Exploratory Data Analysis (EDA) on Retail Data

Overview

This project involves the analysis of three datasets—customers, products, and transactions—to extract valuable insights into customer behavior, sales trends, and product performance. Below is a breakdown of the steps and tasks performed.

---

1. Data Loading and Libraries Setup

- Imported essential Python libraries:
  - `numpy` and `pandas` for data manipulation.
  - `matplotlib` and `seaborn` for data visualization.
- Loaded datasets:
  - **Customers**: `customers.csv`
  - **Products**: `products.csv`
  - **Transactions**: `transactions.csv`

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:
```
plt.style.available
```

Out[3]:
```
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'petroff10',
 'seaborn-v0_8',
 'seaborn-v0_8-bright',
 'seaborn-v0_8-colorblind',
 'seaborn-v0_8-dark',
 'seaborn-v0_8-dark-palette',
 'seaborn-v0_8-darkgrid',
 'seaborn-v0_8-deep',
 'seaborn-v0_8-muted',
 'seaborn-v0_8-notebook',
 'seaborn-v0_8-paper',
 'seaborn-v0_8-pastel',
 'seaborn-v0_8-poster',
 'seaborn-v0_8-talk',
 'seaborn-v0_8-ticks',
 'seaborn-v0_8-white',
 'seaborn-v0_8-whitegrid',
 'tableau-colorblind10']
```

In [8]:
```
sns.set_style("darkgrid")
```

In [9]:
```
# Load the datasets
customers = pd.read_csv("Desktop/Customers.csv")
products = pd.read_csv("Desktop/Products.csv")
transactions = pd.read_csv("Desktop/Transactions.csv")
```

```
Customers Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   CustomerID    200 non-null    object
 1   CustomerName  200 non-null    object
 2   Region        200 non-null    object
 3   SignupDate    200 non-null    object
dtypes: object(4)
memory usage: 6.4+ KB
None

Products Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   ProductID    100 non-null    object 
 1   ProductName  100 non-null    object 
 2   Category     100 non-null    object 
 3   Price        100 non-null    float64
dtypes: float64(1), object(3)
memory usage: 3.2+ KB
None

Transactions Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   TransactionID    1000 non-null   object 
 1   CustomerID       1000 non-null   object 
 2   ProductID        1000 non-null   object 
 3   TransactionDate  1000 non-null   object 
 4   Quantity         1000 non-null   int64  
 5   TotalValue       1000 non-null   float64
 6   Price            1000 non-null   float64
dtypes: float64(2), int64(1), object(4)
memory usage: 54.8+ KB
None
```

2. Data Exploration

- Displayed dataset structures using `.info()`:
  - **Customers**: 200 entries, 4 columns (CustomerID, CustomerName, Region, SignupDate).
  - **Products**: 100 entries, 4 columns (ProductID, ProductName, Category, Price).
  - **Transactions**: 1000 entries, 7 columns (TransactionID, CustomerID, ProductID, TransactionDate, Quantity, TotalValue, Price).
- Verified missing values:

- No missing data in any dataset.
- Checked for duplicate entries:
  - Removed duplicates from all datasets.

In [33]:
```
# Display basic information
print("Customers Dataset:")
print(customers.info(), "\n")
print("Products Dataset:")
print(products.info(), "\n")
print("Transactions Dataset:")
print(transactions.info(), "\n")
```

```
Customers Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   CustomerID    200 non-null    object
 1   CustomerName  200 non-null    object
 2   Region        200 non-null    object
 3   SignupDate    200 non-null    datetime64[ns]
dtypes: datetime64[ns](1), object(3)
memory usage: 6.4+ KB
None

Products Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ProductID    100 non-null    object
 1   ProductName  100 non-null    object
 2   Category     100 non-null    object
 3   Price        100 non-null    float64
dtypes: float64(1), object(3)
memory usage: 3.2+ KB
None

Transactions Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   TransactionID    1000 non-null   object
 1   CustomerID       1000 non-null   object
 2   ProductID        1000 non-null   object
 3   TransactionDate  1000 non-null   datetime64[ns]
 4   Quantity         1000 non-null   int64
 5   TotalValue       1000 non-null   float64
 6   Price            1000 non-null   float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(3)
memory usage: 54.8+ KB
None
```

3. Data Cleaning

- Converted `SignupDate` and `TransactionDate` columns to `datetime` format for analysis.
- Verified dataset consistency after cleaning:
  - All data types aligned with expected values.

In [10]:
```
# Check for missing values
print("Missing Values:\n")
print("Customers:\n", customers.isnull().sum(), "\n")
print("Products:\n", products.isnull().sum(), "\n")
print("Transactions:\n", transactions.isnull().sum(), "\n")

# Convert date columns to datetime
customers['SignupDate'] = pd.to_datetime(customers['SignupDate'])
transactions['TransactionDate'] = pd.to_datetime(transactions['TransactionDate']

# Remove duplicates if any
customers.drop_duplicates(inplace=True)
products.drop_duplicates(inplace=True)
transactions.drop_duplicates(inplace=True)

# Confirm data cleaning
print("Cleaned Data:")
print(customers.info(), "\n")
print(products.info(), "\n")
print(transactions.info(), "\n")


# customers.info(),products.info(),transactions.info()
```

```
Missing Values:

Customers:
 CustomerID      0
CustomerName    0
Region          0
SignupDate      0
dtype: int64

Products:
 ProductID      0
ProductName    0
Category       0
Price          0
dtype: int64

Transactions:
 TransactionID      0
CustomerID         0
ProductID          0
TransactionDate    0
Quantity           0
TotalValue         0
Price              0
dtype: int64

Cleaned Data:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   CustomerID    200 non-null    object
 1   CustomerName  200 non-null    object
 2   Region        200 non-null    object
 3   SignupDate    200 non-null    datetime64[ns]
dtypes: datetime64[ns](1), object(3)
memory usage: 6.4+ KB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   ProductID    100 non-null    object
 1   ProductName  100 non-null    object
 2   Category     100 non-null    object
 3   Price        100 non-null    float64
dtypes: float64(1), object(3)
memory usage: 3.2+ KB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   TransactionID    1000 non-null   object
 1   CustomerID       1000 non-null   object
```

```
 2   ProductID        1000 non-null   object
 3   TransactionDate  1000 non-null   datetime64[ns]
 4   Quantity         1000 non-null   int64
 5   TotalValue       1000 non-null   float64
 6   Price            1000 non-null   float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(3)
memory usage: 54.8+ KB
None
```

4. Data Integration

- Merged the three datasets into a single unified dataset using the following keys:
  - **Customers** merged with **Transactions** using `CustomerID`.
  - **Products** merged with the result using `ProductID`.
- Renamed columns and removed redundant ones (e.g., duplicate `Price` columns after merging).

In [11]:

```
# Merge datasets
merged_data = transactions.merge(customers, on='CustomerID', how='left')
merged_data = merged_data.merge(products, on='ProductID', how='left')

# Display the merged dataset
# print("Merged Data Sample:\n", merged_data.head())


merged_data.head()
```

Out[11]:

	TransactionID	CustomerID	ProductID	TransactionDate	Quantity	TotalValue	Price_x
0	T00001	C0199	P067	2024-08-25 12:38:23	1	300.68	300.68
1	T00112	C0146	P067	2024-05-27 22:23:54	1	300.68	300.68
2	T00166	C0127	P067	2024-04-25 07:38:55	1	300.68	300.68
3	T00272	C0087	P067	2024-03-26 22:55:37	2	601.36	300.68
4	T00363	C0070	P067	2024-03-21 15:10:10	3	902.04	300.68

In [12]:

```
merged_data['Price_x']==merged_data['Price_y']
```

```
Out[12]: 0       True
         1       True
         2       True
         3       True
         4       True
                 ...
         995     True
         996     True
         997     True
         998     True
         999     True
         Length: 1000, dtype: bool
```

```
In [13]: merged_data.drop(columns="Price_x",inplace=True)
         merged_data.rename(columns={"Price_y":"Price"},inplace=True)
```

```
In [14]: !mkdir Desktop/tushar
```

```
mkdir: Desktop/tushar: File exists
```

```
In [15]: merged_data.to_csv(open("Desktop/tushar/final.csv",'wb'))
```

5. Exploratory Data Analysis

A. Descriptive Analysis

- **Customer Distribution by Region**:
  - Created a bar chart showing the number of customers in each region.
- **Signup Trends Over Time**:
  - Plotted customer signup trends over the years using a line chart.

B. Product Analysis

- **Sales by Product Category**:
  - Plotted a horizontal bar chart of total sales by category.
- **Top and Bottom Performing Products**:
  - Identified top 5 and bottom 5 products based on total sales and visualized using bar charts.

C. Customer Segmentation

- **Spending Segments**:
  - Segmented customers into:
    - Low Spenders, Medium Spenders, High Spenders, Premium Spenders.
  - Visualized segment distribution using a bar chart.
- **Churn Analysis**:
  - Defined churned customers as those inactive for more than 90 days.
  - Analyzed churn distribution using a pie chart.

D. Temporal Analysis

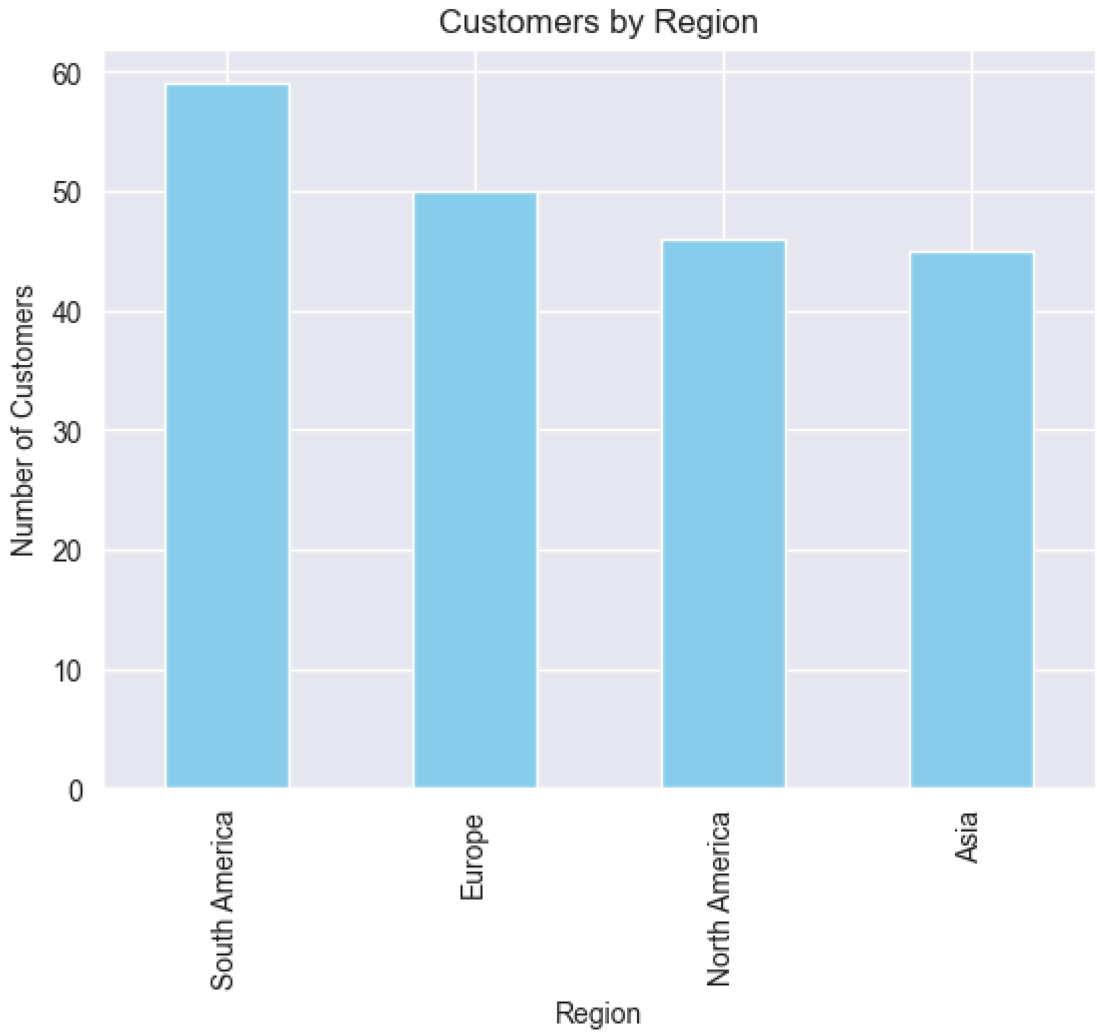
- **Peak Transaction Times**:
  - Analyzed transactions by:
    - Hour of the day (line chart).
    - Day of the week (bar chart).
    - Month (bar chart).
- **Monthly Sales Trend**:
  - Created a line chart to show total sales over months.

E. Transaction Insights

- **Average Transaction Value**:
  - Calculated the average transaction value as **$690.00**.
  - Visualized transaction value distribution using a histogram.

Customers by region

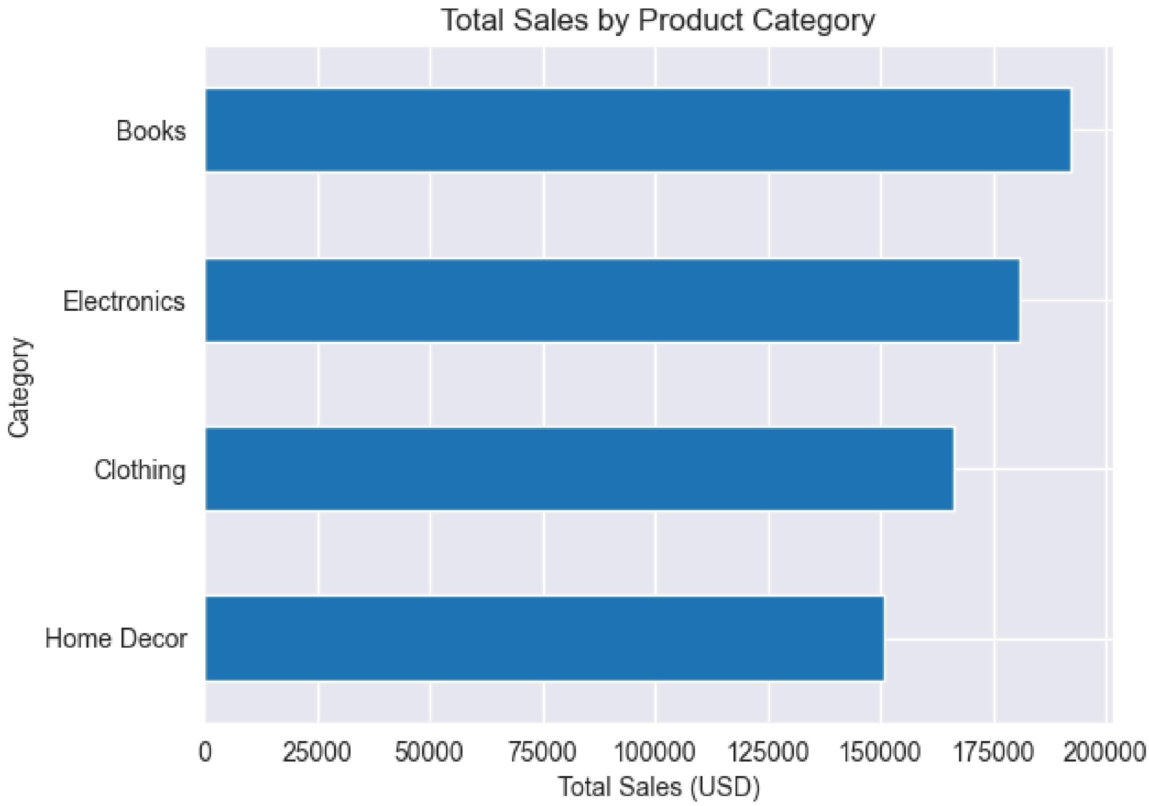
In [16]:
```
# Customers by region
region_counts = customers['Region'].value_counts()
region_counts.plot(kind='bar', color='skyblue', title='Customers by Region')
plt.xlabel('Region')
plt.ylabel('Number of Customers')
plt.show()

# Customer signups over time

customers['SignupDate'].dt.year.value_counts().sort_index().plot(kind='line', ma
plt.xlabel('Year')
plt.ylabel('Number of Signups')
plt.show()
```


Customers by Region
Number of Customers
60
50
40
30
20
10
0
South America
Europe
North America
Asia
Region


Customer Signups Over Time
Number of Signups
80
75
70
65
60
2022.00
2022.25
2022.50
2022.75
2023.00
2023.25
2023.50
2023.75
2024.00
Year

Top Selling Categories

In [18]:
```
category_sales = merged_data.groupby('Category')['TotalValue'].sum().sort_values
category_sales.plot(kind='barh', title='Total Sales by Product Category')
plt.ylabel('Category')
plt.xlabel('Total Sales (USD)')
plt.show()
```



In [21]:
```
merged_data.head()
```

Out[21]:

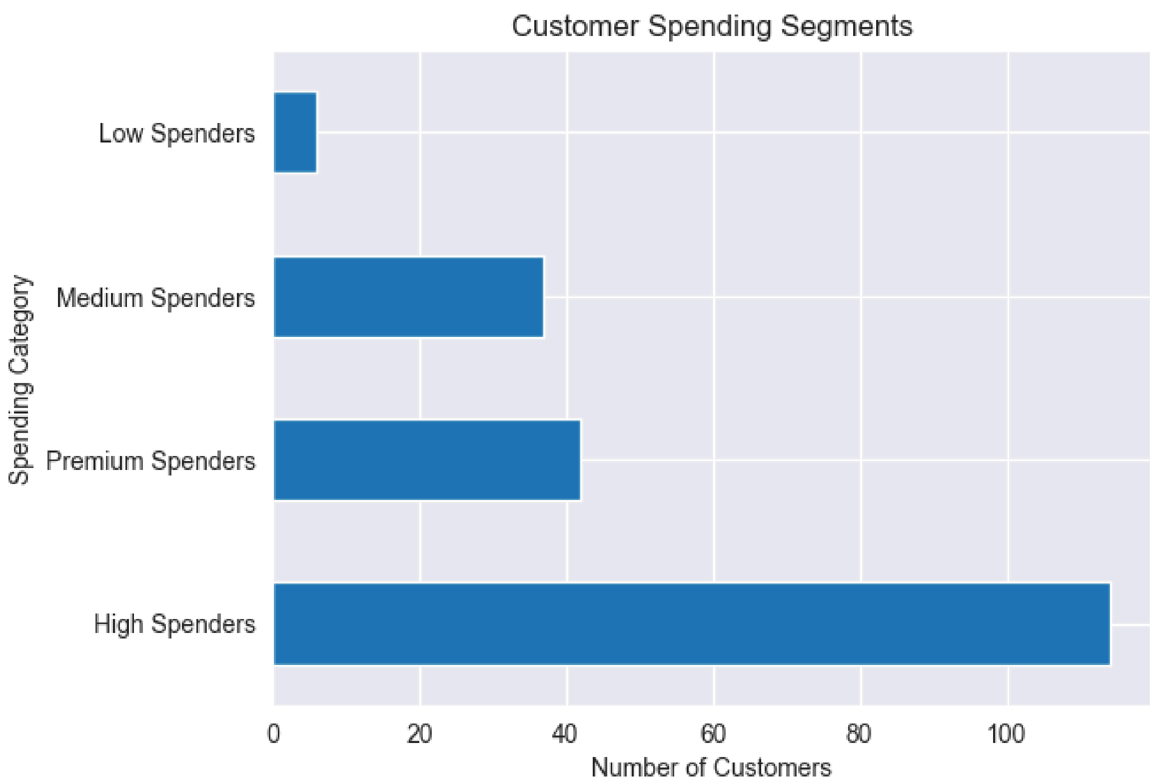
	TransactionID	CustomerID	ProductID	TransactionDate	Quantity	TotalValue	Custor
0	T00001	C0199	P067	2024-08-25 12:38:23	1	300.68	Andre
1	T00112	C0146	P067	2024-05-27 22:23:54	1	300.68	Britta
2	T00166	C0127	P067	2024-04-25 07:38:55	1	300.68	Kathry
3	T00272	C0087	P067	2024-03-26 22:55:37	2	601.36	Travis
4	T00363	C0070	P067	2024-03-21 15:10:10	3	902.04	Timo

In [22]:
```
# Calculate total spending and transaction count for each customer
customer_summary = merged_data.groupby('CustomerID').agg({
    'TotalValue': 'sum',  # Total spending
    'TransactionID': 'count'  # Number of transactions
}).rename(columns={'TotalValue': 'TotalSpending', 'TransactionID': 'TransactionC

# Segmentation based on spending levels
customer_summary['SpendingCategory'] = pd.cut(
    customer_summary['TotalSpending'],
    bins=[0, 500, 2000, 5000, float('inf')],
    labels=['Low Spenders', 'Medium Spenders', 'High Spenders', 'Premium Spender
)

# View top customers
customer_summary.head()
```

Out[22]:

CustomerID	TotalSpending	TransactionCount	SpendingCategory
C0001	3354.52	5	High Spenders
C0002	1862.74	4	Medium Spenders
C0003	2725.38	4	High Spenders
C0004	5354.88	8	Premium Spenders
C0005	2034.24	3	High Spenders

In [23]:
```
# Plot the spending categories
customer_summary['SpendingCategory'].value_counts().plot(kind='barh', title='Cus
plt.ylabel('Spending Category')
plt.xlabel('Number of Customers')
plt.show()
```



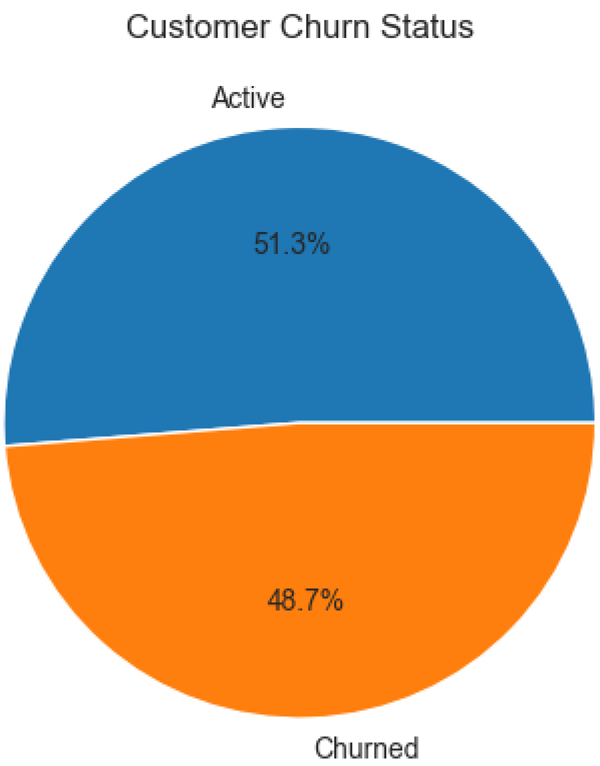
As we can see there are some customer's which spends more

In [24]:
```
# Calculate the most recent transaction date for each customer
last_transaction = merged_data.groupby('CustomerID')['TransactionDate'].max()

# Define the reference date (e.g., today's date)
import datetime
reference_date = datetime.datetime.now()

# Calculate days since last transaction
last_transaction = last_transaction.reset_index()
last_transaction['DaysSinceLastPurchase'] = (reference_date - last_transaction['

# Define churned customers as those who haven't purchased in the last 90 days (a
last_transaction['ChurnStatus'] = last_transaction['DaysSinceLastPurchase'].appl
    lambda x: 'Churned' if x > 90 else 'Active'
)

# View churn analysis
last_transaction.head()
```

Out[24]:

	CustomerID	TransactionDate	DaysSinceLastPurchase	ChurnStatus
0	C0001	2024-11-02 17:04:16	81	Active
1	C0002	2024-12-03 01:41:41	51	Active
2	C0003	2024-08-24 18:54:04	151	Churned
3	C0004	2024-12-23 14:13:52	31	Active
4	C0005	2024-11-04 00:30:22	80	Active

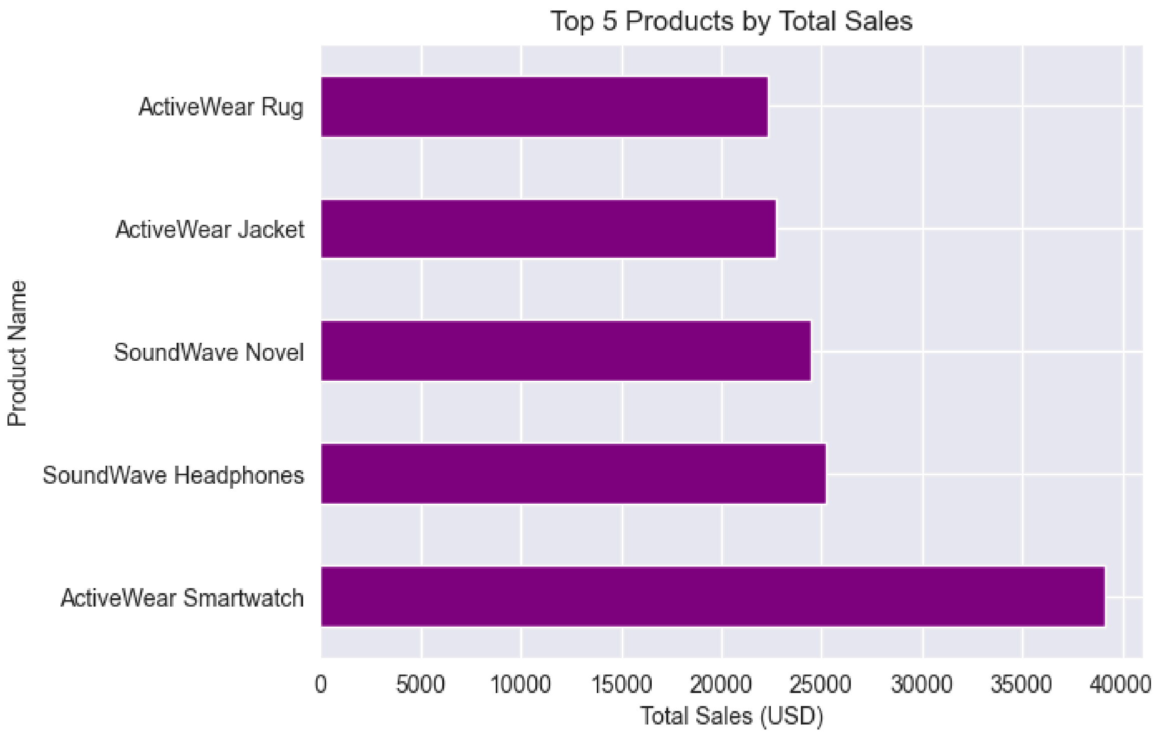
In [25]:
```
# Plot the distribution of churned vs. active customers
last_transaction['ChurnStatus'].value_counts().plot(kind='pie', autopct='%1.1f%%
                                                     title='Customer Churn Status
plt.ylabel('')
plt.show()
```



Customers are declining

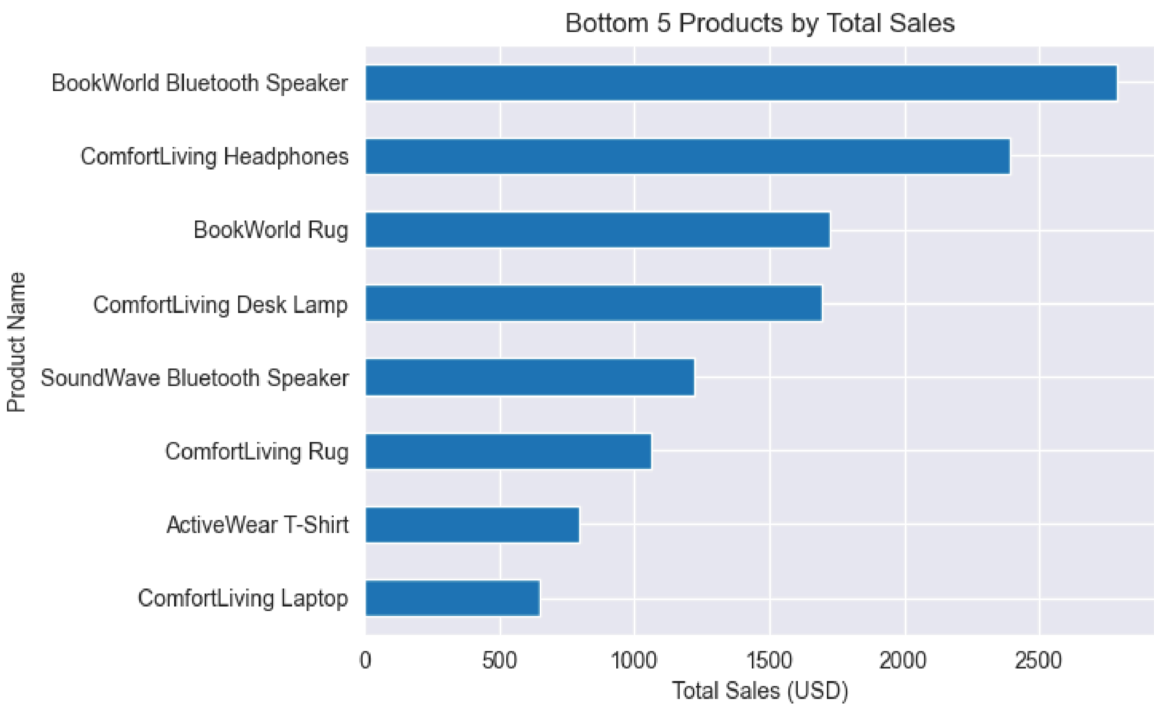
Top Performing products

In [26]:
```
# Top 5 products by total sales
top_products = merged_data.groupby('ProductName')['TotalValue'].sum().sort_value

# Plot top 5 products
top_products.plot(kind='barh', color='purple', title='Top 5 Products by Total Sa
plt.ylabel('Product Name')
plt.xlabel('Total Sales (USD)')
plt.show()
```



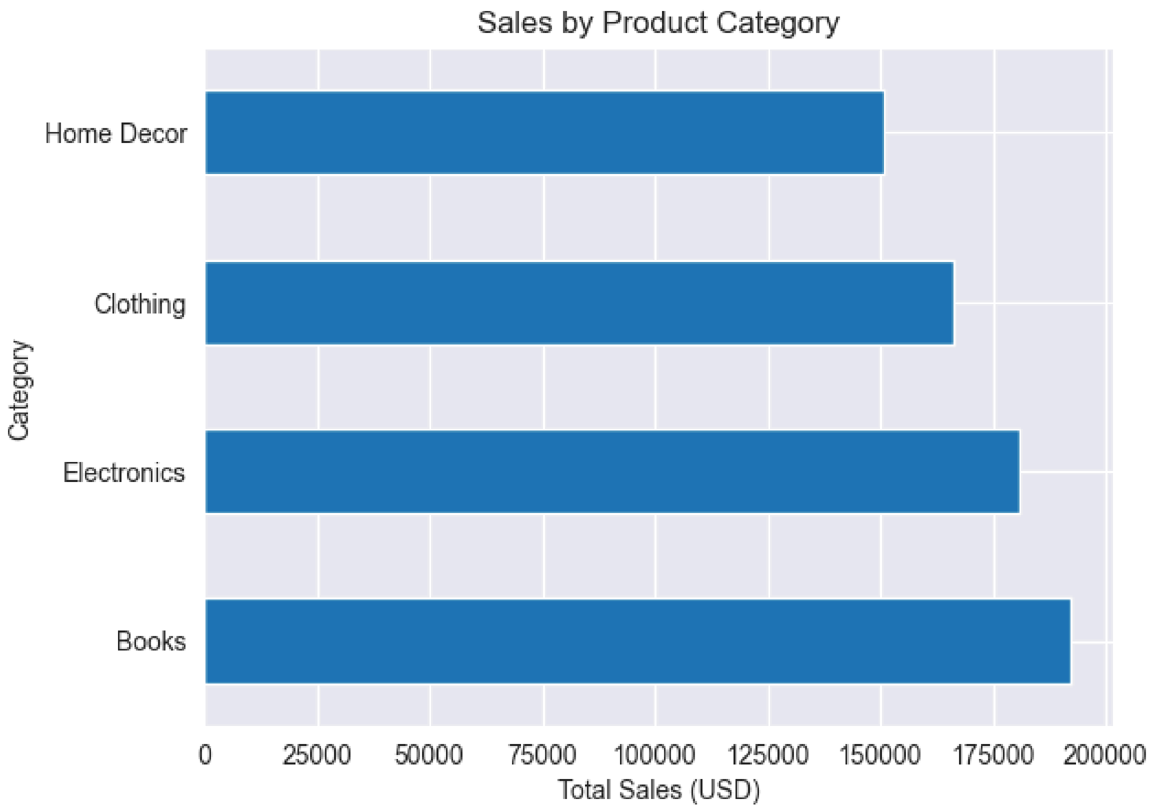
Low Performing Products

In [27]:
```
# Bottom 5 products by total sales
low_products = merged_data.groupby('ProductName')['TotalValue'].sum().sort_value

# Plot bottom 5 products
low_products.plot(kind='barh', title='Bottom 5 Products by Total Sales')
plt.ylabel('Product Name')
plt.xlabel('Total Sales (USD)')
plt.show()
```



Category Analysis

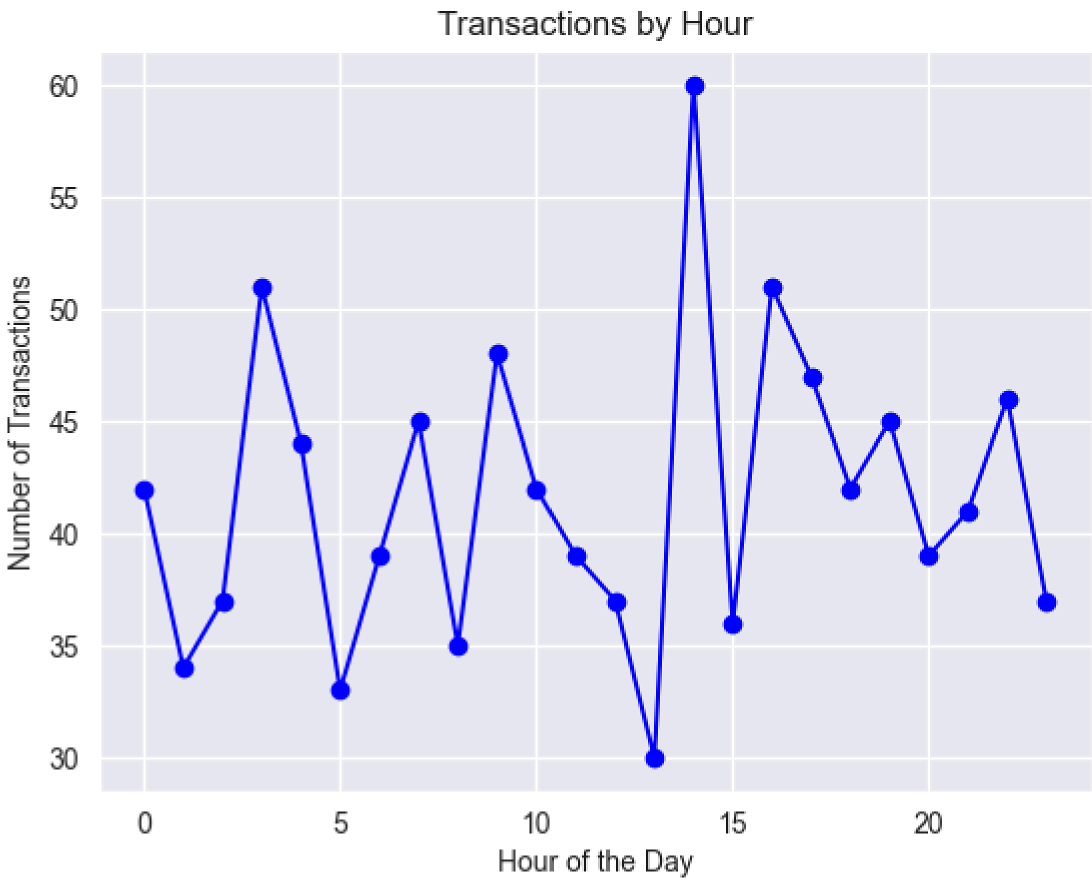
In [28]:
```
# Analyze sales by product category
category_sales = merged_data.groupby('Category')['TotalValue'].sum().sort_values

# Plot sales by category
category_sales.plot(kind='barh', title='Sales by Product Category')
plt.ylabel('Category')
plt.xlabel('Total Sales (USD)')
plt.show()
```

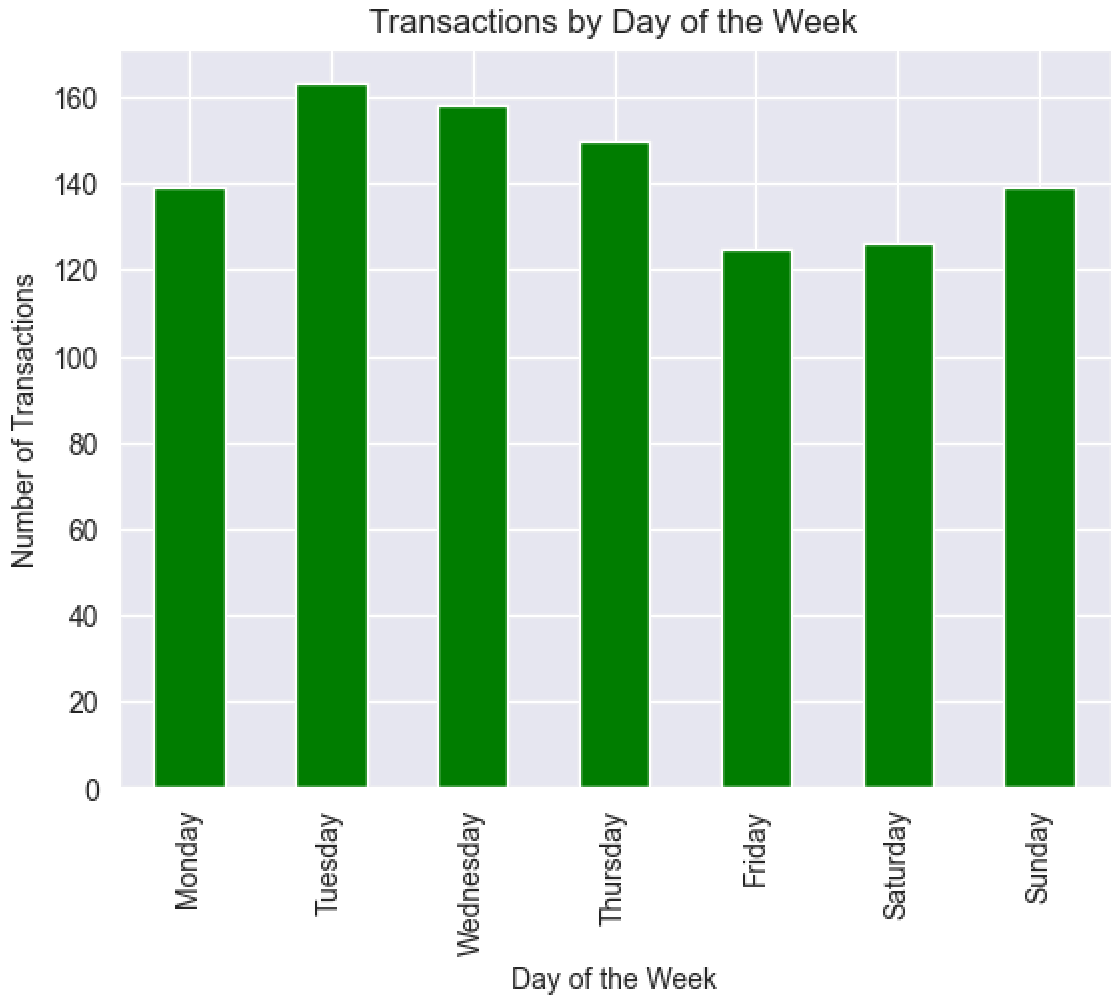


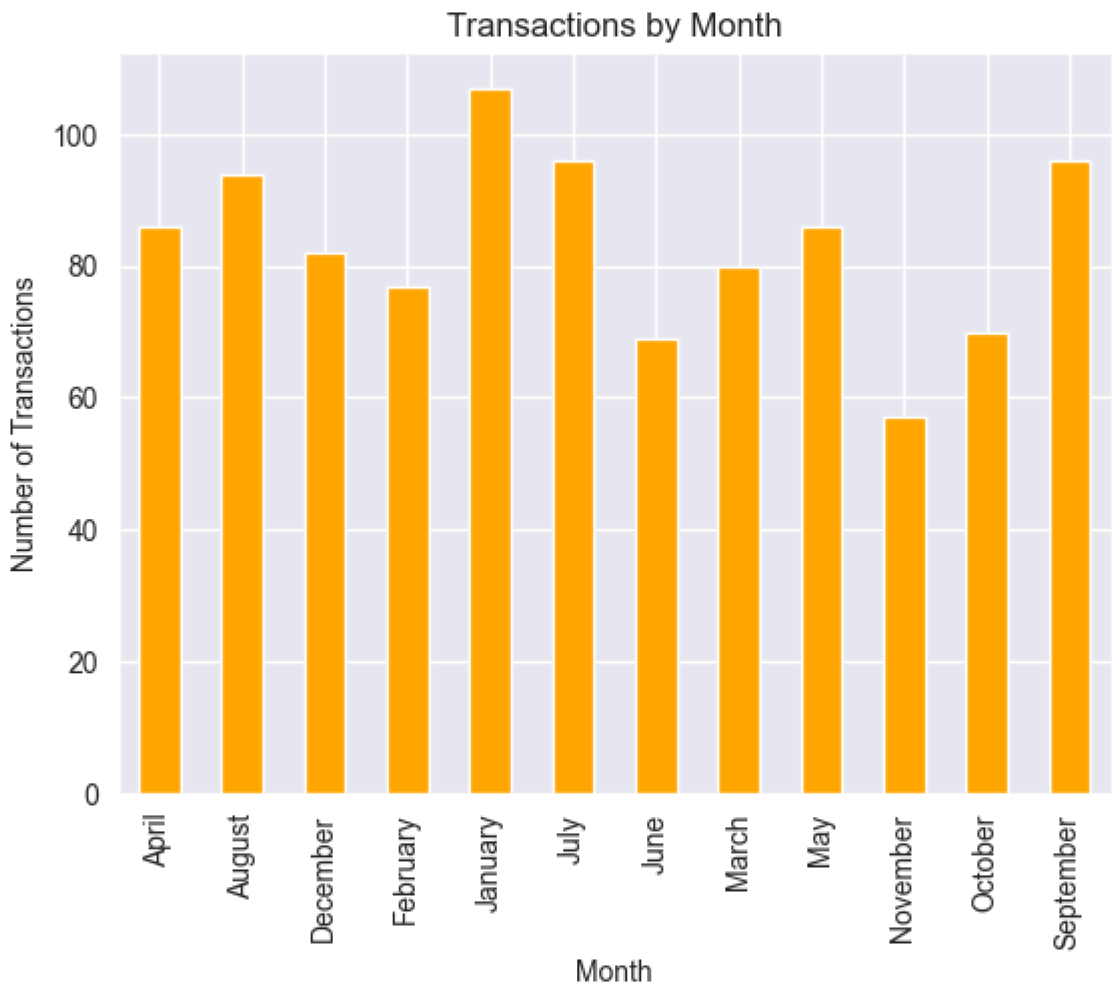
Peak Time

In [29]:
```
# Extract date-related features
merged_data['TransactionDate'] = pd.to_datetime(merged_data['TransactionDate'])
merged_data['Hour'] = merged_data['TransactionDate'].dt.hour
merged_data['Day'] = merged_data['TransactionDate'].dt.day_name()
merged_data['Month'] = merged_data['TransactionDate'].dt.month_name()

# Transactions by hour
hourly_transactions = merged_data.groupby('Hour')['TransactionID'].count()

# Plot hourly transactions
hourly_transactions.plot(kind='line', marker='o', color='blue', title='Transacti
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Transactions')
plt.grid(True)
plt.show()

# Transactions by day of the week
```

```
daily_transactions = merged_data.groupby('Day')['TransactionID'].count().reindex
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday
)

# Plot daily transactions
daily_transactions.plot(kind='bar', color='green', title='Transactions by Day of
plt.xlabel('Day of the Week')
plt.ylabel('Number of Transactions')
plt.show()

# Transactions by month
monthly_transactions = merged_data.groupby('Month')['TransactionID'].count()

# Plot monthly transactions
monthly_transactions.plot(kind='bar', color='orange', title='Transactions by Mon
plt.xlabel('Month')
plt.ylabel('Number of Transactions')
plt.show()
```



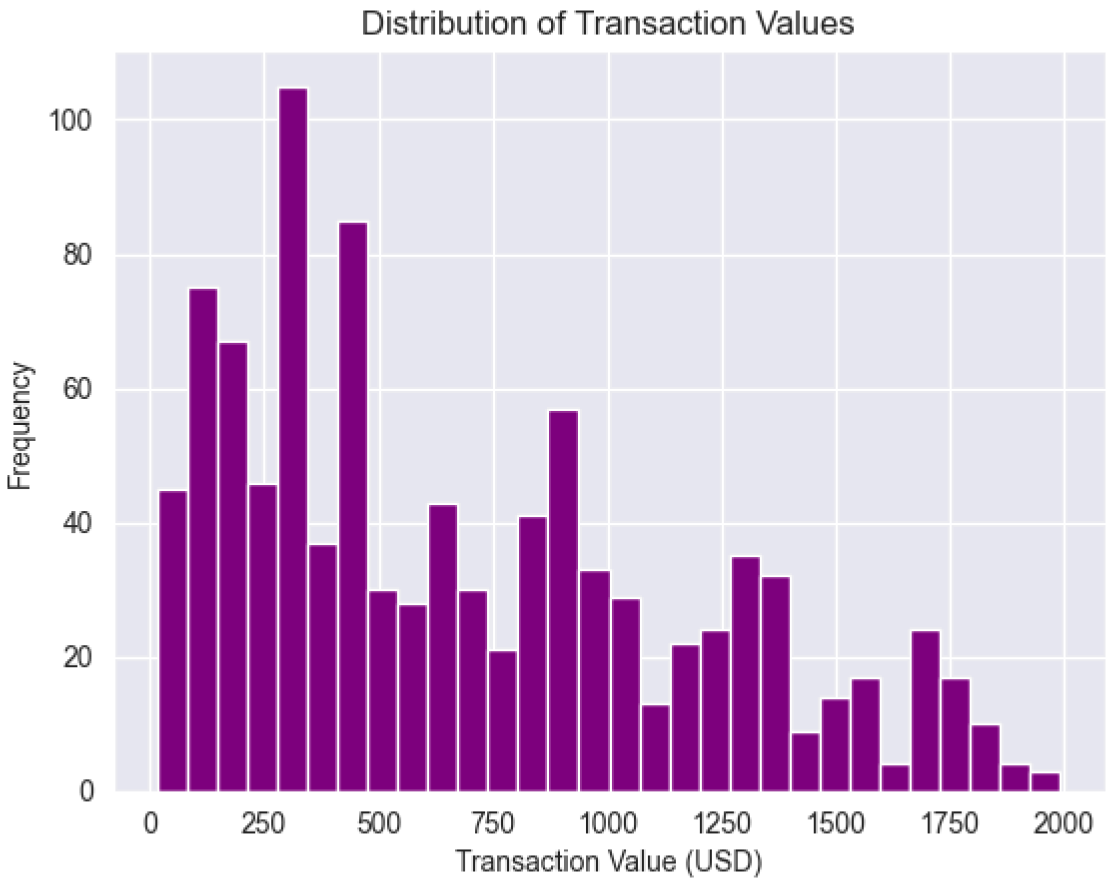

Transactions by Day of the Week
Number of Transactions
0
20
40
60
80
100
120
140
160
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day of the Week


Transactions by Month
Number of Transactions
0
20
40
60
80
100
April
August
December
February
January
July
June
March
May
November
October
September
Month

Average Transection Value

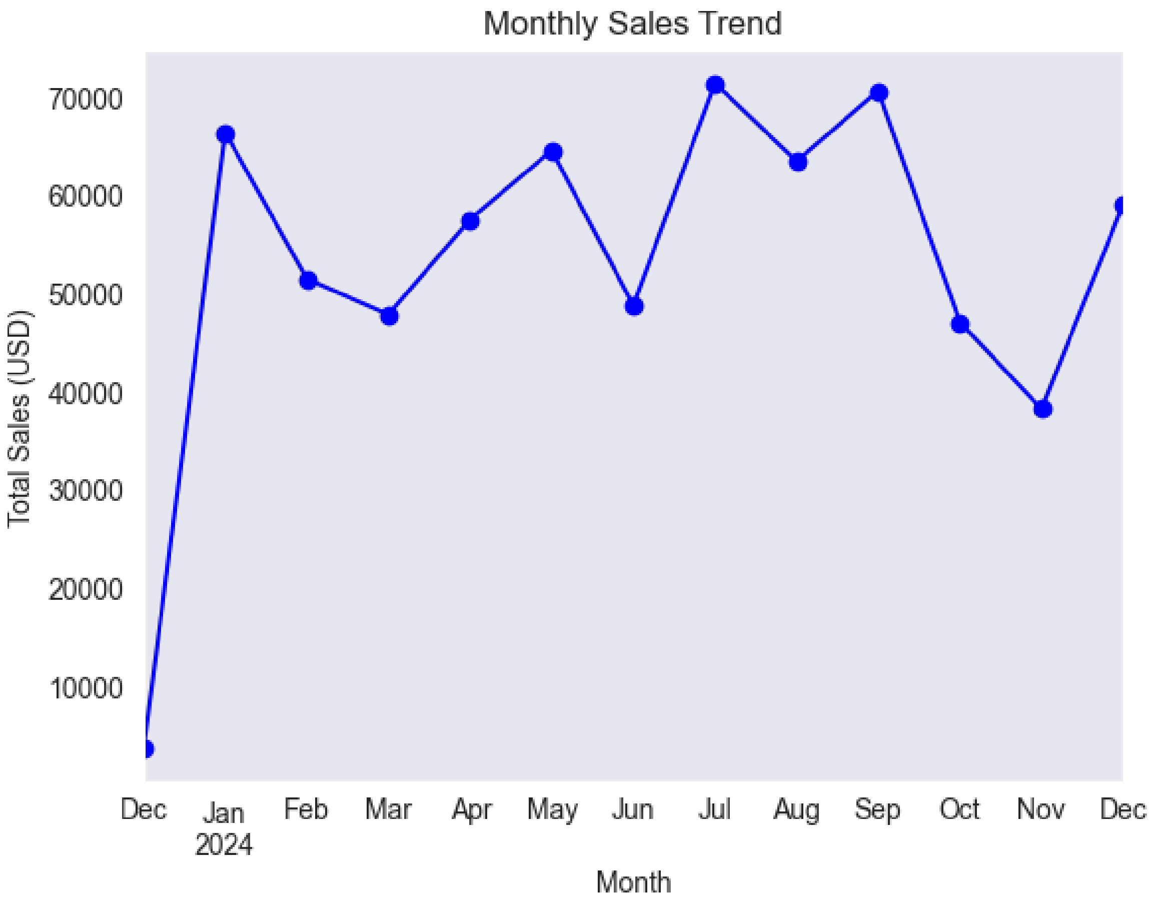
In [30]:
```
# Calculate total and average transaction value
transaction_summary = merged_data.groupby('TransactionID')['TotalValue'].sum()
average_transaction_value = transaction_summary.mean()

print(f"Average Transaction Value: $ {average_transaction_value:.2f}")

# Plot the distribution of transaction values
transaction_summary.plot(kind='hist', bins=30, color='purple', title='Distributi
plt.xlabel('Transaction Value (USD)')
plt.ylabel('Frequency')
plt.show()
```

```
Average Transaction Value: $ 690.00
```



Monthly Sales

In [31]:
```
# Group data by month and calculate total revenue
monthly_sales = merged_data.groupby(merged_data['TransactionDate'].dt.to_period(
monthly_sales.index = monthly_sales.index.to_timestamp()  # Convert to timestamp

# Plot monthly sales trend
monthly_sales.plot(kind='line', marker='o', color='blue', title='Monthly Sales T
plt.xlabel('Month')
plt.ylabel('Total Sales (USD)')
plt.grid()
plt.show()
```



6. Summary of Insights

- **Customer Behavior**:
  - High spenders dominate specific regions, while churn rates indicate declining engagement in certain segments.
- **Product Insights**:
  - Electronics outperform other categories; some low-performing products may require promotions.
- **Sales Trends**:
  - Transactions peak during specific times and days, offering opportunities for targeted marketing.
- **Retention Opportunities**:
  - Re-engagement campaigns are crucial to activate churned customers.

7. Deliverables

- Final cleaned and merged dataset saved as `final.csv`.
- Plots and visualizations for the following:
  - Customer distribution by region.
  - Signup trends over time.
  - Sales trends by category and product.
  - Customer segmentation and churn analysis.
  - Temporal sales trends (hourly, daily, monthly).

- Transaction value distribution.

Conclusion

This EDA project highlights actionable insights to improve customer retention, optimize product performance, and maximize sales. These findings can guide data-driven decisions for enhanced business outcomes.

In [ ]:

```

```