```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
pd.set_option("display.max_columns", None)
pd.set_option("display.float_format", lambda x: f"{x:,.2f}")

sns.set(style="whitegrid", context="talk")

plt.rcParams["font.size"] = 12          # base font size
plt.rcParams["axes.titlesize"] = 14     # title font
plt.rcParams["axes.labelsize"] = 12     # x/y label font
plt.rcParams["xtick.labelsize"] = 10    # x tick labels
plt.rcParams["ytick.labelsize"] = 10    # y tick labels
plt.rcParams["legend.fontsize"] = 10    # legend

df = pd.read_csv("/content/drive/MyDrive/Collab/KNN Project/ecommerce_customer_behavior_dataset_v2.csv")
```

```
df.head()
```

	Order_ID	Customer_ID	Date	Age	Gender	City	Product_Category	Unit_Price	Quantity	Discount_Amount	Total_Amount
0	ORD_000001-1	CUST_00001	2023-05-29	40	Male	Ankara	Books	29.18	1	0.00	29.18
1	ORD_000001-2	CUST_00001	2023-10-12	40	Male	Ankara	Home & Garden	644.40	1	138.05	506.35
2	ORD_000001-3	CUST_00001	2023-12-05	40	Male	Ankara	Sports	332.82	5	0.00	1,664.10
3	ORD_000002-1	CUST_00002	2023-05-11	33	Male	Istanbul	Food	69.30	5	71.05	275.45
4	ORD_000002-2	CUST_00002	2023-06-16	33	Male	Istanbul	Beauty	178.15	3	0.00	534.45

Next steps: Generate code with df | New interactive sheet

EDA

```
df.shape
```

```
(17049, 18)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17049 entries, 0 to 17048
Data columns (total 18 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Order_ID                  17049 non-null  object
 1   Customer_ID               17049 non-null  object
 2   Date                      17049 non-null  object
 3   Age                       17049 non-null  int64
 4   Gender                    17049 non-null  object
 5   City                      17049 non-null  object
 6   Product_Category          17049 non-null  object
 7   Unit_Price                17049 non-null  float64
 8   Quantity                  17049 non-null  int64
 9   Discount_Amount           17049 non-null  float64
 10  Total_Amount              17049 non-null  float64
 11  Payment_Method            17049 non-null  object
 12  Device_Type               17049 non-null  object
 13  Session_Duration_Minutes  17049 non-null  int64
 14  Pages_Viewed              17049 non-null  int64
 15  Is_Returning_Customer     17049 non-null  bool
 16  Delivery_Time_Days        17049 non-null  int64
 17  Customer_Rating           17049 non-null  int64
dtypes: bool(1), float64(3), int64(6), object(8)
memory usage: 2.2+ MB
```

```
df["Date"]=pd.to_datetime(df["Date"])
df["Year"] = df["Date"].dt.year
df["Month"] = df["Date"].dt.month
df["Month_Name"] = df["Date"].dt.month_name()
df["Day"] = df["Date"].dt.day
df["Weekday"] = df["Date"].dt.day_name()
```

```
missing = df.isna().sum().sort_values(ascending=False)
missing_pct = (df.isna().mean() * 100).sort_values(ascending=False)
missing_summary = pd.DataFrame({"Missing": missing, "%": missing_pct})
print(missing_summary)
```

```
                          Missing    %
Order_ID                        0 0.00
Customer_ID                     0 0.00
Date                            0 0.00
Age                             0 0.00
Gender                          0 0.00
City                            0 0.00
Product_Category                0 0.00
Unit_Price                      0 0.00
Quantity                        0 0.00
Discount_Amount                 0 0.00
Total_Amount                    0 0.00
Payment_Method                  0 0.00
Device_Type                     0 0.00
Session_Duration_Minutes        0 0.00
Pages_Viewed                    0 0.00
Is_Returning_Customer           0 0.00
Delivery_Time_Days              0 0.00
Customer_Rating                 0 0.00
Year                            0 0.00
Month                           0 0.00
Month_Name                      0 0.00
Day                             0 0.00
Weekday                         0 0.00
```

```
df.isna().sum()
```

	0
Order_ID	0
Customer_ID	0
Date	0
Age	0
Gender	0
City	0
Product_Category	0
Unit_Price	0
Quantity	0
Discount_Amount	0
Total_Amount	0
Payment_Method	0
Device_Type	0
Session_Duration_Minutes	0
Pages_Viewed	0
Is_Returning_Customer	0
Delivery_Time_Days	0
Customer_Rating	0
Year	0
Month	0
Month_Name	0
Day	0
Weekday	0

dtype: int64

```
for col in df:
    print(col)
    print(df[col].unique())
    print("-"*110)
```

```
Is_Returning_Customer
[ True False]
---------------------------------------------------------------------------------------------------------------------
Delivery_Time_Days
[13  6  9  4  5  7 10  2 14  8  3 11 12 15  1 16 19 20 18 17 21 24 25 23
 22]
---------------------------------------------------------------------------------------------------------------------
Customer_Rating
[4 2 5 3 1]
---------------------------------------------------------------------------------------------------------------------
Year
[2023 2024]
---------------------------------------------------------------------------------------------------------------------
Month
[ 5 10 12  6  2  1  3  7  8  9  4 11]
---------------------------------------------------------------------------------------------------------------------
Month_Name
['May' 'October' 'December' 'June' 'February' 'January' 'March' 'July'
 'August' 'September' 'April' 'November']
---------------------------------------------------------------------------------------------------------------------
Day
[29 12  5 11 16 27  3 13 21 24  7 14  9 15 17  4  8 23  2 25 22 28 20 10
  1 19 18 26 31  6 30]
---------------------------------------------------------------------------------------------------------------------
Weekday
['Monday' 'Thursday' 'Tuesday' 'Friday' 'Wednesday' 'Sunday' 'Saturday']
---------------------------------------------------------------------------------------------------------------------
```

```
df.duplicated().sum()
```

```
np.int64(0)
```

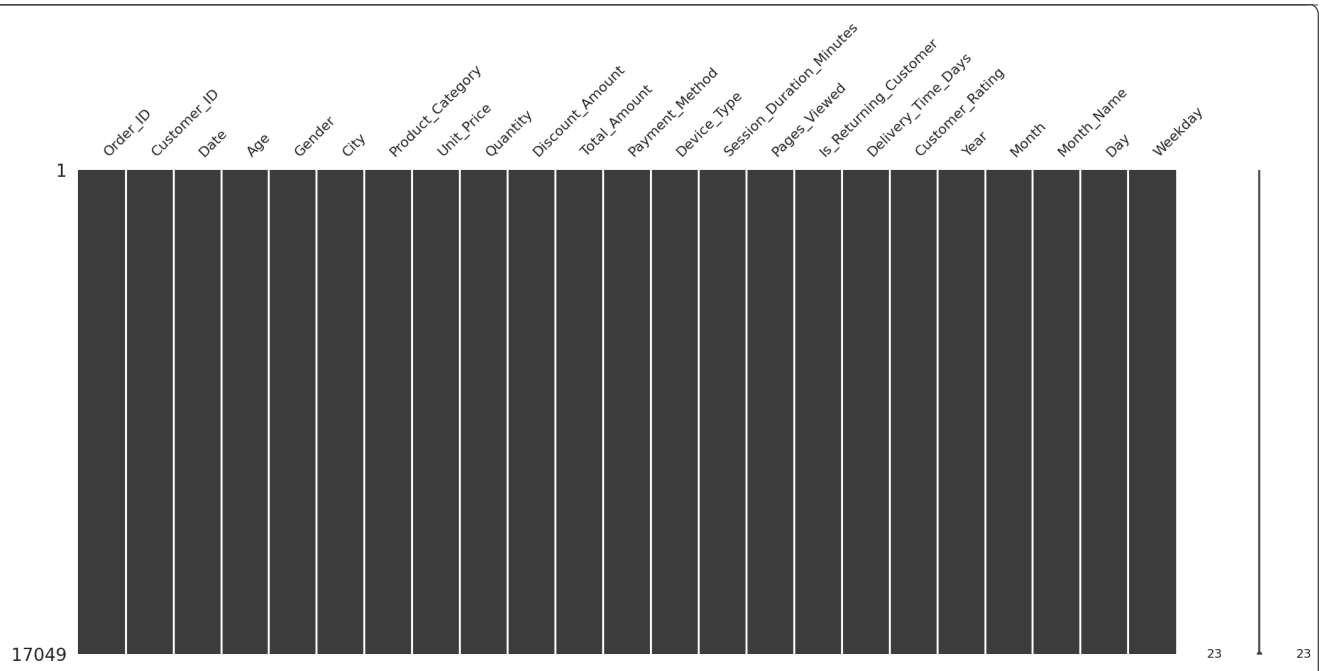
```
df.describe()
```

	Date	Age	Unit_Price	Quantity	Discount_Amount	Total_Amount	Session_Duration_Minutes	Pages_Viewed
count	17049	17,049.00	17,049.00	17,049.00	17,049.00	17,049.00	17,049.00	17,049.00
mean	2023-08-15 01:33:09.723737344	34.95	447.90	3.01	69.79	1,277.44	14.54	9.00
min	2023-01-01 00:00:00	18.00	5.05	1.00	0.00	6.21	4.00	1.00
25%	2023-04-26 00:00:00	26.00	73.26	2.00	0.00	172.97	13.00	7.00
50%	2023-08-16 00:00:00	35.00	174.68	3.00	0.00	455.85	15.00	9.00
75%	2023-12-06 00:00:00	42.00	494.57	4.00	32.71	1,267.75	17.00	11.00
max	2024-03-25 00:00:00	75.00	7,900.01	5.00	6,538.29	37,852.05	26.00	18.00
std	NaN	11.05	722.32	1.42	240.70	2,358.44	2.93	2.26

```
# Negative or zero amounts?
df[df["Total_Amount"] <= 0].head()
```

	Order_ID	Customer_ID	Date	Age	Gender	City	Product_Category	Unit_Price	Quantity	Discount_Amount	Total_Amount	Payment_

```
import missingno as msno
msno.matrix(df)
plt.show()
```



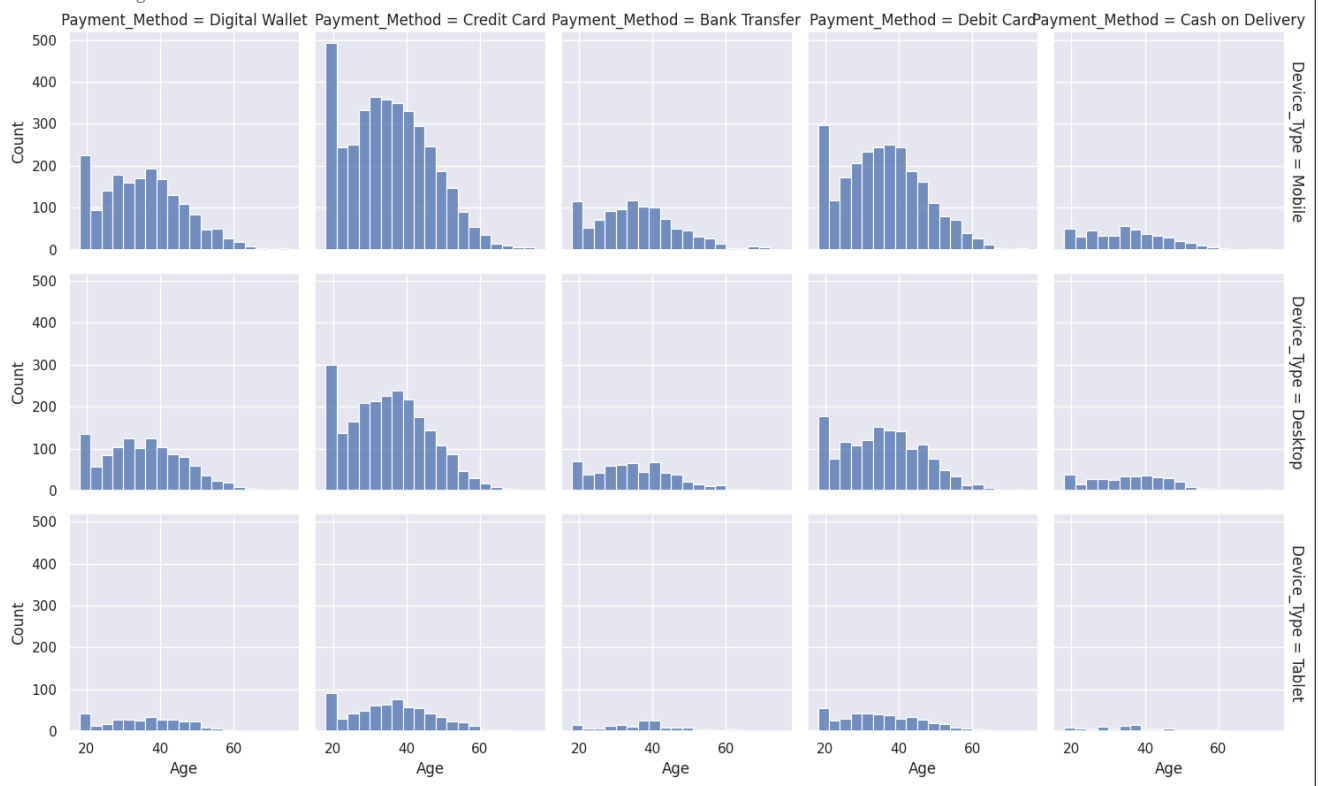
Univariate analysis

```
numeric_cols = ["Age", "Unit_Price", "Quantity", "Discount_Amount",
                "Total_Amount", "Session_Duration_Minutes",
                "Pages_Viewed", "Delivery_Time_Days", "Customer_Rating"]

cat_cols = ["Gender", "City", "Product_Category", "Payment_Method",
            "Device_Type", "Is_Returning_Customer"]

df[numeric_cols].describe()
```

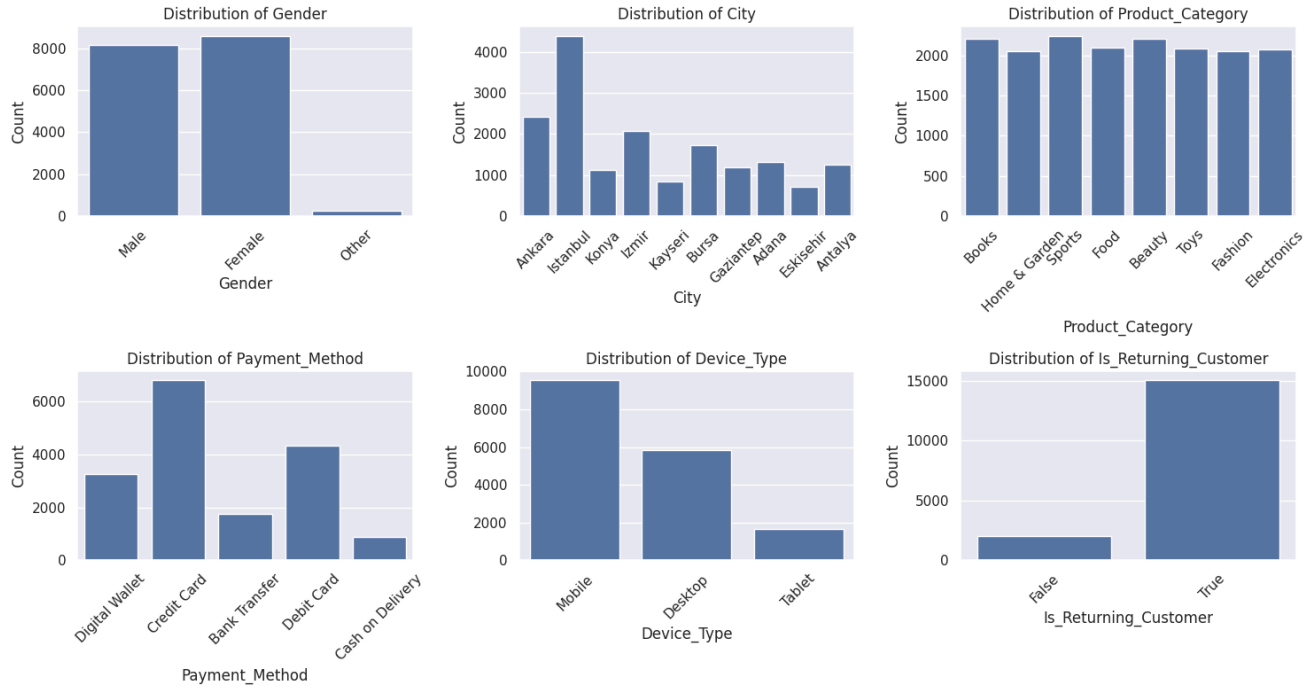
	Age	Unit_Price	Quantity	Discount_Amount	Total_Amount	Session_Duration_Minutes	Pages_Viewed	Delivery_Time_Days
count	17,049.00	17,049.00	17,049.00	17,049.00	17,049.00	17,049.00	17,049.00	17,049.00
mean	34.95	447.90	3.01	69.79	1,277.44	14.54	9.00	6.50
std	11.05	722.32	1.42	240.70	2,358.44	2.93	2.26	3.49
min	18.00	5.05	1.00	0.00	6.21	4.00	1.00	1.00
25%	26.00	73.26	2.00	0.00	172.97	13.00	7.00	4.00
50%	35.00	174.68	3.00	0.00	455.85	15.00	9.00	6.00
75%	42.00	494.57	4.00	32.71	1,267.75	17.00	11.00	8.00
max	75.00	7,900.01	5.00	6,538.29	37,852.05	26.00	18.00	25.00

```
sns.set_theme(style="darkgrid")

sns.displot(
    df, x="Age", col="Payment_Method", row="Device_Type",
    binwidth=3, height=3, facet_kws=dict(margin_titles=True),
)
```

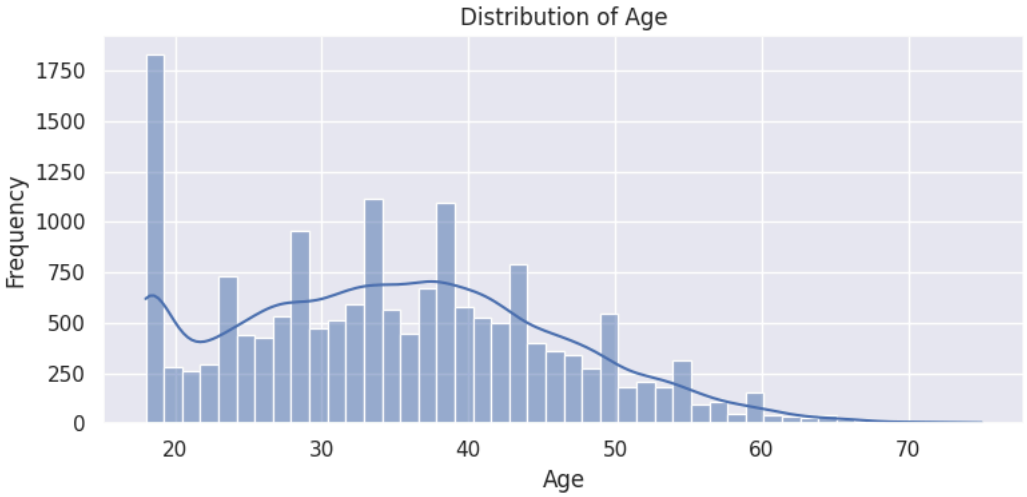
```
<seaborn.axisgrid.FacetGrid at 0x7b66dcba62a0>
```

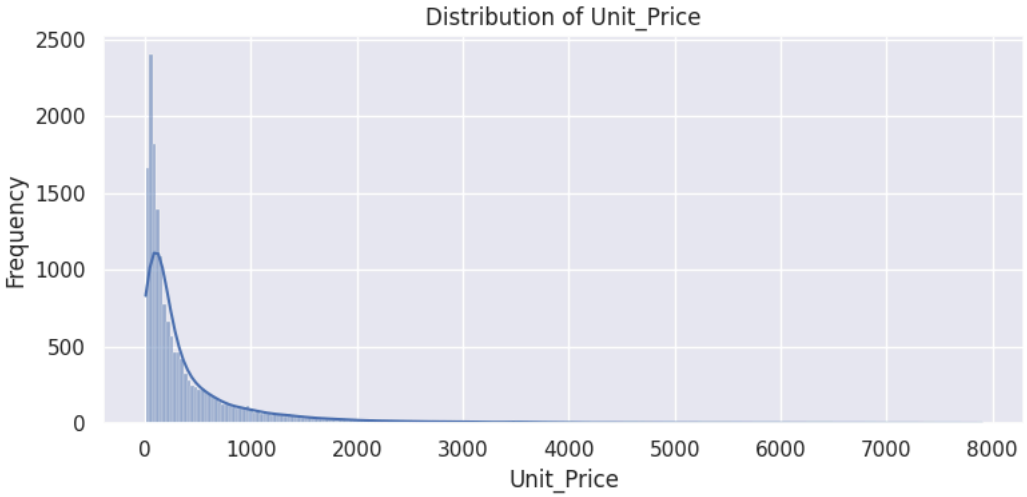


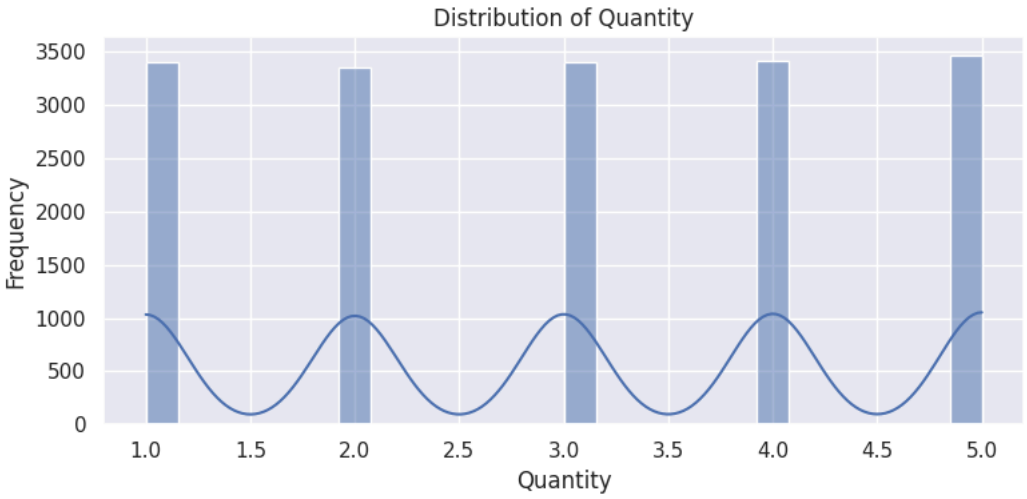

```
import math

# List of categorical columns
cat_cols = ["Gender", "City", "Product_Category", "Payment_Method", "Device_Type", "Is_Returning_Customer"]

n_cols = 3  # how many plots per row
n_rows = math.ceil(len(cat_cols) / n_cols)

fig, axes = plt.subplots(n_rows, n_cols, figsize=(5 * n_cols, 4 * n_rows))
axes = axes.flatten()  # make it easy to iterate

for i, cat in enumerate(cat_cols):
    ax = axes[i]
    sns.countplot(data=df, x=cat, ax=ax)
    ax.set_title(f"Distribution of {cat}")
    ax.set_xlabel(cat)
    ax.set_ylabel("Count")
    ax.tick_params(axis="x", rotation=45)



plt.tight_layout()
plt.show()
```

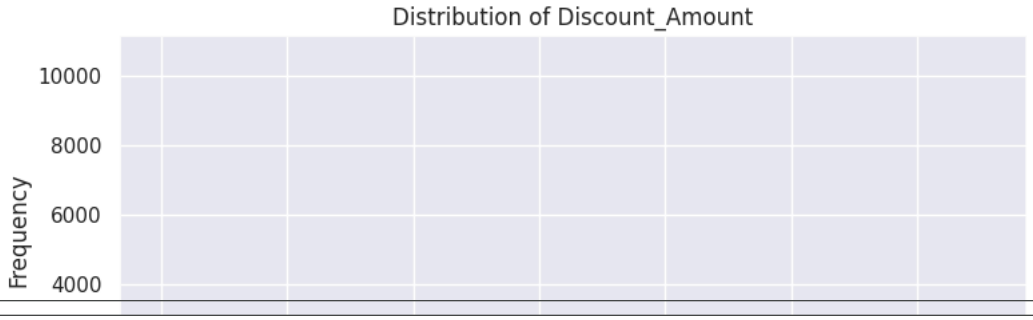



```
# Histograms for numeric variables
for col in numeric_cols:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[col], kde=True)
    plt.title(f"Distribution of {col}")
    plt.xlabel(col)
    plt.ylabel("Frequency")
    plt.tight_layout()
    plt.show()
```



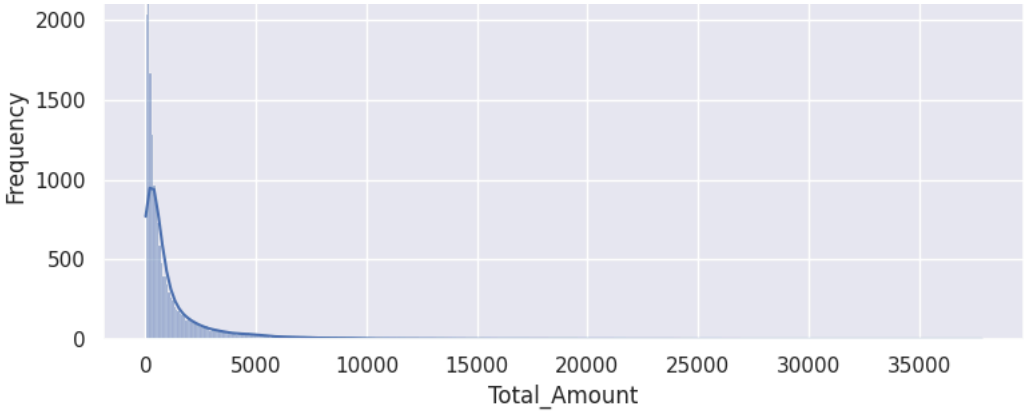


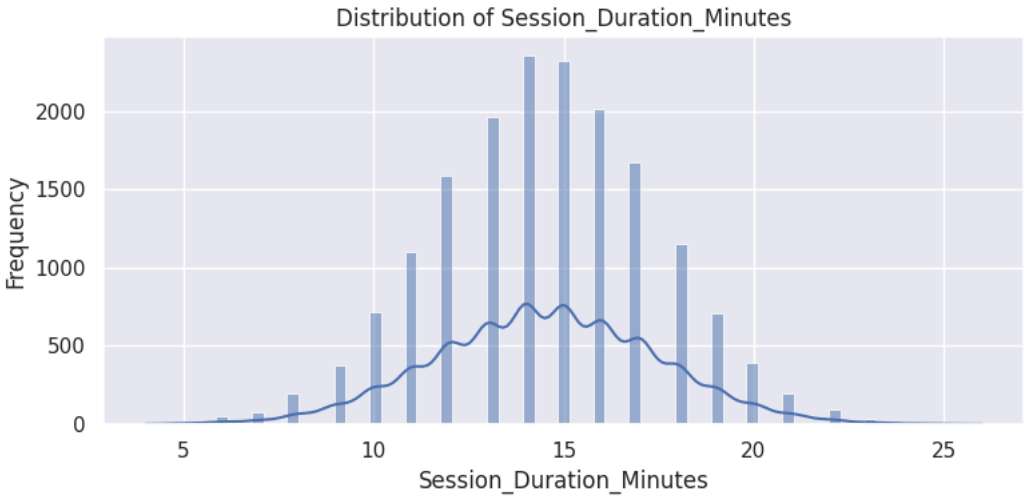


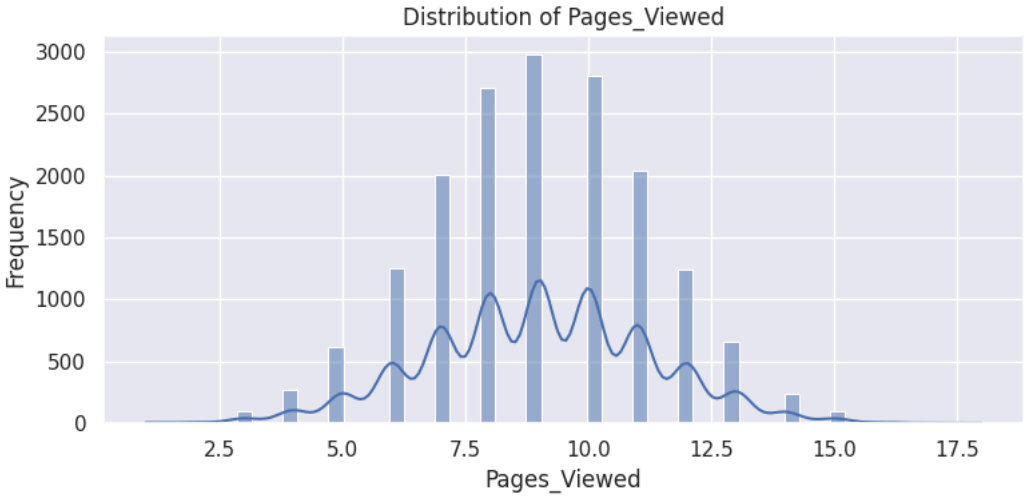


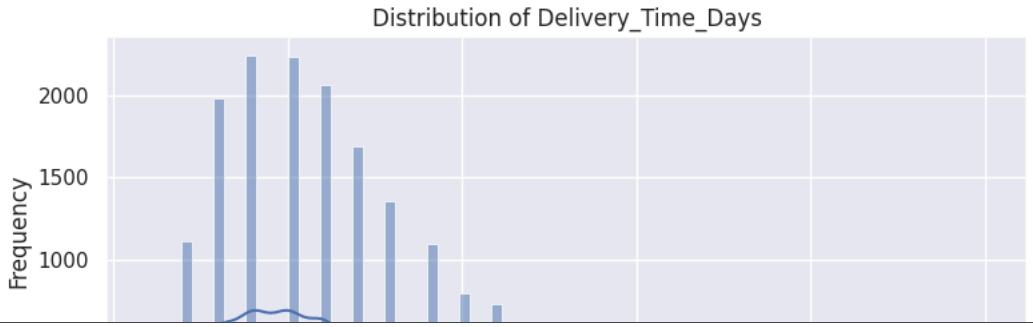

```
# Boxplots for numeric variables (outlier detection)
```

```
for col in numeric_cols:
  plt.figure(figsize=(12, 6))
  sns.boxplot(data=df[col])
  plt.title(f"Boxplots of {col}")
  plt.xticks(rotation=45)
  plt.tight_layout()
  plt.show()
```

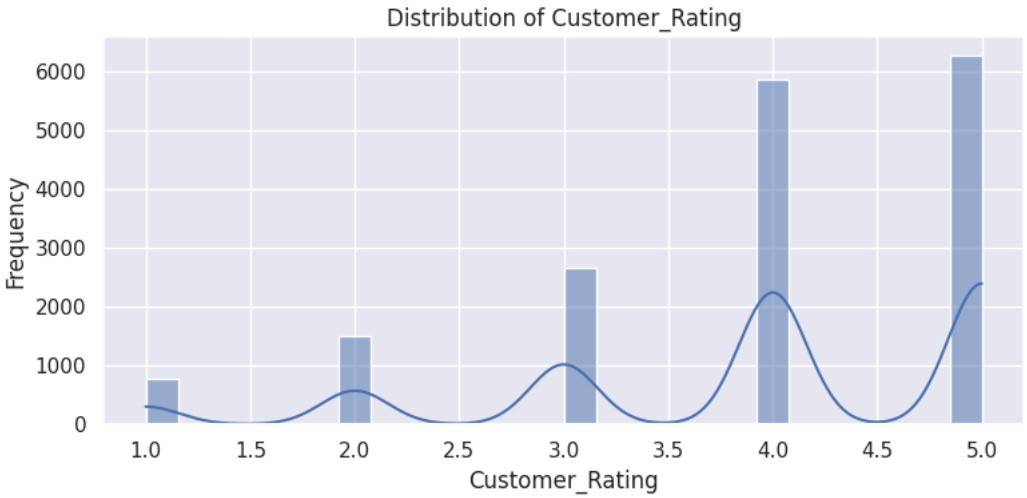


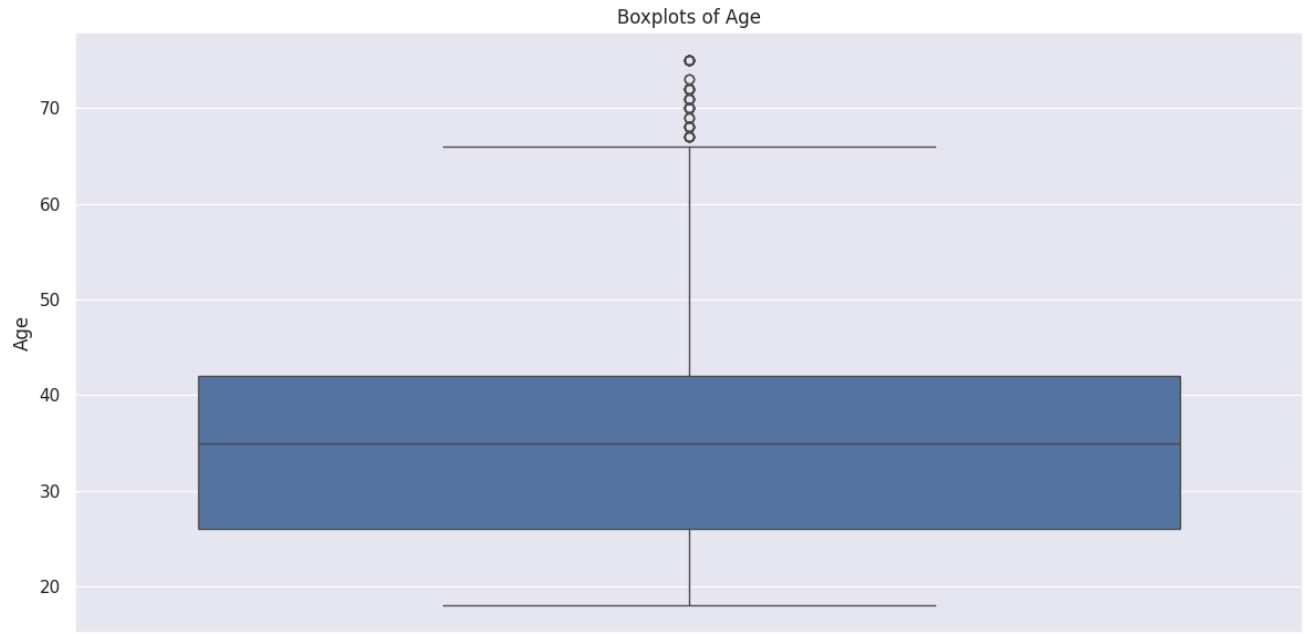


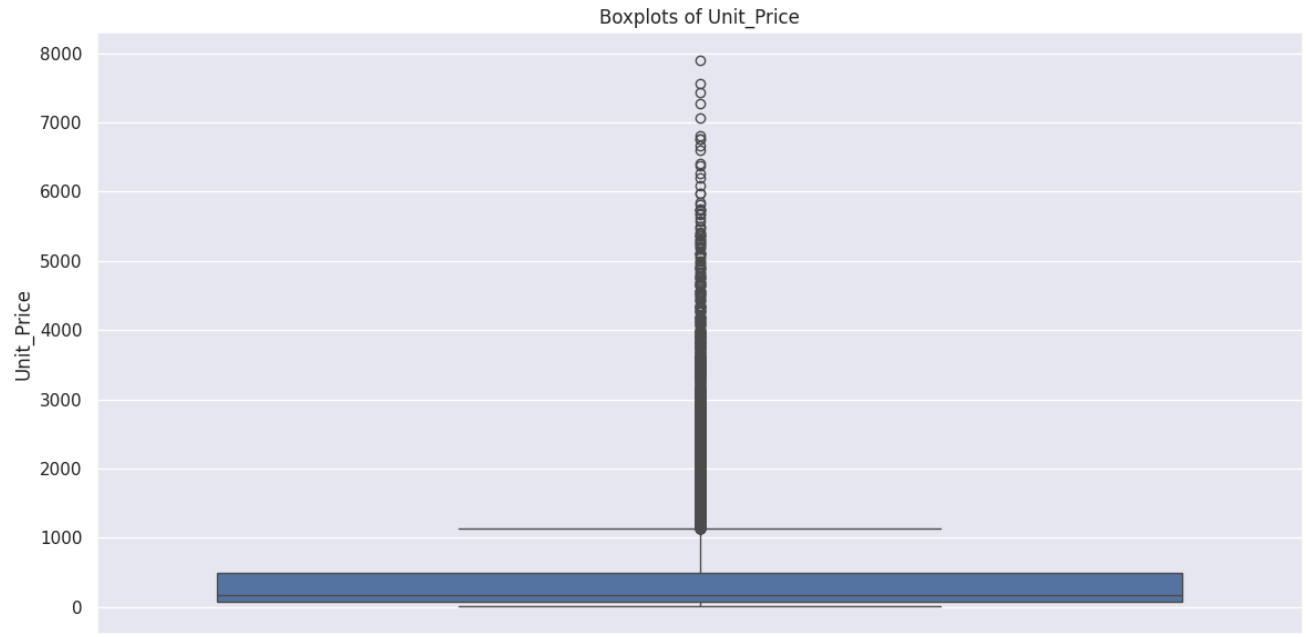


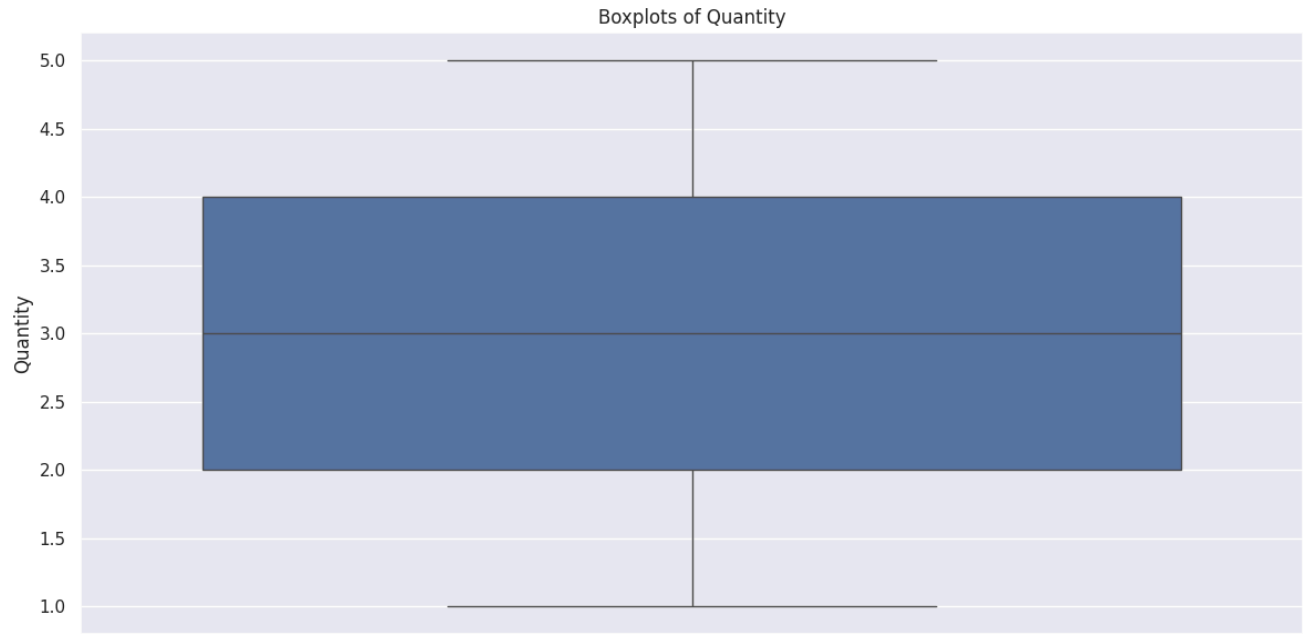


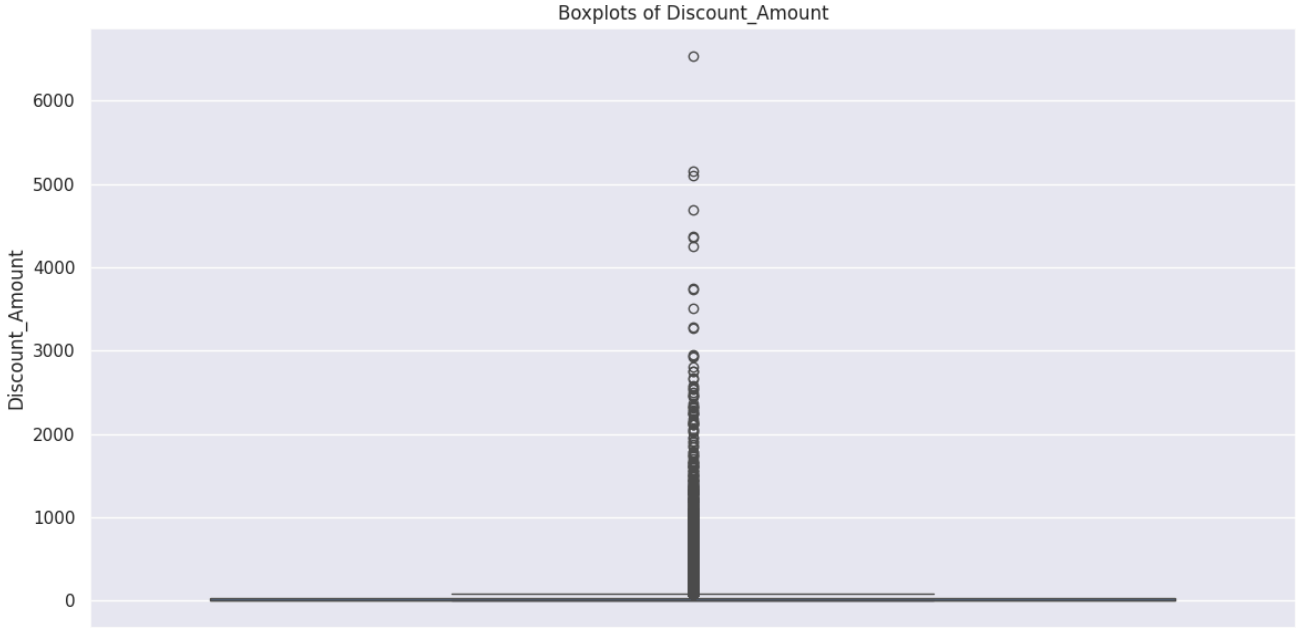

500
0
0
5
10
15
20
25
Delivery_Time_Days

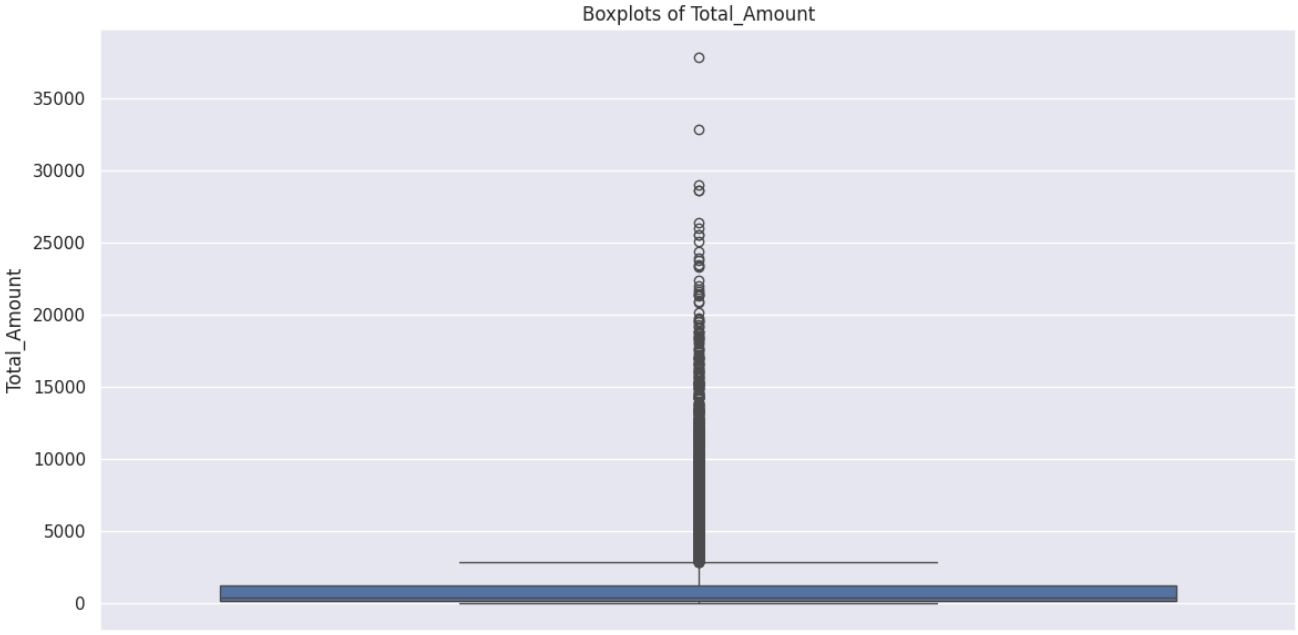

Distribution of Customer_Rating
Frequency
6000
5000
4000
3000
2000
1000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Customer_Rating

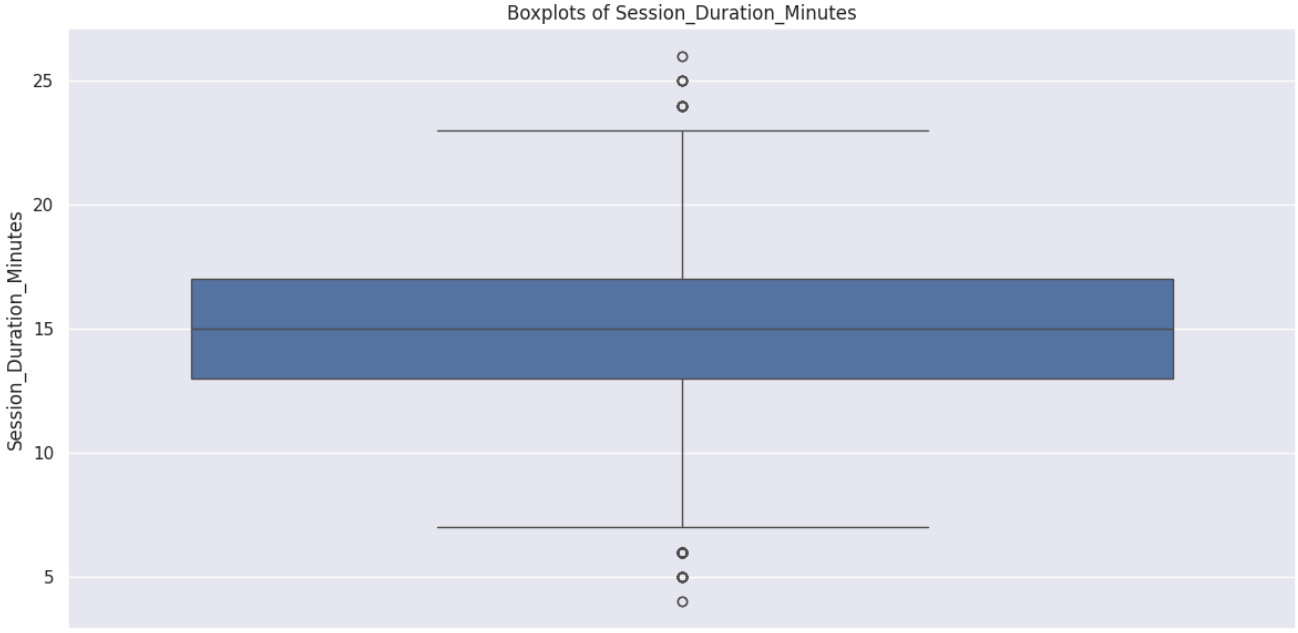

Boxplots of Age
Age
70
60
50
40
30
20


Boxplots of Unit_Price
Unit_Price
8000
7000
6000
5000
4000
3000
2000
1000
0

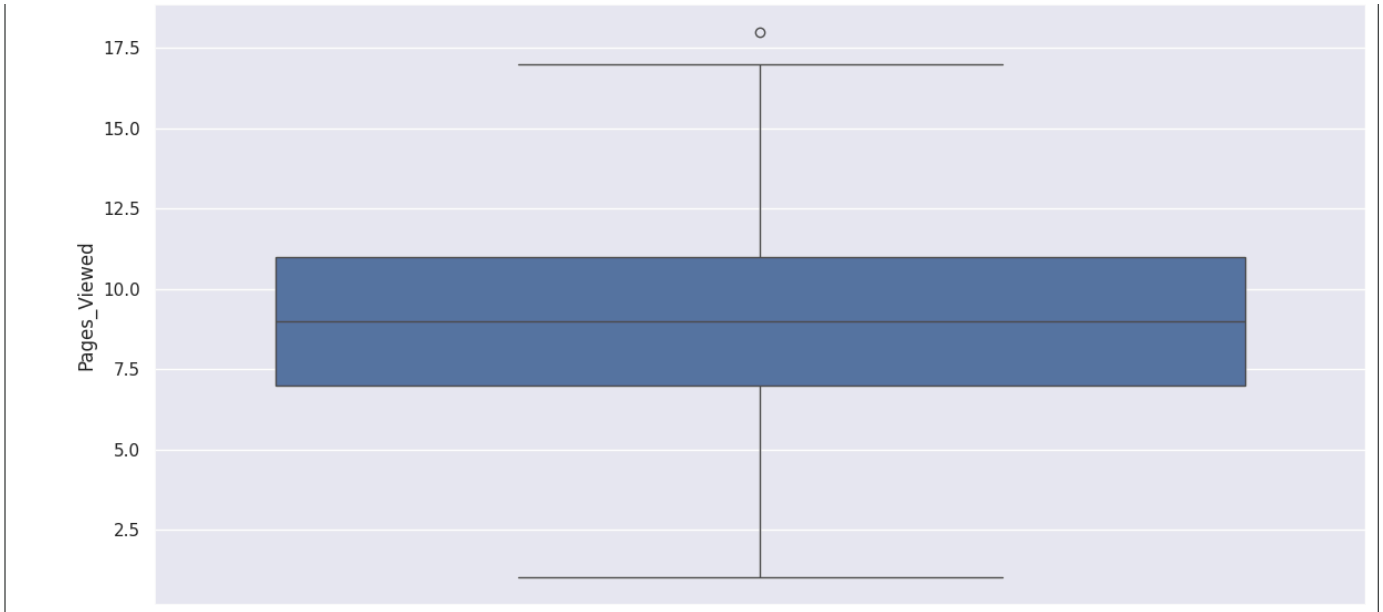

Boxplots of Quantity
Quantity
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0

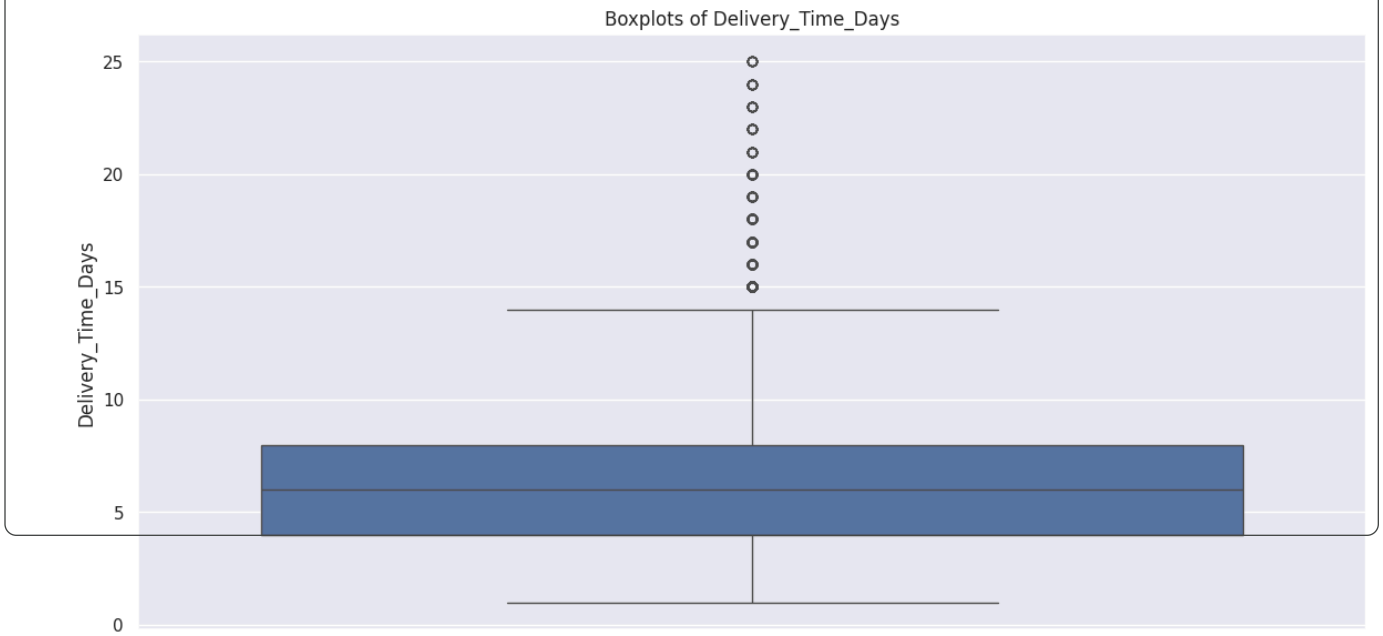


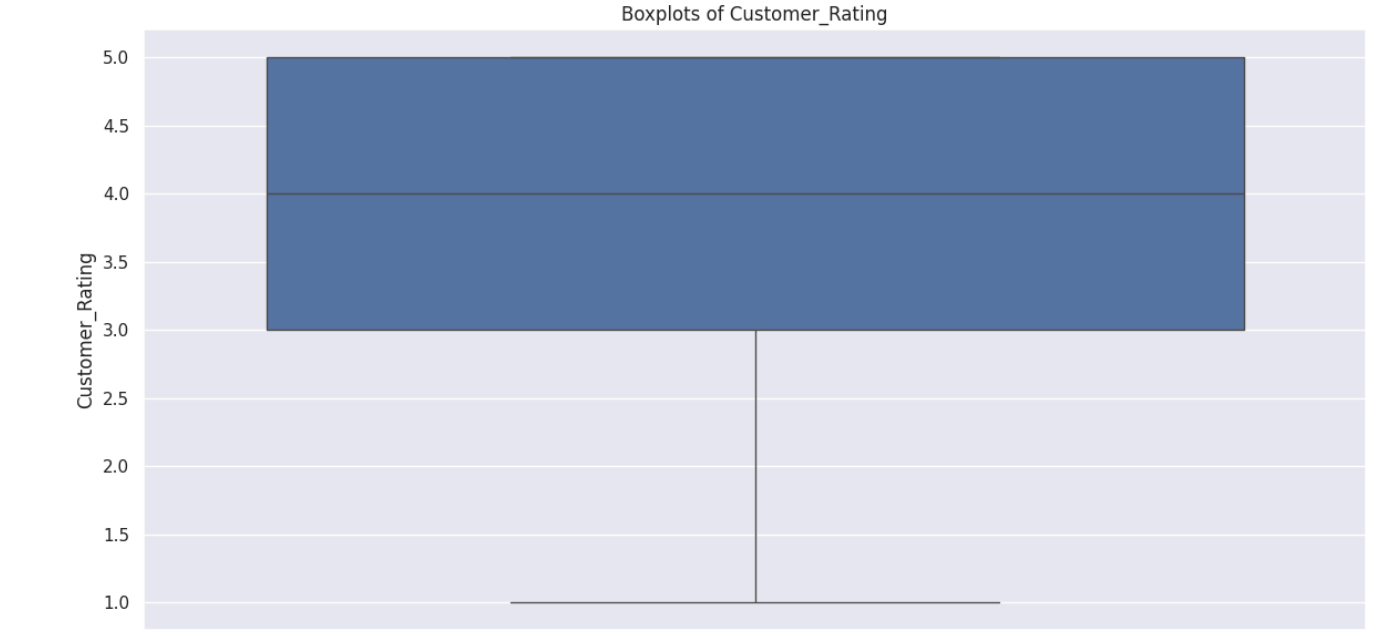




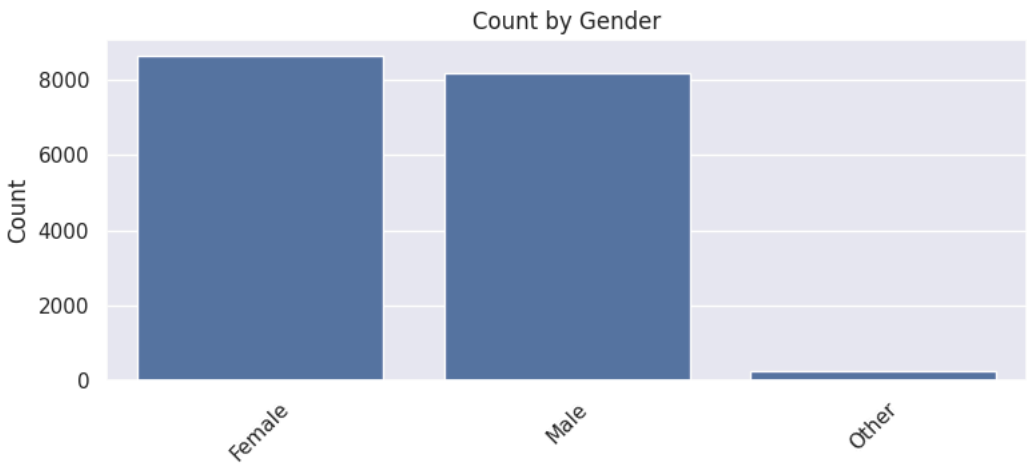
Boxplots of Pages_Viewed

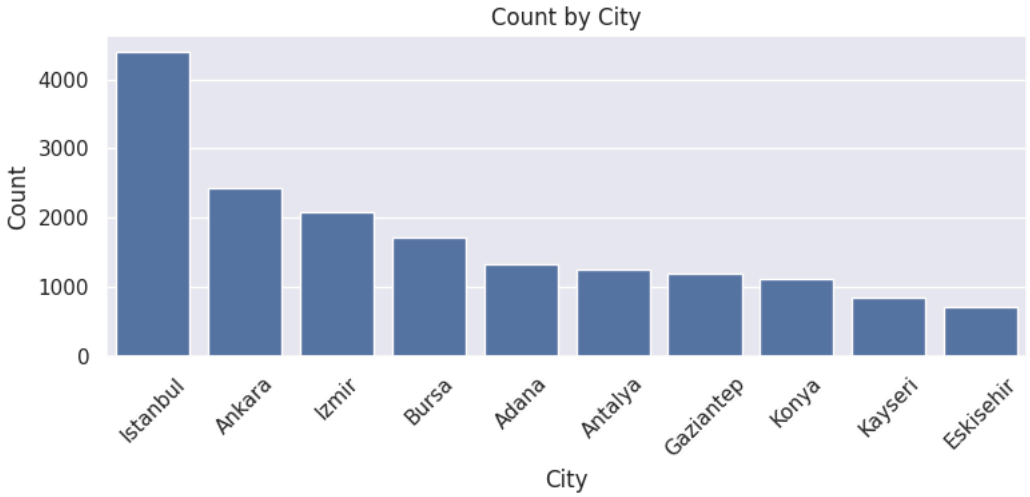

Pages_Viewed
17.5
15.0
12.5
10.0
7.5
5.0
2.5

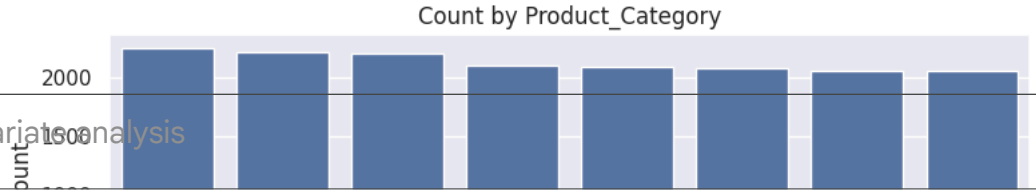

Boxplots of Delivery_Time_Days
Delivery_Time_Days
25
20
15
10
5
0


Boxplots of Customer_Rating
Customer_Rating
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0

```
for col in cat_cols:
    plt.figure(figsize=(8, 4))
    vc = df[col].value_counts().sort_values(ascending=False)
    sns.barplot(x=vc.index.astype(str), y=vc.values)
    plt.title(f"Count by {col}")
    plt.xlabel(col)
    plt.ylabel("Count")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

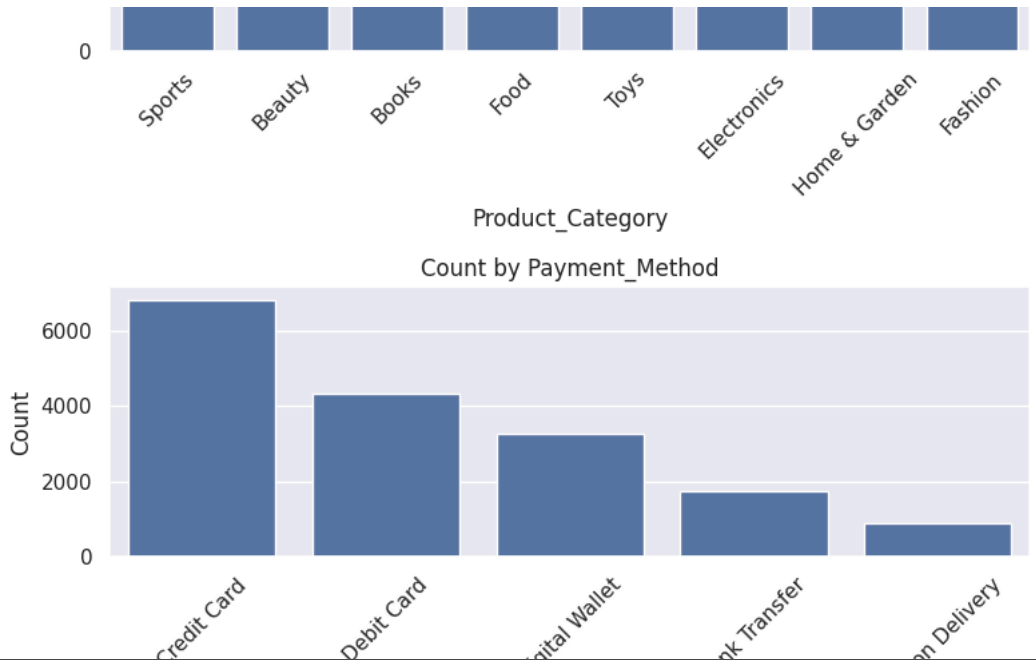


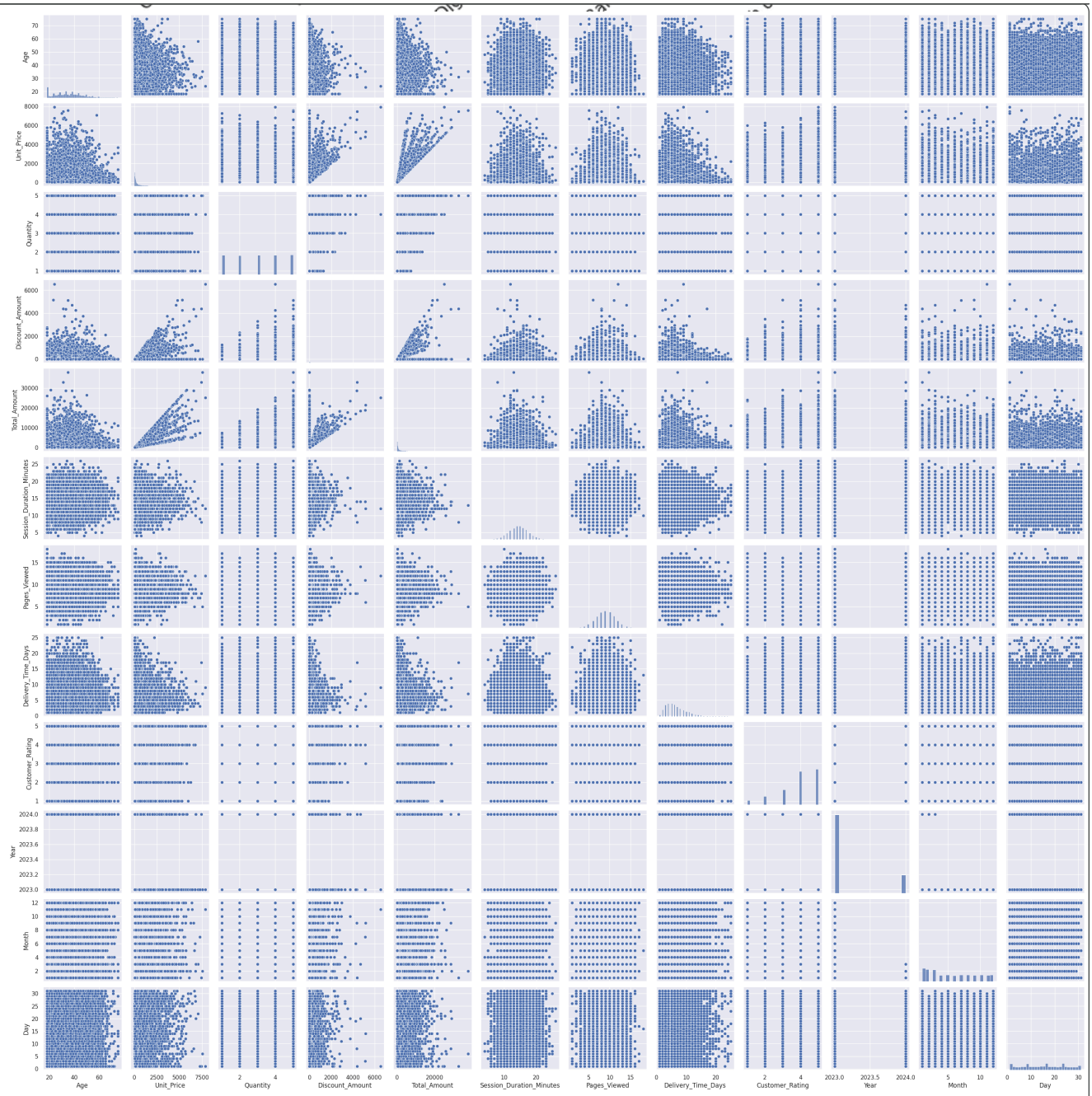




Bivariate analysis

```
sns.pairplot(df.select_dtypes(include=['float', 'int']))
plt.show();
```





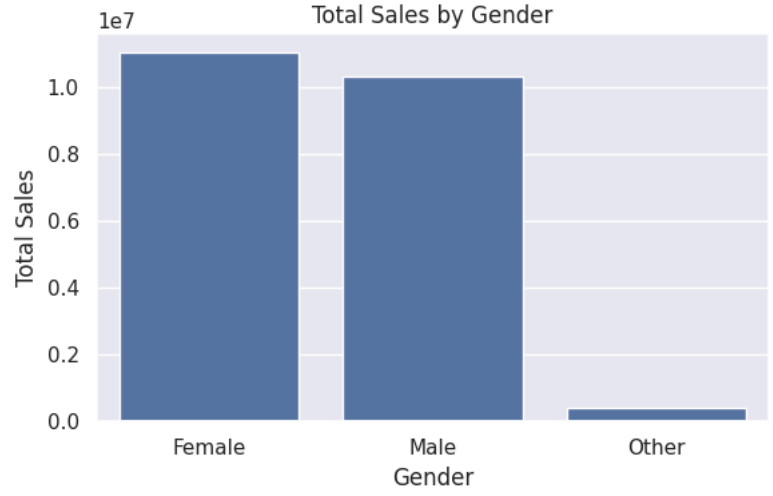

```
# Pivot table sales by city and product category
pivot_city_cat = pd.pivot_table(
    df,
    index="City",
    columns="Product_Category",
```

```
    values="Total_Amount",
    aggfunc="sum",
    fill_value=0,
    margins=True,
    margins_name="Total"
)
pivot_city_cat

# Heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(
    pivot_city_cat.drop("Total", axis=1).drop("Total", axis=0),
    annot=True,
    fmt=".0f"
)
plt.title("Total Sales by City and Product Category")
plt.ylabel("City")
plt.xlabel("Product Category")
plt.tight_layout()
plt.show()
```


```
# Total sales by gender
sales_gender = df.groupby("Gender")["Total_Amount"].sum().sort_values(ascending=False)
print(sales_gender)
```

```
Gender
Female   11,037,984.60
Male     10,343,185.70
Other       397,882.29
Name: Total_Amount, dtype: float64
```

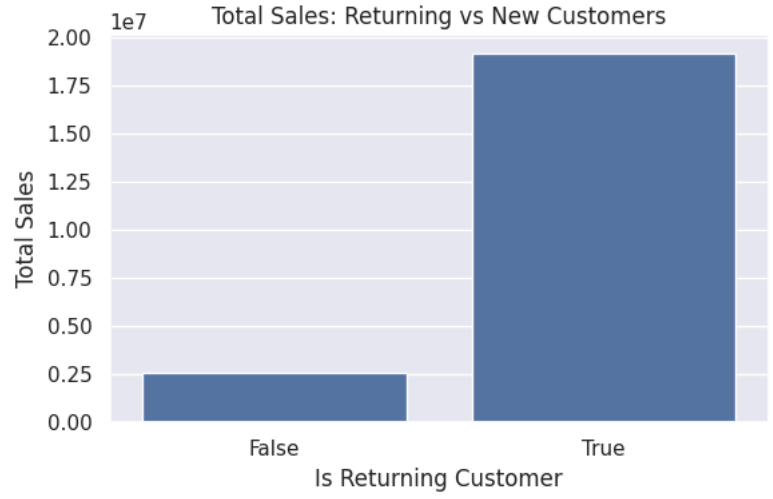
```
plt.figure(figsize=(6, 4))
sns.barplot(x=sales_gender.index, y=sales_gender.values)
plt.title("Total Sales by Gender")
plt.xlabel("Gender")
plt.ylabel("Total Sales")
plt.tight_layout()
plt.show()
```




```
# Total sales by returning vs new customer
sales_returning = df.groupby("Is_Returning_Customer")["Total_Amount"].sum()
print(sales_returning)
```

```
Is_Returning_Customer
False    2,588,332.13
True    19,190,720.46
Name: Total_Amount, dtype: float64
```

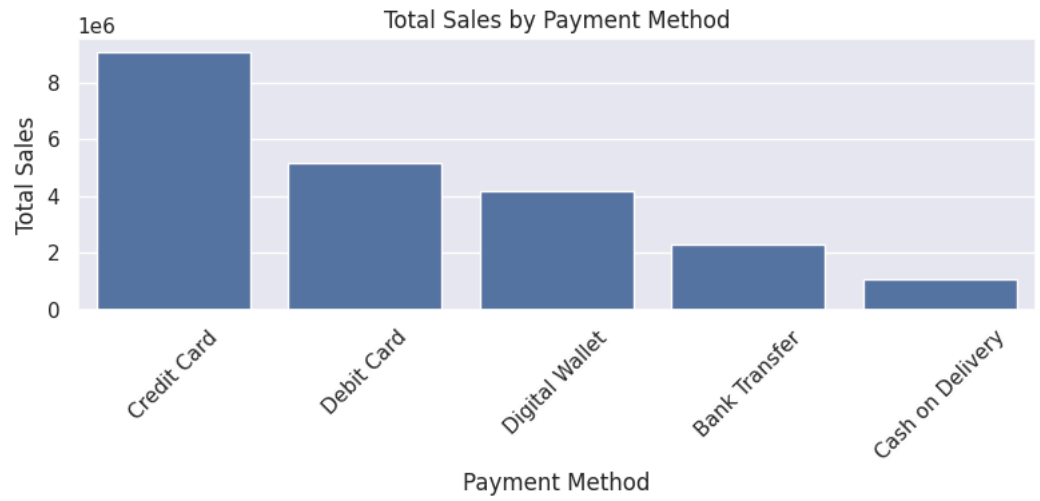
```
plt.figure(figsize=(6, 4))
sns.barplot(x=sales_returning.index.astype(str), y=sales_returning.values)
plt.title("Total Sales: Returning vs New Customers")
plt.xlabel("Is Returning Customer")
plt.ylabel("Total Sales")
plt.tight_layout()
plt.show()
```




```
# Sales by payment method
sales_payment = df.groupby("Payment_Method")["Total_Amount"].sum().sort_values(ascending=False)
print(sales_payment)
```

```
Payment_Method
Credit Card        9,069,714.65
Debit Card         5,152,182.83
Digital Wallet     4,191,842.00
Bank Transfer      2,311,499.25
Cash on Delivery   1,053,813.86
Name: Total_Amount, dtype: float64
```

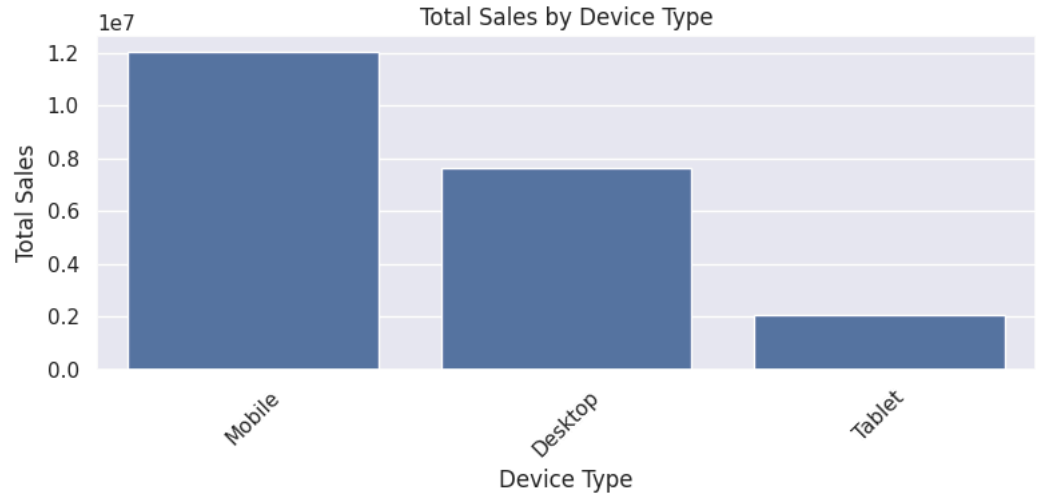
```
plt.figure(figsize=(8, 4))
sns.barplot(x=sales_payment.index, y=sales_payment.values)
plt.title("Total Sales by Payment Method")
plt.xlabel("Payment Method")
plt.ylabel("Total Sales")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
# Sales by device type
sales_device = df.groupby("Device_Type")["Total_Amount"].sum().sort_values(ascending=False)
print(sales_device)
```

```
Device_Type
Mobile    12,028,685.29
Desktop    7,661,437.07
Tablet     2,088,930.23
Name: Total_Amount, dtype: float64
```

```
plt.figure(figsize=(8, 4))
sns.barplot(x=sales_device.index, y=sales_device.values)
plt.title("Total Sales by Device Type")
plt.xlabel("Device Type")
plt.ylabel("Total Sales")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
df.columns
```

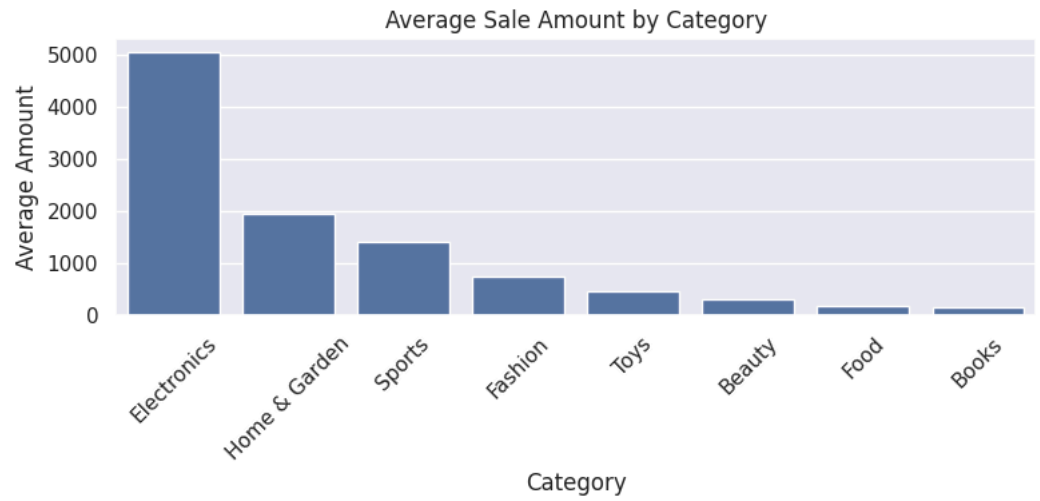
```
Index(['Order_ID', 'Customer_ID', 'Date', 'Age', 'Gender', 'City',
       'Product_Category', 'Unit_Price', 'Quantity', 'Discount_Amount',
       'Total_Amount', 'Payment_Method', 'Device_Type',
```

```
       'Session_Duration_Minutes', 'Pages_Viewed', 'Is_Returning_Customer',
       'Delivery_Time_Days', 'Customer_Rating', 'Year', 'Month', 'Month_Name',
       'Day', 'Weekday'],
      dtype='object')
```

```
# Average Sale Amount by device type
sales_cat = df.groupby("Product_Category")["Total_Amount"].mean().sort_values(ascending=False)
print(sales_cat)
```

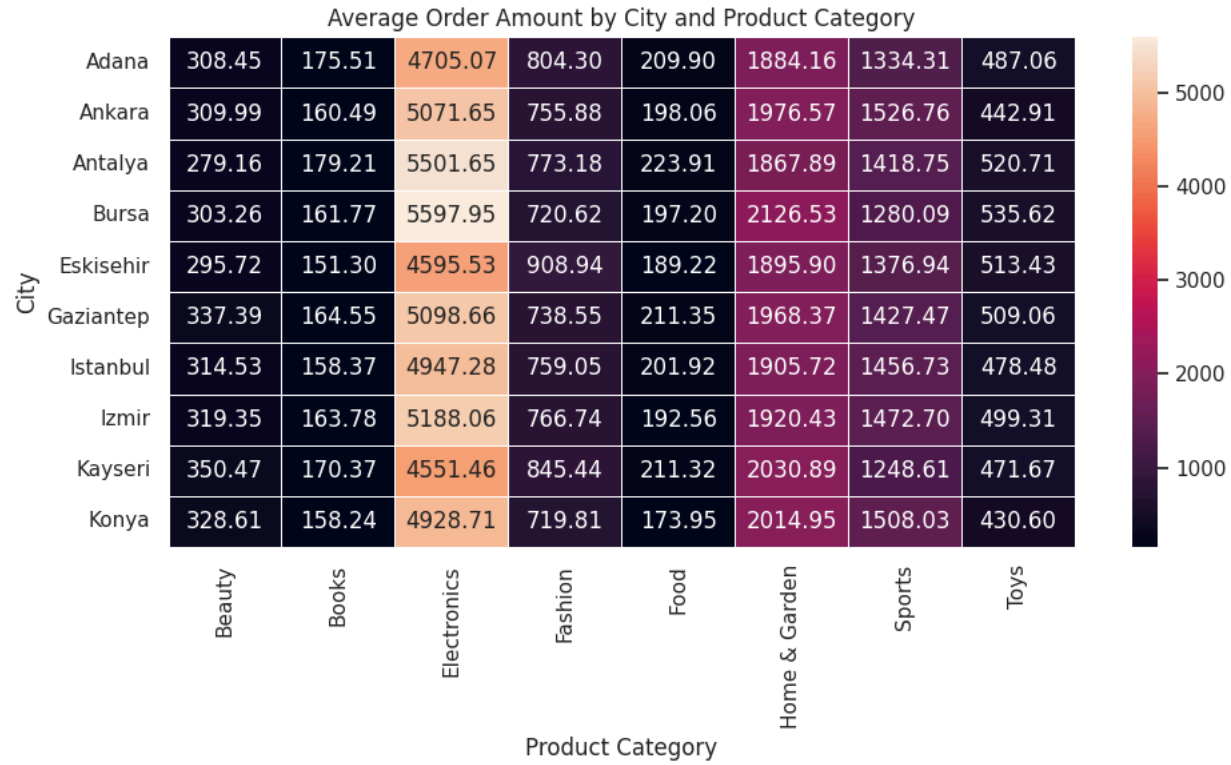
```
Product_Category
Electronics     5,053.95
Home & Garden   1,953.35
Sports          1,425.75
Fashion           767.04
Toys              485.28
Beauty            313.94
Food              200.69
Books             163.37
Name: Total_Amount, dtype: float64
```

```
plt.figure(figsize=(8, 4))
sns.barplot(x=sales_cat.index, y=sales_cat.values)
plt.title("Average Sale Amount by Category")
plt.xlabel("Category")
plt.ylabel("Average Amount")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```




```
pivot_mean = pd.pivot_table(df,
              values='Total_Amount',
              index='City',
              columns=['Product_Category'],
              aggfunc='mean'
              ).round(2)
```

```
plt.figure(figsize=(10, 6))
sns.heatmap(
    pivot_mean,
    annot=True,        # show the numbers
    fmt=".2f",
    linewidths=.5
)
plt.title("Average Order Amount by City and Product Category")
plt.xlabel("Product Category")
plt.ylabel("City")
plt.tight_layout()
plt.show()
```



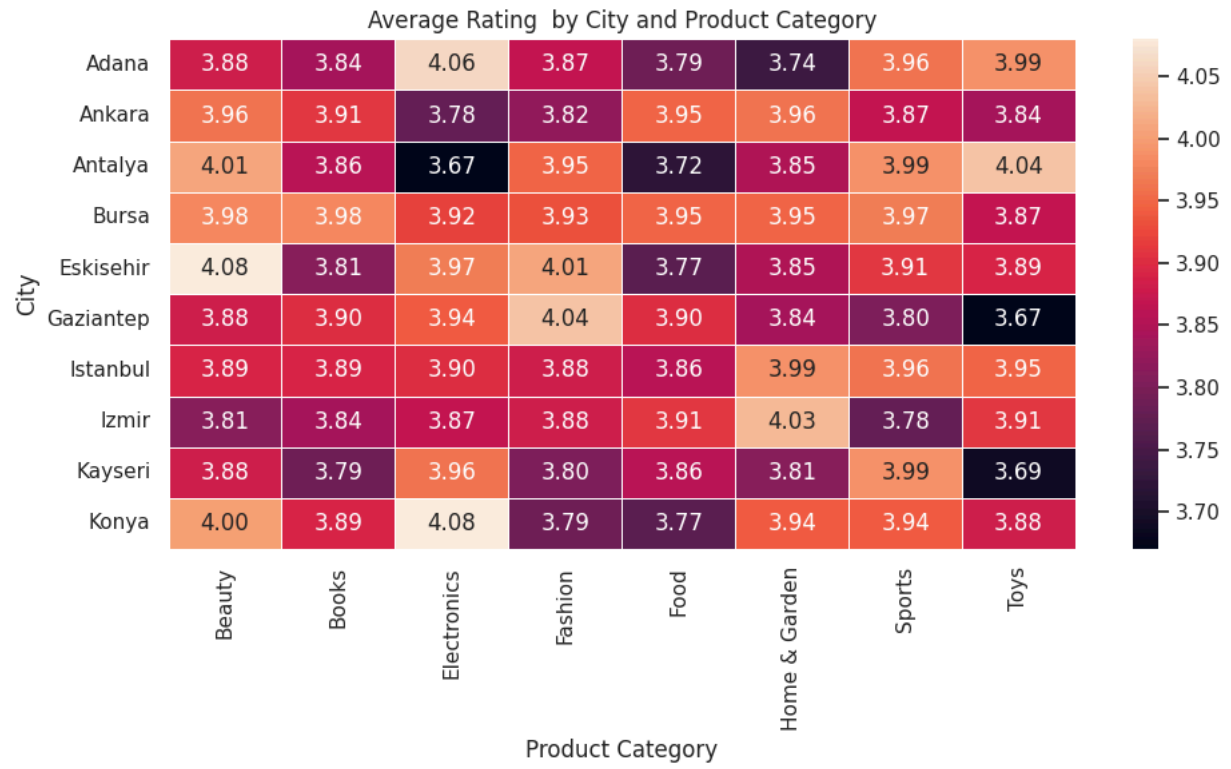

```
pivot_rating = pd.pivot_table(df,
               values='Customer_Rating',
               index='City',
               columns=['Product_Category'],
               aggfunc='mean'
               ).round(2)
pivot_rating
```

Product_Category	Beauty	Books	Electronics	Fashion	Food	Home & Garden	Sports	Toys
City								
Adana	3.88	3.84	4.06	3.87	3.79	3.74	3.96	3.99
Ankara	3.96	3.91	3.78	3.82	3.95	3.96	3.87	3.84
Antalya	4.01	3.86	3.67	3.95	3.72	3.85	3.99	4.04
Bursa	3.98	3.98	3.92	3.93	3.95	3.95	3.97	3.87
Eskisehir	4.08	3.81	3.97	4.01	3.77	3.85	3.91	3.89
Gaziantep	3.88	3.90	3.94	4.04	3.90	3.84	3.80	3.67
Istanbul	3.89	3.89	3.90	3.88	3.86	3.99	3.96	3.95
Izmir	3.81	3.84	3.87	3.88	3.91	4.03	3.78	3.91
Kayseri	3.88	3.79	3.96	3.80	3.86	3.81	3.99	3.69
Konya	4.00	3.89	4.08	3.79	3.77	3.94	3.94	3.88

Next steps: Generate code with pivot_rating | New interactive sheet

```
plt.figure(figsize=(10, 6))
sns.heatmap(
    pivot_rating,
    annot=True,        # show the numbers
    fmt=".2f",
    linewidths=.5
)
plt.title("Average Rating  by City and Product Category")
plt.xlabel("Product Category")
plt.ylabel("City")
```

```
plt.tight_layout()
plt.show()
```




```
bins = [0, 17, 24, 34, 44, 54, 64, 100]
labels = ["0–17", "18–24", "25–34", "35–44", "45–54", "55–64", "65+"]

df["Age_group"] = pd.cut(df["Age"], bins=bins, labels=labels, right=True, include_lowest=True)


df[["Age", "Age_group"]]
```

	Age	Age_group
0	40	35–44
1	40	35–44
2	40	35–44
3	33	25–34
4	33	25–34
...	...	...
17044	44	35–44
17045	24	18–24
17046	24	18–24
17047	24	18–24
17048	24	18–24

17049 rows × 2 columns

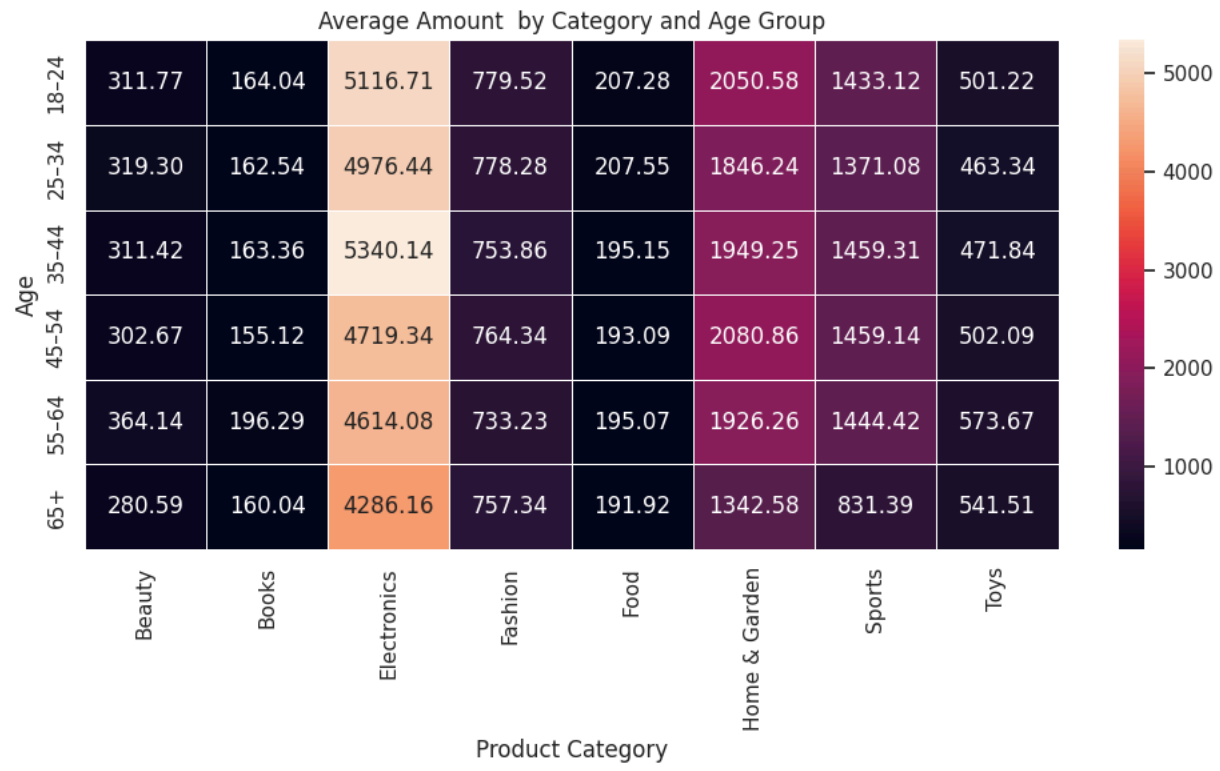
```
pivot_age = pd.pivot_table(df,
                values='Total_Amount',
                index='Age_group',
                columns=['Product_Category'],
                aggfunc='mean'
                ).round(2)
pivot_age
```

```
/tmp/ipython-input-807517238.py:1: FutureWarning: The default value of observed=False is deprecated and will change to observed=
  pivot_age = pd.pivot_table(df,
```

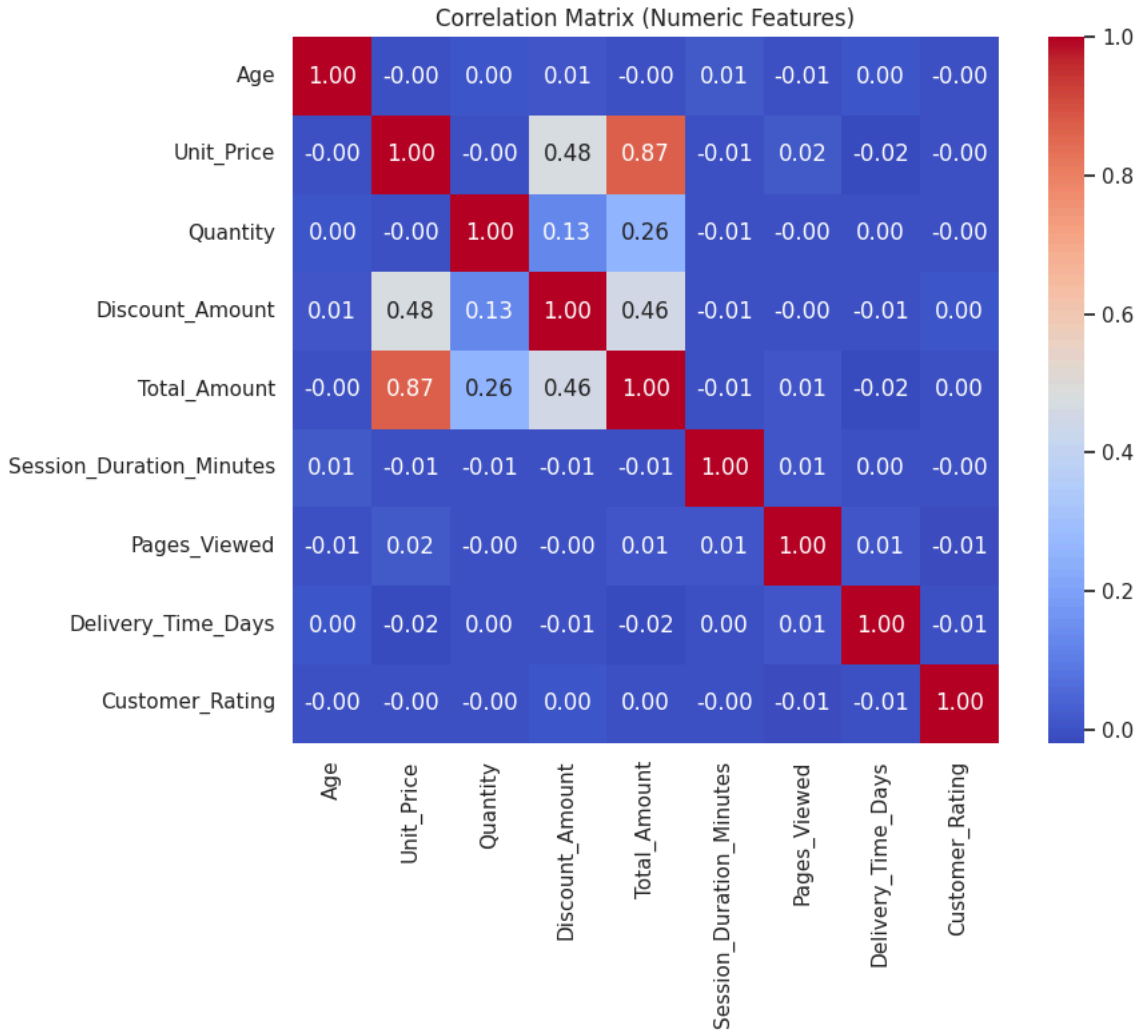
Product_Category	Beauty	Books	Electronics	Fashion	Food	Home & Garden	Sports	Toys
Age_group								
18–24	311.77	164.04	5,116.71	779.52	207.28	2,050.58	1,433.12	501.22
25–34	319.30	162.54	4,976.44	778.28	207.55	1,846.24	1,371.08	463.34
35–44	311.42	163.36	5,340.14	753.86	195.15	1,949.25	1,459.31	471.84
45–54	302.67	155.12	4,719.34	764.34	193.09	2,080.86	1,459.14	502.09
55–64	364.14	196.29	4,614.08	733.23	195.07	1,926.26	1,444.42	573.67
65+	280.59	160.04	4,286.16	757.34	191.92	1,342.58	831.39	541.51

Next steps: Generate code with pivot_age | New interactive sheet

```
plt.figure(figsize=(10, 6))
sns.heatmap(
    pivot_age,
    annot=True,        # show the numbers
    fmt=".2f",
    linewidths=.5
)
plt.title("Average Amount  by Category and Age Group")
plt.xlabel("Product Category")
plt.ylabel("Age")
plt.tight_layout()
plt.show()
```

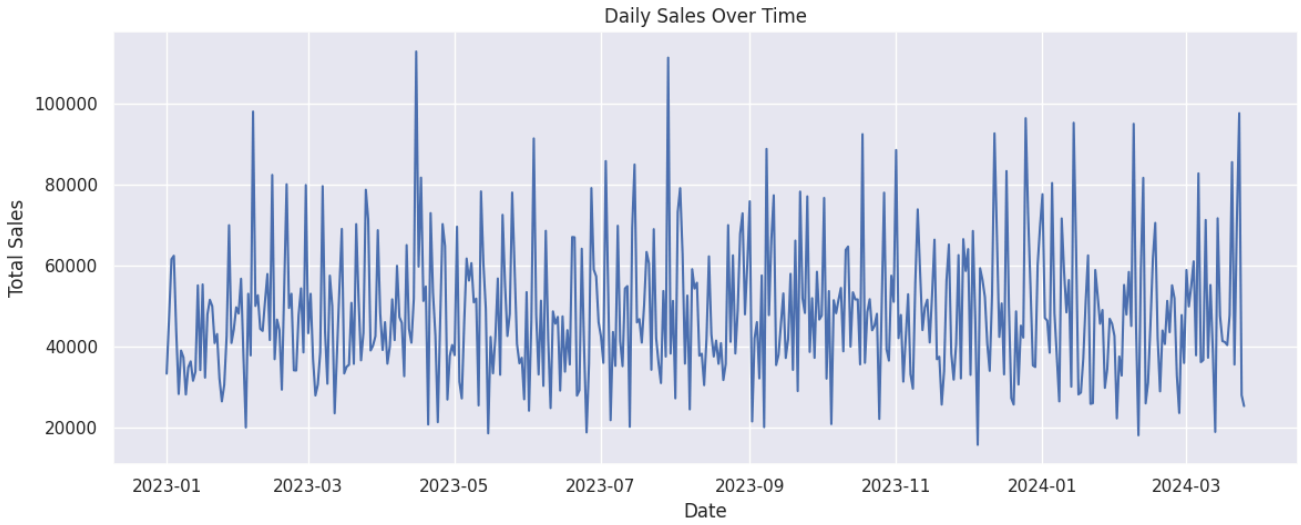


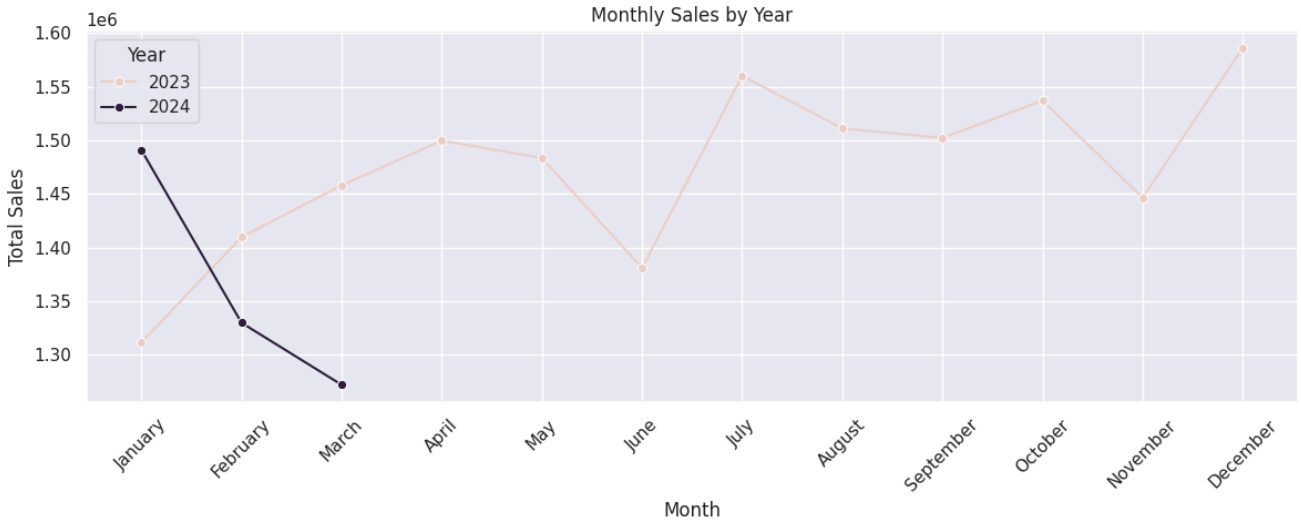

```
corr = df[numeric_cols].corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", square=True)
plt.title("Correlation Matrix (Numeric Features)")
plt.tight_layout()
plt.show()
```

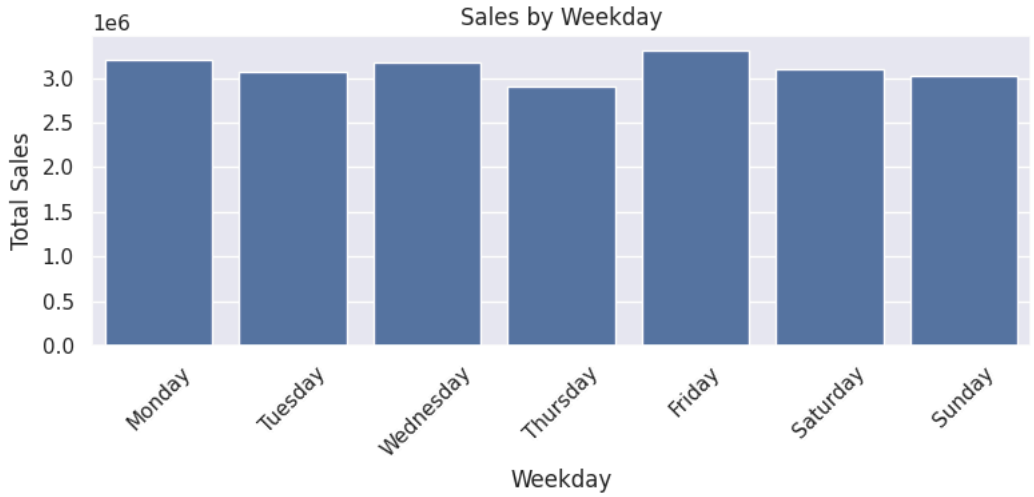



```
# Sales over time (daily)
daily_sales = df.groupby("Date")["Total_Amount"].sum().reset_index()

plt.figure(figsize=(12, 5))
sns.lineplot(data=daily_sales, x="Date", y="Total_Amount")
plt.title("Daily Sales Over Time")
plt.xlabel("Date")
plt.ylabel("Total Sales")
plt.tight_layout()
plt.show()

# Sales by month
monthly_sales = df.groupby(["Year", "Month_Name"])["Total_Amount"].sum().reset_index()

# To keep month order correct:
month_order = ["January", "February", "March", "April", "May", "June",
               "July", "August", "September", "October", "November", "December"]
monthly_sales["Month_Name"] = pd.Categorical(monthly_sales["Month_Name"],
                                             categories=month_order,
                                             ordered=True)
monthly_sales = monthly_sales.sort_values(["Year", "Month_Name"])

plt.figure(figsize=(12, 5))
sns.lineplot(data=monthly_sales, x="Month_Name", y="Total_Amount", hue="Year", marker="o")
plt.title("Monthly Sales by Year")
plt.xlabel("Month")
plt.ylabel("Total Sales")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# Sales by weekday
weekday_sales = df.groupby("Weekday")["Total_Amount"].sum().reset_index()
weekday_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
```

```
weekday_sales["Weekday"] = pd.Categorical(weekday_sales["Weekday"],
                                          categories=weekday_order,
                                          ordered=True)
weekday_sales = weekday_sales.sort_values("Weekday")

plt.figure(figsize=(8, 4))
sns.barplot(data=weekday_sales, x="Weekday", y="Total_Amount")
plt.title("Sales by Weekday")
plt.xlabel("Weekday")
plt.ylabel("Total Sales")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

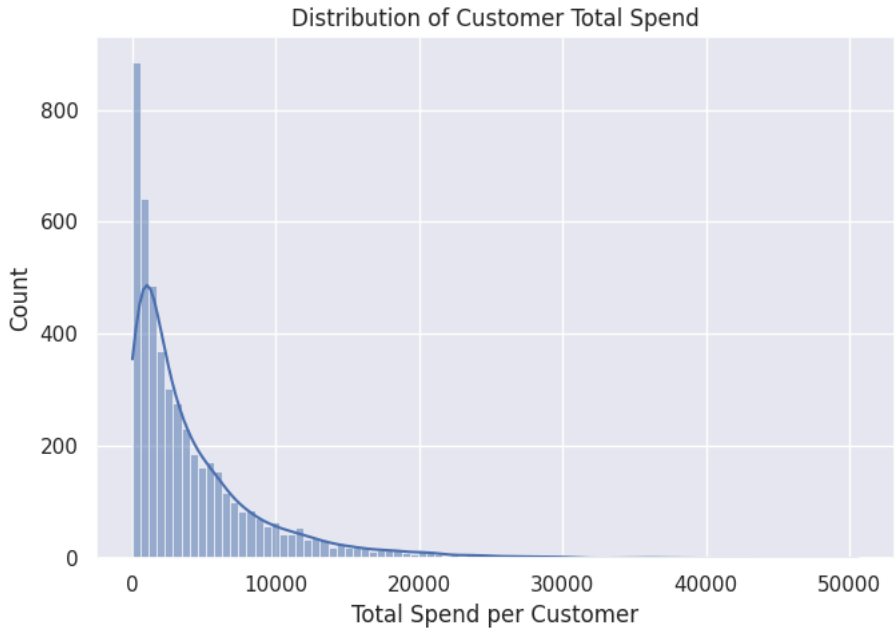







```
# Aggregate per customer
cust = df.groupby("Customer_ID").agg(
    n_orders=("Order_ID", "nunique"),
    total_spent=("Total_Amount", "sum"),
```

```
    avg_order_value=("Total_Amount", "mean"),
    avg_rating=("Customer_Rating", "mean"),
    min_date=("Date", "min"),
    max_date=("Date", "max"),
    returning=("Is_Returning_Customer", "max") ).reset_index()

cust["customer_lifetime_days"] = (cust["max_date"] - cust["min_date"]).dt.days + 1

cust.head()

# Distribution of total spent
plt.figure(figsize=(7, 5))
sns.histplot(cust["total_spent"], kde=True)
plt.title("Distribution of Customer Total Spend")
plt.xlabel("Total Spend per Customer")
plt.tight_layout()
plt.show()

# Total spent vs number of orders
plt.figure(figsize=(7, 5))
sns.scatterplot(data=cust, x="n_orders", y="total_spent", alpha=0.6)
plt.title("Customer Total Spend vs Number of Orders")
plt.xlabel("Number of Orders")
plt.ylabel("Total Spend")
plt.tight_layout()
plt.show()
```




```
sales_category=df.groupby("Product_Category")["Total_Amount"].sum().sort_values(ascending=False)
print(sales_category)
```

```
Product_Category
Electronics     10,481,897.65
Home & Garden    4,023,903.94
Sports           3,205,086.99
Fashion          1,577,035.70
Toys             1,014,237.53
Beauty             694,437.02
Food               422,054.65
Books              360,399.11
Name: Total_Amount, dtype: float64
```

```
sales_city=df.groupby("City")["Total_Amount"].sum().sort_values(ascending=False)
print(sales_city)
```

```
City
Istanbul    5,646,595.78
Ankara      3,052,443.10
Izmir       2,651,743.92
Bursa       2,317,028.10
Antalya     1,661,594.15
Gaziantep   1,573,422.25
Adana       1,512,320.19
```

```
Konya       1,363,152.80
Kayseri     1,091,658.53
Eskisehir     909,093.77
Name: Total_Amount, dtype: float64
```

```
sales_gender=df.groupby("Gender")["Total_Amount"].sum().sort_values(ascending=False)
print(sales_gender)
```

```
Gender
Female   11,037,984.60
Male     10,343,185.70
Other       397,882.29
Name: Total_Amount, dtype: float64
```

```
pivot_sales = pd.pivot_table(
    df,
    index="City",                # rows
    columns="Product_Category",  # columns
    values="Total_Amount",       # what we're aggregating = sales
    aggfunc="sum",               # total sales
    fill_value=0                 # replace NaN with 0
)

print(pivot_sales)
```

```
Product_Category     Beauty     Books  Electronics    Fashion       Food  \
City
Adana             60,764.96 31,240.52   682,235.44 136,731.20  35,263.96
Ankara           108,806.15 47,665.86 1,414,990.84 230,542.60  53,277.10
Antalya           43,270.35 28,673.55   847,253.35 115,976.59  36,945.54
Bursa             60,954.85 36,560.39 1,153,177.97 138,358.67  44,566.66
Eskisehir         22,474.95 14,675.75   436,575.08  73,624.52  16,650.98
Gaziantep         51,282.94 27,645.15   815,785.08 101,920.30  31,068.90
Istanbul         171,733.51 87,105.76 2,730,896.02 414,441.59 108,028.90
Izmir             89,738.61 43,893.20 1,276,263.97 184,785.47  49,872.94
Kayseri           36,448.43 20,785.27   523,417.86  87,079.82  20,286.74
Konya             48,962.27 22,153.66   601,302.04  93,574.94  26,092.93

Product_Category  Home & Garden     Sports       Toys
City
Adana                273,203.48 213,489.41  79,391.22
Ankara               577,157.75 480,929.39 139,073.41
Antalya              293,258.87 217,068.56  79,147.34
Bursa                459,330.17 311,062.98 113,016.41
Eskisehir            182,006.66 118,417.04  44,668.79
Gaziantep            265,730.28 211,265.84  68,723.76
Istanbul           1,019,559.78 849,271.96 265,558.26
Izmir                472,426.12 407,938.39 126,825.22
Kayseri              213,242.93 149,833.73  40,563.75
Konya                267,987.90 245,809.69  57,269.37
```

KNN CLASSIFIER

```
cust = df.groupby("Customer_ID").agg(
    Age=("Age", "first"),                          # age is fixed per customer
    Gender=("Gender", "first"),
    City=("City", "first"),
    Device_Type=("Device_Type", lambda x: x.mode().iat[0] if not x.mode().empty else x.iloc[0]),
    is_returning=("Is_Returning_Customer", "max"), # True if ever returning
    total_spent=("Total_Amount", "sum"),
    n_orders=("Order_ID", "nunique"),
    avg_rating=("Customer_Rating", "mean"),
    first_purchase=("Date", "min"),
    last_purchase=("Date", "max")
).reset_index()
```

```
cust
```

	Customer_ID	Age	Gender	City	Device_Type	is_returning	total_spent	n_orders	avg_rating	first_purchase	last_purch
0	CUST_00001	40	Male	Ankara	Mobile	True	2,199.63	3	3.33	2023-05-29	2023-1
1	CUST_00002	33	Male	Istanbul	Desktop	True	809.90	2	4.00	2023-05-11	2023-0
2	CUST_00003	42	Male	Konya	Desktop	True	3,030.81	2	3.50	2023-02-27	2024-0
3	CUST_00004	53	Male	Izmir	Desktop	False	383.22	1	5.00	2024-02-13	2024-0
4	CUST_00005	32	Male	Ankara	Mobile	True	2,422.73	3	3.67	2023-03-16	2023-0
...	...	...	...	...	...	...	...	...	...	...	
4995	CUST_04996	34	Male	Antalya	Mobile	True	3,001.96	4	3.00	2023-07-07	2024-0
4996	CUST_04997	43	Female	Adana	Mobile	True	15,440.42	4	4.00	2023-01-31	2023-1
4997	CUST_04998	72	Female	Kayseri	Desktop	False	482.90	1	3.00	2023-04-27	2023-0
4998	CUST_04999	44	Male	Antalya	Mobile	False	137.30	1	1.00	2024-01-16	2024-0
4999	CUST_05000	24	Female	Eskisehir	Desktop	True	7,282.41	4	3.75	2023-02-22	2024-0

5000 rows × 11 columns

Next steps: Generate code with cust | New interactive sheet

```
cust["customer_lifetime_days"] = (cust["last_purchase"] - cust["first_purchase"]).dt.days + 1
cust["customer_lifetime_days"] = cust["customer_lifetime_days"].fillna(0)
cust["is_returning"] = cust["is_returning"].astype(int)   # bool → 0/1

cust.head()
```

	Customer_ID	Age	Gender	City	Device_Type	is_returning	total_spent	n_orders	avg_rating	first_purchase	last_purchase
0	CUST_00001	40	Male	Ankara	Mobile	1	2,199.63	3	3.33	2023-05-29	2023-12-05
1	CUST_00002	33	Male	Istanbul	Desktop	1	809.90	2	4.00	2023-05-11	2023-06-16
2	CUST_00003	42	Male	Konya	Desktop	1	3,030.81	2	3.50	2023-02-27	2024-01-03
3	CUST_00004	53	Male	Izmir	Desktop	0	383.22	1	5.00	2024-02-13	2024-02-13
4	CUST_00005	32	Male	Ankara	Mobile	1	2,422.73	3	3.67	2023-03-16	2023-06-21

Next steps: Generate code with cust | New interactive sheet

```
# Threshold for high value (top 25% spenders)
threshold = cust["total_spent"].quantile(0.75)

cust["high_value"] = (cust["total_spent"] >= threshold).astype(int)

cust["high_value"].value_counts()
```

high_value	count
0	3750
1	1250

dtype: int64

```
feature_cols = [
    "Age",
    "Gender",
    "City",
    "Device_Type",
    "is_returning",
    "n_orders",
    "avg_rating",
    "customer_lifetime_days"
]

X = cust[feature_cols]
y = cust["high_value"]
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y  # keep same proportion of high/standard in both sets
)
```

```
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier

numeric_features = ["Age", "is_returning", "n_orders", "avg_rating", "customer_lifetime_days"]
categorical_features = ["Gender", "City", "Device_Type"]

numeric_transformer = Pipeline(steps=[
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("onehot", OneHotEncoder(handle_unknown="ignore"))
])

preprocess = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features),
    ]
)

knn_clf = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", KNeighborsClassifier())
])
```

```
from sklearn.model_selection import GridSearchCV
param_grid = {
}
```

```
Best params: {'model__metric': 'minkowski', 'model__n_neighbors': 9, 'model__weights': 'uniform'}
Best CV F1: 0.4505022273631817
```

```
best_knn = grid_search.best_estimator_
```

```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = best_knn.predict(X_test)

print("Classification report:\n")
print(classification_report(y_test, y_pred, target_names=["Standard", "High-value"]))

print("Confusion matrix:\n")
```