Target
Insight

Target Sales Analytics Capstone

Data Analytics with Advanced SQL | Power BI | MySQL | DAX | Visualization | python | Machine-Learning

www.TargetSalesInsights.com

By: K K Sahani

Overview

- Introduction
- Dataset Description
- Phase 1: SQL Analysis
- Phase 2: Data Visualization
- Phase 3: Predictive Analytics
- Reference

Introduction

This project focuses on analyzing sales, customer behavior, and operational performance using data visualization and analytics. By leveraging datasets such as orders, products, customers, and regions, the goal is to uncover actionable insights that drive informed decision-making. Key areas of analysis include sales trends, customer loyalty, revenue by category, regional performance, and operational metrics like delivery times and return rates.

The project employs Power BI for interactive dashboards and DAX queries for advanced calculations, enabling a clear understanding of business performance and opportunities for optimization. It is designed to assist stakeholders in identifying growth areas, enhancing customer experiences, and improving operational efficiency.

Target
Insigh

Phase 1: SQL Analysis – Advanced Queries

Introduction:
This phase focuses on leveraging advanced SQL queries to analyze sales, customer behavior, and operational performance. Key tasks include identifying top-selling products, loyal customers, and sales trends, as well as evaluating delivery efficiency and cancellations. Advanced techniques like window functions and time-based metrics (MTD, YTD) are used to extract actionable insights for informed decision-making.

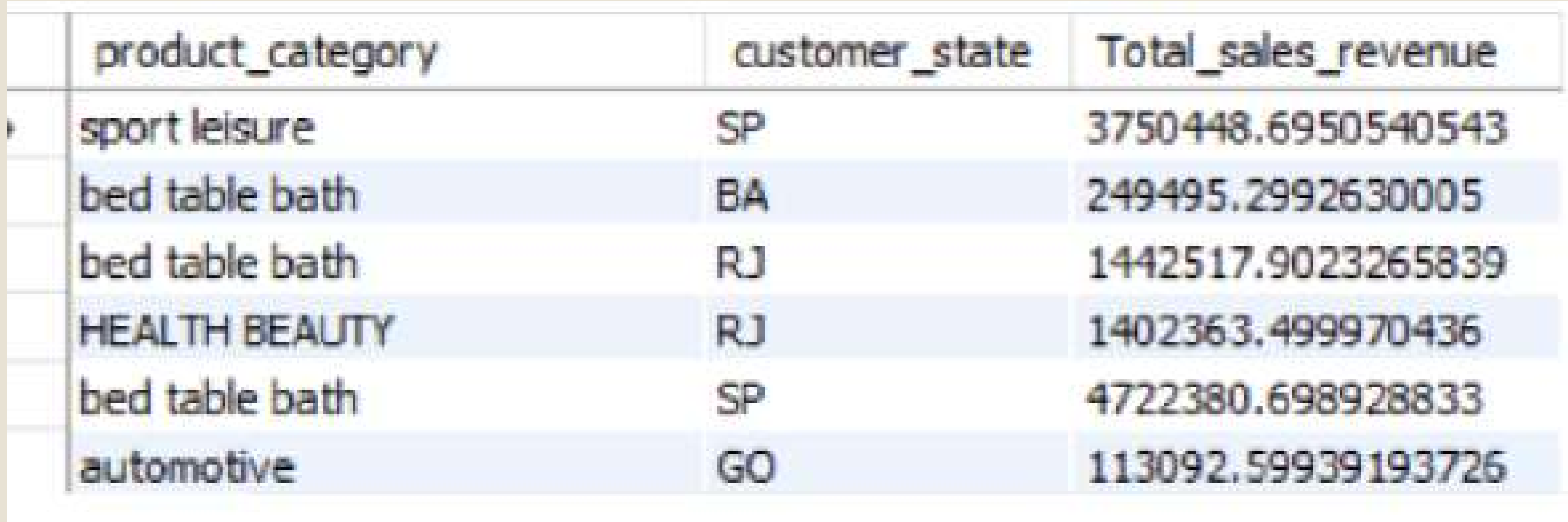
About MySQL

- **Powerful and Open-Source**: A widely used relational database management system (RDBMS).
- **Data Storage and Management**: Used for storing and managing structured data.
- **SQL Querying:** Provides robust tools for querying, analysing, and manipulating data through SQL (Structured Query Language).
- **Scalability**: Can handle large datasets and grow with business needs
- **Reliability**: Known for its stable performance and consistency.
- **Efficiency**: Optimized for fast data processing and complex business queries.

1. Sales PerformancAnalysis:

1.Write a query to calculate total sales revenue per-category, sub-category, and region.

```
SELECT
    p.product_category,
    c.customer_state AS region,
    SUM(pay.payment_value) AS total_sales_revenue
FROM
    ecomerce_sales.order_items oi
JOIN
    ecomerce_sales.products p ON oi.product_id = p.product_id
JOIN
    ecomerce_sales.orders o ON oi.order_id = o.order_id
JOIN
    ecomerce_sales.customers c ON o.customer_id = c.customer_id
JOIN
    ecomerce_sales.payments pay ON o.order_id = pay.order_id
WHERE
    o.order_status = 'delivered'
GROUP BY
    p.product_category,
    c.customer_state
LIMIT 100;
```

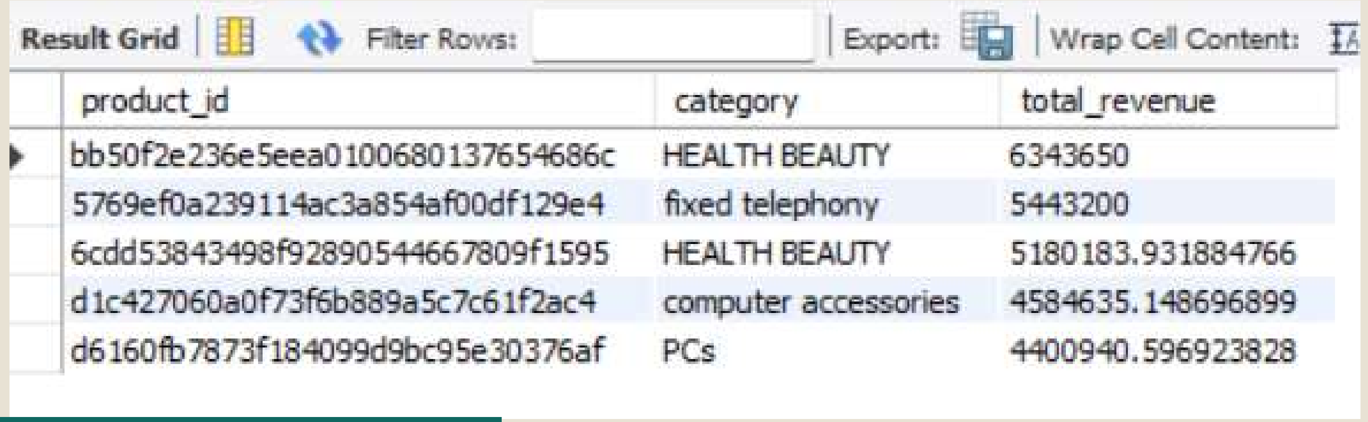


product_category	customer_state	Total_sales_revenue
sport leisure	SP	3750448.6950540543
bed table bath	BA	249495.2992630005
bed table bath	RJ	1442517.9023265839
HEALTH BEAUTY	RJ	1402363.499970436
bed table bath	SP	4722380.698928833
automotive	GO	113092.59939193726

1. Sales PerformancAnalysis:

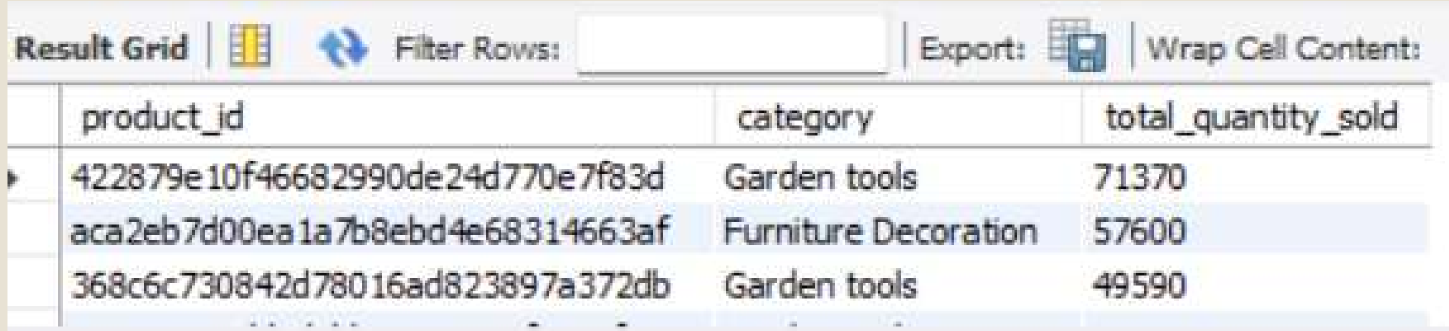
2. Identify the top 5 best-selling products by both sales revenue and quantity sold.

```
-- Top 5 Best-Selling Products by Sales Revenue
SELECT
   p.product_id,
   p.product_category AS category,
   SUM(oi.price * oi.order_item_id) AS total_revenue
FROM
   order_items oi
JOIN
   products p
ON
   oi.product_id = p.product_id
GROUP BY
   p.product_id, p.product_category
ORDER BY
   total_revenue DESC
LIMIT 5;
```



product_id	category	total_revenue
bb50f2e236e5eea0100680137654686c	HEALTH BEAUTY	6343650
5769ef0a239114ac3a854af00df129e4	fixed telephony	5443200
6cdd53843498f92890544667809f1595	HEALTH BEAUTY	5180183.931884766
d1c427060a0f73f6b889a5c7c61f2ac4	computer accessories	4584635.148696899
d6160fb7873f184099d9bc95e30376af	PCs	4400940.596923828

```
-- Top 5 Best-Selling Products by Quantity Sold
SELECT
   p.product_id,
   p.product_category AS category,
   SUM(oi.order_item_id) AS total_quantity_sold
FROM
   order_items oi
JOIN
   products p
ON
   oi.product_id = p.product_id
GROUP BY
   p.product_id, p.product_category
ORDER BY
   total_quantity_sold DESC
LIMIT 5;
```

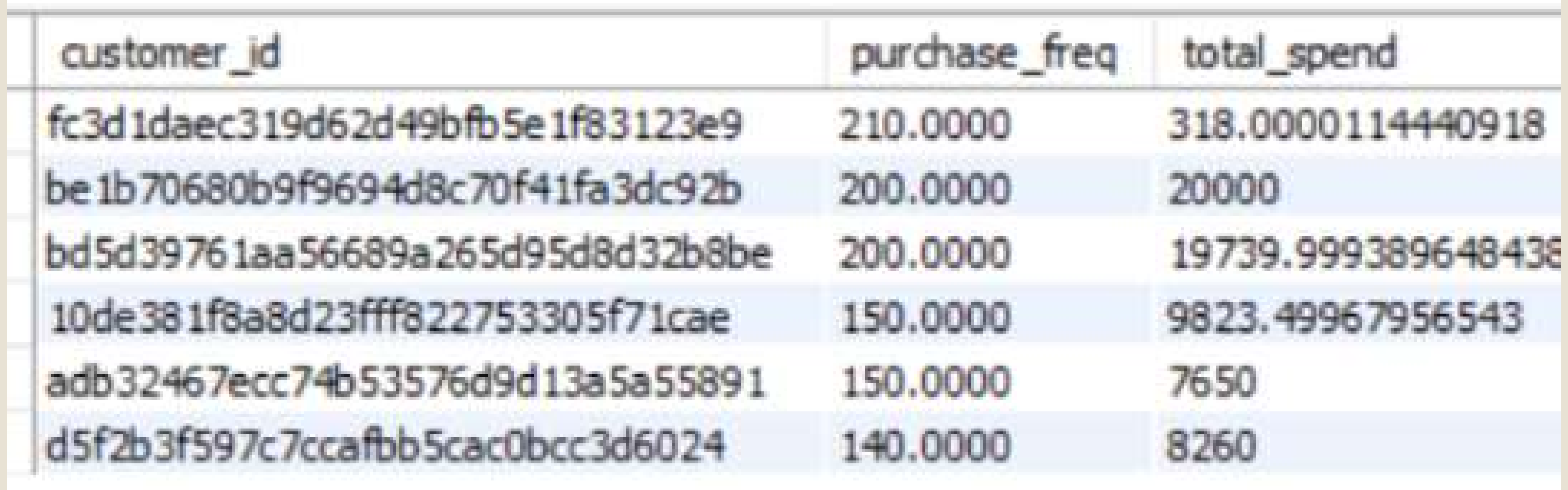


product_id	category	total_quantity_sold
422879e10f46682990de24d770e7f83d	Garden tools	71370
aca2eb7d00ea1a7b8ebd4e68314663af	Furniture Decoration	57600
368c6c730842d78016ad823897a372db	Garden tools	49590

2. Customer Insights:

1. Find the most loyal customers by calculating their purchase frequency and total spend.

```
-  Most Loyal Customers (Purchase Frequency and Total Spend)
SELECT
    c.customer_id,
    c.customer_unique_id,
    COUNT(o.order_id) AS purchase_frequency,
    SUM(oi.price * oi.order_item_id) AS total_spend
FROM
    customers c
JOIN
    orders o
ON
    c.customer_id = o.customer_id
JOIN
    order_items oi
ON
    o.order_id = oi.order_id
GROUP BY
    c.customer_id, c.customer_unique_id
ORDER BY
    purchase_frequency DESC, total_spend DESC
LIMIT 10; -- Top 10 most loyal customers
```



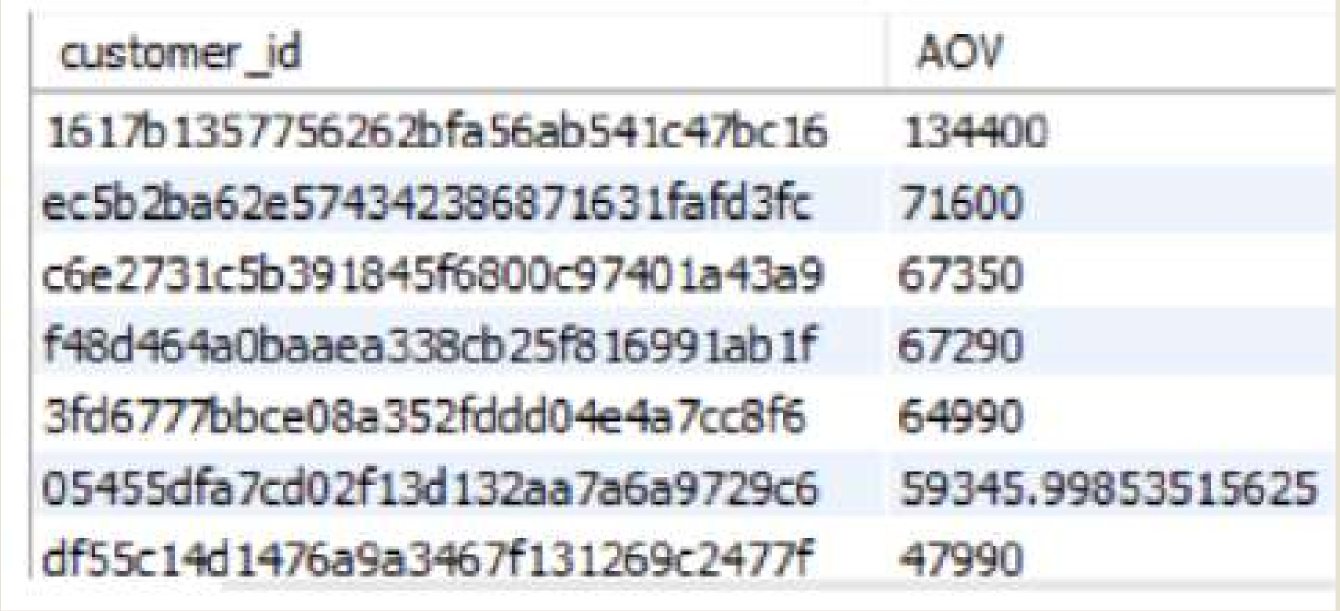
customer_id	purchase_freq	total_spend
fc3d1daec319d62d49bfb5e1f83123e9	210.0000	318.000011440918
be1b70680b9f9694d8c70f41fa3dc92b	200.0000	20000
bd5d39761aa56689a265d95d8d32b8be	200.0000	19739.99389648438
10de381f8a8d23fff822753305f71cae	150.0000	9823.49967956543
adb32467ecc74b53576d9d13a5a55891	150.0000	7650
d5f2b3f597c7ccafbb5cac0bcc3d6024	140.0000	8260

2. Customer Insights:

2. Identify customers with the highest average order value (AOV).

.

```
-- Query 2: Customers with the Highest AOV
SELECT
   c.customer_id,
   c.customer_unique_id,
   SUM(oi.price * oi.order_item_id) / COUNT(DISTINCT o.order_id)
AS average_order_value,
   SUM(oi.price * oi.order_item_id) AS total_spend,
   COUNT(DISTINCT o.order_id) AS total_orders
FROM
   customers c
JOIN
   orders o
ON
   c.customer_id = o.customer_id
JOIN
   order_items oi
ON
   o.order_id = oi.order_id
GROUP BY
   c.customer_id, c.customer_unique_id
ORDER BY
   average_order_value DESC
LIMIT 10; -- Top 10 customers with the highest AOV
```

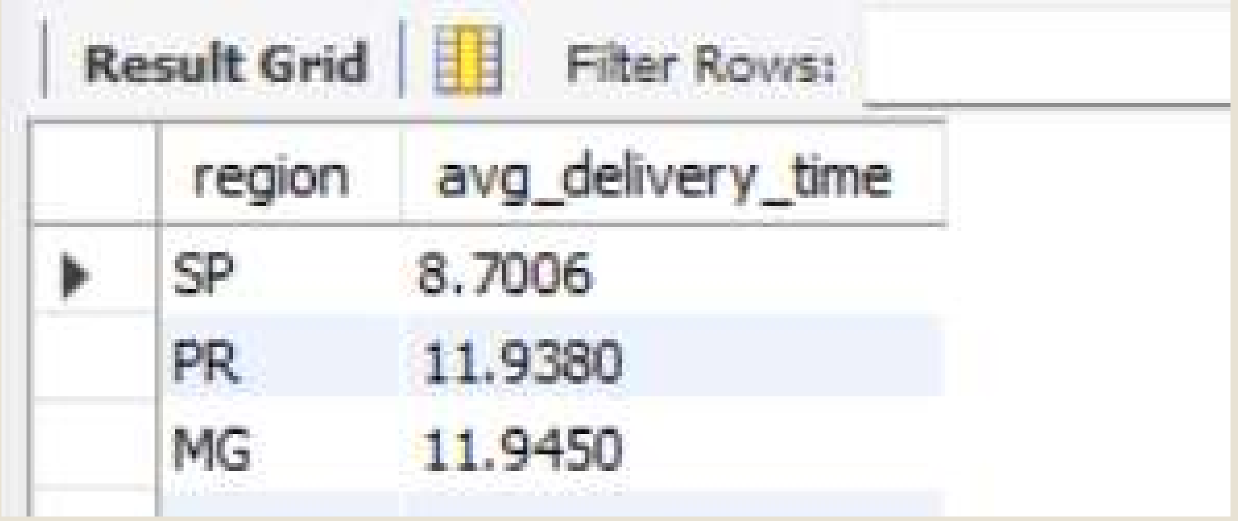


customer_id	AOV
1617b1357756262bfa56ab541c47bc16	134400
ec5b2ba62e574342386871631fafd3fc	71600
c6e2731c5b391845f6800c97401a43a9	67350
f48d464a0baaea338cb25f816991ab1f	67290
3fd6777bbce08a352fddd04e4a7cc8f6	64990
05455dfa7cd02f13d132aa7a6a9729c6	59345.99853515625
df55c14d1476a9a3467f131269c2477f	47990

3. Operational Efficiency:

1. Analyze delivery performance by calculating the average delivery time by region.

```
• --  Average Delivery Time by Region

WITH delivery_time AS (
    SELECT
        o.customer_id,
        DATEDIFF(o.order_delivered_customer_date,
o.order_purchase_timestamp) AS delivery_time
    FROM
        orders o
    WHERE
        o.order_status = 'delivered'
        AND o.order_delivered_customer_date IS NOT NULL
)
SELECT
    c.customer_state AS region,
    AVG(d.delivery_time) AS avg_delivery_time
FROM
    delivery_time d
JOIN
    customers c
    ON d.customer_id = c.customer_id
GROUP BY
    c.customer_state
ORDER BY
    avg_delivery_time ASC;
```

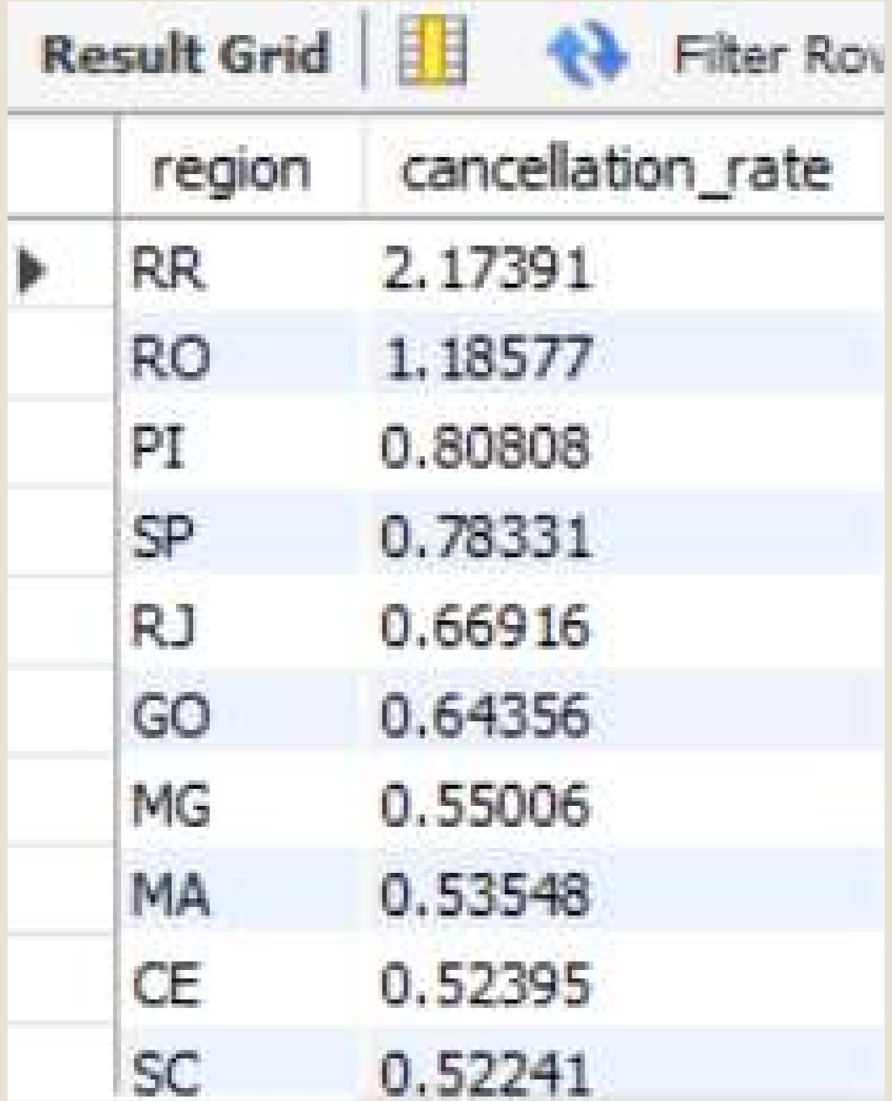


```
   region   avg_delivery_time
▶  SP       8.7006
   PR       11.9380
   MG       11.9450
```

3. Operational Efficiency:

2..Identify regions or products with the highest canceled rates.

```sql
--      Regions with the Highest Canceled Rates
SELECT
    c.customer_state AS region,
     COUNT(CASE WHEN o.order_status = 'canceled' THEN 1 END) * 100.0 / COUNT(*) AS cancellation_rate
FROM
    orders o
JOIN
    customers c
    ON o.customer_id = c.customer_id
GROUP BY
    c.customer_state
ORDER BY
    cancellation_rate DESC
LIMIT 10;
```

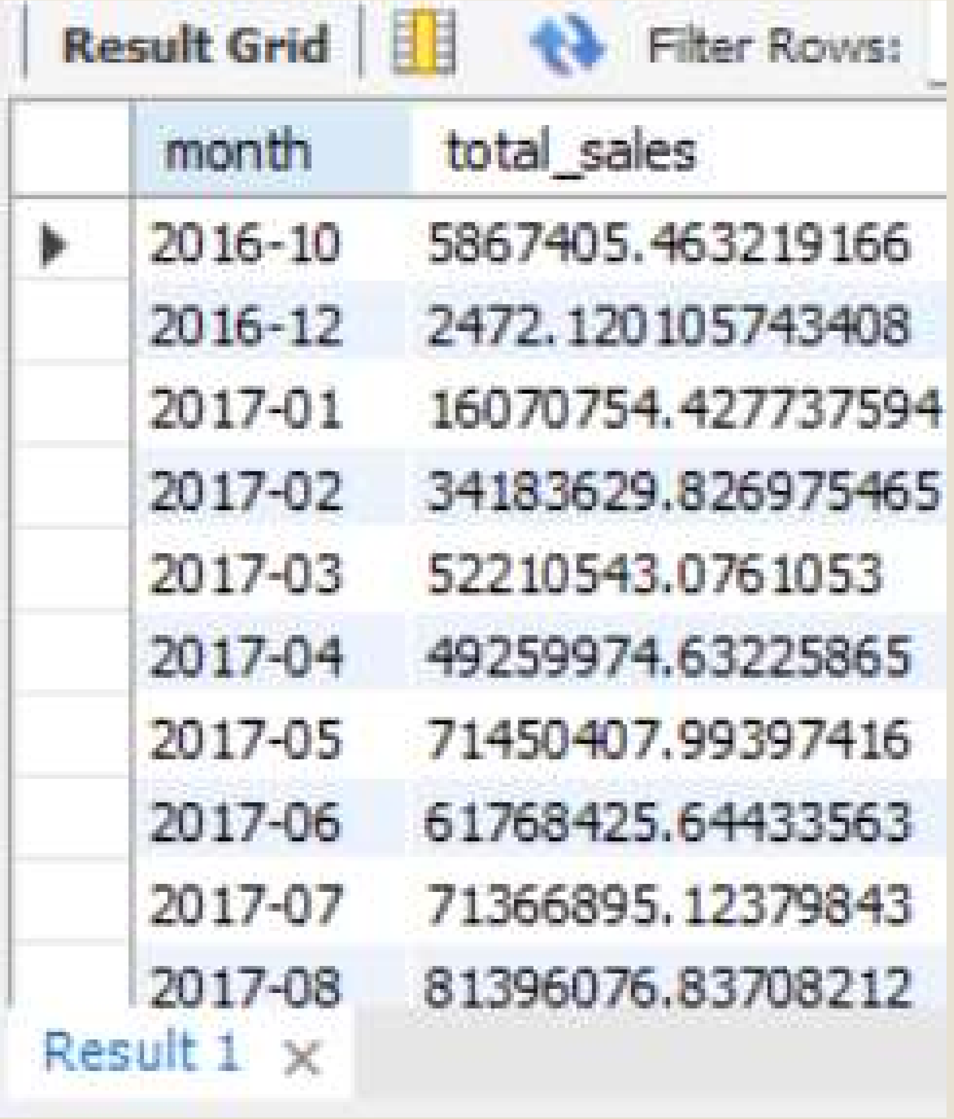


region	cancellation_rate
RR	2.17391
RO	1.18577
PI	0.80808
SP	0.78331
RJ	0.66916
GO	0.64356
MG	0.55006
MA	0.53548
CE	0.52395
SC	0.52241

4. Date and Time Analytics:

1. Write a query to find the monthly sales trend for the last two years.

```
-- Query for Monthly Sales Trend (Last Two Years)
SELECT
    DATE_FORMAT(o.order_purchase_timestamp,
'%Y-%m') AS month,
    SUM(p.payment_value) AS total_sales
FROM
    orders o
JOIN
    payments p
    ON o.order_id = p.order_id
WHERE
    o.order_status = 'delivered'
GROUP BY
    DATE_FORMAT(o.order_purchase_timestamp,
'%Y-%m')
ORDER BY
    month ASC;
```

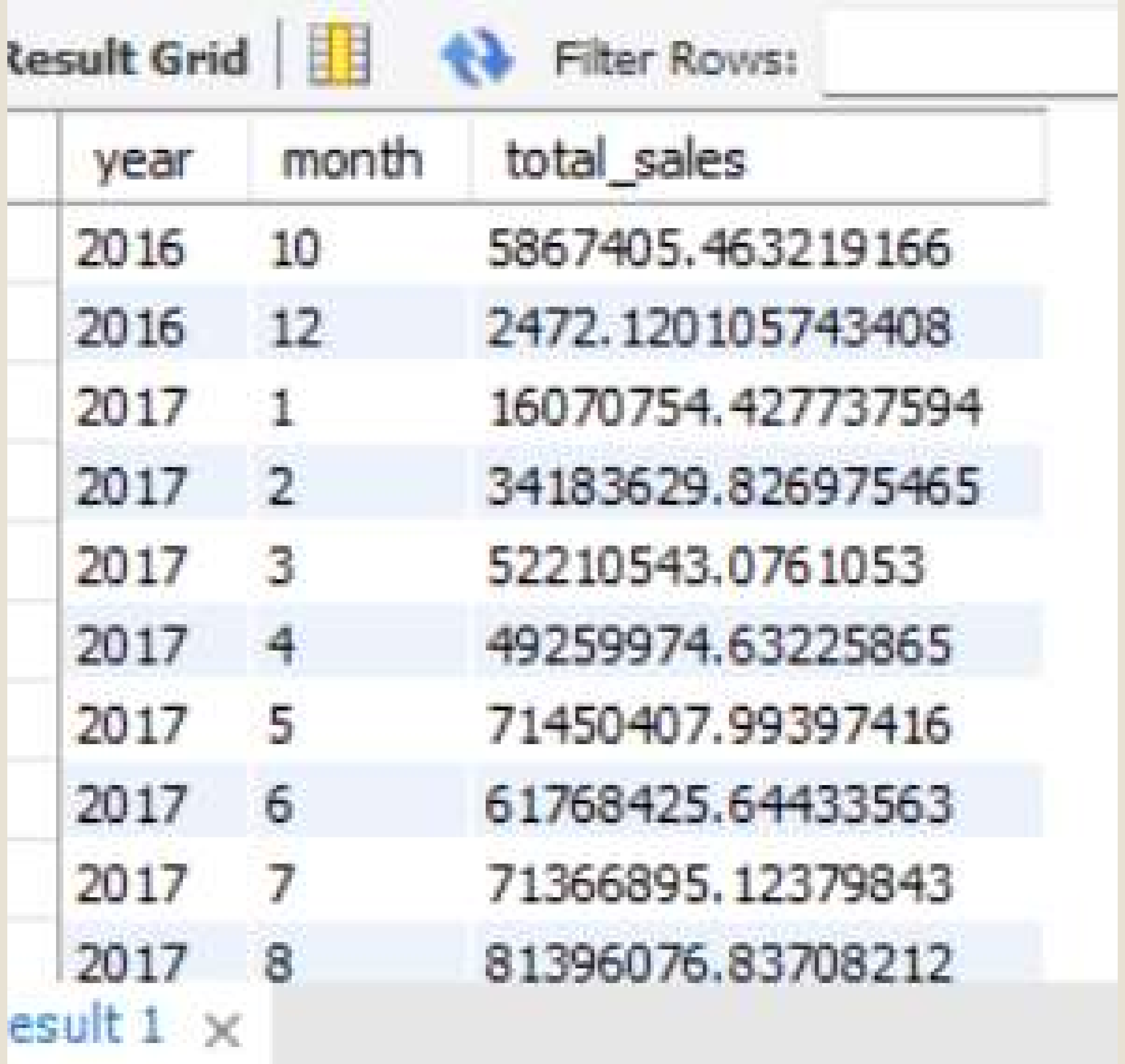


```
month     total_sales
2016-10   5867405.463219166
2016-12   2472.120105743408
2017-01   16070754.427737594
2017-02   34183629.826975465
2017-03   52210543.0761053
2017-04   49259974.63225865
2017-05   71450407.99397416
2017-06   61768425.64433563
2017-07   71366895.12379843
2017-08   81396076.83708212
```

4. Date and Time Analytics:

2. Analyze the seasonality of sales to identify peak months.

```
--  Query for Seasonality Analysis (Peak Months)

SELECT
    YEAR(order_purchase_timestamp) AS year,
    MONTH(order_purchase_timestamp) AS month,
    SUM(p.payment_value) AS total_sales
FROM
    orders o
JOIN
    payments p
    ON o.order_id = p.order_id
WHERE
    o.order_status = 'delivered'
GROUP BY
    YEAR(order_purchase_timestamp),
MONTH(order_purchase_timestamp)
ORDER BY
   year, month;
```

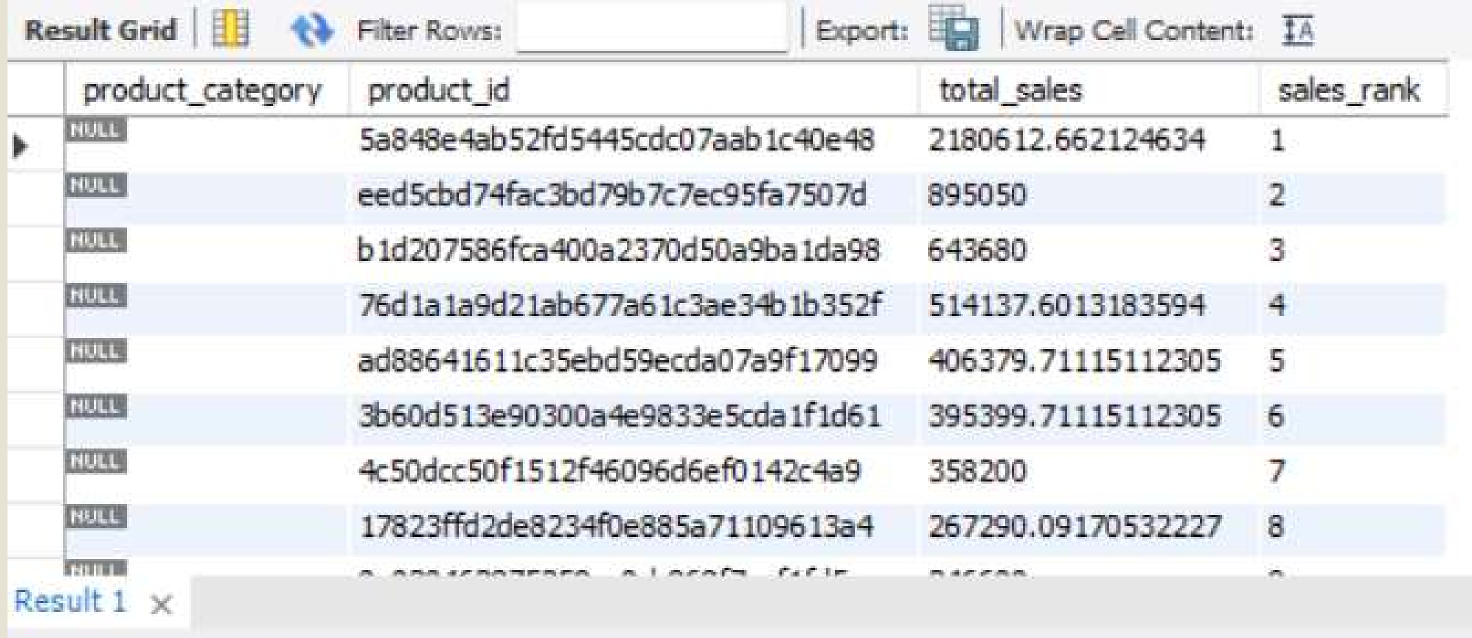


year	month	total_sales
2016	10	5867405.463219166
2016	12	2472.120105743408
2017	1	16070754.427737594
2017	2	34183629.826975465
2017	3	52210543.0761053
2017	4	49259974.63225865
2017	5	71450407.99397416
2017	6	61768425.64433563
2017	7	71366895.12379843
2017	8	81396076.83708212

5. Advanced SQL Queries:

1. Use window functions to rank products based on their sales within each category.

```
-- --  1. Rank Products Based on Sales Within Each
Category using window function
SELECT
    p.product_category,
    oi.product_id,
    SUM(oi.price) AS total_sales,
    RANK() OVER (PARTITION BY p.product_category ORDER
BY SUM(oi.price) DESC) AS sales_rank
FROM
    order_items oi
JOIN
    products p
    ON oi.product_id = p.product_id
GROUP BY
    p.product_category, oi.product_id
ORDER BY
    p.product_category, sales_rank;
```

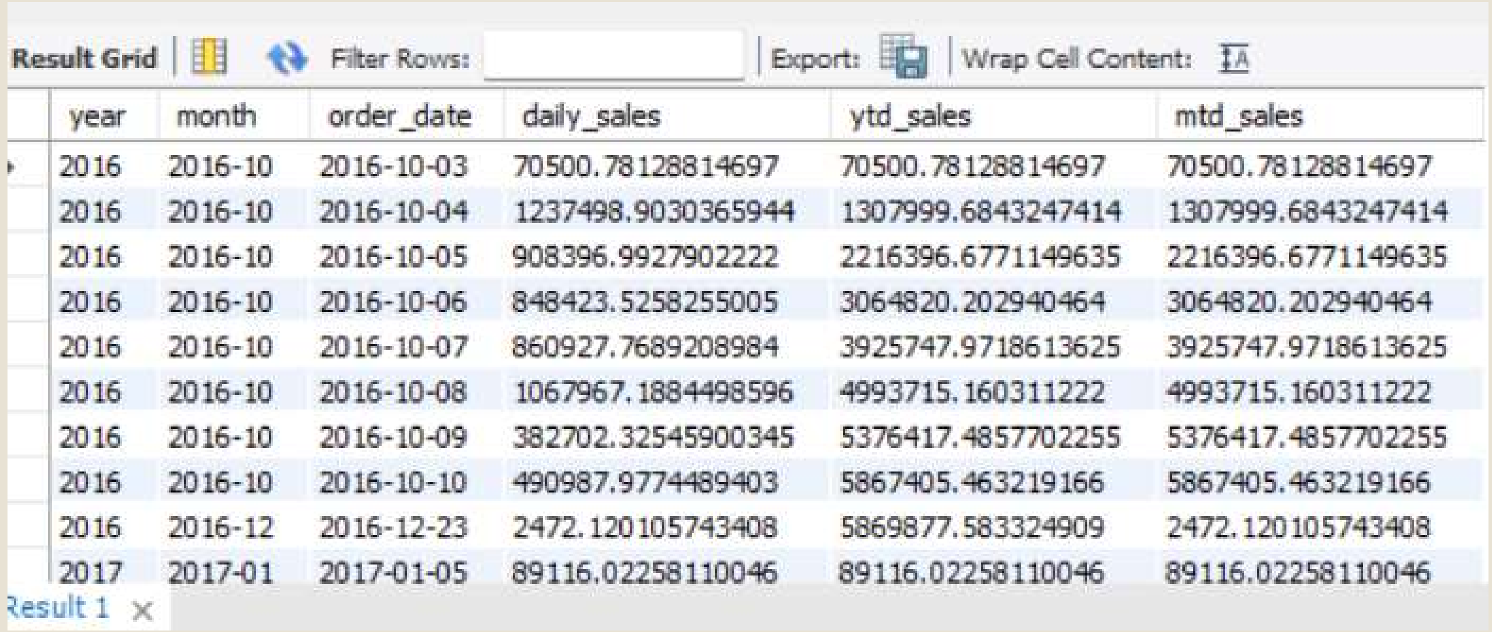


product_category	product_id	total_sales	sales_rank
NULL	5a848e4ab52fd5445cdc07aab1c40e48	2180612.662124634	1
NULL	eed5cbd74fac3bd79b7c7ec95fa7507d	895050	2
NULL	b1d207586fca400a2370d50a9ba1da98	643680	3
NULL	76d1a1a9d21ab677a61c3ae34b1b352f	514137.6013183594	4
NULL	ad88641611c35ebd59ecda07a9f17099	406379.71115112305	5
NULL	3b60d513e90300a4e9833e5cda1f1d61	395399.71115112305	6
NULL	4c50dcc50f1512f46096d6ef0142c4a9	358200	7
NULL	17823ffd2de8234f0e885a71109613a4	267290.09170532227	8

5. Advanced SQL Queries:

2.Calculate month-to-date (MTD) and year-to-date (YTD) sales metrics.

```sql
--  Combining MTD and YTD Metrics
SELECT
    YEAR(o.order_purchase_timestamp) AS year,
    DATE_FORMAT(o.order_purchase_timestamp, '%Y-%m') AS month,
    DATE(o.order_purchase_timestamp) AS order_date,
    SUM(p.payment_value) AS daily_sales,
    SUM(SUM(p.payment_value)) OVER (
        PARTITION BY YEAR(o.order_purchase_timestamp)
        ORDER BY DATE(o.order_purchase_timestamp)
    ) AS ytd_sales,
    SUM(SUM(p.payment_value)) OVER (
        PARTITION BY DATE_FORMAT(o.order_purchase_timestamp, '%Y-%m')
        ORDER BY DATE(o.order_purchase_timestamp)
    ) AS mtd_sales
FROM
    orders o
JOIN
    payments p
    ON o.order_id = p.order_id
WHERE
    o.order_status = 'delivered'
GROUP BY
    year, month, order_date
ORDER BY
    year, month, order_date;
```

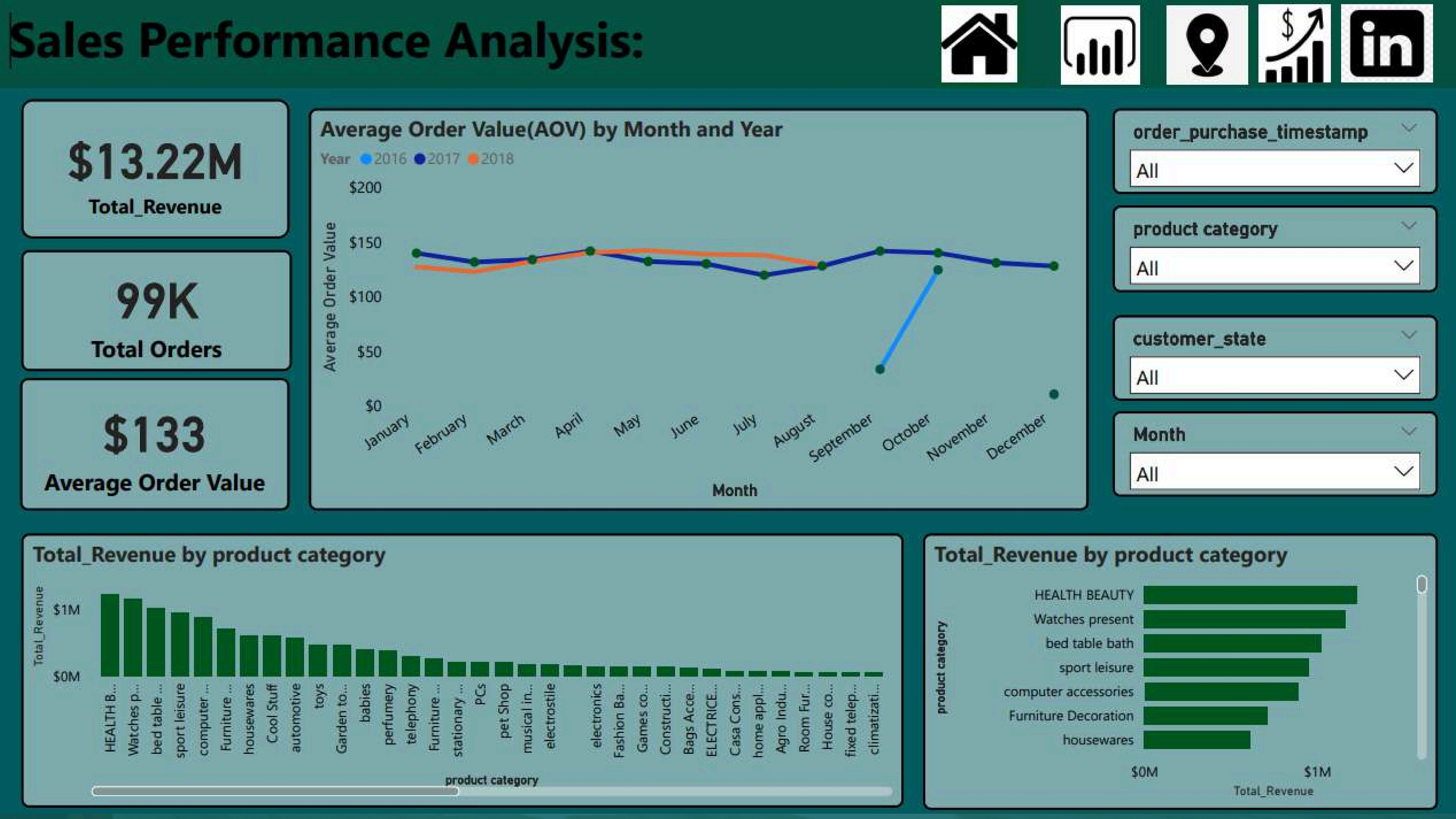


year	month	order_date	daily_sales	ytd_sales	mtd_sales
2016	2016-10	2016-10-03	70500.78128814697	70500.78128814697	70500.78128814697
2016	2016-10	2016-10-04	1237498.9030365944	1307999.6843247414	1307999.6843247414
2016	2016-10	2016-10-05	908396.9927902222	2216396.677149635	2216396.677149635
2016	2016-10	2016-10-06	848423.5258255005	3064820.202940464	3064820.202940464
2016	2016-10	2016-10-07	860927.7689208984	3925747.9718613625	3925747.9718613625
2016	2016-10	2016-10-08	1067967.1884498596	4993715.160311222	4993715.160311222
2016	2016-10	2016-10-09	382702.32545900345	5376417.4857702255	5376417.4857702255
2016	2016-10	2016-10-10	490987.9774489403	5867405.463219166	5867405.463219166
2016	2016-12	2016-12-23	2472.120105743408	5869877.583324909	2472.120105743408
2017	2017-01	2017-01-05	89116.02258110046	89116.02258110046	89116.02258110046

Phase 2: Data Visualization – Power BI

About Power BI

- **DAX Queries**: Used for creating complex calculations and data manipulations.
- **Visuals**: Allows users to create a wide range of charts and visuals (e.g., bar charts, line graphs, maps) to represent data insights.
- **Filters**: Enable users to segment data and focus on specific insights for better analysis.
- **Data Modeling**: Helps structure data relationships to ensure consistency and accuracy in reports.
- **Calendar Table**: Used for time-based analysis, supporting time intelligence functions like year-over-year comparisons and trend analysis.



Sales Performance Analysis:

$13.22M
Total_Revenue

99K
Total Orders

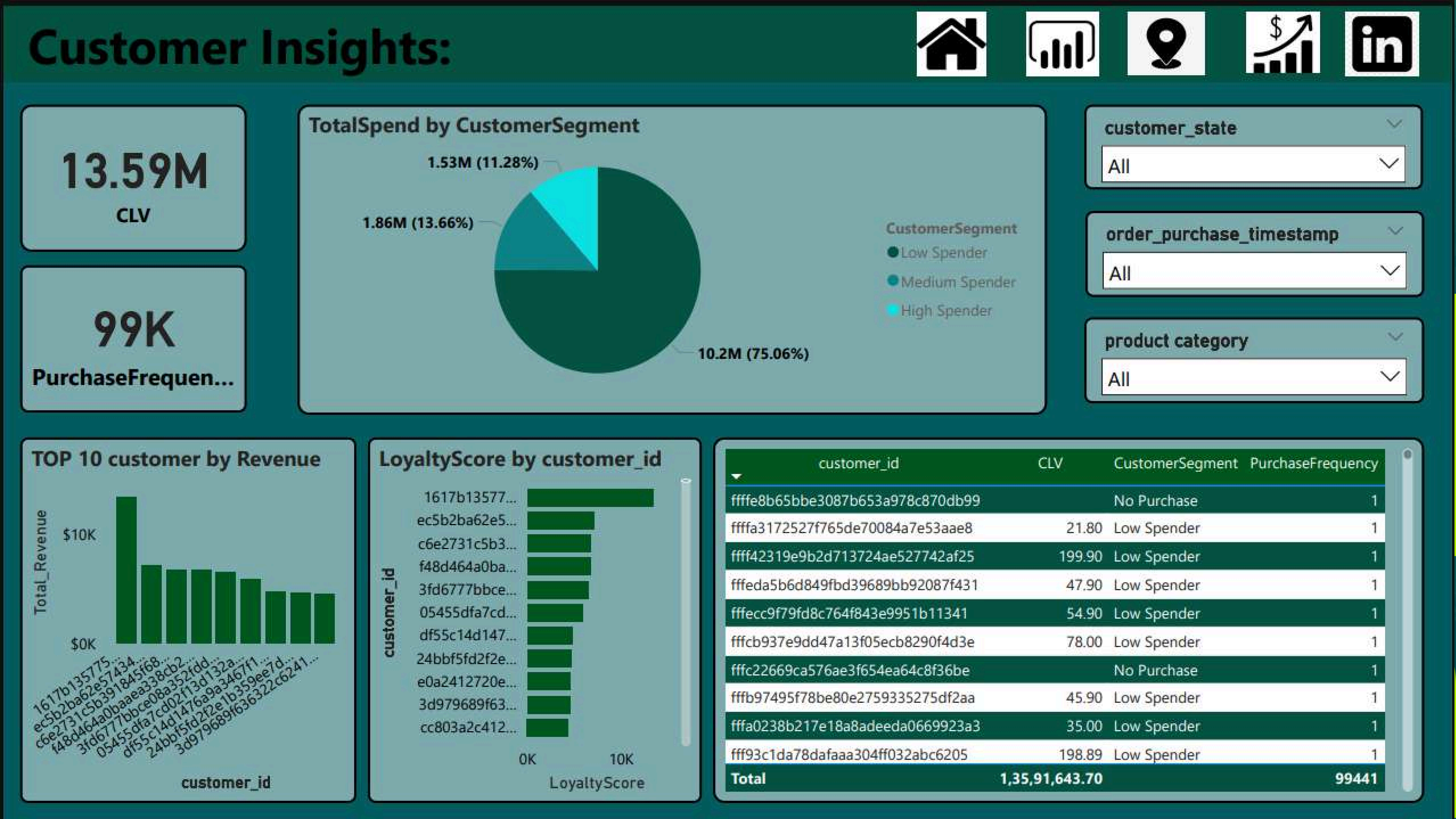
$133
Average Order Value

order_purchase_timestamp: All

product category: All

customer_state: All

Month: All



Customer Insights:

13.59M

CLV

99K

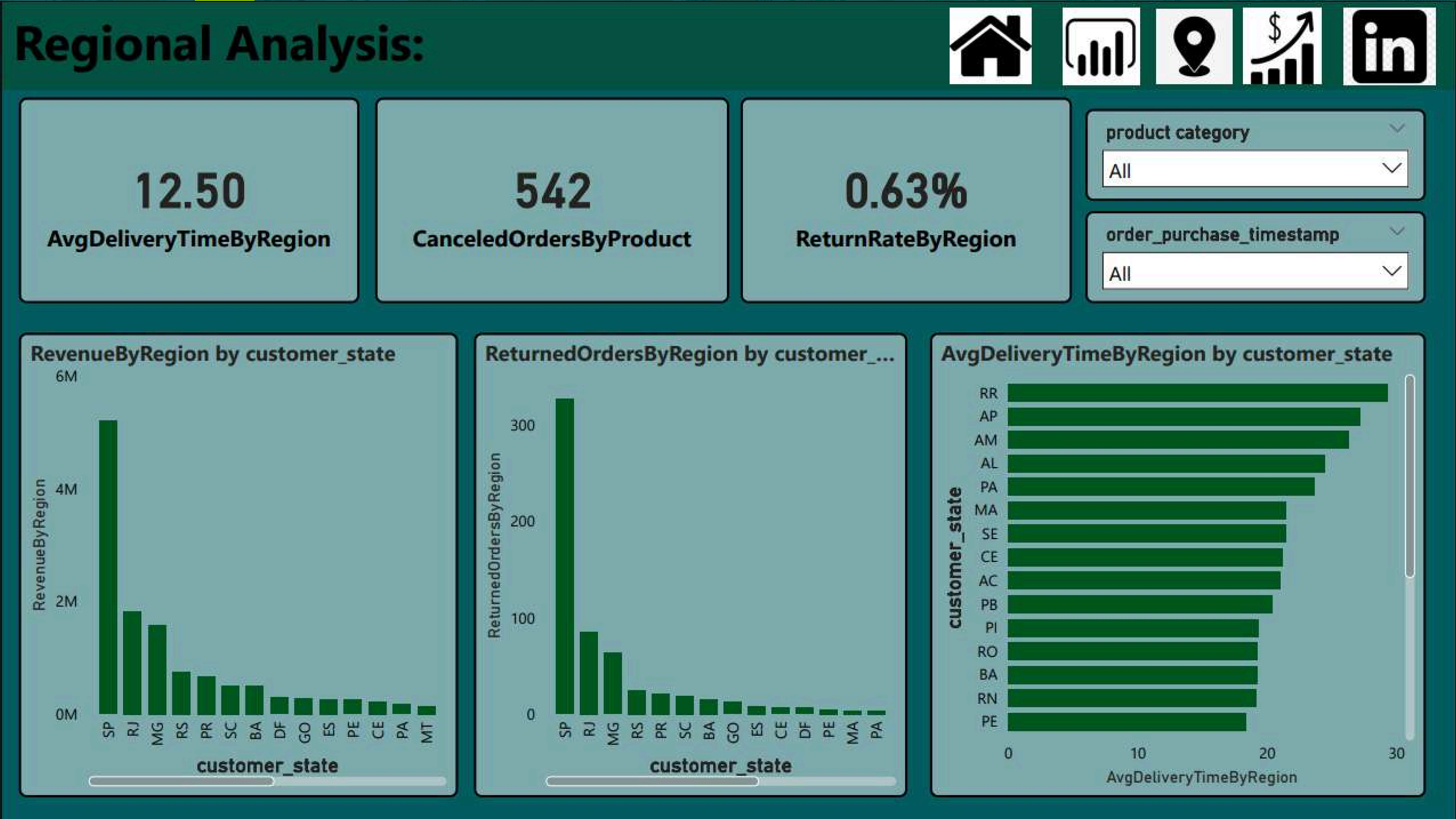
PurchaseFrequen...

customer_state: All

order_purchase_timestamp: All

product category: All

customer_id	CLV	CustomerSegment	PurchaseFrequency
ffffe8b65bbe3087b653a978c870db99		No Purchase	1
ffffa3172527f765de70084a7e53aae8	21.80	Low Spender	1
ffff42319e9b2d713724ae527742af25	199.90	Low Spender	1
fffeda5b6d849fbd39689bb92087f431	47.90	Low Spender	1
fffecc9f79fd8c764f843e9951b11341	54.90	Low Spender	1
fffcb937e9dd47a13f05ecb8290f4d3e	78.00	Low Spender	1
fffc22669ca576ae3f654ea64c8f36be		No Purchase	1
fffb97495f78be80e2759335275df2aa	45.90	Low Spender	1
fffa0238b217e18a8adeeda0669923a3	35.00	Low Spender	1
fff93c1da78dafaaa304ff032abc6205	198.89	Low Spender	1
Total	1,35,91,643.70		99441



Regional Analysis:

12.50
AvgDeliveryTimeByRegion

542
CanceledOrdersByProduct

0.63%
ReturnRateByRegion

product category
All

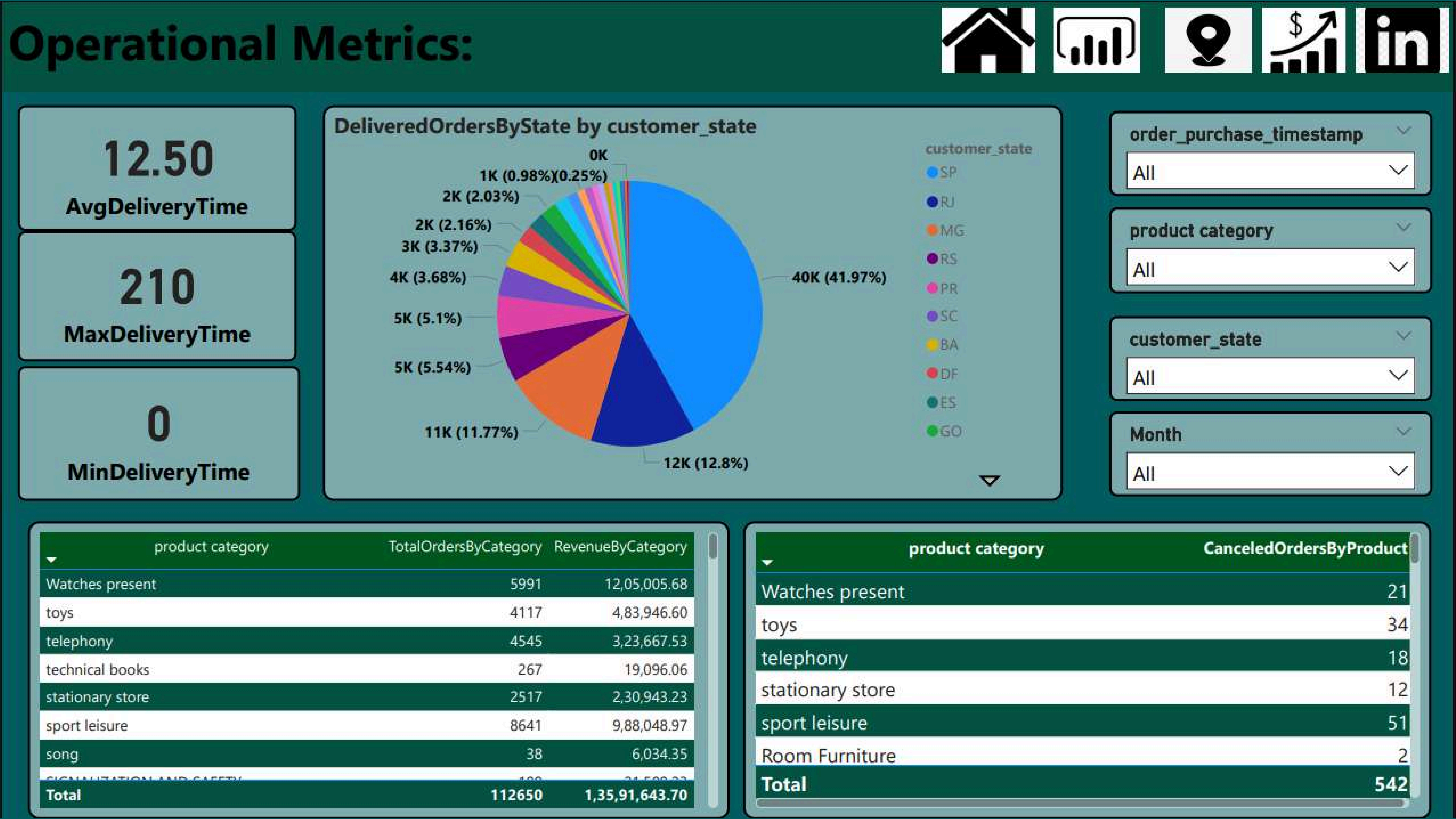
order_purchase_timestamp
All

RevenueByRegion by customer_state

ReturnedOrdersByRegion by customer_...

AvgDeliveryTimeByRegion by customer_state

Operational Metrics:



12.50

AvgDeliveryTime

210

MaxDeliveryTime

0

MinDeliveryTime

DeliveredOrdersByState by customer_state

order_purchase_timestamp: All

product category: All

customer_state: All

Month: All

product category	TotalOrdersByCategory	RevenueByCategory
Watches present	5991	12,05,005.68
toys	4117	4,83,946.60
telephony	4545	3,23,667.53
technical books	267	19,096.06
stationary store	2517	2,30,943.23
sport leisure	8641	9,88,048.97
song	38	6,034.35
SIGNALIZATION AND SAFETY	[illegible]	[illegible]
Total	**112650**	**1,35,91,643.70**

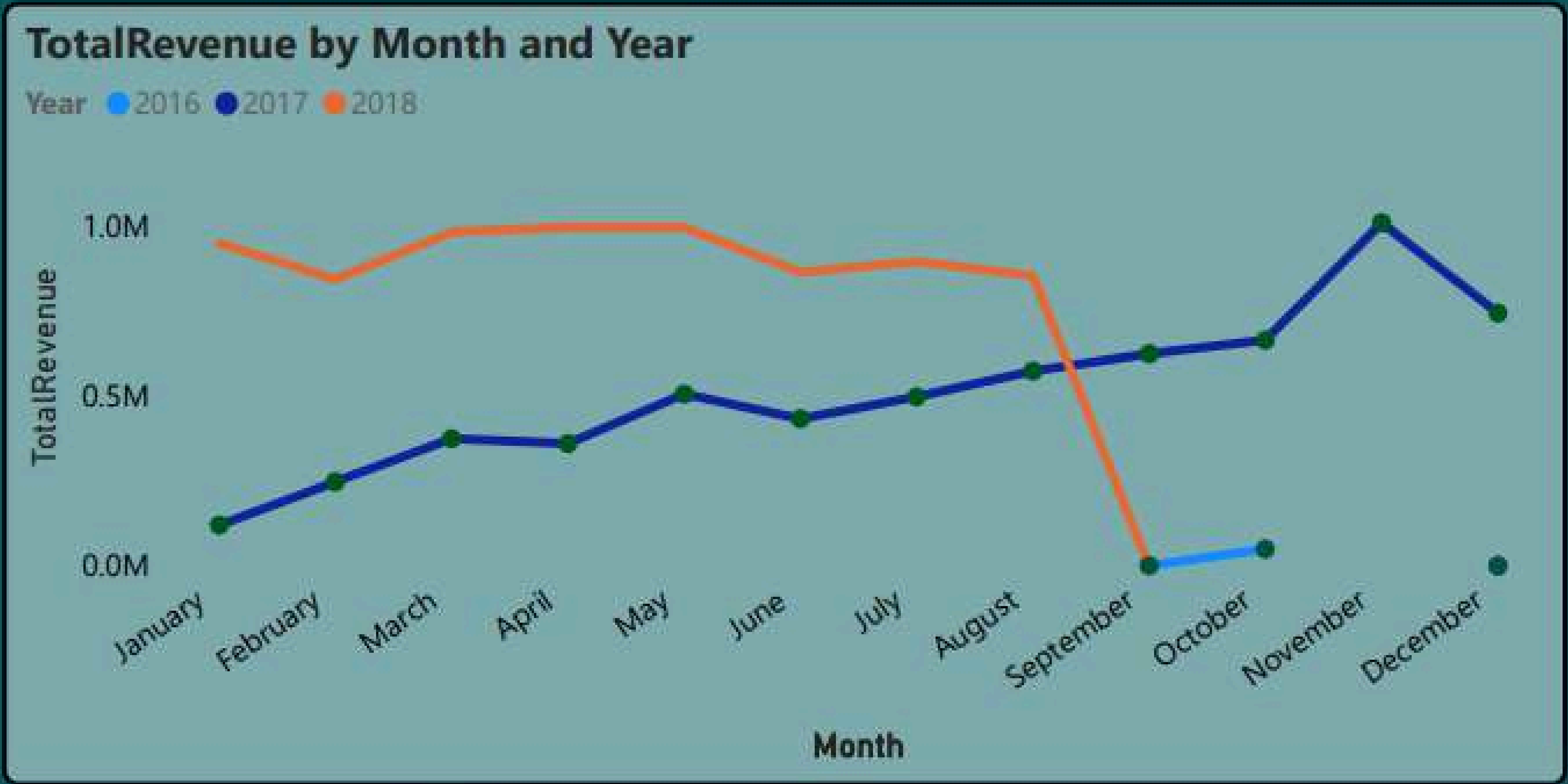
product category	CanceledOrdersByProduct
Watches present	21
toys	34
telephony	18
stationary store	12
sport leisure	51
Room Furniture	2
Total	**542**

Time Trends:

$13.59M
TotalRevenue

0%
MoM_Growth

250.3%
YoY_Growth


TotalRevenue by Month and Year

Year
All

Month
All

customer_state
All

Year	TotalRevenue	YoY_Growth
2017	55,17,496.55	132.98
2018	80,24,098.21	1.10
	893.90	0.00
2016	49,155.04	0.00
Total	**1,35,91,643.70**	**2.50**

product category	MoM_Growth	YoY_Growth	TotalRevenue
Watches present	0.00	3.42	12,05,005.68
toys	0.00	1.86	4,83,946.60
telephony	0.00	2.89	3,23,667.53
technical books	0.01	4.38	19,096.06
stationary store	0.00	4.15	2,30,943.23
sport leisure	0.00	2.43	9,88,048.97
song	0.00	3.01	6,034.25
Total	**0.00**	**2.50**	**1,35,91,643.70**

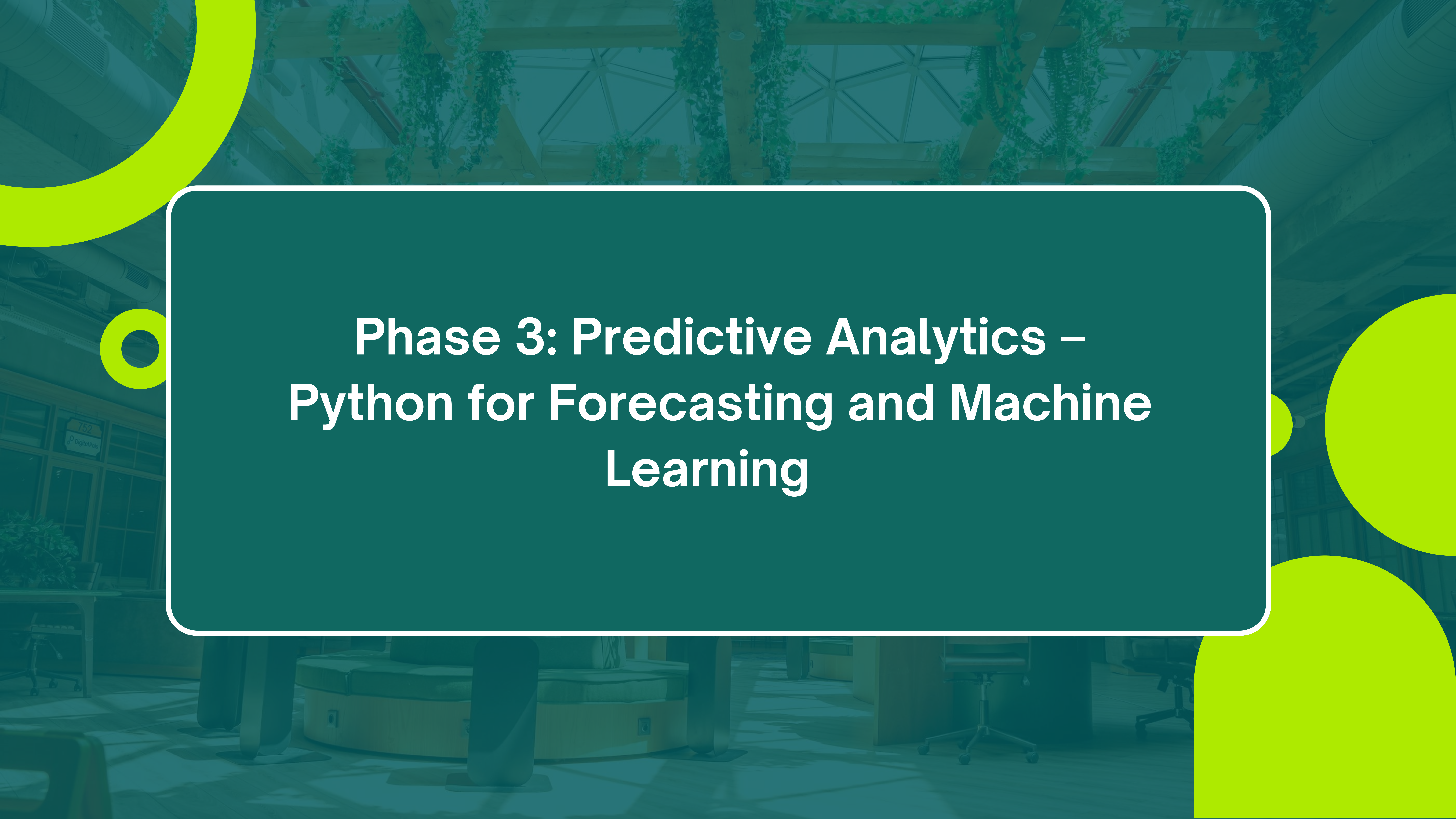
Key Findings

Revenue Insights:
Total revenue: $13.22M, with an Average Order Value (AOV) of $133.04.
Health and Beauty is the top product category, contributing the most to revenue.

Customer Insights:
Loyal customers show high purchase frequency and contribute significantly to revenue.
The top 10 customers were identified using the Loyalty Factor, combining purchase frequency and spending metrics.

Operational Efficiency:
Average delivery time: 12.50 days, with delays in some regions.
Peak sales months and weekday/weekend patterns identified for targeted planning.

Order Insights:
Total products ordered**:** 99K, with 542 cancellations, highlighting areas for improvement.
Strategic seasonal marketing campaigns during peak months to maximize sales.



Phase 3: Predictive Analytics – Python for Forecasting and Machine Learning

About Python

Python Libraries

1. **Pandas**
    - **Purpose**: Data manipulation and analysis.
    - **Usage:** Used for reading, merging, filtering, and handling datasets (e.g., merged_items DataFrame).
2. **NumPy**
    - **Purpose:** Numerical computing.
    - **Usage:** For creating arrays, handling numerical data, and initializing binary data for the sparse matrix.
3. **SciPy**
    - **Module Used**: scipy.sparse.coo_matrix
    - **Purpose:** Efficient storage and computation with sparse matrices.
    - **Usage:** Used to construct a binary sparse matrix to represent the product-order relationship.
4. **scikit-learn**
    - **Module Used:** sklearn.metrics.pairwise.cosine_similarity
    - **Purpose:** Machine learning and statistical tools.
    - **Usage:** Used for computing pairwise cosine similarity to identify product correlations.

Development Environment

- **Jupyter Notebook**
    - **Purpose:** Interactive environment for writing, testing, and visualizing Python code.
    - **Usage:** Used as the primary platform to execute and document the code.

Tasks and Techniques:

Q1. Sales Prediction:

- **Use linear regression to predict monthly sales for the next 12 months.**
- **Identify key factors (e.g., past sales, promotions, and external variables) influencing sales trends**

Step 1: Define the Problem

- We aim to predict monthly sales for the next 12 months using linear regression. We will:
- Use the orders.csv and order_items.csv tables to calculate historical sales.
- Aggregate sales data to derive monthly sales.
- Use the historical data as a basis to train a linear regression model.

Step 2: Prepare the Data

a) Load the necessary tables

- We will focus on the following tables:
- orders.csv: To extract the order dates (order_purchase_timestamp) and associate them with order IDs. order_items.csv: To get the sales amount for each order using the price column.

```
import pandas as pd

# Load datasets
orders = pd.read_csv("E:\BHU\Data_Analyst\Ecommerce\orders.csv")
order_items = pd.read_csv("E:\BHU\Data_Analyst\Ecommerce\order_items.csv")

# Display initial datasets
print(orders.head())
print(order_items.head())
```

b) Merge and clean the data

- **Join orders and order_items on the order_id column to combine order dates with sales data.**

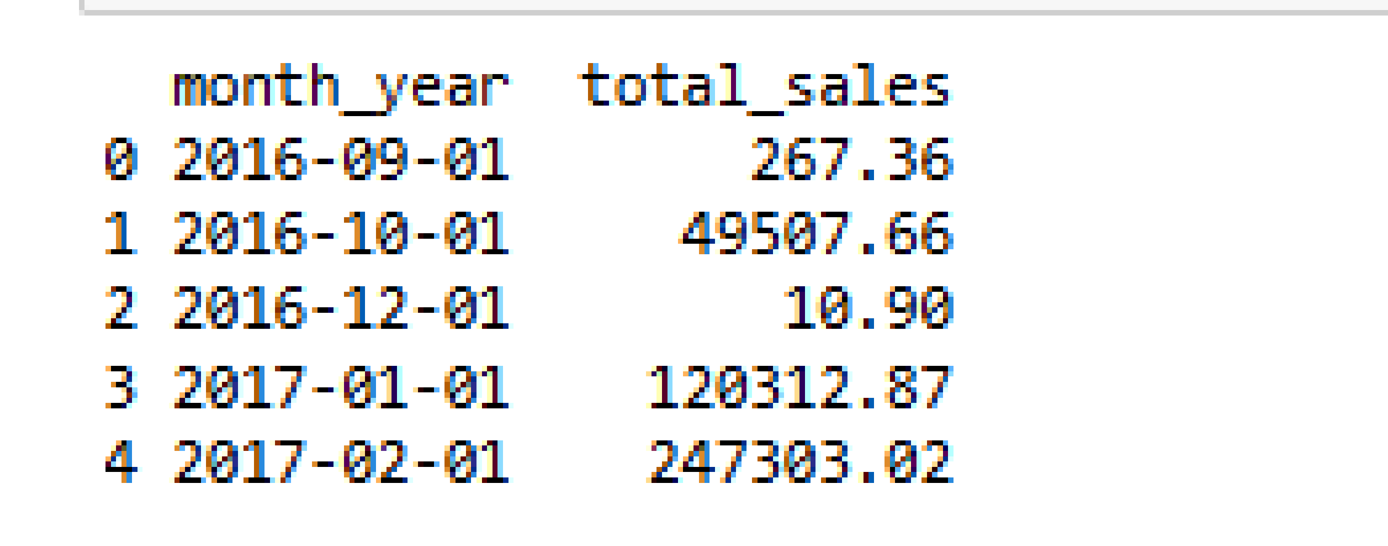
```
# Merge orders with order_items
merged_data = pd.merge(order_items, orders, on="order_id")

# Ensure timestamps are in datetime format
merged_data['order_purchase_timestamp'] = pd.to_datetime(merged_data['order_purchase_timestamp'])

# Drop rows with missing order dates or sales amounts
merged_data = merged_data.dropna(subset=['order_purchase_timestamp', 'price'])

# Display the cleaned merged dataset
print(merged_data.head())
```

c) Aggregate monthly sales

- **Aggregate the sales data (price) by month and calculate total monthly sales.**

```
# Extract month and year
merged_data['month_year'] = merged_data['order_purchase_timestamp'].dt.to_period('M')

# Calculate monthly sales
monthly_sales = merged_data.groupby('month_year')['price'].sum().reset_index()
monthly_sales.rename(columns={'price': 'total_sales'}, inplace=True)

# Convert month_year back to datetime for plotting and modeling
monthly_sales['month_year'] = monthly_sales['month_year'].astype('datetime64[ns]')

# Display monthly sales data
print(monthly_sales.head())
```

Output:-



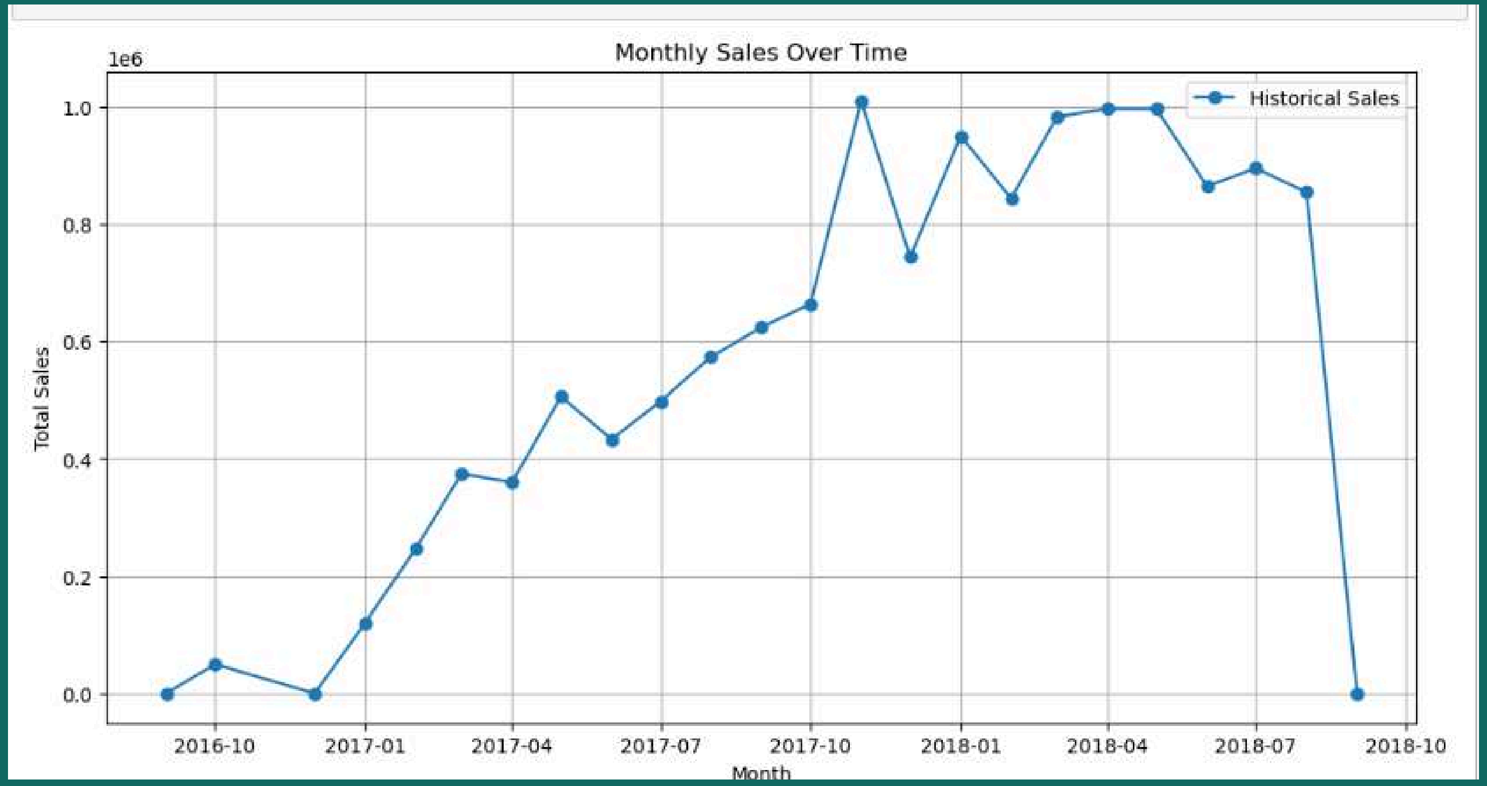
```
  month_year  total_sales
0 2016-09-01       267.36
1 2016-10-01     49507.66
2 2016-12-01        10.90
3 2017-01-01    120312.87
4 2017-02-01    247303.02
```

Step 3: Visualize the Data

- **Visualize historical monthly sales to understand trends and seasonality.**

```
import matplotlib.pyplot as plt

# Plot the historical monthly sales
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales['month_year'], monthly_sales['total_sales'], marker='o', label="Historical Sales")
plt.title("Monthly Sales Over Time")
plt.xlabel("Month")
plt.ylabel("Total Sales")
plt.grid(True)
plt.legend()
plt.show()
```

Output:-



Step 4: Build the Linear Regression Model

a) Prepare the features and target variable

Create a time index (X) and corresponding sales (y).

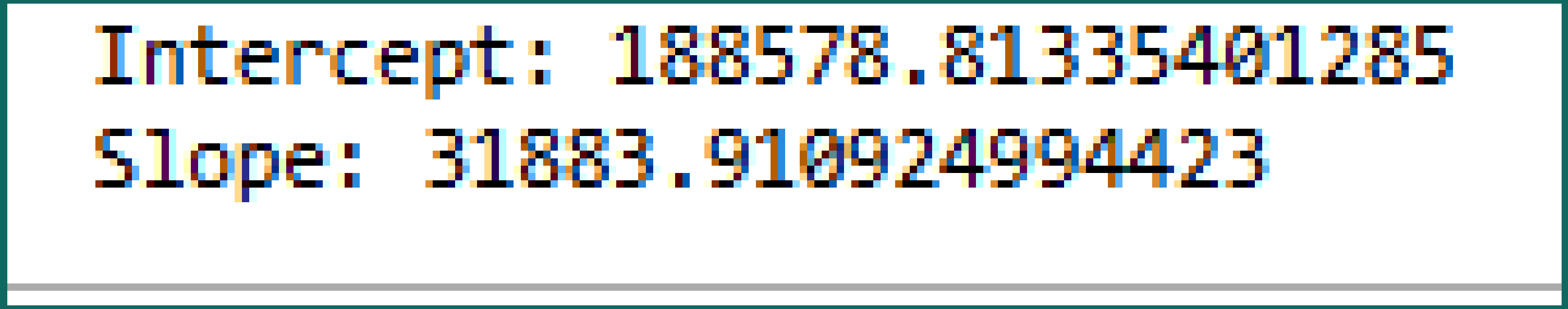
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np

# Create time index as a numerical feature
monthly_sales['time_index'] = np.arange(len(monthly_sales))

# Features (time index) and target (sales)
X = monthly_sales[['time_index']]  # Predictor
y = monthly_sales['total_sales']  # Target

# Split the data into training and test sets (80%-20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

b) Train the model

Train a linear regression model using the training data.

```
# Initialize and train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Display the model coefficients
print(f"Intercept: {model.intercept_}")
print(f"Slope: {model.coef_[0]}")
```

Output:-



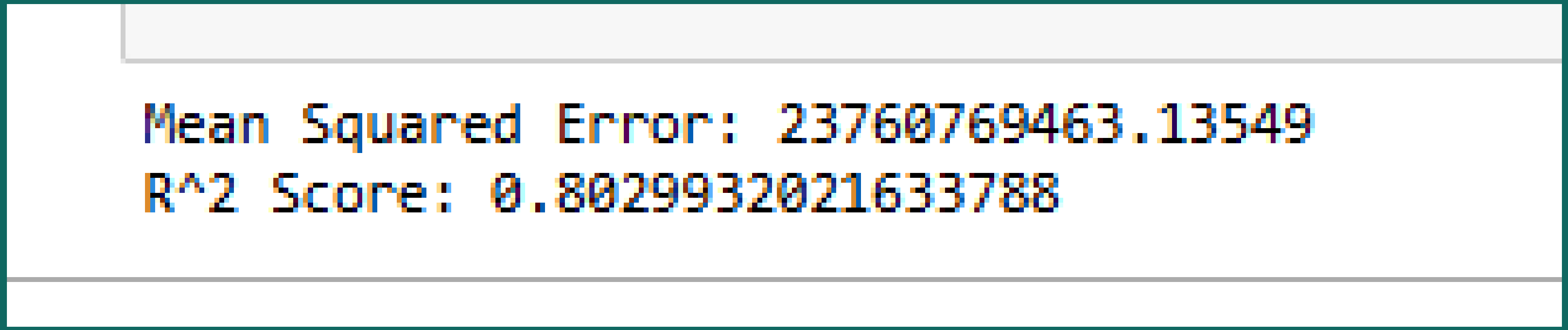
```
Intercept: 188578.81335401285
Slope: 31883.910924994423
```

b) Evaluate the model

Check the model's performance on the test set.

```
from sklearn.metrics import mean_squared_error, r2_score

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")
```

Output:-



```
Mean Squared Error: 23760769463.13549
R^2 Score: 0.8029932021633788
```

Step 5: Forecast Sales for the Next 12 Months

a) Extend the time index

Generate future time indices for the next 12 months.

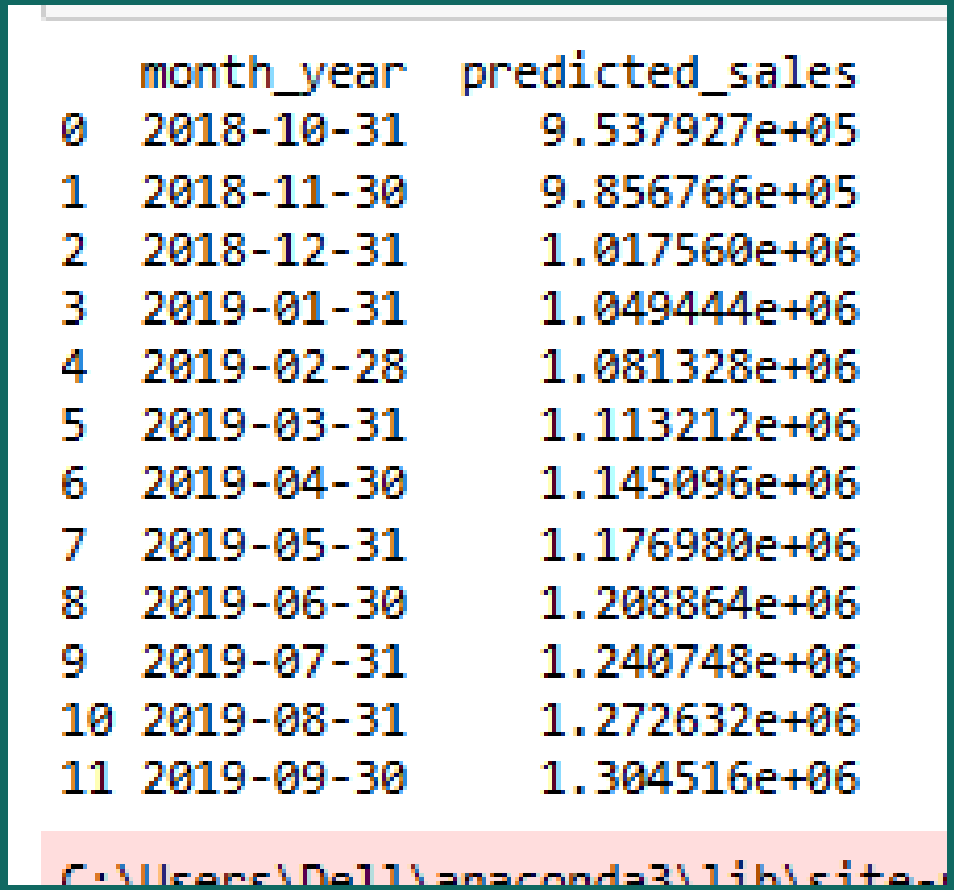
```
# Get the last time index
last_time_index = monthly_sales['time_index'].max()

# Create future time indices
future_time_indices = np.arange(last_time_index + 1, last_time_index + 13).reshape(-1, 1)
```

b) Predict future sales

Use the trained model to forecast sales.

```
# Predict future sales
future_sales = model.predict(future_time_indices)

# Create a DataFrame for future predictions
future_months = pd.date_range(start=monthly_sales['month_year'].max() + pd.offsets.MonthBegin(1),
periods=12, freq='M')
future_sales_df = pd.DataFrame({
    'month_year': future_months,
    'predicted_sales': future_sales
})

# Display future sales predictions
print(future_sales_df)
```

Output:-

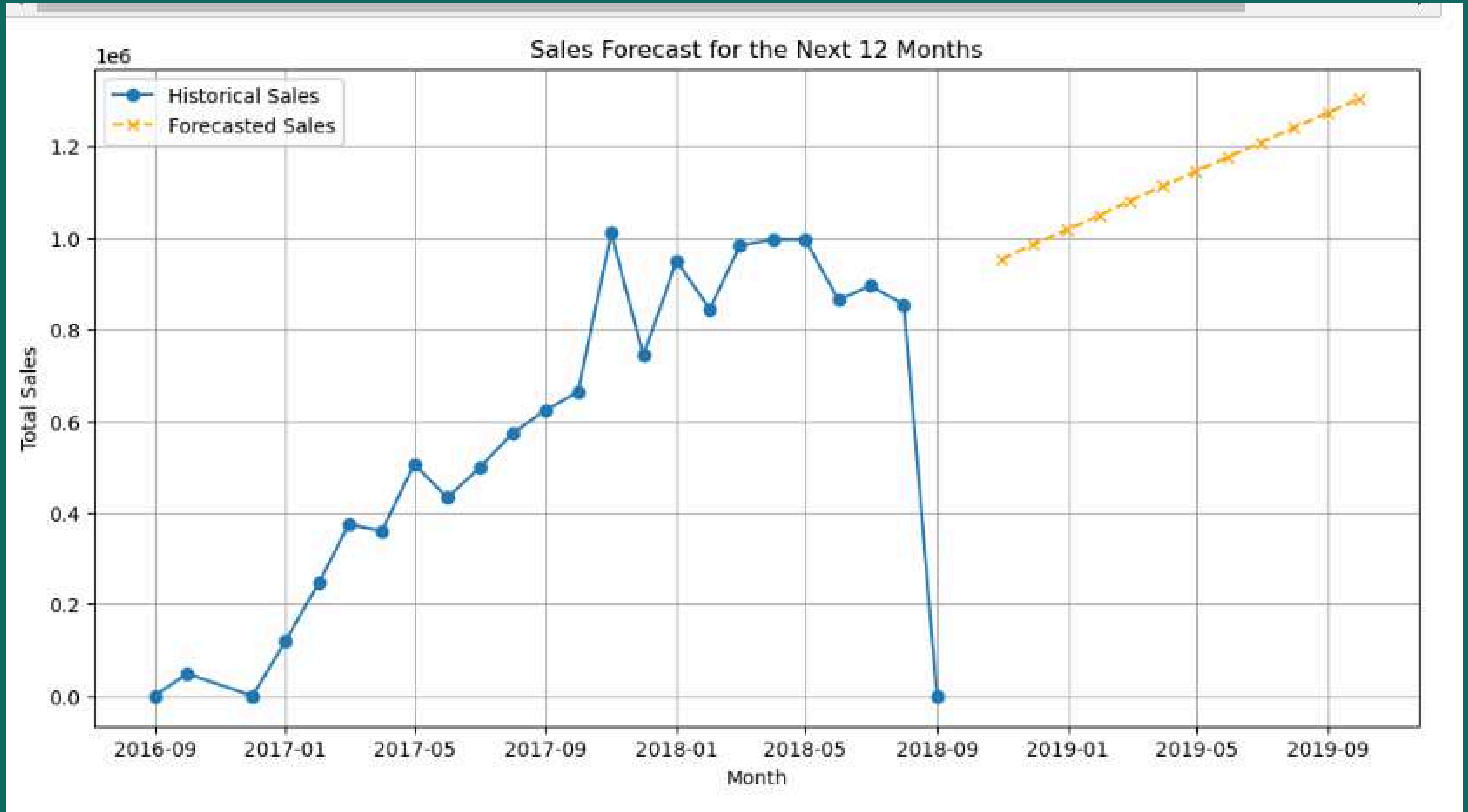


```
   month_year  predicted_sales
0  2018-10-31     9.537927e+05
1  2018-11-30     9.856766e+05
2  2018-12-31     1.017560e+06
3  2019-01-31     1.049444e+06
4  2019-02-28     1.081328e+06
5  2019-03-31     1.113212e+06
6  2019-04-30     1.145096e+06
7  2019-05-31     1.176980e+06
8  2019-06-30     1.208864e+06
9  2019-07-31     1.240748e+06
10 2019-08-31     1.272632e+06
11 2019-09-30     1.304516e+06
C:\Users\Dell\anaconda3\lib\site-
```

c) Visualize the forecast
Combine historical sales with forecasted sales for visualization.

```
# Plot historical and forecasted sales
plt.figure(figsize=(12, 6))
plt.plot(monthly_sales['month_year'], monthly_sales['total_sales'], marker='o', label="Historical Sales")
plt.plot(future_sales_df['month_year'], future_sales_df['predicted_sales'], marker='x', linestyle='--',
color='orange', label="Forecasted Sales")
plt.title("Sales Forecast for the Next 12 Months")
plt.xlabel("Month")
plt.ylabel("Total Sales")
plt.grid(True)
plt.legend()
plt.show()
```

Output:-



Q2. Customer Segmentation (Clustering):

Perform RFM Analysis (Recency, Frequency, Monetary) and use simple clustering methods like hierarchical clustering or manual thresholds to classify customers.

Label customer segments (e.g., Loyal Customers, Discount Seekers) to identify marketing opportunities.

Step 1: Define the Objective

We aim to segment customers using RFM Analysis:

Recency: How recently a customer made a purchase.

Frequency: How often a customer made purchases.

Monetary: How much a customer spent.

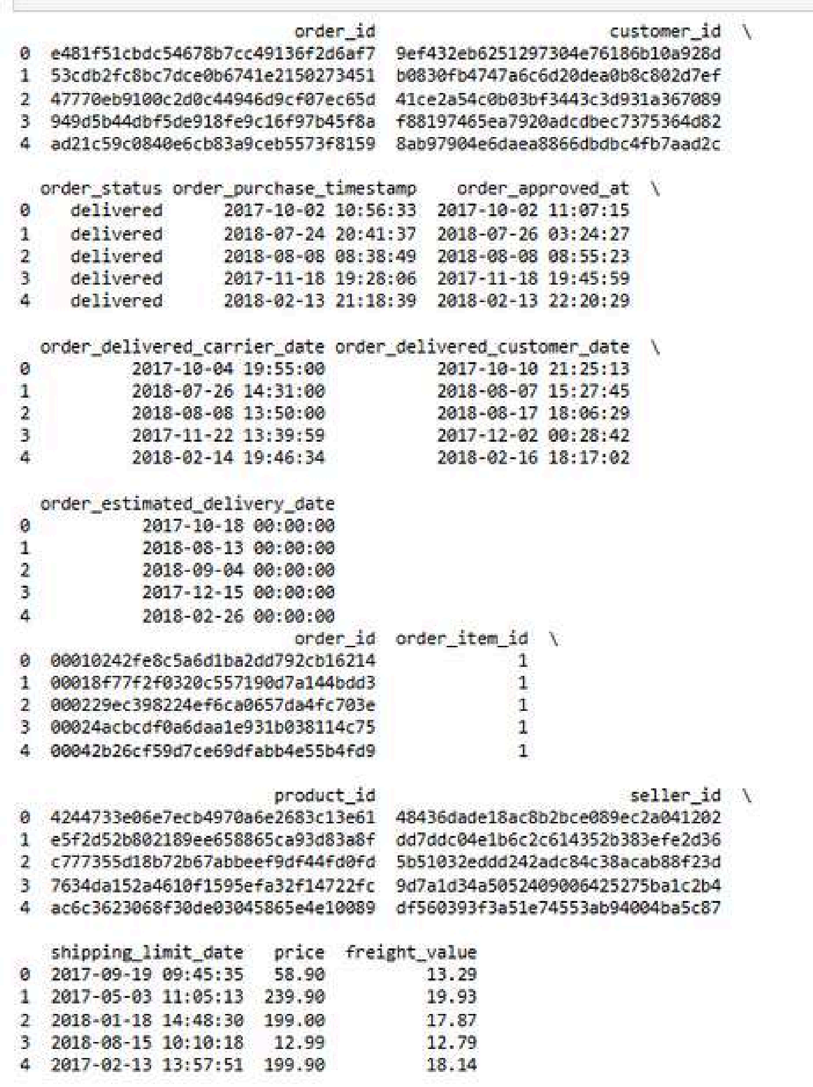
Step 2: Prepare the Data

a) Load the necessary tables

Load the orders.csv and order_items.csv tables.

```
import pandas as pd

# Load datasets
orders = pd.read_csv("E:\BHU\Data_Analyst\Ecommerce\orders.csv")
order_items = pd.read_csv("E:\BHU\Data_Analyst\Ecommerce\order_items.csv")

# Display the first few rows of the datasets
print(orders.head())
print(order_items.head())
```

Output:-



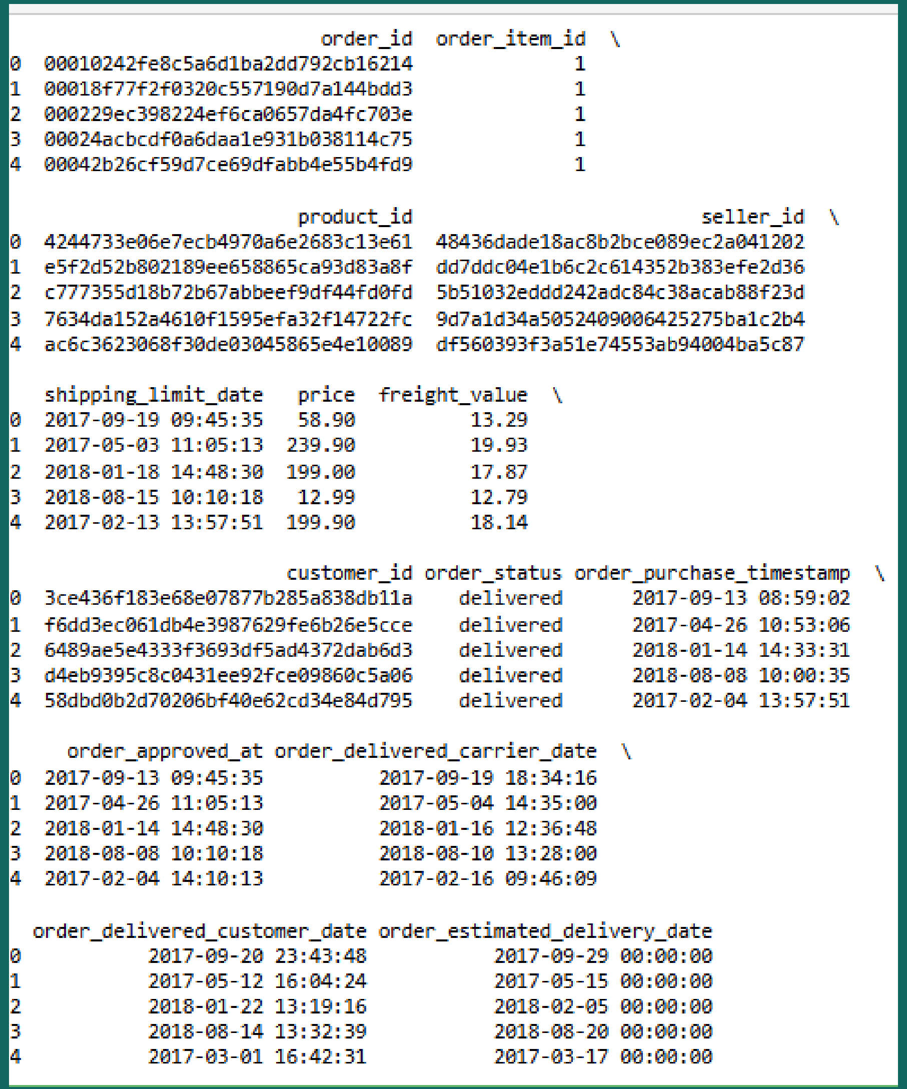
```
                           order_id                       customer_id  \
0  e481f51cbdc54678b7cc49136f2d6af7  9ef432eb6251297304e76186b10a928d
1  53cdb2fc8bc7dce0b6741e2150273451  b0830fb4747a6c6d20dea0b8c802d7ef
2  47770eb9100c2d0c44946d9cf07ec65d  41ce2a54c0b03bf3443c3d931a367089
3  949d5b44dbf5de918fe9c16f97b45f8a  f88197465ea7920adcdbec7375364d82
4  ad21c59c0840e6cb83a9ceb5573f8159  8ab97904e6daea8866dbdbc4fb7aad2c

  order_status order_purchase_timestamp    order_approved_at  \
0    delivered      2017-10-02 10:56:33  2017-10-02 11:07:15
1    delivered      2018-07-24 20:41:37  2018-07-26 03:24:27
2    delivered      2018-08-08 08:38:49  2018-08-08 08:55:23
3    delivered      2017-11-18 19:28:06  2017-11-18 19:45:59
4    delivered      2018-02-13 21:18:39  2018-02-13 22:20:29

  order_delivered_carrier_date order_delivered_customer_date  \
0          2017-10-04 19:55:00           2017-10-10 21:25:13
1          2018-07-26 14:31:00           2018-08-07 15:27:45
2          2018-08-08 13:50:00           2018-08-17 18:06:29
3          2017-11-22 13:39:59           2017-12-02 00:28:42
4          2018-02-14 19:46:34           2018-02-16 18:17:02

  order_estimated_delivery_date
0           2017-10-18 00:00:00
1           2018-08-13 00:00:00
2           2018-09-04 00:00:00
3           2017-12-15 00:00:00
4           2018-02-26 00:00:00
                           order_id  order_item_id  \
0  00010242fe8c5a6d1ba2dd792cb16214              1
1  00018f77f2f0320c557190d7a144bdd3              1
2  000229ec398224ef6ca0657da4fc703e              1
3  00024acbcdf0a6daa1e931b038114c75              1
4  00042b26cf59d7ce69dfabb4e55b4fd9              1

                         product_id                         seller_id  \
0  4244733e06e7ecb4970a6e2683c13e61  48436dade18ac8b2bce089ec2a041202
1  e5f2d52b802189ee658865ca93d83a8f  dd7ddc04e1b6c2c614352b383efe2d36
2  c777355d18b72b67abbeef9df44fd0fd  5b51032eddd242adc84c38acab88f23d
3  7634da152a4610f1595efa32f14722fc  9d7a1d34a5052409006425275ba1c2b4
4  ac6c3623068f30de03045865e4e10089  df560393f3a51e74553ab94004ba5c87

   shipping_limit_date   price  freight_value
0  2017-09-19 09:45:35   58.90          13.29
1  2017-05-03 11:05:13  239.90          19.93
2  2018-01-18 14:48:30  199.00          17.87
3  2018-08-15 10:10:18   12.99          12.79
4  2017-02-13 13:57:51  199.90          18.14
```

b) Merge and clean the data

Join orders and order_items on the order_id column to combine customer information with sales data.

```
# Merge orders with order_items
rfm_data = pd.merge(order_items, orders, on="order_id")

# Ensure timestamps are in datetime format
rfm_data['order_purchase_timestamp'] = pd.to_datetime(rfm_data['order_purchase_timestamp'])

# Drop rows with missing values
rfm_data = rfm_data.dropna(subset=['order_purchase_timestamp', 'price', 'customer_id'])

# Display the merged dataset
print(rfm_data.head())
```

Output:-



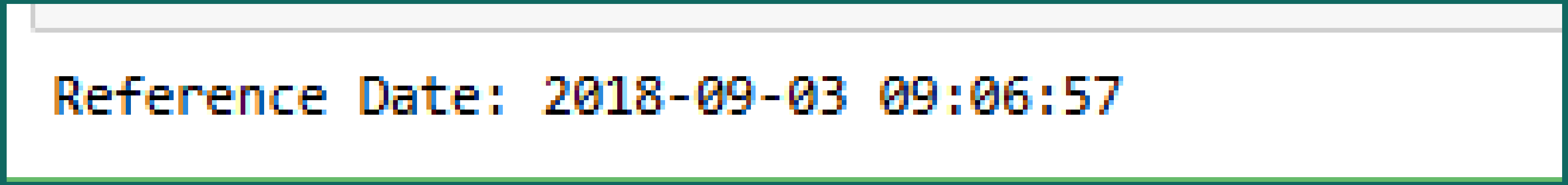
```
                           order_id  order_item_id  \
0  00010242fe8c5a6d1ba2dd792cb16214              1
1  00018f77f2f0320c557190d7a144bdd3              1
2  000229ec398224ef6ca0657da4fc703e              1
3  00024acbcdf0a6daa1e931b038114c75              1
4  00042b26cf59d7ce69dfabb4e55b4fd9              1

                         product_id                         seller_id  \
0  4244733e06e7ecb4970a6e2683c13e61  48436dade18ac8b2bce089ec2a041202
1  e5f2d52b802189ee658865ca93d83a8f  dd7ddc04e1b6c2c614352b383efe2d36
2  c777355d18b72b67abbeef9df44fd0fd  5b51032eddd242adc84c38acab88f23d
3  7634da152a4610f1595efa32f14722fc  9d7a1d34a5052409006425275ba1c2b4
4  ac6c3623068f30de03045865e4e10089  df560393f3a51e74553ab94004ba5c87

   shipping_limit_date   price  freight_value  \
0  2017-09-19 09:45:35   58.90          13.29
1  2017-05-03 11:05:13  239.90          19.93
2  2018-01-18 14:48:30  199.00          17.87
3  2018-08-15 10:10:18   12.99          12.79
4  2017-02-13 13:57:51  199.90          18.14

                        customer_id order_status order_purchase_timestamp  \
0  3ce436f183e68e07877b285a838db11a    delivered      2017-09-13 08:59:02
1  f6dd3ec061db4e3987629fe6b26e5cce    delivered      2017-04-26 10:53:06
2  6489ae5e4333f3693df5ad4372dab6d3    delivered      2018-01-14 14:33:31
3  d4eb9395c8c0431ee92fce09860c5a06    delivered      2018-08-08 10:00:35
4  58dbd0b2d70206bf40e62cd34e84d795    delivered      2017-02-04 13:57:51

     order_approved_at order_delivered_carrier_date  \
0  2017-09-13 09:45:35          2017-09-19 18:34:16
1  2017-04-26 11:05:13          2017-05-04 14:35:00
2  2018-01-14 14:48:30          2018-01-16 12:36:48
3  2018-08-08 10:10:18          2018-08-10 13:28:00
4  2017-02-04 14:10:13          2017-02-16 09:46:09

  order_delivered_customer_date order_estimated_delivery_date
0           2017-09-20 23:43:48           2017-09-29 00:00:00
1           2017-05-12 16:04:24           2017-05-15 00:00:00
2           2018-01-22 13:19:16           2018-02-05 00:00:00
3           2018-08-14 13:32:39           2018-08-20 00:00:00
4           2017-03-01 16:42:31           2017-03-17 00:00:00
```

Step 3: Compute RFM Metrics

a) Define the reference date Choose a reference date (e.g., the most recent date in the dataset).

```
# Reference date for recency calculation
reference_date = rfm_data['order_purchase_timestamp'].max()
print(f"Reference Date: {reference_date}")
```

Output:-



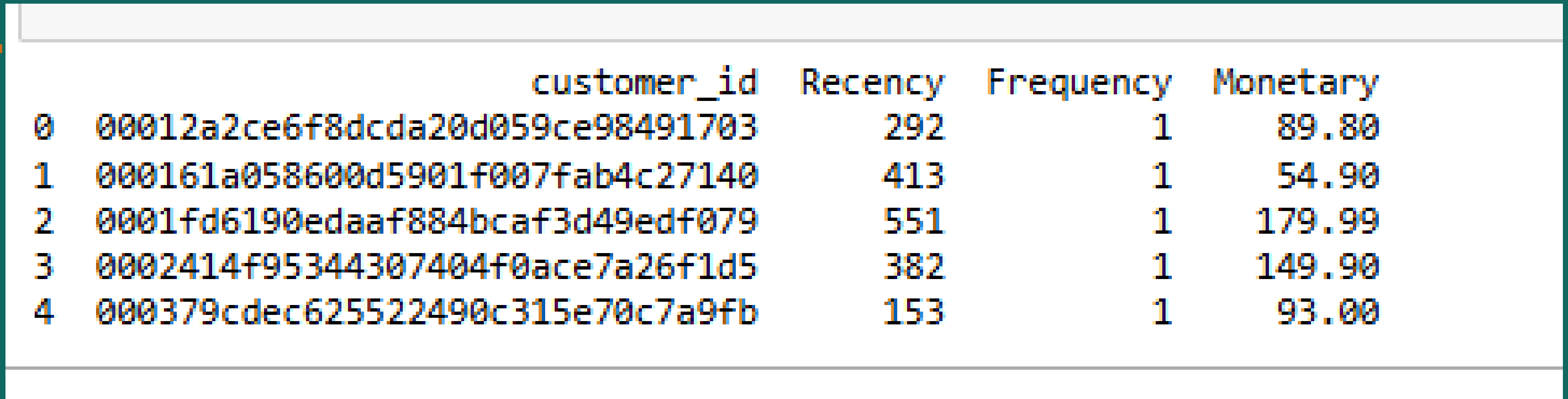
```
Reference Date: 2018-09-03 09:06:57
```

b) Calculate RFM metrics

Group the data by customer_id to calculate Recency, Frequency, and Monetary values.

```
# Group by customer_id
rfm_metrics = rfm_data.groupby('customer_id').agg({
    'order_purchase_timestamp': lambda x: (reference_date - x.max()).days,  # Recency
    'order_id': 'count',  # Frequency
    'price': 'sum'  # Monetary
}).reset_index()

# Rename the columns
rfm_metrics.rename(columns={
    'order_purchase_timestamp': 'Recency',
    'order_id': 'Frequency',
    'price': 'Monetary'
}, inplace=True)

# Display the RFM metrics
print(rfm_metrics.head())
```

Output:-



```
                        customer_id  Recency  Frequency  Monetary
0  00012a2ce6f8dcda20d059ce98491703      292          1     89.80
1  000161a058600d5901f007fab4c27140      413          1     54.90
2  0001fd6190edaaf884bcaf3d49edf079      551          1    179.99
3  0002414f95344307404f0ace7a26f1d5      382          1    149.90
4  000379cdec625522490c315e70c7a9fb      153          1     93.00
```

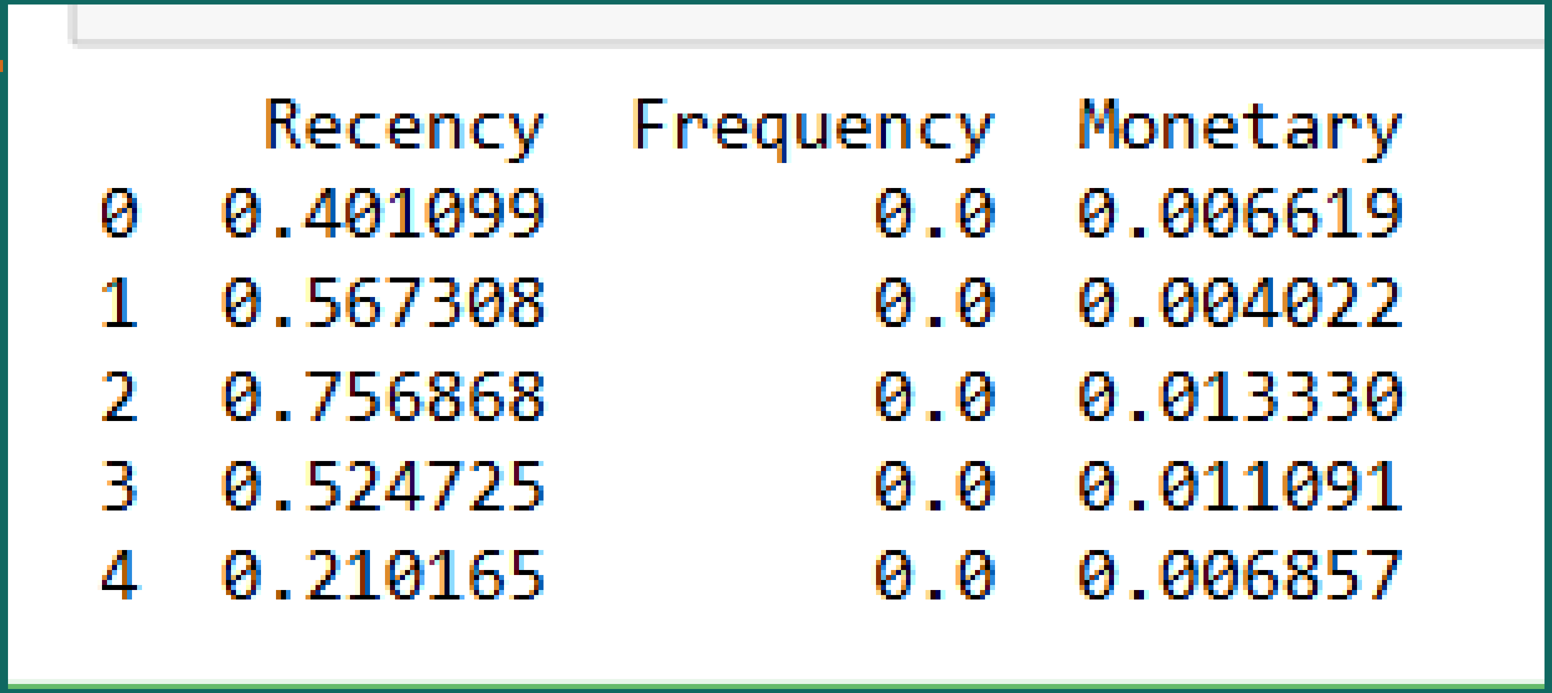
Step 4: Perform Clustering

a) Normalize the data

Since RFM values have different scales, normalize them for clustering.

```
from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler
scaler = MinMaxScaler()

# Normalize the RFM metrics
rfm_normalized = scaler.fit_transform(rfm_metrics[['Recency', 'Frequency', 'Monetary']])

# Convert back to DataFrame
rfm_normalized = pd.DataFrame(rfm_normalized, columns=['Recency', 'Frequency', 'Monetary'])
print(rfm_normalized.head())
```

Output:-



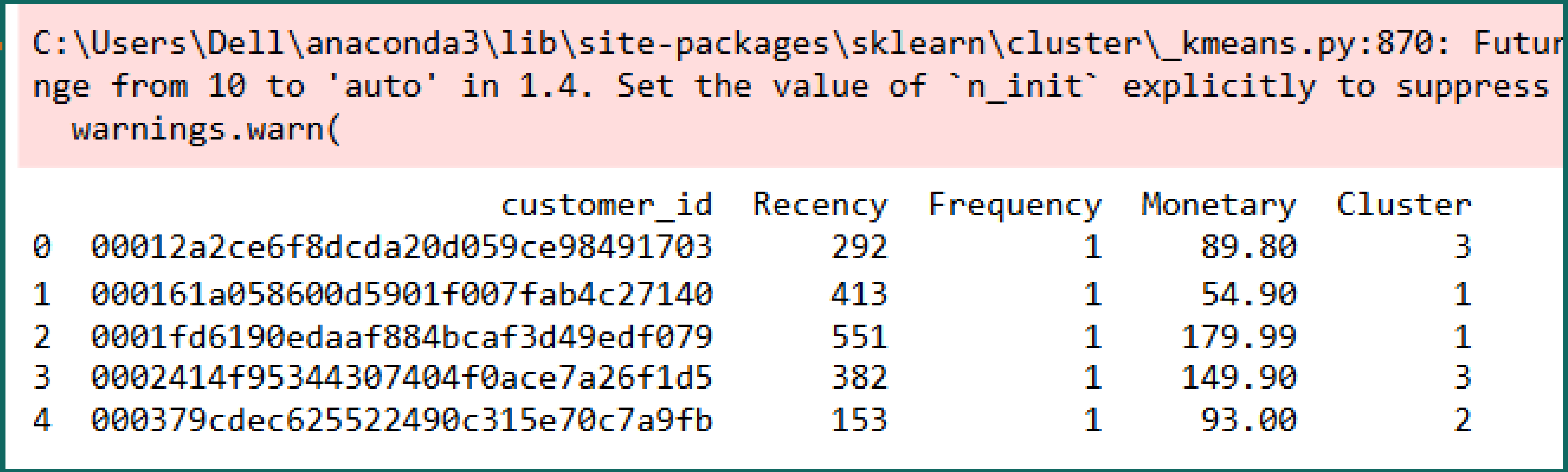
```
    Recency  Frequency  Monetary
0  0.401099        0.0  0.006619
1  0.567308        0.0  0.004022
2  0.756868        0.0  0.013330
3  0.524725        0.0  0.011091
4  0.210165        0.0  0.006857
```

b) Switch to K-Means Clustering

K-Means clustering is computationally efficient and can handle large datasets without requiring a pairwise distance matrix.

```
from sklearn.cluster import KMeans
import numpy as np

# Choose the number of clusters (e.g., 4 clusters)
n_clusters = 4

# Perform K-Means clustering
kmeans = KMeans(n_clusters=n_clusters, random_state=42)
rfm_metrics['Cluster'] = kmeans.fit_predict(rfm_normalized)

# Display the cluster assignments
print(rfm_metrics[['customer_id', 'Recency', 'Frequency', 'Monetary', 'Cluster']].head())
```

Output:-



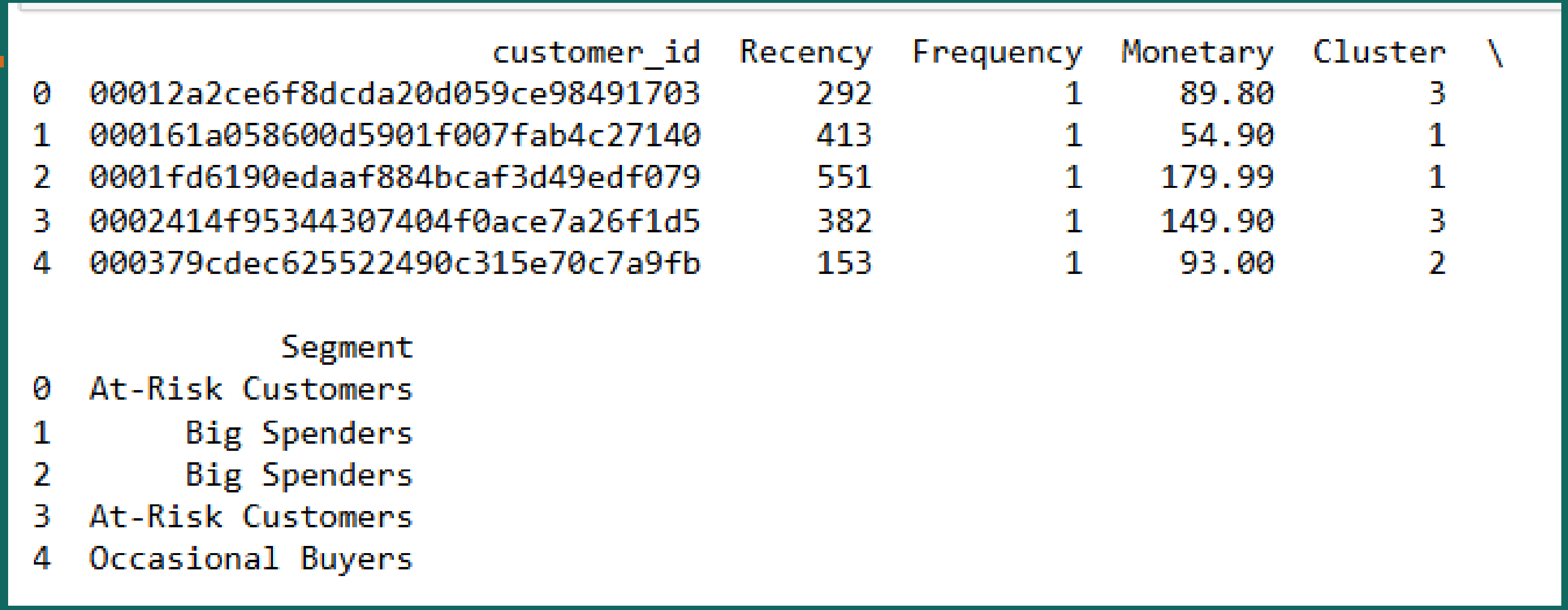
```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: Futur
nge from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
  warnings.warn(

                        customer_id  Recency  Frequency  Monetary  Cluster
0  00012a2ce6f8dcda20d059ce98491703      292          1     89.80        3
1  000161a058600d5901f007fab4c27140      413          1     54.90        1
2  0001fd6190edaaf884bcaf3d49edf079      551          1    179.99        1
3  0002414f95344307404f0ace7a26f1d5      382          1    149.90        3
4  000379cdec625522490c315e70c7a9fb      153          1     93.00        2
```

Step 5: Label Customer Segments

Label the clusters based on RFM values (e.g., Loyal Customers, Discount Seekers).

```
# Define segments based on cluster analysis
def label_segment(row):
    if row['Cluster'] == 0:
        return "Loyal Customers"
    elif row['Cluster'] == 1:
        return "Big Spenders"
    elif row['Cluster'] == 2:
        return "Occasional Buyers"
    elif row['Cluster'] == 3:
        return "At-Risk Customers"
    else:
        return "Other"

# Apply labels to the clusters
rfm_metrics['Segment'] = rfm_metrics.apply(label_segment, axis=1)

# Display customer segments
print(rfm_metrics[['customer_id', 'Recency', 'Frequency', 'Monetary', 'Cluster', 'Segment']].head())
```

Output:-



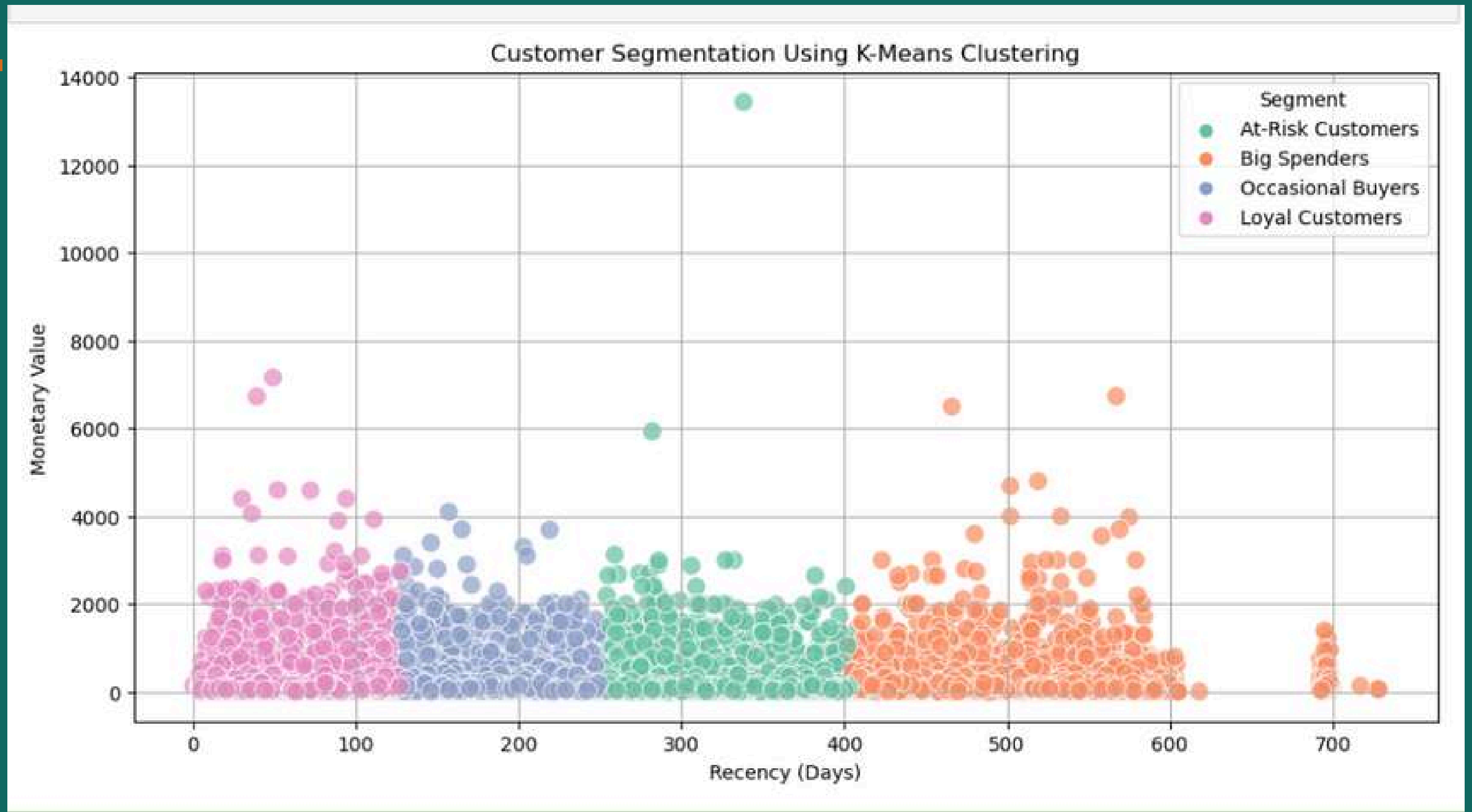
```
                        customer_id  Recency  Frequency  Monetary  Cluster  \
0  00012a2ce6f8dcda20d059ce98491703      292          1     89.80        3
1  000161a058600d5901f007fab4c27140      413          1     54.90        1
2  0001fd6190edaaf884bcaf3d49edf079      551          1    179.99        1
3  0002414f95344307404f0ace7a26f1d5      382          1    149.90        3
4  000379cdec625522490c315e70c7a9fb      153          1     93.00        2

             Segment
0  At-Risk Customers
1       Big Spenders
2       Big Spenders
3  At-Risk Customers
4  Occasional Buyers
```

Step 6: Visualize Segments

Visualize customer segments using scatter plots.

```
import seaborn as sns
import matplotlib.pyplot as plt

# Plot the clusters
plt.figure(figsize=(12, 6))
sns.scatterplot(
    x='Recency', y='Monetary', hue='Segment', data=rfm_metrics,
    palette='Set2', s=100, alpha=0.7
)
plt.title("Customer Segmentation Using K-Means Clustering")
plt.xlabel("Recency (Days)")
plt.ylabel("Monetary Value")
plt.legend(title="Segment")
plt.grid(True)
plt.show()
```

Output:-

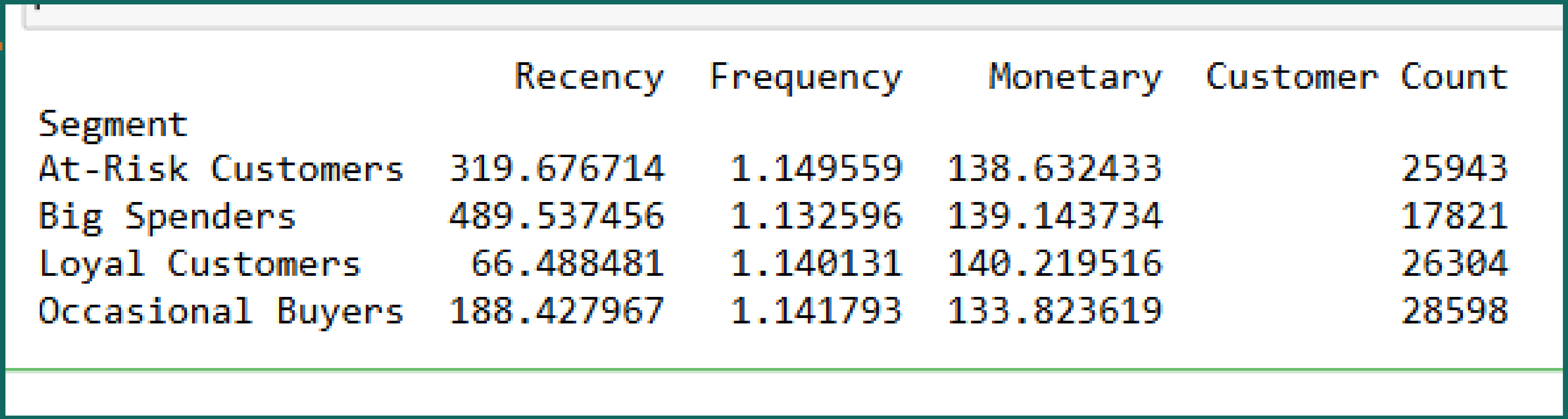


Step 7: Evaluate the Clustering

Analyze the distribution of customers across segments and their key metrics.

```
# Analyze cluster characteristics
cluster_summary = rfm_metrics.groupby('Segment').agg({
    'Recency': 'mean',
    'Frequency': 'mean',
    'Monetary': 'mean',
    'customer_id': 'count'
}).rename(columns={'customer_id': 'Customer Count'})

# Display the cluster summary
print(cluster_summary)
```

Output:-



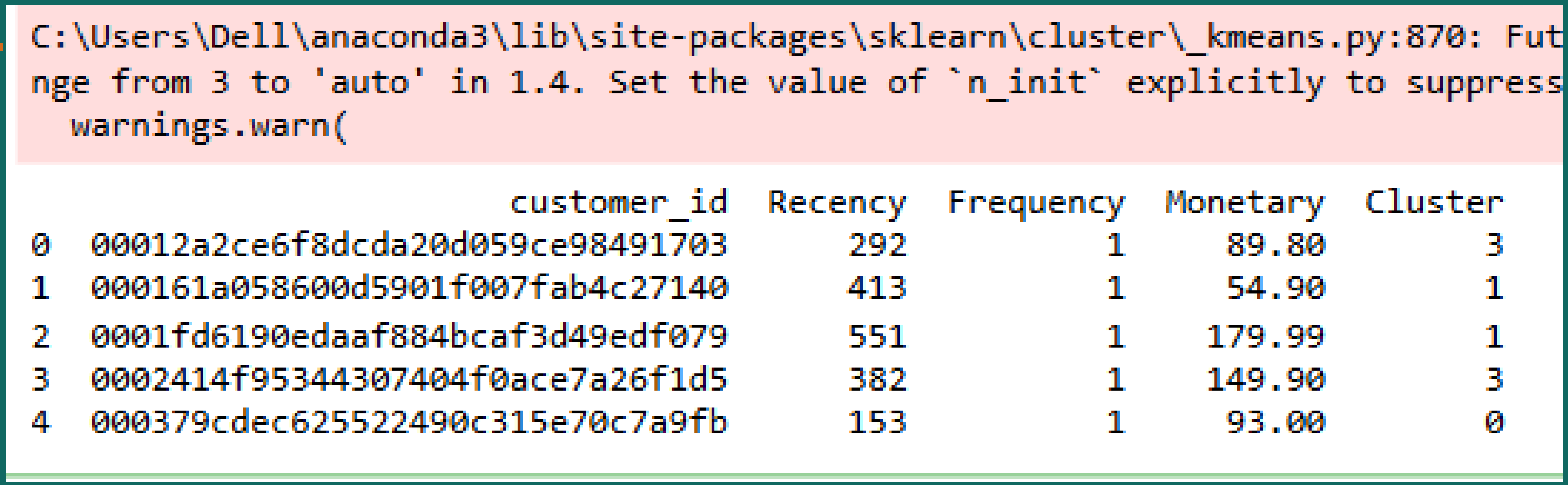
```
                      Recency  Frequency    Monetary  Customer Count
Segment
At-Risk Customers  319.676714   1.149559  138.632433           25943
Big Spenders       489.537456   1.132596  139.143734           17821
Loyal Customers     66.488481   1.140131  140.219516           26304
Occasional Buyers  188.427967   1.141793  133.823619           28598
```

Optional step: MiniBatch K-Means for Larger Datasets

If the dataset is too large even for K-Means, use MiniBatch K-Means, which is a scalable variation of K-Means

```
from sklearn.cluster import MiniBatchKMeans

# Perform MiniBatch K-Means clustering
mini_kmeans = MiniBatchKMeans(n_clusters=n_clusters, random_state=42, batch_size=1000)
rfm_metrics['Cluster'] = mini_kmeans.fit_predict(rfm_normalized)

# Display the updated clusters
print(rfm_metrics[['customer_id', 'Recency', 'Frequency', 'Monetary', 'Cluster']].head())
```

Output:-



```
C:\Users\Dell\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870: Fut
nge from 3 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress
  warnings.warn(

                        customer_id  Recency  Frequency  Monetary  Cluster
0  00012a2ce6f8dcda20d059ce98491703      292          1     89.80        3
1  000161a058600d5901f007fab4c27140      413          1     54.90        1
2  0001fd6190edaaf884bcaf3d49edf079      551          1    179.99        1
3  0002414f95344307404f0ace7a26f1d5      382          1    149.90        3
4  000379cdec625522490c315e70c7a9fb      153          1     93.00        0
```

Q3. Predicting Product Returns:

Build a logistic regression model to predict the likelihood of a product being returned based on features like price, category, and delivery time.

Step 1: Define the Objective

Use the historical data from the datasets (orders.csv, order_items.csv, and products.csv) to predict product returns. Build a logistic regression model to classify whether a product is returned (1) or not (0).

Step 2: Prepare the Data

a) Load the necessary tables Load the orders.csv, order_items.csv, and products.csv tables.

```
import pandas as pd

# Load datasets
orders = pd.read_csv("E:\BHU\Data_Analyst\Ecommerce\orders.csv")
order_items = pd.read_csv("E:\BHU\Data_Analyst\Ecommerce\order_items.csv")
products = pd.read_csv("E:\BHU\Data_Analyst\Ecommerce\products.csv")

# Display the first few rows
print(orders.head())
print(order_items.head())
print(products.head())
```

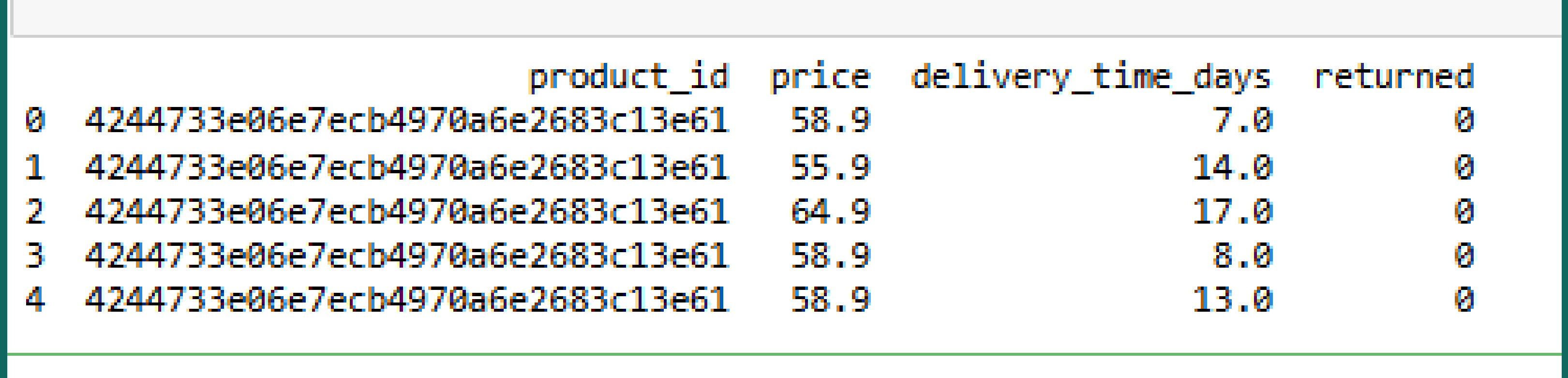
b) Simulate the "product returned" label

Since the dataset does not have explicit return data, we will simulate the return labels:
Assume that orders with a long delivery time and low product price are more likely to be returned. Create a binary column returned (1 for returned, 0 for not returned).

1.Calculate delivery time = order_delivered_customer_date - order_approved_at.
2.Assign returns based on thresholds for delivery time and product price.

```
import numpy as np
import pandas as pd

# Ensure dates are in datetime format, handling errors
orders['order_approved_at'] = pd.to_datetime(orders['order_approved_at'], errors='coerce')
orders['order_delivered_customer_date'] = pd.to_datetime(orders['order_delivered_customer_date'], errors='coerce')

# Calculate delivery time in days (exclude rows where delivery date is NaT)
orders['delivery_time_days'] = (orders['order_delivered_customer_date'] - orders['order_approved_at']).dt.days

# Merge order_items with orders and products
merged_data = pd.merge(order_items, orders, on='order_id', how='inner')
merged_data = pd.merge(merged_data, products, on='product_id', how='inner')

# Drop rows with missing delivery time or price
merged_data = merged_data.dropna(subset=['delivery_time_days', 'price'])

# Simulate returns (binary target variable)
# Assume products with long delivery time (>10 days) and low price (<20) are more likely to be returned
merged_data['returned'] = np.where(
   (merged_data['delivery_time_days'] > 10) & (merged_data['price'] < 20), 1, 0
)

# Display the prepared dataset
print(merged_data[['product_id', 'price', 'delivery_time_days', 'returned']].head())
```

Output:-



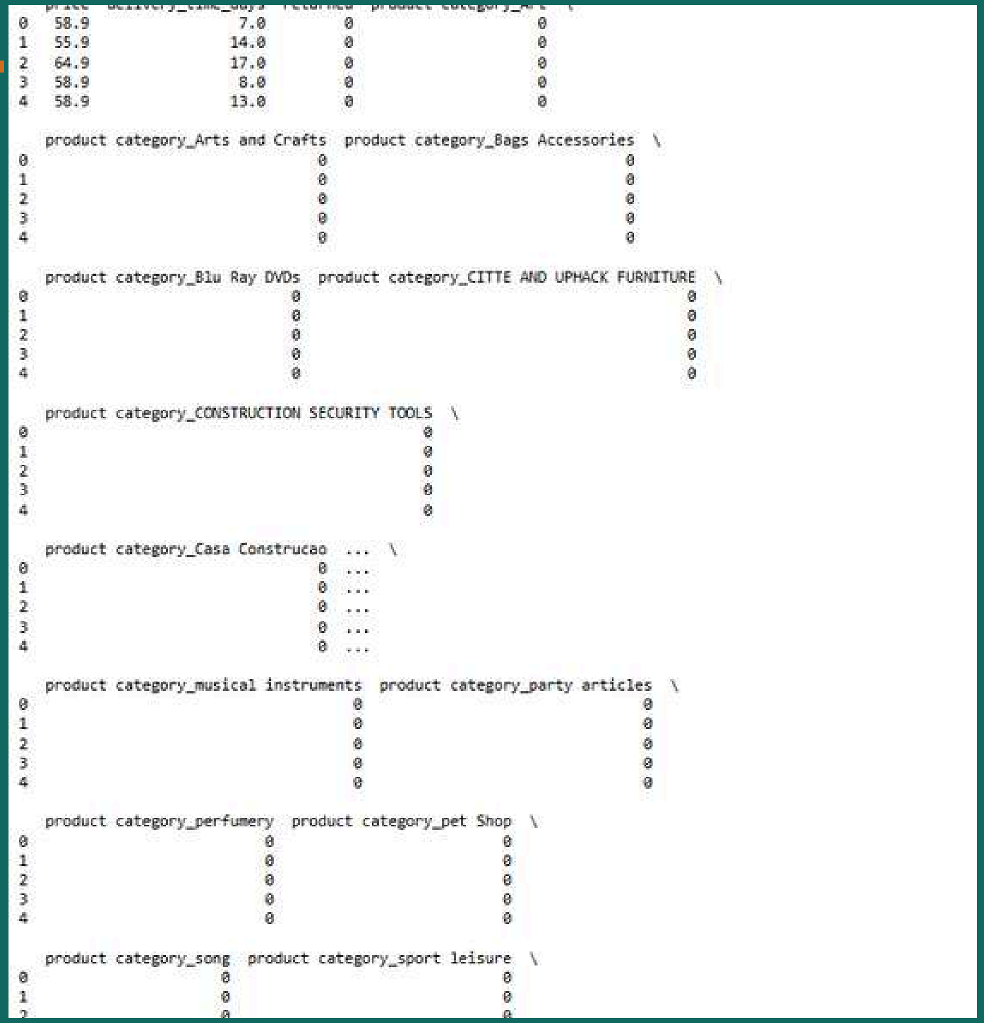
```
                         product_id  price  delivery_time_days  returned
0  4244733e06e7ecb4970a6e2683c13e61   58.9                 7.0         0
1  4244733e06e7ecb4970a6e2683c13e61   55.9                14.0         0
2  4244733e06e7ecb4970a6e2683c13e61   64.9                17.0         0
3  4244733e06e7ecb4970a6e2683c13e61   58.9                 8.0         0
4  4244733e06e7ecb4970a6e2683c13e61   58.9                13.0         0
```

Step 3: Feature Engineering

a) Select Features and Target Variable Identify key features for the model:
Features: price, delivery_time_days, and product category. Target: returned.
Convert categorical data (product category) into numerical format using one-hot encoding.

```
# Select relevant columns
model_data = merged_data[['price', 'delivery_time_days', 'product category', 'returned']]

# One-hot encode the product category
model_data = pd.get_dummies(model_data, columns=['product category'], drop_first=True)

# Display the feature matrix
print(model_data.head())
```

Output:-



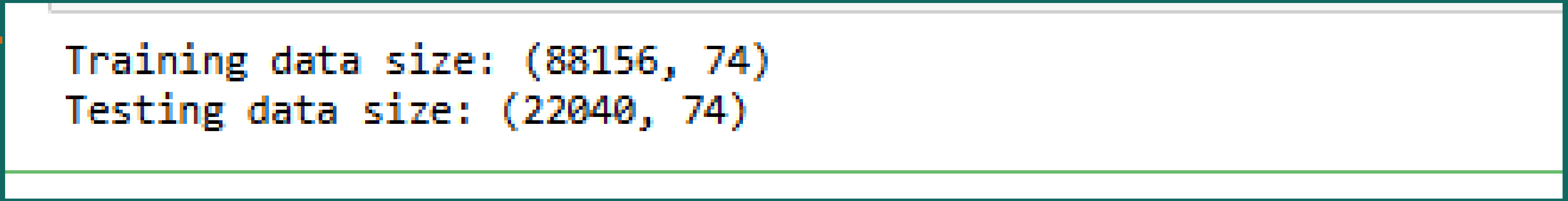
```
   price  delivery_time_days  returned  product category_Art  \
0   58.9                 7.0         0                     0
1   55.9                14.0         0                     0
2   64.9                17.0         0                     0
3   58.9                 8.0         0                     0
4   58.9                13.0         0                     0

   product category_Arts and Crafts  product category_Bags Accessories  \
0                                 0                                  0
1                                 0                                  0
2                                 0                                  0
3                                 0                                  0
4                                 0                                  0

   product category_Blu Ray DVDs  product category_CITTE AND UPHACK FURNITURE  \
0                              0                                            0
1                              0                                            0
2                              0                                            0
3                              0                                            0
4                              0                                            0

   product category_CONSTRUCTION SECURITY TOOLS  \
0                                             0
1                                             0
2                                             0
3                                             0
4                                             0

   product category_Casa Construcao  ...  \
0                                 0  ...
1                                 0  ...
2                                 0  ...
3                                 0  ...
4                                 0  ...

   product category_musical instruments  product category_party articles  \
0                                     0                                0
1                                     0                                0
2                                     0                                0
3                                     0                                0
4                                     0                                0

   product category_perfumery  product category_pet Shop  \
0                           0                          0
1                           0                          0
2                           0                          0
3                           0                          0
4                           0                          0

   product category_song  product category_sport leisure  \
0                      0                               0
1                      0                               0
2                      0                               0
```

b) Split the Data into Training and Testing Sets

Split the dataset into training (80%) and testing (20%) subsets.

```
from sklearn.model_selection import train_test_split

# Define features (X) and target (y)
X = model_data.drop('returned', axis=1)
y = model_data['returned']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print(f"Training data size: {X_train.shape}")
print(f"Testing data size: {X_test.shape}")
```

Output:-



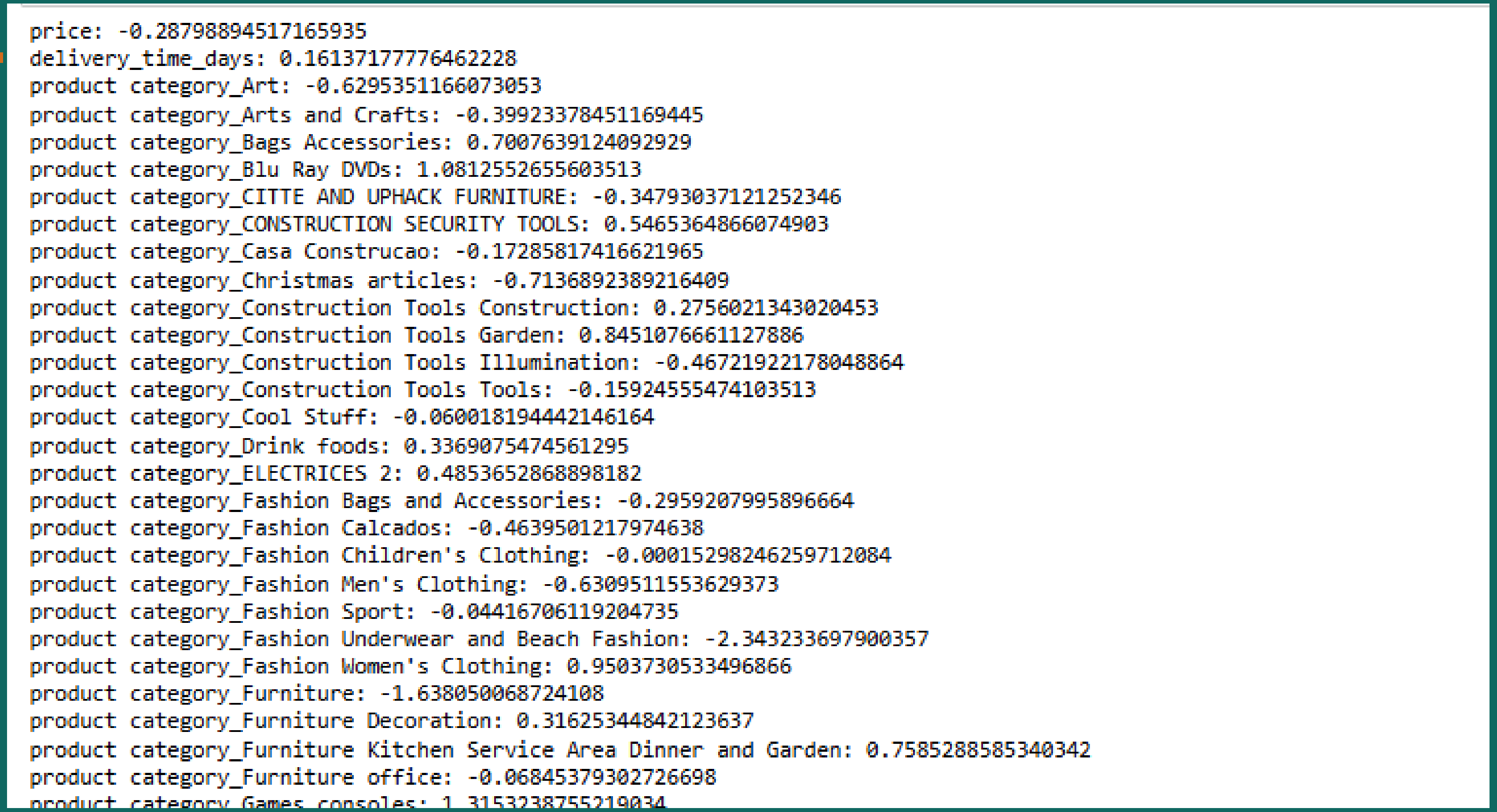
```
Training data size: (88156, 74)
Testing data size: (22040, 74)
```

Step 4: Train the Logistic Regression Model

Train the logistic regression model on the training data.

```
from sklearn.linear_model import LogisticRegression

# Initialize the logistic regression model
logistic_model = LogisticRegression(max_iter=1000, random_state=42)

# Train the model
logistic_model.fit(X_train, y_train)

# Display model coefficients
feature_names = X.columns
coefficients = logistic_model.coef_[0]

for name, coef in zip(feature_names, coefficients):
    print(f"{name}: {coef}")
```

Output:-



```
price: -0.28798894517165935
delivery_time_days: 0.16137177776462228
product category_Art: -0.6295351166073053
product category_Arts and Crafts: -0.39923378451169445
product category_Bags Accessories: 0.7007639124092929
product category_Blu Ray DVDs: 1.0812552655603513
product category_CITTE AND UPHACK FURNITURE: -0.34793037121252346
product category_CONSTRUCTION SECURITY TOOLS: 0.5465364866074903
product category_Casa Construcao: -0.17285817416621965
product category_Christmas articles: -0.7136892389216409
product category_Construction Tools Construction: 0.2756021343020453
product category_Construction Tools Garden: 0.8451076661127886
product category_Construction Tools Illumination: -0.46721922178048864
product category_Construction Tools Tools: -0.15924555474103513
product category_Cool Stuff: -0.060018194442146164
product category_Drink foods: 0.3369075474561295
product category_ELECTRICES 2: 0.4853652868898182
product category_Fashion Bags and Accessories: -0.2959207995896664
product category_Fashion Calcados: -0.46395012179746 38
product category_Fashion Children's Clothing: -0.00015298246259712084
product category_Fashion Men's Clothing: -0.6309511553629373
product category_Fashion Sport: -0.04416706119204735
product category_Fashion Underwear and Beach Fashion: -2.343233697900357
product category_Fashion Women's Clothing: 0.9503730533496866
product category_Furniture: -1.638050068724108
product category_Furniture Decoration: 0.31625344842123637
product category_Furniture Kitchen Service Area Dinner and Garden: 0.7585288585340342
product category_Furniture office: -0.06845379302726698
product category_Games consoles: 1.3153238755219034
```

Step 5: Evaluate the Model

a) Make Predictions Predict on the testing data.

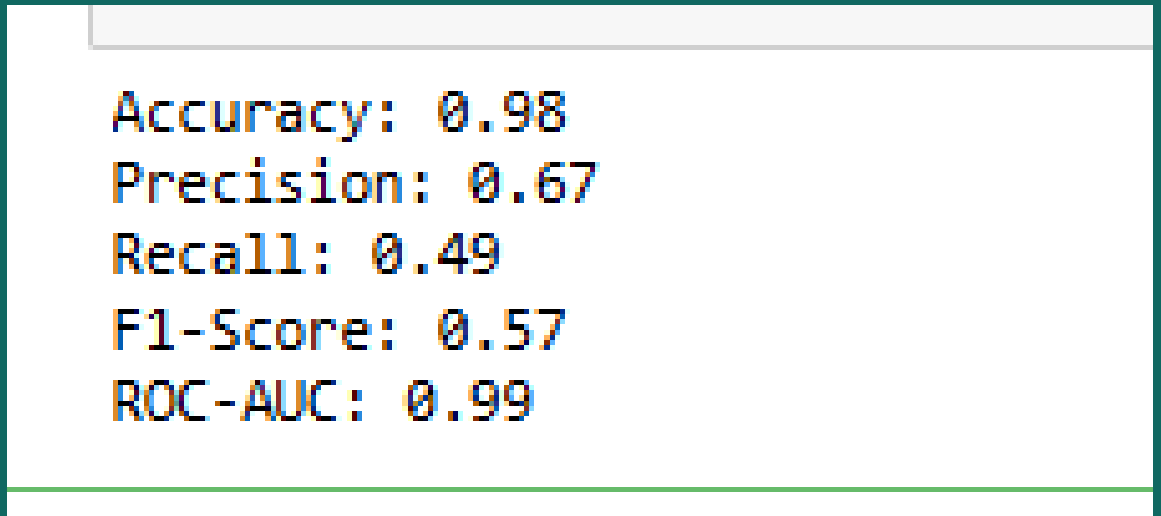
```
# Make predictions
y_pred = logistic_model.predict(X_test)
y_pred_prob = logistic_model.predict_proba(X_test)[:, 1]  # Probability of return
```

b) Evaluate the Model's Performance

Use accuracy, precision, recall, and F1-score to evaluate performance.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_prob)

# Print metrics
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print(f"ROC-AUC: {roc_auc:.2f}")
```

Output:-



```
Accuracy: 0.98
Precision: 0.67
Recall: 0.49
F1-Score: 0.57
ROC-AUC: 0.99
```

Step 6: Interpret Results

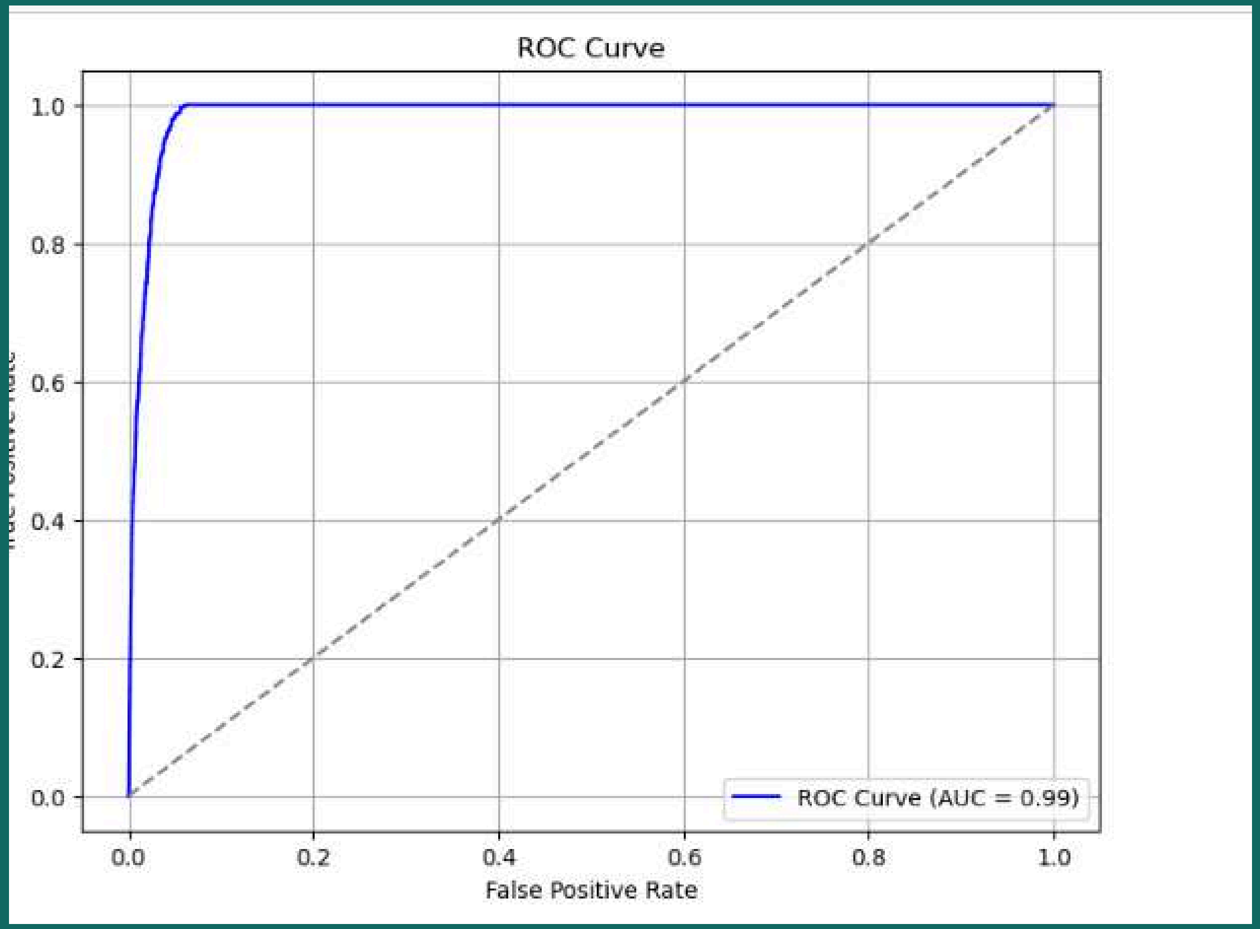
1. Analyze which factors contribute the most to predicting returns:
2. Feature Importance: Use the coefficients from the logistic regression model.
3. Customer Patterns: Examine the most common conditions leading to product returns.

Step 7: Optional - Visualize Results

a) Plot the ROC Curve Visualize the model’s ability to distinguish between returned and non-returned products.

```
from sklearn.metrics import roc_curve

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})", color='blue')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid()
plt.show()
```

Output:-



Q4. Cross-Sell/Upsell Opportunities:

Use correlation analysis and simple rules-based approaches to identify frequently purchased product combinations.
Provide recommendations for cross-selling or bundling products.

Steps

- Merge Necessary Datasets We need to combine order_items with products to associate product details with each order.
- Identify Frequently Purchased Product Combinations Group by order_id and analyze which products are purchased together. Use a co-occurrence matrix or market basket analysis (e.g., Apriori algorithm) to identify frequent combinations.
- Correlation Analysis Compute a co-purchase correlation matrix to quantify relationships between product pairs.
- Provide Recommendations Based on the analysis, suggest product bundles or cross-selling strategies.

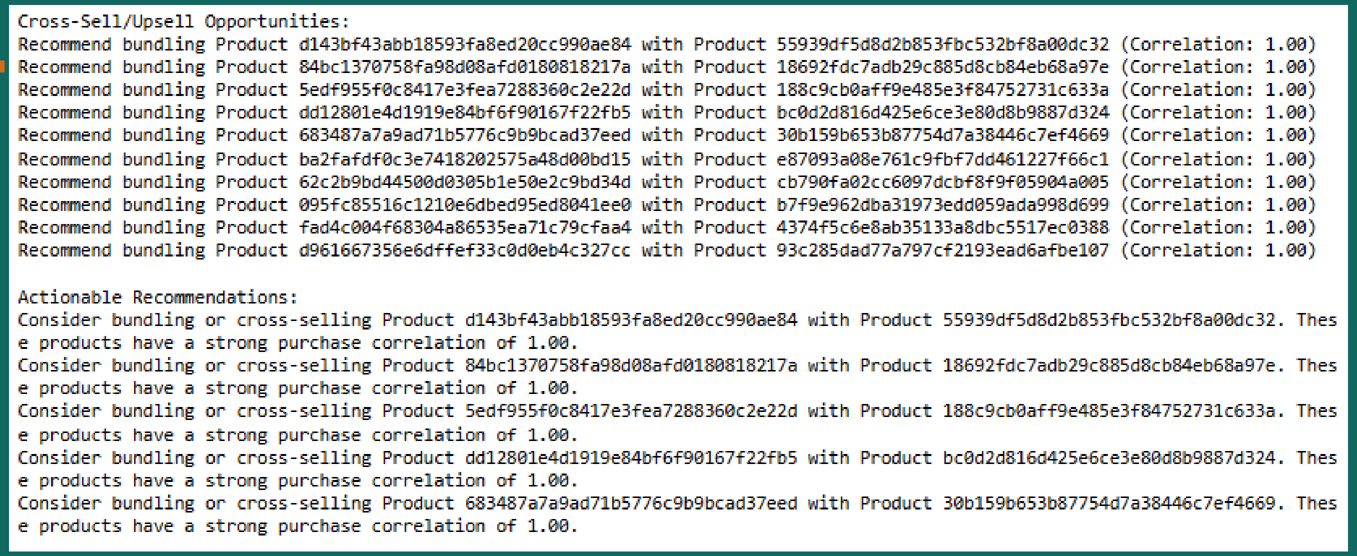
code:-

```
from scipy.sparse import coo_matrix
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd
```

```
# Step 0: Merge order_items with product details
merged_items = pd.merge(order_items, products, on='product_id', how='inner')

# Clean and inspect the merged data
merged_items = merged_items.dropna(subset=['order_id', 'product_id'])  # Remove rows with missing order_id or product_id
merged_items['order_id'] = merged_items['order_id'].astype(str)  # Ensure order_id is a string
merged_items['product_id'] = merged_items['product_id'].astype(str)  # Ensure product_id is a string

# Step 1: Extract unique order_ids and product_ids
order_ids = merged_items['order_id'].unique()
product_ids = merged_items['product_id'].unique()

# Step 2: Create the binary sparse matrix
# Map orders and products to unique integer indices
order_index = {order_id: idx for idx, order_id in enumerate(order_ids)}
product_index = {product_id: idx for idx, product_id in enumerate(product_ids)}

# Map rows and columns for the sparse matrix
rows = merged_items['order_id'].map(order_index)
cols = merged_items['product_id'].map(product_index)
data = np.ones(len(merged_items), dtype=int)

# Build a sparse binary matrix
sparse_matrix = coo_matrix((data, (rows, cols)), shape=(len(order_ids), len(product_ids)))

# Step 3: Compute the co-purchase correlation matrix
# Transpose and convert the sparse matrix to CSR format for efficient row-based operations
dense_matrix = sparse_matrix.T.tocsr()

# Compute pairwise product correlations using cosine similarity
correlation_matrix = cosine_similarity(dense_matrix, dense_matrix)
```

```
# Step 4: Identify strong correlations for recommendations
# Flatten and filter strong correlations
product_ids_array = np.array(product_ids)  # Convert product IDs to a numpy array
correlated_pairs = []  # List to store correlated pairs

threshold = 0.6  # Define the threshold for strong correlations
for i in range(correlation_matrix.shape[0]):
    for j in range(i + 1, correlation_matrix.shape[1]):  # Avoid self-correlations
        if correlation_matrix[i, j] > threshold:
            correlated_pairs.append(
                (product_ids_array[i], product_ids_array[j], correlation_matrix[i, j])
            )

# Sort by correlation in descending order
correlated_pairs = sorted(correlated_pairs, key=lambda x: x[2], reverse=True)

# Step 5: Display cross-sell/upsell recommendations
print("Cross-Sell/Upsell Opportunities:")
for pair in correlated_pairs[:10]:  # Display top 10 recommendations
    print(f"Recommend bundling Product {pair[0]} with Product {pair[1]} (Correlation:
{pair[2]:.2f})")

# Provide actionable recommendations
print("\nActionable Recommendations:")
for pair in correlated_pairs[:5]:  # Focus on the top 5 for actionable insights
    print(
        f"Consider bundling or cross-selling Product {pair[0]} with Product {pair[1]}."
        f" These products have a strong purchase correlation of {pair[2]:.2f}."
    )
```

Output:-



```
Cross-Sell/Upsell Opportunities:
Recommend bundling Product d143bf43abb18593fa8ed20cc990ae84 with Product 55939df5d8d2b853fbc532bf8a00dc32 (Correlation: 1.00)
Recommend bundling Product 84bc1370758fa98d08afd0180818217a with Product 18692fdc7adb29c885d8cb84eb68a97e (Correlation: 1.00)
Recommend bundling Product 5edf955f0c8417e3fea7288360c2e22d with Product 188c9cb0aff9e485e3f84752731c633a (Correlation: 1.00)
Recommend bundling Product dd12801e4d1919e84bf6f90167f22fb5 with Product bc0d2d816d425e6ce3e80d8b9887d324 (Correlation: 1.00)
Recommend bundling Product 683487a7a9ad71b5776c9b9bcad37eed with Product 30b159b653b87754d7a38446c7ef4669 (Correlation: 1.00)
Recommend bundling Product ba2fafdf0c3e7418202575a48d00bd15 with Product e87093a08e761c9fbf7dd461227f66c1 (Correlation: 1.00)
Recommend bundling Product 62c2b9bd44500d0305b1e50e2c9bd34d with Product cb790fa02cc6097dcbf8f9f05904a005 (Correlation: 1.00)
Recommend bundling Product 095fc85516c1210e6dbed95ed8041ee0 with Product b7f9e962dba31973edd059ada998d699 (Correlation: 1.00)
Recommend bundling Product fad4c004f68304a86535ea71c79cfaa4 with Product 4374f5c6e8ab35133a8dbc5517ec0388 (Correlation: 1.00)
Recommend bundling Product d961667356e6dffef33c0d0eb4c327cc with Product 93c285dad77a797cf2193ead6afbe107 (Correlation: 1.00)

Actionable Recommendations:
Consider bundling or cross-selling Product d143bf43abb18593fa8ed20cc990ae84 with Product 55939df5d8d2b853fbc532bf8a00dc32. Thes
e products have a strong purchase correlation of 1.00.
Consider bundling or cross-selling Product 84bc1370758fa98d08afd0180818217a with Product 18692fdc7adb29c885d8cb84eb68a97e. Thes
e products have a strong purchase correlation of 1.00.
Consider bundling or cross-selling Product 5edf955f0c8417e3fea7288360c2e22d with Product 188c9cb0aff9e485e3f84752731c633a. Thes
e products have a strong purchase correlation of 1.00.
Consider bundling or cross-selling Product dd12801e4d1919e84bf6f90167f22fb5 with Product bc0d2d816d425e6ce3e80d8b9887d324. Thes
e products have a strong purchase correlation of 1.00.
Consider bundling or cross-selling Product 683487a7a9ad71b5776c9b9bcad37eed with Product 30b159b653b87754d7a38446c7ef4669. Thes
e products have a strong purchase correlation of 1.00.
```

Resource Page

Target Insight

THANK TOU

For watching this presentation

K K Sahani

☎ +91 72557964940

✉ krishnakumarsahani.bhumca22@gmail.com

🌐 https://www.linkedin.com/in/krishna-kumar-sahani-79867225a/

📍 chandighar,India