Customer Shopping Behaviour Analysis

A Detailed Data Analytics Project

Completed by: **Aditya Nanda**

December 11, 2025

Introduction

This project explores customer shopping behaviour using Python, SQL, and Power BI. The dataset contains 3900 customer transaction records, including demographics, purchase amounts, product attributes, subscription status, shipping preferences, and payment methods. The objective of this project was to perform thorough exploratory data analysis, clean and transform raw data, engineer new features for deeper insights, store the processed data in a SQL database, and finally create a Power BI dashboard for visual interpretation. All work in this project was completed independently.

Python Data Processing and Analysis

1. Loading the Dataset

```
import pandas as pd
df = pd.read_csv('customer_shopping_behavior.csv')
df.head()
```

The dataset was loaded using Pandas, one of the most widely used Python libraries for data manipulation. The `read_csv` function converts the CSV file into a structured DataFrame, enabling efficient analysis. The `df.head()` command displays the first five rows, helping verify that the file was loaded correctly and allowing quick inspection of column names and data types. This initial examination ensures that no formatting issues, missing headers, or unexpected characters exist before deeper analysis begins.

2. Understanding Dataset Structure

```
df.info()
df.describe(include='all')
```

`df.info()` provides a summary of column names, data types, and the number of nonnull entries. This step confirmed that the dataset has 3900 rows and 18 columns, with only the `review_rating` column containing missing values. Understanding data types is critical because incorrect dtypes can break numeric calculations or aggregations.

`df.describe(include='all')` gives statistical summaries for all columns, both numeric and categorical. This highlights patterns such as most common items, popular categories, range of ages, and spending distribution. These insights guide later steps, such as grouping strategies, transformations, and missing value treatment.

3. Missing Value Treatment

```
df.isnull().sum()
df['Review Rating'] = df.groupby('Category')['Review Rating']
                            .transform(lambda x: x.fillna(x.median()))
```

After identifying that only `review_rating` had missing values, a targeted imputation strategy was used. Instead of filling missing values with a global median, the median was computed within each product category. This ensures that items in categories with inherently higher or lower ratings retain realistic values. Using `groupby` and `transform` ensures that missing ratings are replaced without affecting column alignment. This method produces more accurate and nuanced imputations compared to global replacement.

4. Standardising Column Names

```
df.columns = df.columns.str.lower()
df.columns = df.columns.str.replace(' ','_')
df = df.rename(columns={'purchase_amount_(usd)':'purchase_amount'})
```

Column names were standardised to snake_case (lowercase with underscores). This improves readability, reduces typing errors, and ensures compatibility across SQL, Python, and BI tools. Removing spaces prevents syntax issues when referencing columns in queries. Renaming `purchase_amount_(usd)` to `purchase_amount` simplifies usage and improves clarity. Clean column naming is essential in large-scale analytics workflows.

5. Creating Age Groups Using Quantiles

```
labels=['Young Adult','Adult','Middle-aged','Senior']
df['age_group'] = pd.qcut(df['age'], q=4, labels=labels)
```

A new categorical feature, `age_group`, was created by dividing customers into four evenly sized segments based on quantiles. Unlike fixed bins, quantiles ensure that each group contains approximately the same number of customers, leading to balanced comparison across age ranges. This transformation is particularly useful for demographic segmentation and allows easier visualisation and statistical analysis in the dashboard phase.

6. Converting Frequency Labels to Numeric Values

```
frequency_mapping = {
 'Fortnightly':14,'Weekly':7,'Monthly':30,'Quarterly':90,
 'Bi-Weekly':14,'Annually':365,'Every 3 Months':90
}
df['purchase_frequency_days'] = df['frequency_of_purchases']\
                                    .map(frequency_mapping)
```

The `frequency_of_purchases` column originally contained textual labels. Converting them into numeric values enables quantitative analysis, such as calculating purchase cycles or predicting customer value. Mapping repeated labels like "Fortnightly" and "Bi-Weekly" to the same number ensures uniformity. This feature facilitates downstream tasks such as segmentation and modelling.

7. Removing a Redundant Column

```
(df['discount_applied'] == df['promo_code_used']).all()
df = df.drop('promo_code_used',axis=1)
```

A comparison revealed that both columns contained identical values. Redundant columns add no analytical value and may mislead models or inflate dimensionality. Removing `promo_code_used` streamlines the dataset and avoids multicollinearity in future analytics. This check demonstrates good data hygiene practice.

Data Export to SQL

```
from sqlalchemy import create_engine

engine = create_engine("postgresql+psycopg2://postgres:password
                        @localhost:5432/customer_behaviour")
```

```
df.to_sql("customer", engine, if_exists="replace", index=False)
```

The cleaned dataset was exported to a PostgreSQL database using SQLAlchemy. This enables scalable querying, integration with BI tools, and external processing. Replacing any existing table ensures that the latest version of the cleaned dataset is always available for SQL processing.

SQL Analysis and Interpretation

Top Purchased Categories

```
SELECT category, COUNT(*) AS total_orders
FROM customer
GROUP BY category
ORDER BY total_orders DESC;
```

This query identifies the most popular shopping categories based on order count. It helps businesses understand which product segments attract the highest customer interest.

Customer Spending by Gender

```
SELECT gender, SUM(purchase_amount) AS total_spent
FROM customer
GROUP BY gender;
```

This analysis reveals purchasing power differences between genders, useful for targeted marketing campaigns.

Subscription Status Effect

```
SELECT subscription_status, SUM(purchase_amount) AS revenue
FROM customer
GROUP BY subscription_status;
```

This identifies how much revenue comes from subscribed vs. unsubscribed users, helping assess the value of subscription programs.

Age Group Spending

```
SELECT age_group, AVG(purchase_amount) AS avg_spending
FROM customer
```

```
GROUP BY age_group;
```

This determines which age demographic spends the most, an important factor for personalised product recommendations.

Payment Method Preferences

```
SELECT payment_method, COUNT(*) AS usage_count
FROM customer;
```

This shows preferred payment options. Businesses can optimise checkout experiences based on this information.

Power BI Dashboard

The Power BI dashboard visualises patterns extracted from the cleaned dataset. It includes:

- Category-wise sales and revenue contribution
- Age group distribution and purchase trends
- Gender-based purchasing insights
- Payment method usage visualisation
- Subscription and discount impact
- Seasonal purchasing patterns

Each visual is designed to help interpret customer behaviour intuitively and assist decision-making.

Dashboard Screenshot 1

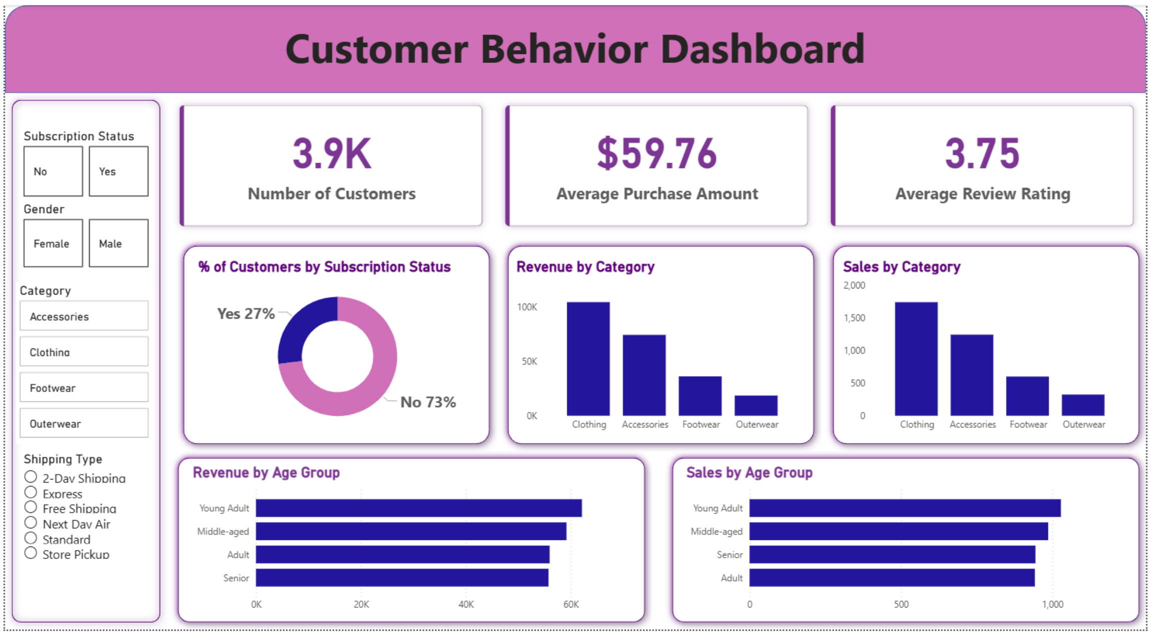

Figure 1: Customer Behaviour Overview

Conclusion

This project provides a complete end-to-end analysis of customer shopping behaviour. The Python workflow covered data inspection, cleaning, and feature engineering with detailed reasoning behind each step. SQL queries extracted deeper insights, while Power BI visualisations presented these patterns clearly. The entire project was completed independently and demonstrates strong understanding of data analysis workflows.