# Customer Shopping Behavior Analysis Report

# 1. Introduction

This report explains the complete workflow used to analyze customer shopping behavior using Python (Jupyter Notebook), PostgreSQL (pgAdmin), and Power BI. The goal was to clean data, load it into a database, run SQL analysis, and build an interactive dashboard.

# 2. Dataset Overview

A screenshot of a computer

AI-generated content may be incorrect.

# 3. Data Cleaning in Jupyter Notebook

Steps performed:

- Loaded CSV file into pandas

- Cleaned column names

- Created age\_group column

- Checked missing values

A screenshot of a computer

AI-generated content may be incorrect.

4. Feature Engineering

- Created age\_group column using custom logic

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

5. Loading Data into PostgreSQL

- Created database: customer\_behavior

- Created table: customer

- Loaded cleaned DataFrame into SQL

A screenshot of a computer

AI-generated content may be incorrect.

6. SQL Analysis in pgAdmin

Example Query:

SELECT "Gender", SUM("Purchase Amount (USD)") AS revenue

FROM customer

GROUP BY "Gender";

Included SQL Questions:

- Total revenue by gender

A screenshot of a computer

AI-generated content may be incorrect.

- Discount users vs. average purchase

A screenshot of a computer

AI-generated content may be incorrect.

- Top products by rating

A screenshot of a computer

AI-generated content may be incorrect.

- Shipping type comparison

A screenshot of a computer

AI-generated content may be incorrect.

- Subscriber behavior

A screenshot of a computer

AI-generated content may be incorrect.

- Top discount products

A screenshot of a computer

AI-generated content may be incorrect.

- Customer segmentation

A screenshot of a computer

AI-generated content may be incorrect.

- Top 3 purchased products per category

A screenshot of a computer

AI-generated content may be incorrect.

- Repeat buyers & subscription

A screenshot of a computer

AI-generated content may be incorrect.

- Revenue by age group

A screenshot of a computer

AI-generated content may be incorrect.

# 7. Power BI Dashboard

KPIs Created:

- Average Purchase Amount

- Average Review Rating

- Number of Customers

Visuals:

- Bar charts

- Pie charts

- Segmentation visuals

[Insert Dashboard Screenshot Here]

8. Key Insights

- Customers spend an average of ~59 USD.

- Review rating average is 3.75.

- Total customer count is ~4000.

- Subscribers generally spend more.

- Females generate slightly higher revenue.

A screenshot of a computer

AI-generated content may be incorrect.

# All the Codes:

**Data Preparation in Jupyter Notebook**

1. **Load the CSV file**
2. import pandas as pd
3. df = pd.read\_csv("customer\_shopping\_behavior.csv")
4. df.head()
5. df.info()

I used df.head() and df.info() to check rows, columns, and data types.

1. **Basic cleaning**
   * Checked for missing values.
   * Verified columns such as Customer ID, Age, Gender, Category, Purchase Amount (USD), Review Rating, Season, Payment Method, Subscription Status, Previous Purchases etc.
   * Ensured numeric columns were in the correct format.
2. **Create age\_group column (for later analysis)**
3. labels = ['Young Adult', 'Adult', 'Middle-aged', 'Senior']
4. df['age\_group'] = pd.qcut(df['Age'], q=4, labels=labels)

This splits customers into four age groups.

1. **Connect Jupyter to PostgreSQL and create database/table**
2. from sqlalchemy import create\_engine
3. import psycopg2
4. username = "postgres"
5. password = "your\_password"
6. host = "localhost"
7. port = "5432"
8. database = "customer\_behavior"

I first created the database (if it did not exist) using psycopg2, then created an engine:

engine = create\_engine(

f"postgresql+psycopg2://{username}:{password}@{host}:{port}/{database}"

)

table\_name = "customer"

df.to\_sql(table\_name, engine, if\_exists="replace", index=False)

print(f"Data successfully loaded into table '{table\_name}' in database '{database}'.")

This step moved the cleaned DataFrame from Jupyter into PostgreSQL.

**4. SQL Analysis in PostgreSQL (pgAdmin)**

Using pgAdmin, I connected to the customer\_behavior database and ran several SQL queries on the customer table.

Examples of questions and queries:

1. **Total revenue by gender**
2. SELECT "Gender",
3. SUM("Purchase Amount (USD)") AS revenue
4. FROM customer
5. GROUP BY "Gender";
6. **Top 5 products by average review rating**
7. SELECT "Item Purchased",
8. AVG("Review Rating") AS avg\_rating
9. FROM customer
10. GROUP BY "Item Purchased"
11. ORDER BY avg\_rating DESC
12. LIMIT 5;
13. **Customer segments (New, Returning, Loyal)**
14. WITH customer\_type AS (
15. SELECT "Customer ID",
16. "Previous Purchases",
17. CASE
18. WHEN "Previous Purchases" = 1 THEN 'New'
19. WHEN "Previous Purchases" BETWEEN 2 AND 10 THEN 'Returning'
20. ELSE 'Loyal'
21. END AS customer\_segment
22. FROM customer
23. )
24. SELECT customer\_segment,
25. COUNT(\*) AS customer\_count
26. FROM customer\_type
27. GROUP BY customer\_segment;
28. **Revenue by age group**

Since age\_group was created in Jupyter and stored in PostgreSQL, I used:

SELECT age\_group,

SUM("Purchase Amount (USD)") AS total\_revenue

FROM customer

GROUP BY age\_group

ORDER BY total\_revenue DESC;

These SQL queries helped generate summaries that I later used in Power BI.

**5. Connecting Power BI to PostgreSQL**

In Power BI Desktop:

1. Click **Get data → PostgreSQL database**.
2. Enter:
   * **Server:** localhost:5432
   * **Database:** customer\_behavior
   * Mode: **Import**
3. Provide PostgreSQL username and password (postgres and the password used in pgAdmin).
4. Load the public customer table into Power BI.

**6. DAX Measures in Power BI**

To build KPIs, I created the following measures:

1. **Number of Customers**
2. Number of Customers =
3. COUNT('public customer'[Customer ID])
4. **Average Purchase Amount**
5. Average Purchase Amount =
6. AVERAGE('public customer'[Purchase Amount (USD)])
7. **Average Review Rating**
8. Average Review Rating =

AVERAGE('public customer'[Review Rating])

# 9. Conclusion

This project demonstrates a full analytics pipeline:

Python → Data cleaning

PostgreSQL → Database + SQL reporting

Power BI → Dashboard for insights

# Recommendations

Based on the findings from Python, SQL, and Power BI, the retail company can take the following actions to improve customer engagement, increase sales, and strengthen long-term loyalty:

**1. Focus marketing campaigns on high-value customer groups**

* Adults and Middle-aged groups generate the highest revenue.
* These segments should receive targeted promotions, loyalty rewards, and personalized product recommendations.

**2. Improve retention for low-spending groups**

* Young Adults and Seniors show lower spending.
* The company should test student discounts, seasonal bundles, or senior-friendly offers to increase purchase frequency.

**3. Strengthen discount and review-based strategies**

* Customers who use discounts still spend more than the average purchase amount.
* This means discounts **do not reduce revenue** — they encourage buying behavior.
* High review ratings also connect with higher sales, so promoting top-rated items will boost conversions.

**4. Promote products with high shipping success**

* Standard and Express shipping comparisons show that faster shipping improves customer trust.
* Offer "Free Express Shipping" on high-value orders to increase cart size.

**5. Boost subscription rates through repeat buyers**

* Repeat buyers (5+ past purchases) are more likely to subscribe.
* Offering exclusive perks for subscribers can convert these loyal customers.

**6. Optimize product categories based on what sells the most**

* Identify the top 3 purchased products per category and stock them more aggressively.
* Use Power BI dashboards to monitor category-level performance monthly.

**7. Continue improving customer experience through data**

* The current dashboard can be expanded with:
  + Monthly revenue trends
  + Customer lifetime value
  + Basket analysis (frequently bought together)