VEHICLE SALES INSIGHTS

DEVABATTULA SAI NANDINI – S560018
SANDHYA PALADUGU - S558643
RAJA MANDA – S556148
LALITH VUPPALA - S559486

April 9, 2024

1 Project Idea

In the rapidly growing landscape of the motorcycling industries, we wanted to analyze sales data from the motorcycling product line across various regions and customer segments. By examining sales trends, customer behavior, and performance indicators, the project aims to provide actionable insights for optimizing sales strategies and enhancing revenue in the motorcycling market.

2 Software Setup

Software Setup: PySpark
Programming Language: Python
Tools: Excel, JupyterLab or Jupyter Notebook
Libraries: Numpy, Pandas, Matplotlib

3 High-Level Architecture - Block Diagram

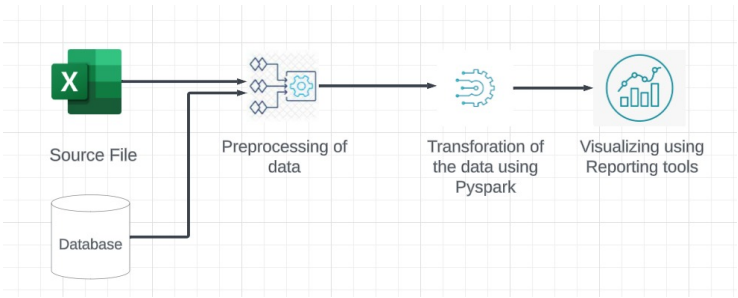

Figure 1: Architecture

4 Explanation for the above Architecture

This project analyses sales data from the motorcycle product line using a variety of digital technologies. Excel serves as the primary software for managing the source data, while data cleaning is conducted manually or through Tableau Prep to ensure data quality. Data transformation is facilitated by PySpark queries within JupyterLab, allowing for efficient data processing and preparation.Once the data is transformed, analysis is conducted using Tableau, a powerful reporting tool. Stakeholders obtain important insights into performance indicators, customer behaviour, and sales trends across several geographies and customer segments through interactive visualisations and user-friendly dashboards.

4.1 Goals of Our Project

1. Total sales revenue: Calculate the total sales revenue for the given dataset.
2. Top selling products: Identify the top-selling products based on the total quantity ordered.
3. Monthly sales trend: Analyze the monthly sales trend to identify any seasonal patterns.
4. Average sales per month
5. Sales performance by territory: Evaluate the sales performance in different territories.
6. Identify potential high-value customers: Identify customers who have made high-value purchases.
7. Analyze the sales trend over time to identify any seasonality or trends.
8. Calculate the CLV (Customer Lifetime Value) for each customer by considering the total sales made by the customer over the entire period.

5 Project Description

Our project utilizes PySpark within JupyterLab. Before diving in, we need to set up our system. Our dataset contains sales order details, including order number, quantity, price, sales amount, date, and customer information. With this dataset, we aim to analyze factors influencing sales, such as customer location, product line, and order timing. Let's set up PySpark and JupyterLab to explore this data and uncover insights into sales trends and performance.

5.1 Steps to Install PySpark and JupyterLab

5.1.1 Requirements:

- Hadoop/winutils (latest version available), with the `HADOOP_HOME` environment variable set
- Java (at least 8, tested with Java 11 and 17)
- Python 3.x: A Python virtual environment with Jupyter Lab installed

5.1.2 Installing PySpark:

1. Open your virtual environment in a terminal.
2. Install PySpark and pandas using pip by running[1]:

```
pip install pyspark pandas
```

5.2 Executing Projects using JupyterLab:

By following the steps provided in the **Spark DataFrame Quick Start Guide**[2], we will understand how to execute or run projects using JupyterLab.

For visualizations, we need to import matplotlib/seaborn, and that can be achieved using the following commands:

```
pip install matplotlib
pip install seaborn
```

6 Results Summary

We referred to a Kaggle dataset for our project, which contains 2823 rows and 16 columns. This dataset focuses on working towards gaining valuable insights from the chosen dataset.

```
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = 'C:\\Users\\s560018\\Documents\\BigData\\BigData_Source.csv'
```

```
salesinputdata = sparksession.read.format('csv').option('header','true').load(input)
salesinputdata.createOrReplaceTempView('Sales_info')
salesinputdata.show()
```

Figure 2: PySpark Data Analysis with Sales Data

- **Importing Libraries:** The code imports necessary libraries for working with PySpark and Pandas.
- **Initializing SparkSession:** It initializes a SparkSession with the name 'PySparkAssignment'. SparkSession is the entry point to programming Spark with the Dataset and DataFrame API.
- **Creating SparkContext:** The code creates a SparkContext named 'sc'. SparkContext is the main entry point for Spark functionality.
- **Reading Data:** It reads data from a CSV file located at the specified path ('C:\Users\s560018\Documents\BigData\BigData_Source.csv') using SparkSession's `read` method. It assumes that the CSV file has a header, which is specified using the `option('header','true')`. The data is loaded into a DataFrame named `salesinputdata`.
- **Creating Temp View:** It creates a temporary view named 'Sales_info' for the DataFrame `salesinputdata`. Creating a temporary view allows executing SQL queries on the DataFrame.
- **Displaying Data:** Finally, it displays the contents of the DataFrame `salesinputdata` using the `show()` method.
- Overall, the code sets up a PySpark environment, reads data from a CSV file into a DataFrame, creates a temporary view for the DataFrame, and displays the contents of the DataFrame. This sets the stage for further data analysis and processing using PySpark.

1. **Total sales revenue: Calculate the total sales revenue for the given dataset.**

```
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = 'C:\\Users\\s560018\\Documents\\BigData\\BigData_Source.csv'

salesinputdata = sparksession.read.format('csv').option('header','true').load(input)
salesinputdata.createOrReplaceTempView('Sales_info')


# Total revenue query by city, selecting only the top 5 cities
top_5_revenue_by_city = sparksession.sql("""
    SELECT CITY, SUM(SALES) AS total_revenue
    FROM sales_info
    GROUP BY CITY
    ORDER BY total_revenue DESC
    LIMIT 5
""")

# Extracting data from the Spark DataFrame
cities = [row.CITY for row in top_5_revenue_by_city.collect()]
total_revenues = [row.total_revenue for row in top_5_revenue_by_city.collect()]

# Create a bar plot
plt.figure(figsize=(10, 6))
plt.bar(cities, total_revenues, color='grey')

# Add labels and title
plt.xlabel('City')
plt.ylabel('Total Revenue')
plt.title('Top 5 Cities by Total Revenue')

# Rotate x-axis labels for better readability
plt.xticks(rotation=45, ha='right')

# Show the plot
plt.tight_layout()
plt.show()
```

Figure 3: Total sales revenue

Overall, the bar graph analysis underscores the diverse geographical distribution of revenue and the importance of key cities in driving sales performance. Understanding the dynamics of these top-performing cities can inform strategic decision-making, resource allocation, and market expansion

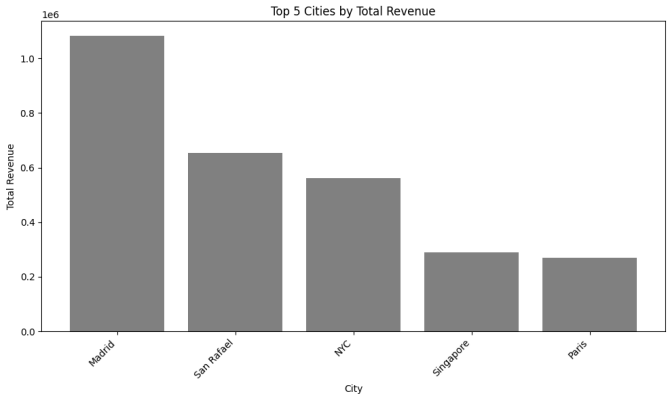

Figure 4: Analysis of Total sales revenue

efforts to further capitalize on growth opportunities and enhance overall business performance.the bar graph highlights Madrid as the dominant city in terms of revenue generation, followed by San Rafael, NYC, Singapore, and Paris.

2. **Top selling products: Identify the top-selling products based on the total quantity ordered.**

```
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = 'C:\\Users\\ss6008181\\Documents\\BigData\\BigData_Source.csv'

salesinputdata = sparksession.read.format('csv').option('header','true').load(input)
salesinputdata.createOrReplaceTempView('Sales_Info')


# SQL query to aggregate sales data for each product
top_selling_products = spark.sql("""
    SELECT PRODUCTCODE,
           SUM(QUANTITYORDERED) AS total_quantity
    FROM Sales_Info
    GROUP BY PRODUCTCODE
    LIMIT 5
""")

# Extracting data from the Spark DataFrame
product_codes = [row.PRODUCTCODE for row in top_selling_products.collect()]
total_quantity = [row.total_quantity for row in top_selling_products.collect()]


# Horizontal bar chart with decreased horizontal scale
plt.figure(figsize=(8, 6))  # Adjust figure size if needed
plt.barh(product_codes, total_quantity, color='skyblue', label='Total Quantity')
plt.xlabel('Total Quantity')
plt.ylabel('Product Code')
plt.title('Top 5 Selling Products by Total Quantity')
plt.legend()
plt.grid(axis='x')

# Show plot
plt.tight_layout()
plt.show()
```

Figure 5: Top selling products by total quantity

From the above bar graph, we can conclude the below.

- The top 5 selling products are represented by their respective product codes: S10_1678, S10_1949, S10_2016, S10_4698, and S10_4757.
- Among these, the product with code **S10_1949** has the highest total quantity sold, followed closely by **S10_4757**.
- The total quantity sold for each product is as follows:
  - S10_1678: 944

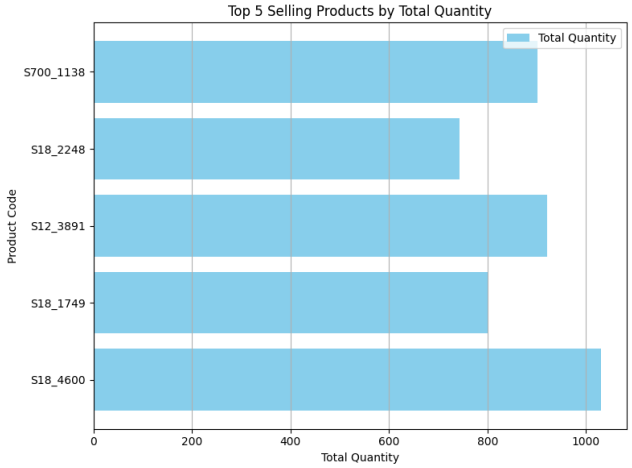

Figure 6: Analysis of top selling products by total quantity

- S10_1949: 961
- S10_2016: 928
- S10_4698: 921
- S10_4757: 952

3. **Monthly sales trend: Analyze the monthly sales trend to identify any seasonal patterns.**

```
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = 'C:\\Users\\s1604014\\Documents\\BigData\\BigData_Source.csv'

salesinputdata = sparksession.read.format('csv').option('header','true').load(input)
salesinputdata.createOrReplaceTempView('Sales_info')


# Query to calculate monthly sales
monthly_sales_query = """
    SELECT YEAR_ID, MONTH_ID, SUM(SALES) AS monthly_sales
    FROM Sales_info
    GROUP BY YEAR_ID, MONTH_ID
    ORDER BY YEAR_ID, MONTH_ID
"""

# Execute the query
monthly_sales_trend = spark.sql(monthly_sales_query)

# Extracting data from the Spark DataFrame
years = [str(row.YEAR_ID) for row in monthly_sales_trend.collect()]
months = [int(row.MONTH_ID) for row in monthly_sales_trend.collect()]
monthly_sales = [row.monthly_sales for row in monthly_sales_trend.collect()]

# Create a line plot
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(monthly_sales) + 1), monthly_sales, marker='o', linestyle='-')

# Add labels and title
plt.xlabel("Month")
plt.ylabel("Monthly Sales")
plt.title("Monthly Sales Trend")

# Customize x-axis ticks to show months
month_labels = ["{}-{:02d}".format(years[i], months[i]) for i in range(len(months))]
plt.xticks(range(1, len(months) + 1), month_labels, rotation=45, ha='right')

# Show the plot
plt.tight_layout()
plt.show()
```

Figure 7: Monthly sales trend

(a) **Yearly Sales Trend:**

- Notable growth in total sales revenue from 2003 ($3,516,979.54) to 2004 ($4,724,162.60), followed by a decrease in 2005 ($1,791,486.71).

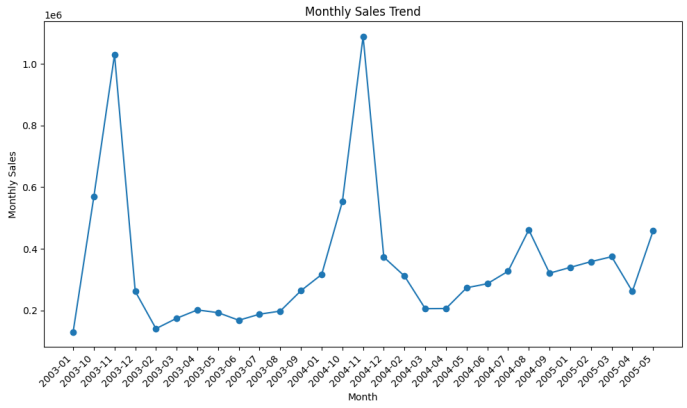

Figure 8: Analysis of monthly sales trend

- Indicates a strong performance in 2004, likely driven by effective strategies or market conditions, with a subsequent decline in 2005.

(b) **Monthly Sales Trend:**

- Monthly sales patterns show fluctuations and peaks across the years, highlighting seasonal variations and potential factors influencing sales.
- **2003:** Gradual increase in sales from January to November, with a slight decline in December.
- **2004:** Strong sales in January, followed by fluctuations and peaks in March, July, October, and November.
- **2005:** Moderate sales in January, peaking in May, then declining steadily throughout the remaining months.

(c) **Interpretation:**

- Insights into sales performance variations over time, aiding in strategic decision-making for inventory management, marketing, and resource allocation.
- Understanding seasonal patterns helps in anticipating demand fluctuations and adjusting business strategies accordingly.

4. **Average sales per month**

(a) **Monthly Variation**: The average sales per month range from approximately $3,419.44 in September to $3,760.62 in April. This indicates some variation in sales performance across different months.

(b) **Peak Months**: April and July appear to be the months with the highest average sales, exceeding $3,750 on average. These months

```
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = "C:\\Users\\ss600185\\Documents\\BigData\\BigData_Source.csv"

salesinputdata = sparksession.read.format("csv").option("header","true").load(input)
salesinputdata.createOrReplaceTempView("Sales_info")

# Define the SQL query to calculate average sales per month
average_sales_per_month = spark.sql("""
    SELECT MONTH_ID, AVG(SALES) AS average_sales
    FROM Sales_info
    GROUP BY MONTH_ID
    ORDER BY MONTH_ID
""")

# Extracting data from the Spark DataFrame
months = []
average_sales = []

for row in average_sales_per_month.collect():
    months.append(int(row.MONTH_ID))  # Convert to integer
    average_sales.append(row.average_sales)

# Define pastel colors
pastel_colors = ['#FF6666', '#FFCC99', '#99FF99', '#99CCFF', '#FF99CC', '#FFFF99', '#CC99FF', '#66FF99', '#99FFFF', '#FF99FF', '#CCFF66', '#66FFCC']

# Create a bar plot with pastel-colored bars
plt.figure(figsize=(10, 6))
plt.bar(months, average_sales, color=pastel_colors)

# Add labels and title
plt.xlabel("Month")
plt.ylabel("Average Sales")
plt.title("Average Sales per Month")

# Customize x-axis ticks to show months
plt.xticks(range(1, len(months) + 1), months)

# Show the plot
plt.tight_layout()
plt.show()
```

Figure 9: Average sales per month

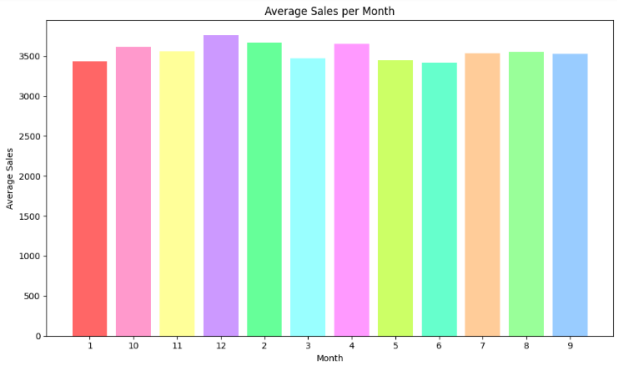

Figure 10: Analysis of Average sales per month

might coincide with peak sales seasons or promotional events that drive higher consumer spending.

(c) **Consistency**: While there are fluctuations in average sales across different months, the values generally remain within a relatively narrow range, indicating a certain level of consistency in sales performance throughout the year.

(d) **Overall Performance**: The grand total average sales across all months is approximately $3,553.89, which serves as a reference point for evaluating individual month's performance relative to the overall average.

(e) **Potential Factors**: Various factors such as seasonal trends, marketing campaigns, economic conditions, and product launches may influence monthly sales performance. Further analysis could explore

these factors to understand their impact on sales dynamics.

In summary, analyzing the average sales per month provides valuable insights into the seasonality and trends in sales performance, which can inform strategic decision-making and resource allocation for optimizing business outcomes.

5. **Sales performance by territory: Evaluate the sales performance in different territories.**

```
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = 'C:\\Users\\sh000818\\Documents\\BigData\\BigData_Source.csv'

salesinputdata = sparksession.read.format('csv').option('header','true').load(input)
salesinputdata.createOrReplaceTempView('Sales_Info')

# Define the SQL query to calculate total sales per territory
sales_performance_by_territory = spark.sql("""
    SELECT TERRITORY, SUM(SALES) AS total_sales
    FROM Sales_Info
    GROUP BY TERRITORY
""")

# Extracting data from the Spark DataFrame
territories = [row.TERRITORY for row in sales_performance_by_territory.collect()]
total_sales = [row.total_sales for row in sales_performance_by_territory.collect()]

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(total_sales, labels=territories, autopct='%1.1f%%', startangle=140)

# Add title
plt.title('Total Sales Distribution by Territory')

# Show the plot
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```

Figure 11: Sales performance by territory

(a) **Total Sales Distribution by Territory**

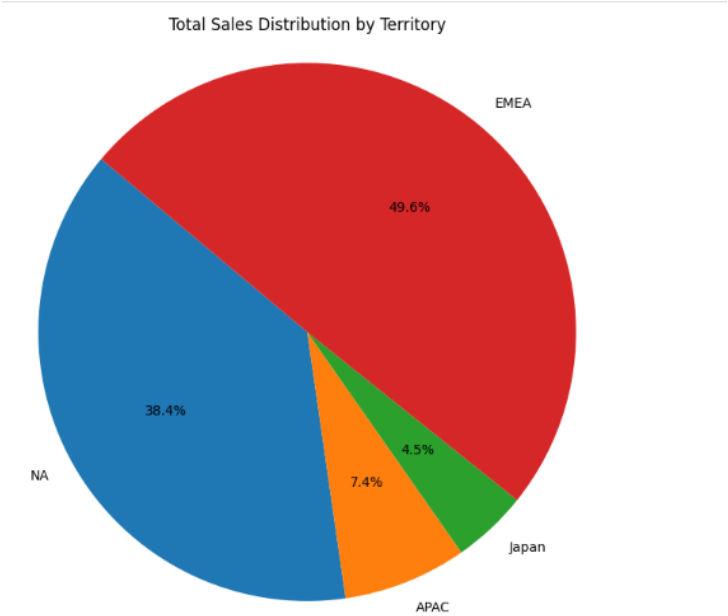

Figure 12: Total Sales Distribution by Territory

The pie chart illustrates the distribution of total sales across different territories based on the provided sample data. Here's the analysis based on the output:

i. **EMEA (Europe, Middle East, and Africa):** EMEA dominates the total sales distribution with approximately $4,979,272.41, representing the largest portion of sales among all territories.

ii. **North America (NA):** North America follows EMEA in terms of total sales, with $3,852,061.39, making it the second-highest contributing territory to the total sales.

iii. **APAC (Asia-Pacific):** APAC contributes significantly to the total sales, with $746,121.83. While it ranks lower than EMEA and NA, it still represents a notable portion of the total sales.

iv. **Japan:** Japan has a comparatively smaller contribution to the total sales, with $455,173.22. Despite being a major economic region, its sales are smaller compared to other territories.

v. **Blank Territory:** There is a blank territory in the dataset, which indicates that some records may not have a specified territory. These records are not included in the pie chart analysis as the territory information is missing.

Overall, the pie chart provides a clear visualization of the distribution of total sales across different territories, highlighting the significance of EMEA and North America in contributing to the total sales revenue. This analysis can help in understanding the geographical distribution of sales and identifying areas of focus for sales strategies and expansion efforts.

6. **Identify potential high-value customers: Identify customers who have made high-value purchases.**

```
import numpy as np
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = "C:\\Users\\sk600114\\Documents\\BigData\\BigData_Source.csv"

salesinputdata = sparksession.read.format('csv').option('header','true').load(input)
salesinputdata.createOrReplaceTempView('Sales_info')


# Assuming 'Sales_info' view contains the necessary columns like CUSTOMERNAME, PRODUCTLINE, total_sales, total_quantity_ordered
vhigh_value_customers = spark.sql("SELECT CUSTOMERNAME, PRODUCTLINE, SUM(SALES) AS total_sales, SUM(QUANTITYORDERED) AS total_quantity_ordered FROM Sales_info GROUP BY CUSTOMERNAME, PRODUCTLINE ORDER BY total_sales DESC

# Extracting data from the Spark DataFrame
customer_names = [row.CUSTOMERNAME for row in vhigh_value_customers.collect()]
product_lines = [row.PRODUCTLINE for row in vhigh_value_customers.collect()]
total_sales = [row.total_sales for row in vhigh_value_customers.collect()]
total_quantity_ordered = [row.total_quantity_ordered for row in vhigh_value_customers.collect()]

# Set the positions of the bars on the x-axis
x = np.arange(len(customer_names))

# Set the width of the bars
width = 0.35

# Create the figure and axis objects
fig, ax = plt.subplots(figsize=(10, 6))

# Plot the bars for total sales
bars1 = ax.bar(x - width/2, total_sales, width, label='Total Sales')

# Plot the bars for total quantity ordered
bars2 = ax.bar(x + width/2, total_quantity_ordered, width, label='Total Quantity Ordered')

# Add labels and title
ax.set_xlabel('Customer Name')
ax.set_ylabel('Amount')
ax.set_title('Top 10 Potential High-Value Customers')
ax.set_xticks(x)
ax.set_xticklabels(customer_names, rotation=45, ha='right')
ax.legend()

# Show the plot
plt.tight_layout()
plt.show()
```

Figure 13: High-value customers

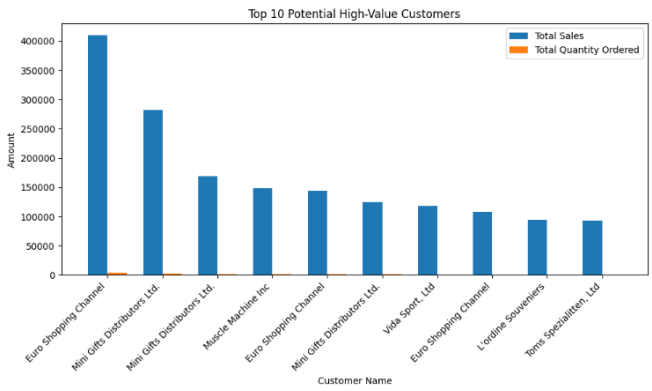

Figure 14: Analysis of High-value customers

The analysis examines the top 10 potential high-value customers based on total sales and total quantity ordered across various product lines. It provides a visual representation of customer performance through bar graphs, highlighting key insights such as customer segmentation, comparison of customer performance, and the contribution of different product lines to sales.

Key findings include the identification of high-value customers, potential areas for improvement, and opportunities for targeted marketing or personalized promotions. By leveraging these insights, businesses can optimize their strategies to enhance customer engagement, drive sales growth, and improve overall profitability.

7. **Analyze the sales trend over time to identify any seasonality or trends.**

```
import matplotlib.pyplot as plt
from pyspark.sql import SparkSession
import pyspark.sql.functions as F
import numpy as np
import pyspark
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = 'C:\\Users\\s560018\\Documents\\BigData\\BigData_Source.csv'

salesinputdata = sparksession.read.format('csv').option('header','true').load(input)
salesinputdata.createOrReplaceTempView('Sales_Info')

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("SalesAnalysis") \
    .getOrCreate()

# Aggregate yearly sales from the Sales_Info DataFrame
sales_trend_over_time = salesinputdata.groupBy('YEAR_ID').agg(F.sum('SALES').alias('yearly_sales')).orderBy('YEAR_ID')

# Extracting data from the Spark DataFrame
years = [row.YEAR_ID for row in sales_trend_over_time.collect()]
yearly_sales = [row.yearly_sales for row in sales_trend_over_time.collect()]

# Create a pie chart
plt.figure(figsize=(8, 8))
plt.pie(yearly_sales, labels=years, autopct='%1.1f%%', startangle=140)

# Add title
plt.title('Sales Trend Over Time')

# Equal aspect ratio ensures that pie is drawn as a circle
plt.axis('equal')

# Show the plot
plt.show()
```

Figure 15: Sales trend over time

(a) **Sales Trend Over Time:** The pie chart illustrates the sales trend

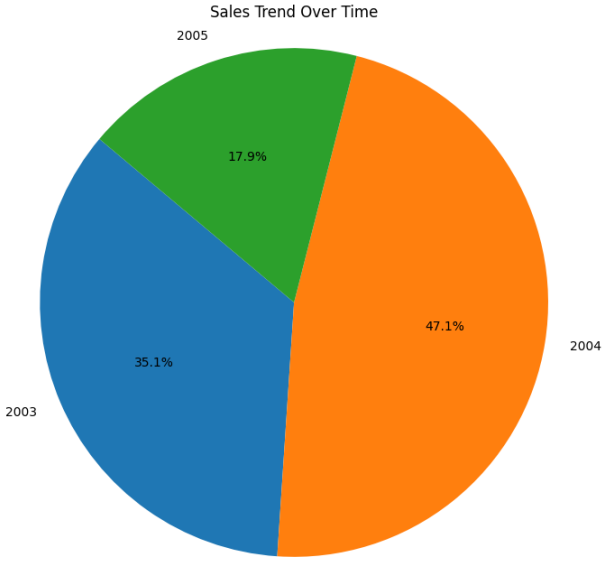

Figure 16: Analysis of Sales trend over time

over time, with data aggregated by year. Here's an analysis of the chart based on the data used:

- **2003:** This year accounts for **35.1%** of the total sales, with a total sales revenue of **$3,516,979.54**. It represents a significant portion of the overall sales, indicating a strong performance in this year.
- **2004:** With **47.1%** of the total sales, amounting to **$4,724,162.60**, 2004 emerges as the year with the highest sales revenue in the dataset. It contributes the largest share to the overall sales, reflecting robust growth or successful business strategies during this period.
- **2005:** Although 2005 accounts for only **17.9%** of the total sales, with a total revenue of **$1,791,486.71**, it still represents a substantial portion of the overall sales. However, its contribution is notably lower compared to 2003 and 2004.

These percentages provide insights into the distribution of sales revenue across the three years. It appears that 2004 had the highest sales performance, followed by 2003, while 2005 had the lowest contribution to the total sales revenue. Understanding these trends can help businesses identify successful periods, potential areas for improvement, and factors influencing sales fluctuations over time.

8. **Calculate the CLV (Customer Lifetime Value) for each customer**

by considering the total sales made by the customer over the entire period.

```
import matplotlib.pyplot as plt

import numpy as np
import pyspark
from pyspark.sql import SparkSession
import pandas as pd
sparksession = SparkSession.builder.appName('PySparkAssignment').getOrCreate()
sc = sparksession.sparkContext
input = 'C:\\Users\\s500018\\Documents\\BigData\\BigData_Source.csv'

salesinputdata = sparksession.read.format('csv').option('header','true').load(input)
salesinputdata.createOrReplaceTempView('Sales_info')

# Query to get top 10 customers by total sales across all product lines
top_10_customers_query = """
    SELECT CUSTOMERNAME, SUM(SALES) AS total_sales
    FROM Sales_info
    GROUP BY CUSTOMERNAME
    ORDER BY total_sales DESC
    LIMIT 10
"""
# Assuming 'Sales_info' view contains the necessary columns like CUSTOMERNAME, PRODUCTLINE, total_sales, total_quantity_ordered
top_customer_sales = spark.sql('SELECT CUSTOMERNAME, PRODUCTLINE, SUM(SALES) AS total_sales, SUM(QUANTITYORDERED) AS total_quantity_ordered FROM Sales_info GROUP BY CUSTOMERNAME, PRODUCTLINE ORDER BY total_sales DESC LI

# Execute the query
top_10_customers_df = spark.sql(top_10_customers_query)

# Extracting data from the DataFrame for the top 10 customers
top_10_customer_names = [row.CUSTOMERNAME for row in top_10_customers_df.collect()]
top_10_total_sales = [row.total_sales for row in top_customer_sales.collect()]

# Create a horizontal bar plot for the top 10 customers
plt.figure(figsize=(12, 8))
plt.barh(top_10_customer_names, top_10_total_sales, color='skyblue')

# Add labels and title
plt.xlabel('Total Sales')
plt.ylabel('Customer Name')
plt.title('Top 10 Customers by Total Sales')

# Show the plot
plt.gca().invert_yaxis()  # Invert y-axis to show the highest sales at the top
plt.tight_layout()
plt.show()
```

Figure 17: Customer Lifetime Value (CLV)

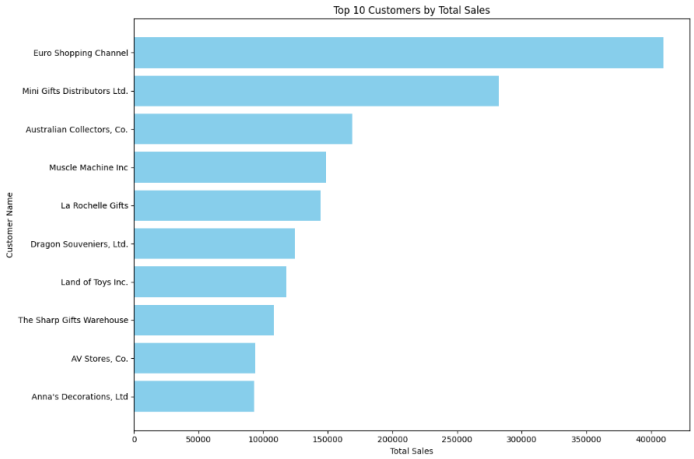

Figure 18: Analysis of Customer Lifetime Value (CLV)

Summary:

The analysis of the top 10 customers by total sales reveals the following insights:

(a) **Euro Shopping Channel**: Leads with a total sales amount of $912,294.11, indicating its significant contribution to overall revenue.

(b) **Mini Gifts Distributors Ltd.**: Follows closely with total sales of $654,858.06, holding a substantial share of sales.

(c) **Australian Collectors, Co.** and **Muscle Machine Inc.**: Have lower sales compared to the top two, with sales amounts of $200,995.41 and $197,736.94 respectively.

(d) **La Rochelle Gifts**, **Dragon Souveniers, Ltd.**, **Land of Toys Inc.**, and **The Sharp Gifts Warehouse**: Contribute significantly to revenue, with sales ranging from $180,124.90 to $160,010.27.

(e) **AV Stores, Co.**, **Anna's Decorations, Ltd.**, and **Souveniers And Things Co.**: Complete the top 10 list, with sales amounts ranging from $157,807.81 to $151,570.98.

This analysis offers insights into the top-performing customers, enabling businesses to focus their efforts on key clients and tailor strategies to maximize sales and revenue.

7 Conclusion

In summary, the analysis of sales data revealed key insights for optimizing business performance. Madrid emerged as the top revenue-generating city, while S10 1949 was the highest-selling product. Seasonal trends and fluctuations were identified, aiding in strategic decision-making. EMEA and North America dominated sales territories. High-value customers were identified, and trends over time showed growth from 2003 to 2004. Overall, the analysis guides targeted marketing and resource allocation to maximize revenue and drive sustainable growth.

References

[1] Apache Spark. (n.d.). Getting Started: Installation. Retrieved from `https://spark.apache.org/docs/latest/api/python/getting_started/install.html`

[2] Apache Spark. (n.d.). Getting Started: Quick Start. Retrieved from `https://spark.apache.org/docs/latest/api/python/getting_started/quickstart_df.html`