# Customer Shopping Behavior Analysis

**Acknowledgments**

This is my first project I built to perform exploratory data analysis. Amlan Mohanty's YouTube video helped me to understand the process. Check out his tutorial: COMPLETE Data Analytics Portfolio Project in 6 EASY Steps.

**1. Project Overview**

This project analyzes customer shopping behavior using transactional data from 3,900 purchases across various product categories. The goal is to uncover insights into spending patterns, customer segments, product preferences, and subscription behavior to guide strategic business decisions.

**2. Dataset Summary**

- Rows: 3,900
- Columns: 18
- Key Features:
    - Customer demographics (Age, Gender, Location, Subscription Status)
    - Purchase details (Item Purchased, Category, Purchase Amount, Season, Size, Color)
    - Shopping behavior (Discount Applied, Promo Code Used, Previous Purchases, Frequency of Purchases, Review Rating, Shipping Type)
- Missing Data: 37 values in Review Rating column

**3. Exploratory Data Analysis using Python**

- We began with data preparation and cleaning in Python:
- Data Loading: Imported the dataset using pandas.

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

Python

```
<bound method DataFrame.info of       Customer ID  Age  Gender Item Purchased     Category  \
0               1   55    Male        Blouse     Clothing
1               2   19    Male       Sweater     Clothing
2               3   50    Male         Jeans     Clothing
3               4   21    Male       Sandals     Footwear
4               5   45    Male        Blouse     Clothing
...           ...  ...     ...           ...          ...
3895         3896   40  Female        Hoodie     Clothing
3896         3897   52  Female      Backpack  Accessories
3897         3898   46  Female          Belt  Accessories
3898         3899   44  Female         Shoes     Footwear
3899         3900   52  Female       Handbag  Accessories
```

- **Initial Exploration**: Used df.info() to check structure and df.describe() for summary statistics.

Here is a cleaned summary statistics table (based on your provided data):

```python
# To check the summary statastics
df.describe()
# By default it gives summary for Numerical Data only
# To see numerical and categorical both use

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

Python

| | Customer ID | Age | Gender | Item Purchased | Category | Purchase Amount (USD) | Location | Size | Color | Season | Review Rating | Subscription Status | Shipping Type | Discount Applied | Promo Code Used | Pre Purc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 3900.000000 | 3900.000000 | 3900 | 3900 | 3900 | 3900.000000 | 3900 | 3900 | 3900 | 3900 | 3863.000000 | 3900 | 3900 | 3900 | 3900 | 3900.0 |
| unique | NaN | NaN | 2 | 25 | 4 | NaN | 50 | 4 | 25 | 4 | NaN | 2 | 6 | 2 | 2 | |
| top | NaN | NaN | Male | Blouse | Clothing | NaN | Montana | M | Olive | Spring | NaN | No | Free Shipping | No | No | |
| freq | NaN | NaN | 2652 | 171 | 1737 | NaN | 96 | 1755 | 177 | 999 | NaN | 2847 | 675 | 2223 | 2223 | |
| mean | 1950.500000 | 44.068462 | NaN | NaN | NaN | 59.764359 | NaN | NaN | NaN | NaN | 3.750065 | NaN | NaN | NaN | NaN | 25.3 |

**Categorical highlights**:

- Gender: Male (most frequent)
- Item Purchased: Blouse (171 times)
- Category: Clothing (most common)
- Location: Montana (most frequent)
- Season: Spring (most common)
- Subscription Status: No (2847)
- Shipping Type: Free Shipping (most common)
- Discount Applied: No (2223)
- **Missing Data Handling**: Checked for null values (df.isnull().sum()) and imputed missing values in the **Review Rating** column using the median rating of each product category.

```python
# Now Lets Check the missing Values
df.isnull().sum()
```

Python

```
Customer ID               0
Age                       0
Gender                    0
Item Purchased            0
Category                  0
Purchase Amount (USD)     0
Location                  0
Size                      0
Color                     0
Season                    0
Review Rating            37
Subscription Status       0
Shipping Type             0
Discount Applied          0
Promo Code Used           0
Previous Purchases        0
```

```python
# We impute missing values using the median review rating within each product category

#  That way if a clothing item is missing its review rating, it gets replaced with the median review rating of a clothing item, not a footwear item.

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

Python

```python
df.isnull().sum()
```

Python

```
Customer ID               0
Age                       0
Gender                    0
Item Purchased            0
Category                  0
Purchase Amount (USD)     0
Location                  0
Size                      0
Color                     0
Season                    0
Review Rating             0
Subscription Status       0
Shipping Type             0
```

- Column Standardization: Renamed columns to snake_case for better readability and documentation.

```python
df.columns = df.columns.str.lower()
df.columns = df.columns.str.replace(' ','_')
df.columns
```

```
Index(['customer_id', 'age', 'gender', 'item_purchased', 'category',
       'purchase_amount_(usd)', 'location', 'size', 'color', 'season',
       'review_rating', 'subscription_status', 'shipping_type',
       'discount_applied', 'promo_code_used', 'previous_purchases',
       'payment_method', 'frequency_of_purchases'],
      dtype='object')
```

Now let's take a look at the column names again. We can see that most of the column names look clean and consistent. However, the **purchase_amount_usd** column still contains unit information (usd) in the name. While this is not wrong, it can be simplified for better readability and analysis. Since the currency unit is already known or documented elsewhere, we can keep the column name concise. Therefore, we will rename **purchase_amount_(usd) to purchase_amount**. This improves clarity while keeping the naming convention consistent.

```python
df = df.rename(columns={'purchase_amount_(usd)':'purchase_amount'})

df.columns
```

```
Index(['customer_id', 'age', 'gender', 'item_purchased', 'category',
       'purchase_amount', 'location', 'size', 'color', 'season',
       'review_rating', 'subscription_status', 'shipping_type',
       'discount_applied', 'promo_code_used', 'previous_purchases',
       'payment_method', 'frequency_of_purchases'],
      dtype='object')
```

All column names are now standardized using snake_case, making the dataset clean and consistent.

- Feature Engineering:

With a well-prepared dataset, we can now move on to feature engineering. Feature engineering helps us derive new insights by transforming existing variables into more meaningful features. Age as a continuous variable is useful, but for segmentation and analysis, grouping ages can be more insightful. Therefore, we will create a new categorical column called age_group. This column will group customers into four meaningful age segments:

- Young Adult
- Adult
- Middle-aged
- Senior

These age groups will help in:

- Customer segmentation
- Behavioral analysis
- Targeted marketing insights
- Easier visualization in dashboards

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

Python

```python
df[['age','age_group']].head(10)
```

Python

| | age | age_group |
|---|---|---|
| 0 | 55 | Middle-aged |
| 1 | 19 | Young Adult |
| 2 | 50 | Middle-aged |
| 3 | 21 | Young Adult |
| 4 | 45 | Middle-aged |
| 5 | 46 | Middle-aged |
| 6 | 63 | Senior |
| 7 | 27 | Young Adult |
| 8 | 26 | Young Adult |
| 9 | 57 | Middle-aged |

Next, we will create another engineered feature called purchase_frequency_days. The dataset currently contains a column named frequency_of_purchases. This column describes how often a customer shops, but it is stored in textual form (e.g., weekly, monthly, quarterly). While text labels are easy to read, they are not ideal for numerical analysis. Working with text makes it difficult to:

- Compare customers quantitatively
- Calculate averages or trends
- Build metrics or models

To make the data more analysis-friendly, we will convert these textual frequencies into numerical values.

Specifically, we will map each frequency to the approximate number of days between purchases. This results in a new numerical column called purchase_frequency_days. By doing this, we transform qualitative information into quantitative data. This makes it significantly easier to:

- Compare customer purchasing behavior
- Perform aggregations and calculations
- Use the feature in statistical analysis or modelling

To convert purchase frequency from text to numeric values, we first create a mapping dictionary. This dictionary, called frequency_mapping, links each textual frequency to its equivalent number of days between purchases.

For example:

- Weekly → 7 days
- Fortnightly → 14 days
- Monthly → 30 days
- Quarterly → 90 days

Using a dictionary ensures:

- Clear and transparent logic
- Easy modification if business assumptions change

Once the mapping is defined, we use the map() function. The map() function replaces each textual value in the frequency_of_purchases column with the corresponding numeric value from the dictionary. The result is a new numerical feature called purchase_frequency_days. This transformation makes the data quantitative, consistent, and easier to analyze.

```python
# create column purchase_frequency_days
# convert textual frequency in numbers

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)
```

Python

```python
df[['purchase_frequency_days', 'frequency_of_purchases']].head(10)
```

Python

| | purchase_frequency_days | frequency_of_purchases |
|---|---|---|
| 0 | 14.0 | Fortnightly |
| 1 | 14.0 | Fortnightly |
| 2 | 7.0 | Weekly |
| 3 | 7.0 | Weekly |
| 4 | 365.0 | Annually |
| 5 | 7.0 | Weekly |
| 6 | 90.0 | Quarterly |
| 7 | 7.0 | Weekly |
| 8 | 365.0 | Annually |
| 9 | 90.0 | Quarterly |

After converting purchase frequency into numeric days, we now have a much clearer view of customer purchasing behavior. Instead of vague terms like weekly or monthly, we can now analyze the exact number of days between purchases, which is extremely valuable for:

- Customer segmentation
- Loyalty analysis
- Behavioral comparisons

Next, let's examine two related columns:

- discount_applied
- promo_code_used

At first glance, these two columns may appear to represent the same information. It's easy to assume that if a promo code was used, a discount must have been applied. However, in real-world business scenarios:

Discounts can be applied without promo codes

Examples include:

- Automatic seasonal sales
- Member-exclusive discounts
- System-applied price reductions

This raises important data questions:

- Do we actually need both columns?

- Do they always contain the same values?
- Is one column redundant?

Before making any assumptions or dropping columns, we should validate this using data. Therefore, the next step is to analyze and compare these two columns to understand their relationship.

```python
df[['purchase_frequency_days', 'frequency_of_purchases']].head(10)
```

Python

| | purchase_frequency_days | frequency_of_purchases |
|---|---|---|
| 0 | 14.0 | Fortnightly |
| 1 | 14.0 | Fortnightly |
| 2 | 7.0 | Weekly |
| 3 | 7.0 | Weekly |
| 4 | 365.0 | Annually |
| 5 | 7.0 | Weekly |
| 6 | 90.0 | Quarterly |
| 7 | 7.0 | Weekly |
| 8 | 365.0 | Annually |
| 9 | 90.0 | Quarterly |

```python
(df['discount_applied'] == df['promo_code_used']).all()
```

Python

```
* We compare the discount_applied and promo_code_used columns using an equality check to see if they contain the same values.
* By applying .all() to the comparison, we verify whether every row matches across both columns.
* The result returns True, which means both columns carry exactly the same information.
* Since promo_code_used does not add any additional insight, it is redundant.
* Keeping redundant columns can increase complexity without improving analysis.
* Therefore, we remove the promo_code_used column from the dataset using df.drop() to keep the data clean and streamlined.
```

markdown

Data Consistency Check: Verified if discount_applied and promo_code_used were redundant; dropped promo_code_used as it was identical in most cases.

```python
df = df.drop('promo_code_used', axis=1)
```

Python

- Now promo code use will be dropped. Let's check.
- And you can see there is no promo code used. We only have discount applied.

All right.

```python
df.columns
```

Python

**Database Integration:** Connected Python to MySQL and loaded the cleaned DataFrame for SQL analysis (using mysql-connector-python or sqlalchemy).

```
pip install pymysql sqlalchemy
```
Python

```
from sqlalchemy import create_engine
from urllib.parse import quote_plus

username = "root"
password = "Pass@123"
host = "localhost"
port = 3306
database = "mydatabase"

# Safest way
engine = create_engine(
    f"mysql+pymysql://{username}:{quote_plus(password)}@{host}:{port}/{database}"
)

print("Engine created successfully!")
# Now you can use engine with pandas, etc.
# df.to_sql(..., con=engine, ...)
```
Python

```
# Write DataFrame to My SQL
# Before running this you need to create database in mysql

table_name = 'mytable'      # Choose any table name
df.to_sql(table_name, engine, if_exists = "replace", index = False)

print(f"Data Successfully loaded into table '{table_name}' in database '{database}.")
```
Python

```
# Read back sample
pd.read_sql("select * from mytable limit 5;" , engine)
```
Python

Till Here Data is successfully loaded to database. Now in MySQL:

## 4. Data Analysis using SQL (Business Transactions)

```
create database mydatabase;
use mydatabase;
select * from mytable limit 20; -- We can see the dataset fully loaded to mysql and now we can solve the business questions

-- 1. What is the total revenue generated by male vs. female customers?
select gender, sum(purchase_amount) as revenue
from mytable
group by gender;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| gender | revenue |
|---|---|
| Male | 157890 |
| Female | 75191 |

```
-- 2. Which Customer used a discount but still spent more than the average purchase amount?
select customer_id, purchase_amount
from mytable
where discount_applied = 'Yes' and purchase_amount >= (select avg(purchase_amount) from mytable); --
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| customer_id | purchase_amount |
|---|---|
| 2 | 64 |
| 3 | 73 |
| 4 | 90 |
| 7 | 85 |

```sql
-- 3. Which are the top 5 products with the highest average review rating?
select item_purchased, round(avg(review_rating), 2) as "Average Product Rating"
from mytable
group by item_purchased
order by avg(review_rating) desc
limit 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| item_purchased | Average Product Rating |
|---|---|
| Gloves | 3.86 |
| Sandals | 3.84 |
| Boots | 3.82 |
| Hat | 3.8 |
| Skirt | 3.78 |

```sql
-- 4. Compare the average purchase amounts between standard and express shipping
select shipping_type,
round(avg(purchase_amount),2)
from mytable
where shipping_type in('Standard', 'Express')
group by shipping_type;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| shipping_type | round(avg(purchase_amount),2) |
|---|---|
| Express | 60.48 |
| Standard | 58.46 |

```sql
-- 5. Do subscribed customers spend more? Compare average spend and total revenue
-- between subscribers and non subscribers
select subscription_status,
count(customer_id) as total_customers,
round(avg(purchase_amount),2) as avg_spend,
round(sum(purchase_amount),2) as total_revenue
from mytable
group by subscription_status
order by total_revenue, avg_spend desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| subscription_status | total_customers | avg_spend | total_revenue |
|---|---|---|---|
| Yes | 1053 | 59.49 | 62645 |
| No | 2847 | 59.87 | 170436 |

```sql
-- 6. Which 5 products have the highest percentage of purchases with discount applied
select item_purchased,
round(100 * sum(case when discount_applied = 'Yes' then 1 else 0 end)/count(*),2) as discount_rate
from mytable
group by item_purchased
order by discount_rate desc
limit 5;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

| item_purchased | discount_rate |
|---|---|
| Hat | 50.00 |
| Sneakers | 49.66 |
| Coat | 49.07 |
| Sweater | 48.17 |
| Pants | 47.37 |

```sql
/* 7. Segment customers into new, Returning, and Loyal based on their total number of previous purchases,
and show the count of each segment */
with customer_type as (
select customer_id, previous_purchases,
CASE
when previous_purchases = 1 then 'New'
when previous_purchases between 2 and 10 then 'Returning'
else 'Loyal'
end as customer_segment
from mytable
)
select customer_segment, count(*) as "Number of Customers"
from customer_type
group by customer_segment;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| customer_segment | Number of Customers |
|---|---|
| Loyal | 3116 |
| Returning | 701 |
| New | 83 |

```sql
-- 8. What are the top 3 most purchased products within each category?
with item_counts as (
select category,
item_purchased,
count(customer_id) as total_orders,
row_number() over (partition by category order by count(customer_id) desc) as item_rank
from mytable
group by category, item_purchased
)

select item_rank, category, item_purchased, total_orders
from item_counts
where item_rank <= 3;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| item_rank | category | item_purchased | total_orders |
|---|---|---|---|
| 1 | Accessories | Jewelry | 171 |
| 2 | Accessories | Sunglasses | 161 |
| 3 | Accessories | Belt | 161 |
| 1 | Clothing | Blouse | 171 |
| 2 | Clothing | Pants | 171 |

```sql
-- 9. Are customers who are repeat buyers (more than 5 previous purchases) also likely to subscribe?
select subscription_status,
count(customer_id) as repeat_buyers
from mytable
where previous_purchases > 5
group by subscription_status;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| subscription_status | repeat_buyers |
|---|---|
| Yes | 958 |
| No | 2518 |

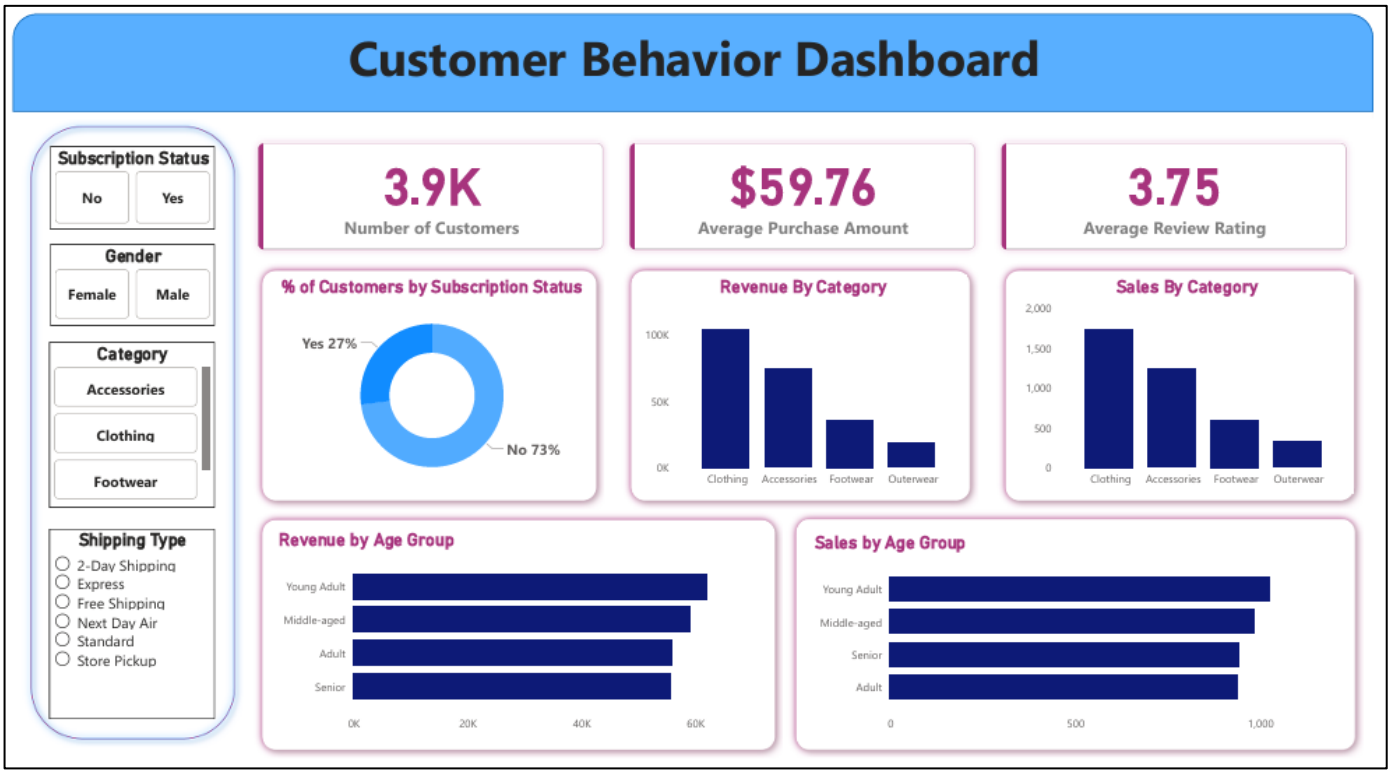
```sql
-- 10. What is the revenue contribution of each age group?
select age_group,
sum(purchase_amount) as total_revenue
from mytable
group by age_group
order by total_revenue desc;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

| age_group | total_revenue |
|---|---|
| Young Adult | 62143 |
| Middle-aged | 59197 |
| Adult | 55978 |
| Senior | 55763 |

**5. Dashboard in Power BI**

Finally, we built an interactive dashboard in **Power BI** to present insights visually.




**Key visuals include**:

- Pie chart: Subscription Status (73% No, 27% Yes)
- Cards: Total Customers (3.9K), Avg. Purchase Amount ($59.76), Avg. Review Rating (3.75)
- Bar charts: Revenue by Age Group, Sales by Category, Revenue by Gender
- Other: Shipping Type distribution, Top products, etc.

**6. Business Recommendations**

- **Boost Subscriptions** – Promote exclusive benefits for subscribers.
- **Customer Loyalty Programs** – Reward repeat buyers to move them into the “Loyal” segment.
- **Review Discount Policy** – Balance sales boosts with margin control.
- **Product Positioning** – Highlight top-rated and best-selling products in campaigns.
- **Targeted Marketing** – Focus efforts on high-revenue age groups and express-shipping users.