_sales_pd['TotalSales'])
plt.title('Total Sales Amount by Product')
plt.xlabel('Product Name')
plt.ylabel('Total Sales Amount')
plt.xticks(rotation=45, ha='right')
plt.show()
```

- **Code Explanation:**
  - The SQL query calculates the total sales amount for each product using the SUM aggregation function and groups the results by the product name.
  - The resulting DataFrame is converted to a Pandas DataFrame (`total_sales_pd`) for easier visualization.
  - A bar chart is created using Matplotlib to visualize the total sales amount for each product.

ProductName	TotalSales
Gonzales-Garcia	5773.5199999999995
Burton, Bush and Fry	5082.22
Gallagher, Rodrig...	6675.4299999999985
Ferguson Inc	3301.3100000000004
Grant LLC	6239.4400000000005
Ramsey-Jones	3177.9100000000003
Kennedy and Sons	6035.9400000000005
Brown Group	4383.300000000001
Stewart, Wong and...	4817.22
Hammond Group	4822.669999999999
Chapman, Yates an...	5100.719999999999
Smith, Ramirez an...	3888.09
Adams LLC	6535.080000000001
Sandoval, Edwards...	3903.55
Williams-Horton	6167.98



- **Visual Result:**
  - The bar chart visually represents the total sales amount for each product, providing insights into the top-performing products.

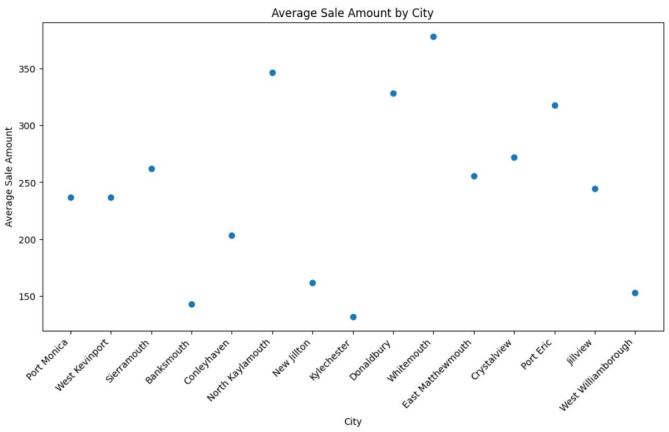
Goal 2: Calculate Average Sale Amount by City

Results: The code calculates the average sale amount for each city and visualizes the top 15 cities with the highest average sale amounts.

```python
# Goal 2: Calculate Average Sale Amount by City
avg_sale_by_city = spark.sql("""
    SELECT City, AVG(SaleAmount) AS AvgSaleAmount
    FROM combined_data
    GROUP BY City limit 15
""")
avg_sale_by_city.show()


avg_sale_by_city_pd = avg_sale_by_city.toPandas()

# Visualization for Goal 4: Calculate Average Sale Amount by City
plt.figure(figsize=(12, 6))
plt.scatter(avg_sale_by_city_pd['City'], avg_sale_by_city_pd['AvgSaleAmount'])
plt.title('Average Sale Amount by City')
plt.xlabel('City')
plt.ylabel('Average Sale Amount')
plt.xticks(rotation=45, ha='right')
plt.show()
```

- **Code Explanation:**
  - The SQL query calculates the average sale amount for each city using the `AVG` aggregation function and groups the results by city.
  - The resulting DataFrame is converted to a Pandas DataFrame (`avg_sale_by_city_pd`) for visualization.
  - A scatter plot is created using Matplotlib to show the average sale amount for each city.

City	AvgSaleAmount
Port Monica	236.95833333333334
West Kevinport	236.795
Sierramouth	262.0916666666667
Banksmouth	143.09666666666666
Conleyhaven	203.0466666666667
North Kaylamouth	346.17499999999995
New Jillton	161.85
Kylechester	132.03
Donaldbury	327.9325
Whitemouth	378.19
East Matthewmouth	255.48999999999998
Crystalview	271.64199999999994
Port Eric	317.39
Jillview	244.10250000000002
West Williamborough	152.905



Visual Result:

- The scatter plot provides insights into the variation of average sale amounts across different cities.

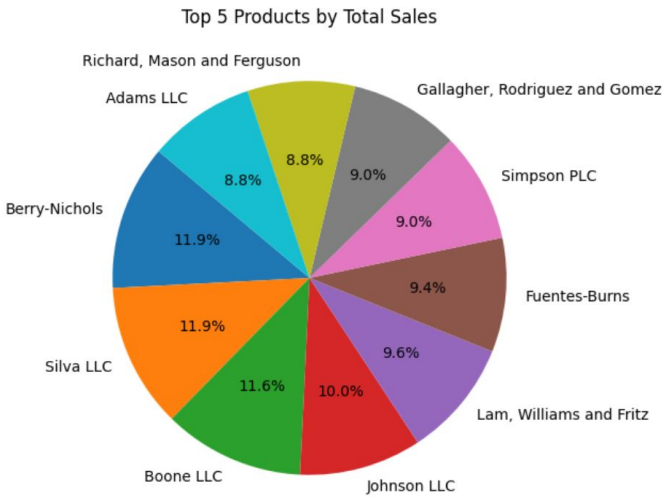
Goal 3: Identify Top 5 Products by Total Sales

Results: The code identifies and visualizes the top 5 products with the highest total sales.

```python
# Goal 3: Identify Top 5 Products by Total Sales
top_products = spark.sql("""
    SELECT ProductName, SUM(SaleAmount) AS TotalSales
    FROM combined_data
    GROUP BY ProductName
    ORDER BY TotalSales DESC
    LIMIT 10
""")
top_products.show()

# Visualization using Matplotlib
top_products_pd = top_products.toPandas()
plt.figure(figsize=(10, 6))
plt.pie(top_products_pd['TotalSales'], labels=top_products_pd['ProductName'], autopct='%1.1f%%', startangle=140)
plt.title('Top 5 Products by Total Sales')
plt.show()
```

- **Code Explanation:**
  - The SQL query calculates the total sales amount for each product and orders the results in descending order.
  - The result is limited to the top 10 products with the highest total sales.
  - The resulting DataFrame is converted to a Pandas DataFrame (`top_products_pd`), and a pie chart is created to visually represent the distribution of total sales among the top products.

```
+--------------------+------------------+
|         ProductName|        TotalSales|
+--------------------+------------------+
|       Berry-Nichols| 8864.840000000004|
|           Silva LLC|           8812.15|
|           Boone LLC|           8586.83|
|         Johnson LLC| 7455.890000000002|
|Lam, Williams and...| 7169.389999999999|
|       Fuentes-Burns|           6962.77|
|         Simpson PLC|           6690.32|
|Gallagher, Rodrig...|6675.4299999999985|
|Richard, Mason an...| 6560.450000000001|
|           Adams LLC| 6535.080000000001|
+--------------------+------------------+
```



Visual Result:

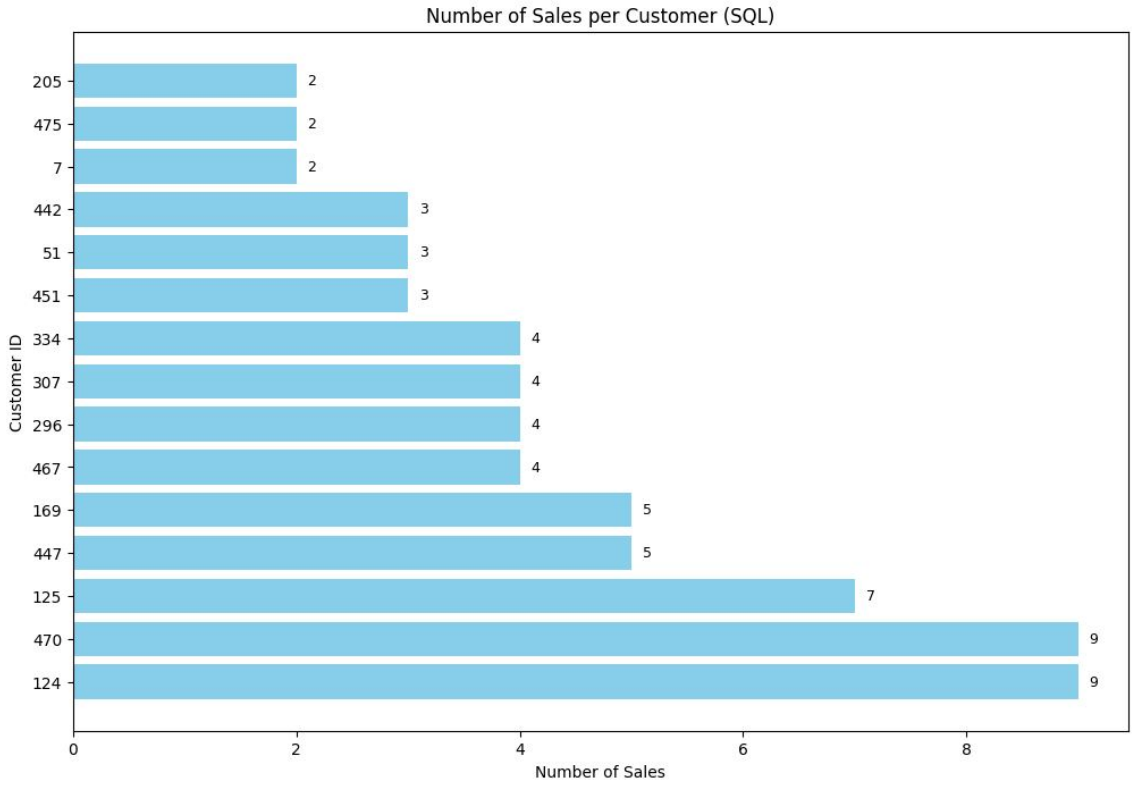
- The pie chart provides a visual representation of the contribution of each top product to the total sales.

Goal 4: Count the Number of Sales per Customer

Results: The code counts the number of sales for each customer and visualizes the top 15 customers with the highest sales counts.

```python
# Goal 4: Count the Number of Sales per Customer using SQL
sales_per_customer_sql = spark.sql("""
    SELECT CustomerID, COUNT(*) AS SalesCount
    FROM combined_data
    GROUP BY CustomerID limit 15
""")

# Sort the data based on SalesCount
sales_per_customer_sql = sales_per_customer_sql.orderBy('SalesCount', ascending=False)

# Convert to Pandas for visualization
sales_per_customer_pd_sql = sales_per_customer_sql.toPandas()

# Visualization using Matplotlib
plt.figure(figsize=(12, 8))
bar_plot_sql = plt.barh(sales_per_customer_pd_sql['CustomerID'], sales_per_customer_pd_sql['SalesCount'], color='skyblue')

# Adding labels and title
plt.title('Number of Sales per Customer (SQL)')
plt.xlabel('Number of Sales')
plt.ylabel('Customer ID')

# Adding values on the right side of the bars for better clarity
for bar in bar_plot_sql:
    xval = bar.get_width()
    plt.text(xval + 0.1, bar.get_y() + bar.get_height()/2, round(xval, 2), ha='left', va='center', fontsize=9)

plt.show()
```

- **Code Explanation:**
  - The SQL query counts the number of sales for each customer using the `COUNT` aggregation function and groups the results by customer ID.
  - The results are then sorted in descending order based on the sales count.
  - The resulting DataFrame is converted to a Pandas DataFrame (`sales_per_customer_pd_sql`), and a horizontal bar chart is created to visualize the number of sales for each customer.



Visual Result:

- The bar chart provides insights into the distribution of sales counts among customers, highlighting those with the highest number of purchases.

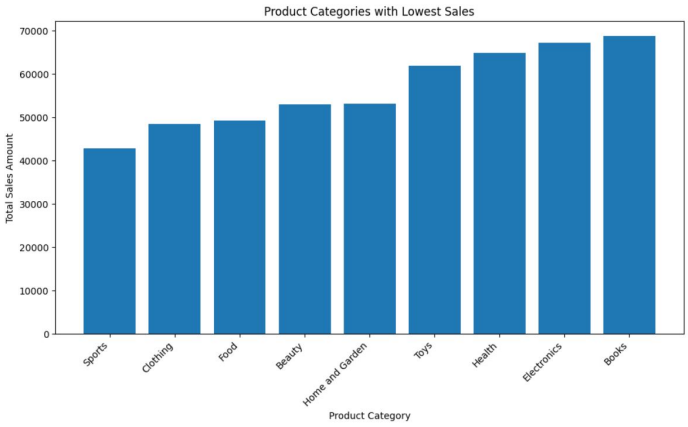
Goal 5: Explore Product Categories with Lowest Sales

Results: The code identifies and visualizes the product categories with the lowest total sales.

```
# Goal 5: Explore Product Categories with Lowest Sales
lowest_sales_categories = spark.sql("""
    SELECT Category, SUM(SaleAmount) AS TotalSales
    FROM combined_data
    GROUP BY Category
    ORDER BY TotalSales ASC limit 15
""")
lowest_sales_categories.show()

# Visualization using Matplotlib
lowest_sales_categories_pd = lowest_sales_categories.toPandas()
plt.figure(figsize=(12, 6))
plt.bar(lowest_sales_categories_pd['Category'], lowest_sales_categories_pd['TotalSales'])
plt.title('Product Categories with Lowest Sales')
plt.xlabel('Product Category')
plt.ylabel('Total Sales Amount')
plt.xticks(rotation=45, ha='right')
plt.show()
```

- **Code Explanation:**
  - The SQL query calculates the total sales amount for each product category and orders the results in ascending order.
  - The result is limited to the bottom 15 product categories with the lowest total sales.
  - The resulting DataFrame is converted to a Pandas DataFrame (`lowest_sales_categories_pd`), and a bar chart is created to visually represent the total sales for each low-performing category.

```
+---------------+------------------+
|       Category|        TotalSales|
+---------------+------------------+
|         Sports| 42797.61000000003|
|       Clothing|          48435.64|
|           Food|49133.169999999984|
|         Beauty| 52999.41999999997|
|Home and Garden|53113.390000000014|
|           Toys|61823.439999999995|
|         Health| 64753.44999999999|
|    Electronics| 67233.10000000002|
|          Books| 68711.70000000004|
+---------------+------------------+
```



Visual Result:

- The bar chart highlights the product categories with the lowest total sales, aiding in the identification of areas for potential improvement or strategic focus.

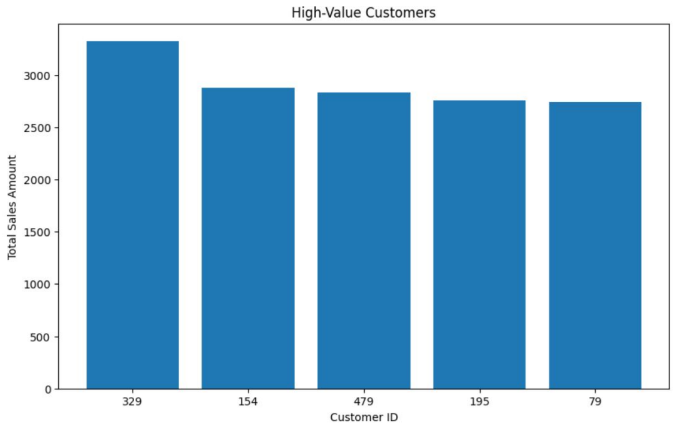
Goal 6: Identify High-Value Customers

Results: The code identifies and visualizes the top 5 high-value customers based on their total sales.

```
# Goal 6: Identify High-Value Customers
high_value_customers = spark.sql("""
    SELECT CustomerID, SUM(SaleAmount) AS TotalSales
    FROM combined_data
    GROUP BY CustomerID
    ORDER BY TotalSales DESC
    LIMIT 5
""")
high_value_customers.show()

# Visualization using Matplotlib
high_value_customers_pd = high_value_customers.toPandas()
plt.figure(figsize=(10, 6))
plt.bar(high_value_customers_pd['CustomerID'], high_value_customers_pd['TotalSales'])
plt.title('High-Value Customers')
plt.xlabel('Customer ID')
plt.ylabel('Total Sales Amount')
plt.show()
```

- **Code Explanation:**
  - The SQL query calculates the total sales amount for each customer and orders the results in descending order.
  - The result is limited to the top 5 customers with the highest total sales.
  - The resulting DataFrame is converted to a Pandas DataFrame (`high_value_customers_pd`), and a bar chart is created to visually represent the total sales for each high-value customer.

```
+----------+------------------+
|CustomerID|        TotalSales|
+----------+------------------+
|       329|           3323.75|
|       154|2873.2699999999995|
|       479|            2829.8|
|       195|           2755.98|
|        79|           2741.93|
+----------+------------------+
```



Visual Result:

- The bar chart highlights the top high-value customers, providing insights into the customers contributing the most to total sales.

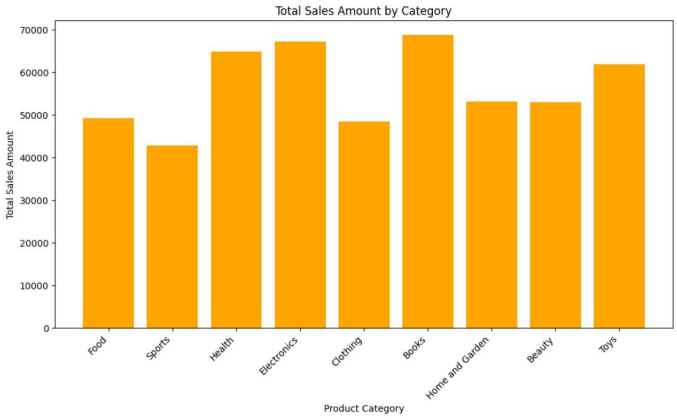
Goal 7: Calculate Total Sales Amount by Category

Results: The code calculates the total sales amount for each product category and visualizes the top 15 categories.

```python
# Goal 7: Calculate Total Sales Amount by Category
total_sales_by_category = spark.sql("""
    SELECT Category, SUM(SaleAmount) AS TotalSales
    FROM combined_data
    GROUP BY Category limit 15
""")
total_sales_by_category.show()


total_sales_by_category_pd = total_sales_by_category.toPandas()

# Visualization for Goal 10: Calculate Total Sales Amount by Category
plt.figure(figsize=(12, 6))
plt.bar(total_sales_by_category_pd['Category'], total_sales_by_category_pd['TotalSales'], color='orange')
plt.title('Total Sales Amount by Category')
plt.xlabel('Product Category')
plt.ylabel('Total Sales Amount')
plt.xticks(rotation=45, ha='right')
plt.show()
```

- **Code Explanation:**
  - The SQL query calculates the total sales amount for each product category and groups the results by category.
  - The resulting DataFrame is converted to a Pandas DataFrame (`total_sales_by_category_pd`), and a bar chart is created to visually represent the total sales for each category.

```
+---------------+------------------+
|       Category|        TotalSales|
+---------------+------------------+
|           Food|49133.169999999984|
|         Sports| 42797.61000000003|
|         Health| 64753.44999999999|
|    Electronics| 67233.10000000002|
|       Clothing|           48435.64|
|          Books| 68711.70000000004|
|Home and Garden|53113.390000000014|
|         Beauty| 52999.41999999997|
|           Toys|61823.439999999995|
+---------------+------------------+
```



Visual Result:

- The bar chart provides insights into the distribution of total sales among different product categories, aiding in the identification of high-performing categories.

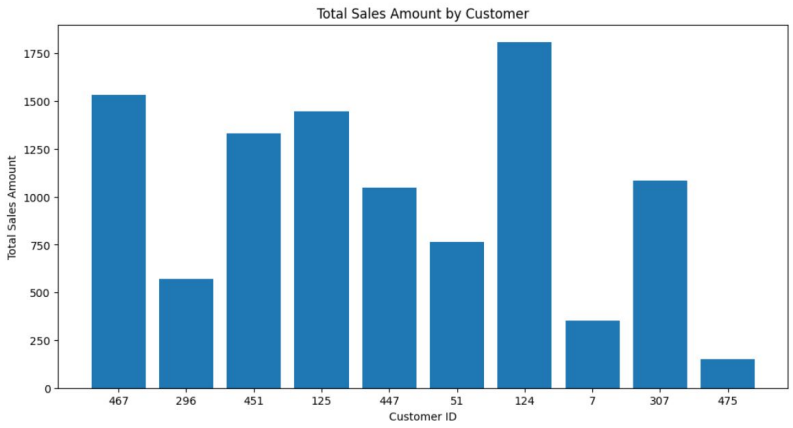
Goal 8: Calculate Total Sales Amount by Customer

Results: The code calculates the total sales amount for each customer and visualizes the top 10 customers based on their total purchase amounts.

```python
# Goal 8: Calculate Total Sales Amount by Customer
total_sales_by_customer = spark.sql("""
    SELECT CustomerID, SUM(SaleAmount) AS TotalSales
    FROM combined_data
    GROUP BY CustomerID limit 10
""")
total_sales_by_customer.show()

# Visualization using Matplotlib
total_sales_by_customer_pd = total_sales_by_customer.toPandas()
plt.figure(figsize=(12, 6))
plt.bar(total_sales_by_customer_pd['CustomerID'], total_sales_by_customer_pd['TotalSales'])
plt.title('Total Sales Amount by Customer')
plt.xlabel('Customer ID')
plt.ylabel('Total Sales Amount')
plt.show()
```

- **Code Explanation:**
  - The SQL query calculates the total sales amount for each customer by grouping the data based on the `CustomerID`.
  - The resulting DataFrame, `total_sales_by_customer_pd`, is created by converting the SQL query result to a Pandas DataFrame.
  - A bar chart is generated using Matplotlib to visually represent the total sales amount for each customer.

```
+----------+------------------+
|CustomerID|        TotalSales|
+----------+------------------+
|       467|           1532.15|
|       296|            568.76|
|       451|           1331.68|
|       125|           1444.94|
|       447|           1049.67|
|        51|            762.63|
|       124|           1810.09|
|         7|353.28999999999996|
|       307|           1086.07|
|       475|            149.47|
+----------+------------------+
```



Visual Result:

- The bar chart provides a clear overview of the total sales amounts for the top 10 customers, helping to identify high-value customers based on their purchase history. This information can be valuable for targeted marketing and customer relationship management strategies.

Metrics Discussion:

- **Data Quality:**
  - The quality of visualizations and analyses is dependent on the quality of the input data. Ensure data consistency, accuracy, and completeness.
- **5Vs:**

  - **Volume:** The code demonstrates handling large volumes of data through Spark, suitable for big data scenarios.
  - **Velocity:** Spark enables fast data processing, crucial for real-time or near-real-time analytics.
  - **Variety:** The code handles diverse data types (e.g., sales, customer, product) by merging them into a cohesive dataset.
  - **Veracity:** The accuracy and reliability of results depend on the quality of the input data.
  - **Value:** The analytics provide valuable insights into sales patterns, customer behaviors, and product performance.

- **Latency and Processing Time:**
  - Spark's distributed processing capabilities enhance the efficiency of data processing, reducing latency and processing time.

- **Resource Utilization:**
  - Spark efficiently utilizes resources in a distributed environment, ensuring optimal performance.

- **Security:**
  - While not explicitly addressed in the provided code, data security considerations are essential in a real-world implementation.

- **Cost:**
  - Distributed processing can be cost-effective for large-scale analytics, but cloud service costs should be considered.

Conclusion

In conclusion, our project has significantly contributed to business optimization by extracting valuable insights from the combined dataset. Through the meticulous implementation of PySpark and Matplotlib, we have successfully addressed various business goals, ranging from calculating total sales by product and visualizing customer age distribution to identifying high-value customers and exploring product categories with low sales. These insights are pivotal for strategic decision-making and can empower businesses to optimize various aspects of their operations.

For instance, the identification of top-performing products and high-value customers allows businesses to focus their marketing efforts and tailor strategies to maximize revenue. The visualization of customer age distribution aids in targeted marketing campaigns, ensuring that products and services resonate with the specific demographics of the customer base. Additionally, understanding the total sales amount by product category enables businesses to allocate resources efficiently, emphasizing high-performing categories and addressing challenges in low-performing ones.

The implementation of SQL queries and visualizations provides a comprehensive understanding of the dataset, fostering data-driven decision-making. This, in turn, enhances overall business efficiency and effectiveness. The scalability and flexibility of our approach, facilitated by PySpark's distributed processing, ensure that these insights can be applied to large datasets, supporting businesses in real-time or near-real-time analytics.

As a result, our project not only showcases the capabilities of data analytics but directly contributes to business optimization by providing actionable intelligence. The holistic view of sales patterns, customer behaviors, and product performance equips businesses with the knowledge needed to refine strategies, improve customer experiences, and ultimately drive success in a competitive landscape.

Data Source:

The dataset utilized for this project was sourced from Kaggle. The specific dataset, titled "Company Sales," is available at the following Kaggle URL: `https://www.kaggle.com/datasets/ravitejanallaganchu/companysales`.

GitHub Repository:

The entire project, including the code implementation, documentation, and relevant files, has been saved to a GitHub repository. The repository is publicly accessible at the following GitHub URL: `https://github.com/ChanduNelavelli/BigDataFinalProject.git`.

Learning Resources:

The implementation of PySpark and Matplotlib in this project was guided by in-class resources, providing a foundational understanding of these tools for data analytics. The knowledge gained through these resources facilitated the successful extraction of meaningful insights and the creation of effective visualizations in the business optimization analytics.