Capstone Project (Retail Domain)

Project Name: Retail_Sales_Insights
Prepared by: Bishowjith Ghosh
Date: September 2024
Master's Program in Data Science

In [1]:
```
# Importing required libraries for data cleaning and transformation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Load the dataset

In [2]:
```
file_path = "retail_data.csv"
df = pd.read_csv(file_path)

# Display the first few rows of the dataset to understand its structure
df.head()
```

Out[2]:

	Transaction_ID	Customer_ID	Name	Email	Phone	Address	City	State	Zipcode	Country	...	Total_Amount	Product_Category	Product_Brand	Produc
0	8691788.0	37249.0	Michelle Harrington	Ebony39@gmail.com	1.414787e+09	3959 Amanda Burgs	Dortmund	Berlin	77985.0	Germany	...	324.086270	Clothing	Nike	
1	2174773.0	69749.0	Kelsey Hill	Mark36@gmail.com	6.852900e+09	82072 Dawn Centers	Nottingham	England	99071.0	UK	...	806.707815	Electronics	Samsung	
2	6679610.0	30192.0	Scott Jensen	Shane85@gmail.com	8.362160e+09	4133 Young Canyon	Geelong	New South Wales	75929.0	Australia	...	1063.432799	Books	Penguin Books	Ch
3	7232460.0	62101.0	Joseph Miller	Mary34@gmail.com	2.776752e+09	8148 Thomas Creek Suite 100	Edmonton	Ontario	88420.0	Canada	...	2466.854021	Home Decor	Home Depot	
4	4983775.0	27901.0	Debra Coleman	Charles30@gmail.com	9.098268e+09	5813 Lori Ports Suite 269	Bristol	England	48704.0	UK	...	248.553049	Grocery	Nestle	Ch

5 rows × 30 columns

In [3]:
```
df.columns
```

Out[3]:
```
Index(['Transaction_ID', 'Customer_ID', 'Name', 'Email', 'Phone', 'Address',
       'City', 'State', 'Zipcode', 'Country', 'Age', 'Gender', 'Income',
       'Customer_Segment', 'Date', 'Year', 'Month', 'Time', 'Total_Purchases',
       'Amount', 'Total_Amount', 'Product_Category', 'Product_Brand',
       'Product_Type', 'Feedback', 'Shipping_Method', 'Payment_Method',
       'Order_Status', 'Ratings', 'products'],
      dtype='object')
```

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 302010 entries, 0 to 302009
Data columns (total 30 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   Transaction_ID    301677 non-null  float64
 1   Customer_ID       301702 non-null  float64
 2   Name              301628 non-null  object
 3   Email             301663 non-null  object
 4   Phone             301648 non-null  float64
 5   Address           301695 non-null  object
 6   City              301762 non-null  object
 7   State             301729 non-null  object
 8   Zipcode           301670 non-null  float64
 9   Country           301739 non-null  object
 10  Age               301837 non-null  float64
 11  Gender            301693 non-null  object
 12  Income            301720 non-null  object
 13  Customer_Segment  301795 non-null  object
 14  Date              301651 non-null  object
 15  Year              301660 non-null  float64
 16  Month             301737 non-null  object
 17  Time              301660 non-null  object
 18  Total_Purchases   301649 non-null  float64
 19  Amount            301653 non-null  float64
 20  Total_Amount      301660 non-null  float64
 21  Product_Category  301727 non-null  object
 22  Product_Brand     301729 non-null  object
 23  Product_Type      302010 non-null  object
 24  Feedback          301826 non-null  object
 25  Shipping_Method   301673 non-null  object
 26  Payment_Method    301713 non-null  object
 27  Order_Status      301775 non-null  object
 28  Ratings           301826 non-null  float64
 29  products          302010 non-null  object
dtypes: float64(10), object(20)
memory usage: 69.1+ MB
```

In [5]:
```
df.describe()
```

Out[5]:

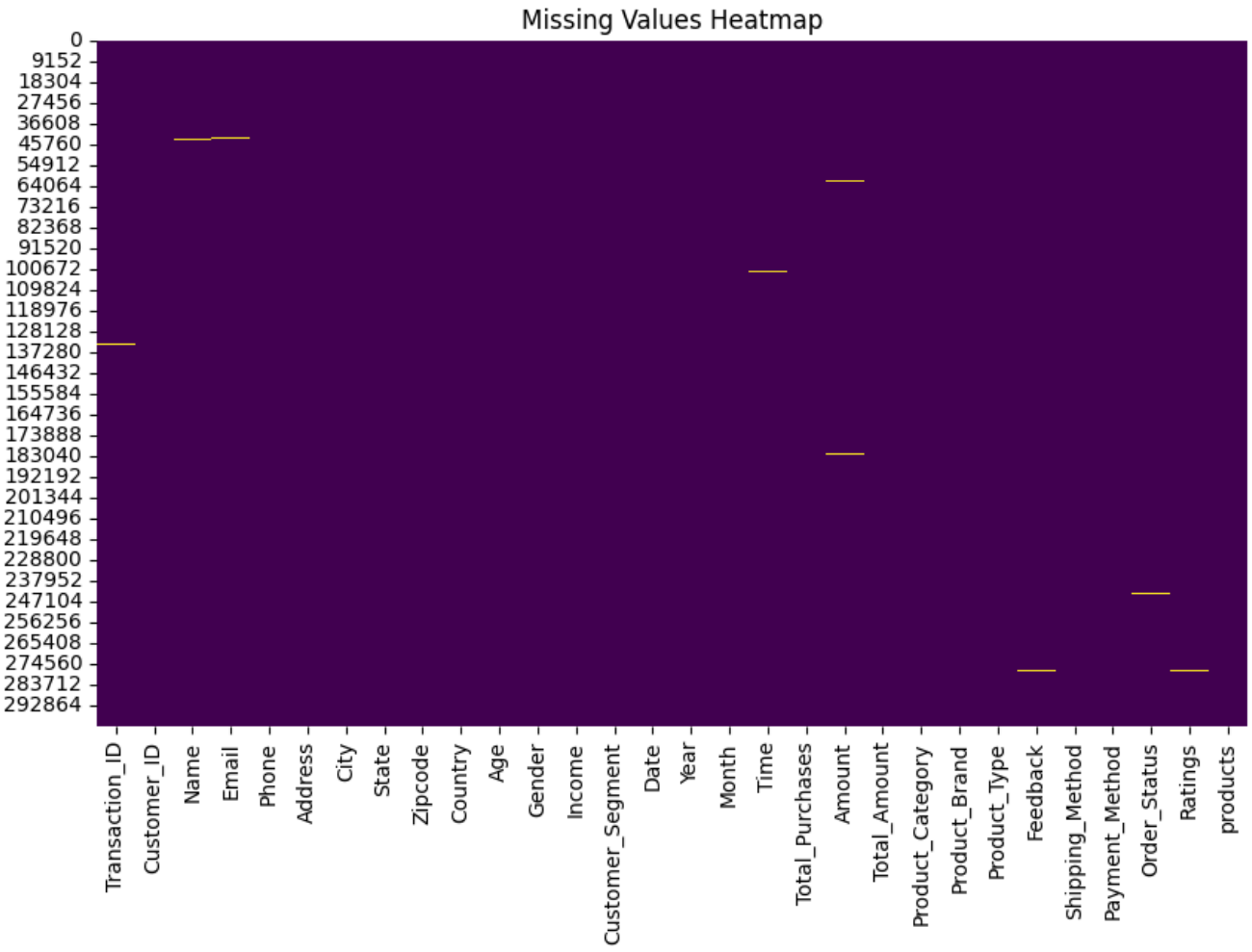
	Transaction_ID	Customer_ID	Phone	Zipcode	Age	Year	Total_Purchases	Amount	Total_Amount	Ratings
count	3.016770e+05	301702.000000	3.016480e+05	301670.000000	301837.000000	301660.000000	301649.000000	301653.000000	301660.000000	301826.000000
mean	5.495823e+06	55006.553934	5.501464e+09	50298.951019	35.481326	2023.165113	5.359729	255.163659	1367.651156	3.162670
std	2.595565e+06	26005.675200	2.596017e+09	28972.807134	15.021933	0.371283	2.868575	141.389640	1128.998515	1.320827
min	1.000007e+06	10000.000000	1.000049e+09	501.000000	18.000000	2023.000000	1.000000	10.000219	10.003750	1.000000
25%	3.247930e+06	32469.250000	3.255061e+09	25425.000000	22.000000	2023.000000	3.000000	132.890764	438.724278	2.000000
50%	5.499657e+06	55012.000000	5.505812e+09	50602.500000	32.000000	2023.000000	5.000000	255.470969	1041.117547	3.000000
75%	7.739509e+06	77511.000000	7.749860e+09	75252.000000	46.000000	2023.000000	8.000000	377.672606	2029.999853	4.000000
max	9.999995e+06	99999.000000	9.999996e+09	99949.000000	70.000000	2024.000000	10.000000	499.997911	4999.625796	5.000000

Check for missing values in the dataset and print the summary

In [6]:
```
missing_value = df.isnull().sum()
print("Missing values: \n", missing_value)
```

```
Missing values:
 Transaction_ID      333
Customer_ID         308
Name                382
Email               347
Phone               362
Address             315
City                248
State               281
Zipcode             340
Country             271
Age                 173
Gender              317
Income              290
Customer_Segment    215
Date                359
Year                350
Month               273
Time                350
Total_Purchases     361
Amount              357
Total_Amount        350
Product_Category    283
Product_Brand       281
Product_Type          0
Feedback            184
Shipping_Method     337
Payment_Method      297
Order_Status        235
Ratings             184
products              0
dtype: int64
```

In [7]:
```
# Visualize missing data with a heatmap to understand the distribution
plt.figure(figsize=(10,6))
sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
plt.title('Missing Values Heatmap')
plt.show()
```



Handling Missing Values

In [8]:
```
# Handle missing values for numerical and categorical columns

# Numerical Columns:
# Fill missing values with the median since it’s less sensitive to outliers than the mean.
numerical_columns = df.select_dtypes(include=[np.number]).columns

for col in numerical_columns:
    if df[col].isnull().sum() > 0:  # Check if the column has missing values
        median_value = df[col].median()
        df[col].fillna(median_value, inplace=True)

# Categorical Columns:
# Fill missing values with a placeholder, such as 'Unknown'.
categorical_columns = df.select_dtypes(include=['object']).columns
```

```
for col in categorical_columns:
    if df[col].isnull().sum() > 0:  # Check if the column has missing values
        df[col].fillna('Unknown', inplace=True)

# Recheck for missing values after filling
missing_values_after_fill = df.isnull().sum()
print("Missing Values After Filling:\n", missing_values_after_fill)
```

```
Missing Values After Filling:
 Transaction_ID      0
Customer_ID          0
Name                 0
Email                0
Phone                0
Address              0
City                 0
State                0
Zipcode              0
Country              0
Age                  0
Gender               0
Income               0
Customer_Segment     0
Date                 0
Year                 0
Month                0
Time                 0
Total_Purchases      0
Amount               0
Total_Amount         0
Product_Category     0
Product_Brand        0
Product_Type         0
Feedback             0
Shipping_Method      0
Payment_Method       0
Order_Status         0
Ratings              0
products             0
dtype: int64
```

Handle duplicate

In [9]:
```
# Check for duplicates in the dataset
duplicate_rows = df[df.duplicated()]
print("Number of duplicate rows:", len(duplicate_rows))
```

```
Number of duplicate rows: 4
```

In [10]:
```
# Remove duplicates
df.drop_duplicates(inplace=True)
```

In [11]:
```
# Confirm that duplicates are removed
print("Number of rows after removing duplicates:", len(df))
```

```
Number of rows after removing duplicates: 302006
```

Clean String Data

In [12]:
```
# Clean unnecessary spaces and special characters, but retain '@'
string_columns = df.select_dtypes(include=['object']).columns

# Strip leading/trailing spaces and remove special characters except '@'
df[string_columns] = df[string_columns].apply(lambda x: x.str.strip().str.replace('[^a-zA-Z0-9@\s]', '', regex=True))

# Display a sample of cleaned string data
df[string_columns].head()
```

Out[12]:

	Name	Email	Address	City	State	Country	Gender	Income	Customer_Segment	Date	Month	Time	Product_Category	Product_Brand	Product_Ty
0	Michelle Harrington	Ebony39@gmailcom	3959 Amanda Burgs	Dortmund	Berlin	Germany	Male	Low	Regular	9182023	September	220355	Clothing	Nike	Sho
1	Kelsey Hill	Mark36@gmailcom	82072 Dawn Centers	Nottingham	England	UK	Female	Low	Premium	12312023	December	84204	Electronics	Samsung	Tab
2	Scott Jensen	Shane85@gmailcom	4133 Young Canyon	Geelong	New South Wales	Australia	Male	Low	Regular	4262023	April	40629	Books	Penguin Books	Childre
3	Joseph Miller	Mary34@gmailcom	8148 Thomas Creek Suite 100	Edmonton	Ontario	Canada	Male	High	Premium	050823	May	145517	Home Decor	Home Depot	To
4	Debra Coleman	Charles30@gmailcom	5813 Lori Ports Suite 269	Bristol	England	UK	Male	Low	Premium	011024	January	165407	Grocery	Nestle	Chocola

Clean Numeric Data

In [13]:
```
# Step 1: Remove all non-numeric characters from the 'Phone' column
df['Phone'] = df['Phone'].astype(str).str.replace('[^0-9]', '', regex=True)

# Step 2: Check the length of phone numbers and find unusual values
df['Phone_Length'] = df['Phone'].str.len()  # Create a new column for phone number lengths

# Step 3: Identify phone numbers that are too short or too long (e.g., less than 10 digits or more than 15 digits)
invalid_phones = df[(df['Phone_Length'] < 10) | (df['Phone_Length'] > 15)]

# Display rows with invalid phone numbers
print("Invalid phone numbers (less than 10 digits or more than 15 digits):")
invalid_phones[['Phone', 'Phone_Length']]
```

```
Invalid phone numbers (less than 10 digits or more than 15 digits):
```

Out[13]:

	Phone	Phone_Length

In [14]:
```
# Drop rows with invalid phone numbers (less than 10 digits or more than 15 digits)
df_cleaned_phones = df[~((df['Phone_Length'] < 10) | (df['Phone_Length'] > 15))]
```

```
# Check the size of the dataset after cleaning
df_cleaned_phones.shape
```

Out[14]: (302006, 31)

In [15]:
```
# Replace invalid phone numbers with NaN
df['Phone'] = np.where((df['Phone_Length'] < 10) | (df['Phone_Length'] > 15), np.nan, df['Phone'])

# Verify the changes
df[['Phone', 'Phone_Length']].head()
```

Out[15]:

	Phone	Phone_Length
0	14147868010	11
1	68528999870	11
2	83621604490	11
3	27767517240	11
4	90982676350	11

In [16]:
```
# Check for phone numbers that are too long
long_phone_numbers = df[df['Phone'].str.len() > 20]
print("Phone numbers that exceed 20 characters:")
print(long_phone_numbers)

# Check for any non-numeric characters
invalid_phone_numbers = df[df['Phone'].str.contains('[^0-9]', regex=True)]
print("Phone numbers with non-numeric characters:")
print(invalid_phone_numbers)
```

```
Phone numbers that exceed 20 characters:
Empty DataFrame
Columns: [Transaction_ID, Customer_ID, Name, Email, Phone, Address, City, State, Zipcode, Country, Age, Gender, Income, Customer_Segment, Date, Year, Month, Time, Total_Purcha
ses, Amount, Total_Amount, Product_Category, Product_Brand, Product_Type, Feedback, Shipping_Method, Payment_Method, Order_Status, Ratings, products, Phone_Length]
Index: []

[0 rows x 31 columns]
Phone numbers with non-numeric characters:
Empty DataFrame
Columns: [Transaction_ID, Customer_ID, Name, Email, Phone, Address, City, State, Zipcode, Country, Age, Gender, Income, Customer_Segment, Date, Year, Month, Time, Total_Purcha
ses, Amount, Total_Amount, Product_Category, Product_Brand, Product_Type, Feedback, Shipping_Method, Payment_Method, Order_Status, Ratings, products, Phone_Length]
Index: []

[0 rows x 31 columns]
```

In [17]:
```
# Check for non-numeric values in the 'Income' column
non_numeric_income = df[~df['Income'].str.replace('[\$,]', '', regex=True).str.isnumeric()]

# Display the non-numeric values
print("Non-numeric values in 'Income':")
non_numeric_income['Income'].unique()
```

```
Non-numeric values in 'Income':
```

Out[17]: array(['Low', 'High', 'Medium', 'Unknown'], dtype=object)

In [18]:
```
# Display a few rows of the Date and Time columns to investigate the issue
print(df[['Date', 'Time']].head(10))
```

```
      Date    Time
0   9182023  220355
1  12312023   84204
2   4262023   40629
3    050823  145517
4    011024  165407
5   9212023  232427
6   6262023  133551
7   3242023  101256
8    010624  143826
9    100423  222740
```

In [19]:
```
# Convert 'Date' to proper datetime format (MMDDYYYY)
df['Date'] = pd.to_datetime(df['Date'], format='%m%d%Y', errors='coerce')
```

In [20]:
```
# Convert 'Time' from HHMMSS to HH:MM:SS format
def convert_time(x):
    if pd.isna(x):
        return None  # Handle missing values
    try:
        # Ensure the time format is HHMMSS and convert to HH:MM:SS
        return pd.to_datetime(str(x).zfill(6), format='%H%M%S').time()
    except ValueError:
        return None  # Handle invalid time formats

# Apply the conversion function to the 'Time' column
df['Time'] = df['Time'].apply(convert_time)
```

In [21]:
```
# Display the first 10 rows to verify the changes
print(df[['Date', 'Time']].head(10))
# Display the first 10 rows to verify the changes
print(df[['Date', 'Time']].head(10))
```

```
        Date      Time
0 2023-09-18  22:03:55
1 2023-12-31  08:42:04
2 2023-04-26  04:06:29
3        NaT  14:55:17
4        NaT  16:54:07
5 2023-09-21  23:24:27
6 2023-06-26  13:35:51
7 2023-03-24  10:12:56
8        NaT  14:38:26
9        NaT  22:27:40
        Date      Time
0 2023-09-18  22:03:55
1 2023-12-31  08:42:04
2 2023-04-26  04:06:29
3        NaT  14:55:17
4        NaT  16:54:07
5 2023-09-21  23:24:27
6 2023-06-26  13:35:51
7 2023-03-24  10:12:56
8        NaT  14:38:26
9        NaT  22:27:40
```

In [22]:
```
# Display rows where the 'Date' column is NaT
missing_date_rows = df[df['Date'].isna()]
print(missing_date_rows[['Date', 'Time']])
```

```
       Date      Time
3       NaT  14:55:17
4       NaT  16:54:07
8       NaT  14:38:26
9       NaT  22:27:40
12      NaT  00:00:47
...     ...       ...
301997  NaT  08:30:10
302000  NaT  14:47:57
302002  NaT  17:37:41
302008  NaT  11:20:31
302009  NaT  11:44:36

[119113 rows x 2 columns]
```

In [23]:
```python
# Drop rows where the 'Date' column is NaT
df_cleaned = df.dropna(subset=['Date'])
```

In [24]:
```python
# Check if there are still any missing values in 'Date' and 'Time' columns
missing_dates = df['Date'].isna().sum()
missing_times = df['Time'].isna().sum()

print(f"Missing dates: {missing_dates}")
print(f"Missing times: {missing_times}")
```

```
Missing dates: 119113
Missing times: 350
```

In [25]:
```python
# Drop rows where 'Time' is missing
df = df.dropna(subset=['Time'])

# Check again for missing values
missing_times = df['Time'].isna().sum()
print(f"Missing times after dropping rows: {missing_times}")
```

```
Missing times after dropping rows: 0
```

Correct Data Types

In [26]:
```python
# Convert the 'Year' column from float to integer (handle any NaN values before conversion)
if df['Year'].isnull().sum() > 0:
    df['Year'].fillna(df['Year'].median(), inplace=True)  # Fill missing values if any before conversion
df['Year'] = df['Year'].astype('int64')

# Verify the conversion of 'Year'
print("Data type of 'Year' column after conversion:", df['Year'].dtype)

# Convert 'Customer_ID' to integer if it doesn't contain letters (assuming all values are numeric)
if df['Customer_ID'].isnull().sum() > 0:
    df['Customer_ID'].fillna(0, inplace=True)  # Fill NaN with 0 or another default value
df['Customer_ID'] = df['Customer_ID'].astype('int64')

# Convert 'Zipcode' to string
df['Zipcode'] = df['Zipcode'].astype('str')

# Verify the data types after conversion
print("Data types after all conversions:")
print(df.dtypes)
```

```
Data type of 'Year' column after conversion: int64
Data types after all conversions:
Transaction_ID             float64
Customer_ID                  int64
Name                        object
Email                       object
Phone                       object
Address                     object
City                        object
State                       object
Zipcode                     object
Country                     object
Age                        float64
Gender                      object
Income                      object
Customer_Segment            object
Date                datetime64[ns]
Year                         int64
Month                       object
Time                        object
Total_Purchases            float64
Amount                     float64
Total_Amount               float64
Product_Category            object
Product_Brand               object
Product_Type                object
Feedback                    object
Shipping_Method             object
Payment_Method              object
Order_Status                object
Ratings                    float64
products                    object
Phone_Length                 int64
dtype: object
```

Analyze the Total Sales per State (using pandas)

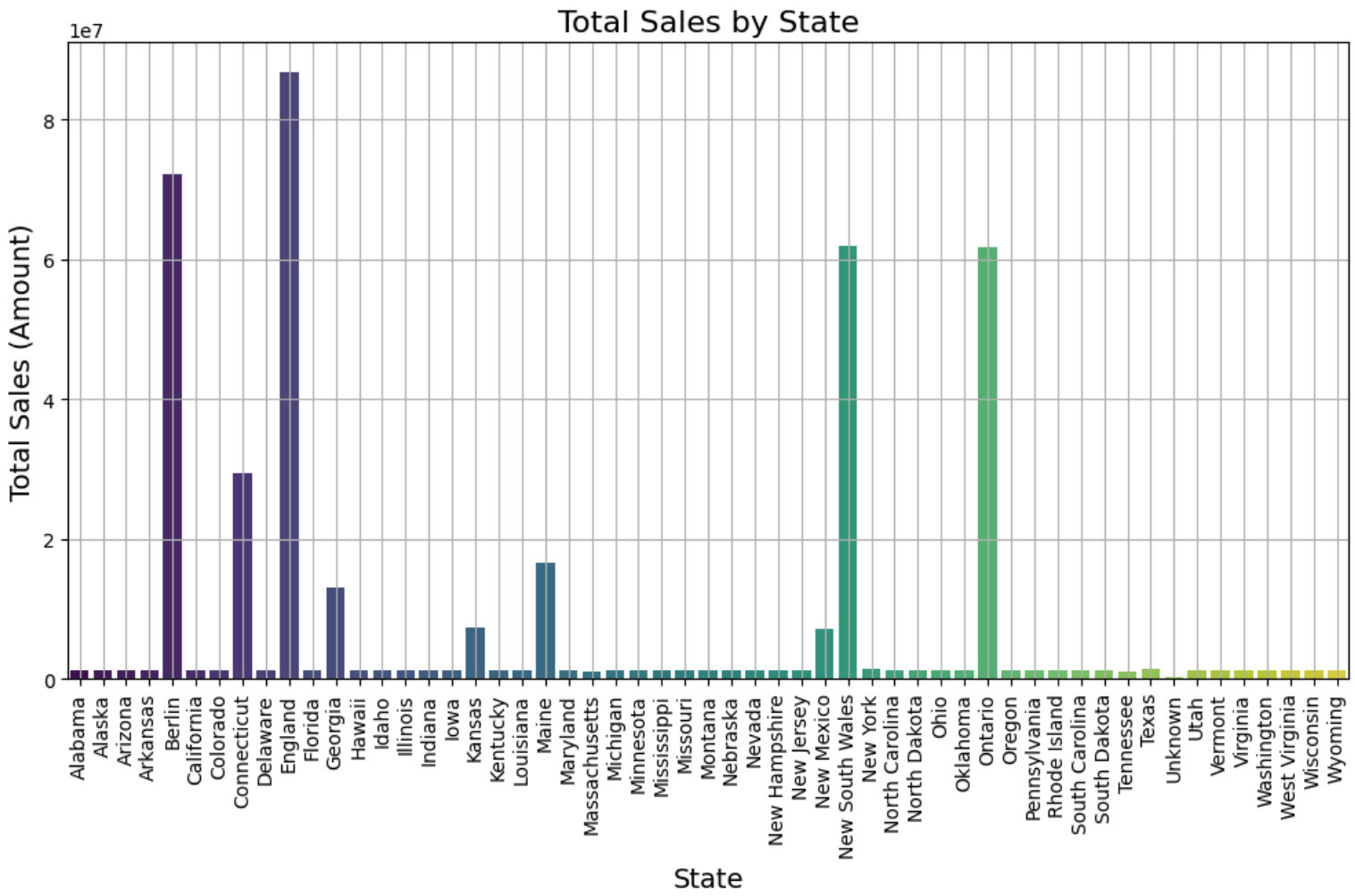
In [27]:
```python
# Group by 'State' and calculate total sales per state
state_sales = df.groupby('State')['Total_Amount'].sum().reset_index()

# Sort by total sales and display the top 5 states
top_states = state_sales.sort_values(by='Total_Amount', ascending=False).head(5)
print("Top 5 States by Total Sales:\n", top_states)
```

```
Top 5 States by Total Sales:
              State  Total_Amount
9           England  8.662934e+07
4            Berlin  7.210351e+07
33  New South Wales  6.198066e+07
39          Ontario  6.169588e+07
7       Connecticut  2.936984e+07
```

Visualize Total Sales by State (using seaborn)

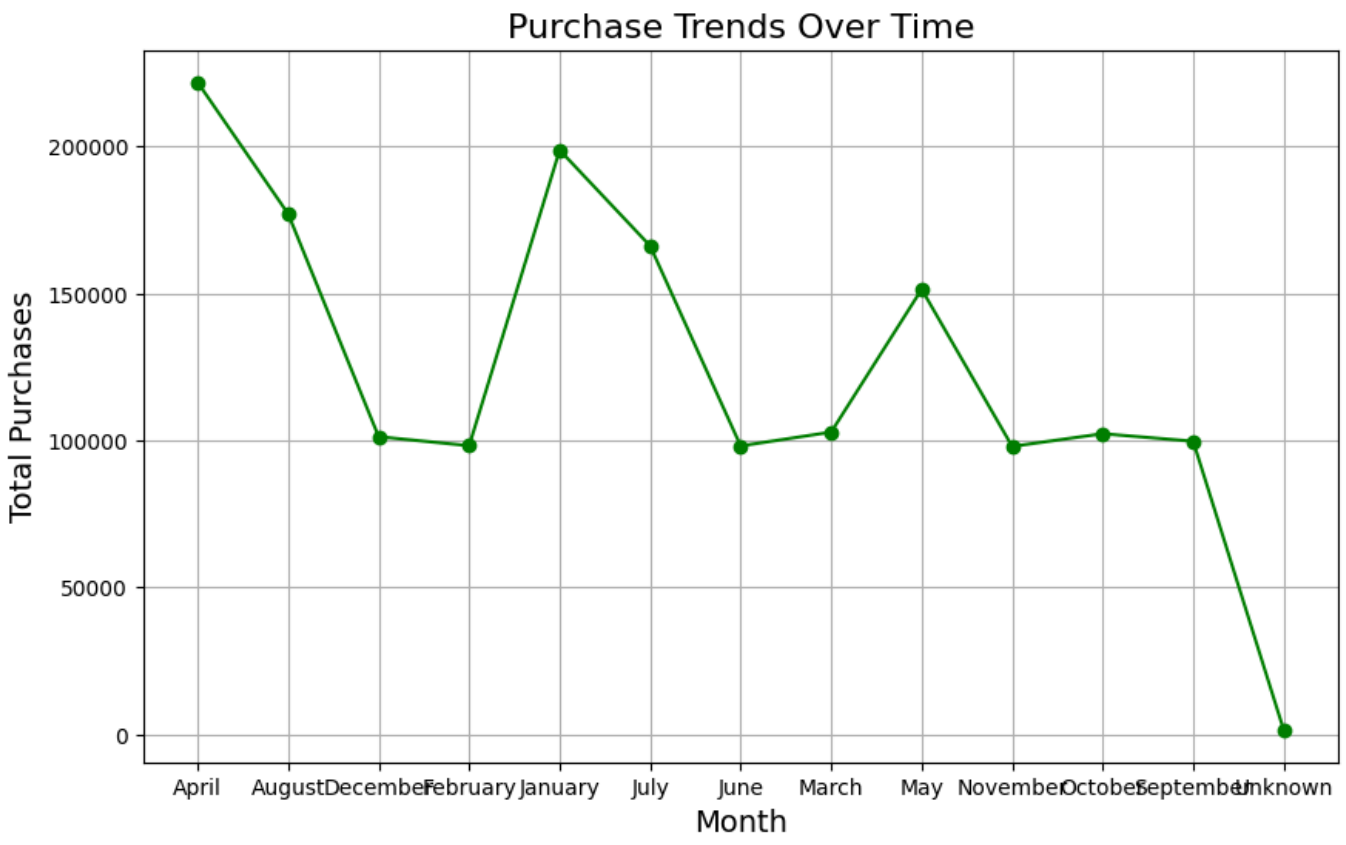
In [28]:
```python
# Visualize the total sales by state using a bar plot
plt.figure(figsize=(12, 6))
sns.barplot(x='State', y='Total_Amount', data=state_sales, palette='viridis')
plt.title('Total Sales by State', fontsize=16)
```

```
plt.xlabel('State', fontsize=14)
plt.ylabel('Total Sales (Amount)', fontsize=14)
plt.xticks(rotation=90)
plt.grid(True)
plt.show()
```



Plot the Purchase Trends Over Time (using matplotlib)

In [29]:
```
# Group by 'Month' to calculate total purchases per month
monthly_purchases = df.groupby('Month')['Total_Purchases'].sum().reset_index()

# Line plot to show purchase trends over time
plt.figure(figsize=(10, 6))
plt.plot(monthly_purchases['Month'], monthly_purchases['Total_Purchases'], marker='o', linestyle='-', color='green')
plt.title('Purchase Trends Over Time', fontsize=16)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Total Purchases', fontsize=14)
plt.grid(True)
plt.show()
```



Find Customers with Total Amount Above a Certain Threshold

In [30]:
```
# Filter customers with Total_Amount greater than 10,000
high_spenders = df[df['Total_Amount'] > 10000][['Customer_ID', 'Name', 'Total_Amount']]

# Display the top 5 high spenders
print("High Spenders (Total Amount > 10,000):\n", high_spenders.head())
```

```
High Spenders (Total Amount > 10,000):
 Empty DataFrame
Columns: [Customer_ID, Name, Total_Amount]
Index: []
```

Find the Average Purchase Amount by Customer Segment

In [31]:
```
# Group by 'Customer_Segment' and calculate the average Total_Amount
avg_spend_by_segment = df.groupby('Customer_Segment')['Total_Amount'].mean().reset_index()

# Display the average spending by customer segment
print("Average Purchase Amount by Customer Segment:\n", avg_spend_by_segment)
```

```
Average Purchase Amount by Customer Segment:
   Customer_Segment  Total_Amount
0               New   1367.647289
1           Premium   1362.848507
2           Regular   1368.879087
3           Unknown   1533.729559
```

Sort the data by Total_Amount in descending order

In [32]:
```
# Sort the data by Total_Amount in descending order
sorted_data = df.sort_values(by='Total_Amount', ascending=False)

# Reset the index and drop the old one
sorted_data_reset = sorted_data.reset_index(drop=True)

# Display the first few rows of the reset DataFrame
sorted_data_reset.head()
```

Out[32]:

	Transaction_ID	Customer_ID	Name	Email	Phone	Address	City	State	Zipcode	Country	...	Product_Category	Product_Brand	Product_Type	Feedba
0	3955465.0	16696	Sarah Smith	Daniel79@gmailcom	73408120570	255 Nicole Highway	Portsmouth	England	96017.0	UK	...	Electronics	Samsung	Television	B
1	9671388.0	23578	Casey Herrera	Alexandra87@gmailcom	42618660800	44959 Antonio Highway	New York	Kansas	65322.0	USA	...	Clothing	Adidas	Jacket	B
2	6851931.0	80039	Morgan Oconnor	Kelly34@gmailcom	94915526160	080 Chang Cape	Kitchener	Ontario	35277.0	Canada	...	Grocery	CocaCola	Juice	Avera
3	7518888.0	14409	Erica Green	Amanda25@gmailcom	83045647010	28381 Donovan Radial Apt 969	San Francisco	Maine	40543.0	USA	...	Books	Penguin Books	NonFiction	Go
4	6580873.0	67425	Jesse Hughes	Brandon23@gmailcom	69890182070	942 Carpenter Stream	Nuremberg	Berlin	27224.0	Germany	...	Clothing	Zara	Shirt	Go

5 rows × 31 columns

Use iloc to select rows 10 to 15 and specific columns

In [33]:
```
# Use iloc to select rows 10 to 15 and specific columns
selected_data_iloc = df.iloc[10:16, [df.columns.get_loc('Customer_ID'), df.columns.get_loc('Income'), df.columns.get_loc('Total_Amount')]]

# Display the selected data
print(selected_data_iloc)
```

```
    Customer_ID  Income  Total_Amount
10        19136     Low    363.927479
11        66883  Medium    364.830567
12        31930  Medium   1618.793610
13        74671  Medium   3157.353142
14        98300    High   1786.356235
15        64995  Medium     71.171704
```

Group by Product_Brand and calculate total purchases for each brand

In [34]:
```
# Group by Product_Brand and calculate total purchases for each brand
total_purchases_by_brand = df.groupby('Product_Brand')['Total_Purchases'].sum().reset_index()

# Display the total purchases by brand
print(total_purchases_by_brand)
```

```
       Product_Brand  Total_Purchases
0             Adidas          97468.0
1              Apple          96366.0
2   Bed Bath  Beyond          97404.0
3           BlueStar          12008.0
4           CocaCola          98271.0
5      HarperCollins          97574.0
6         Home Depot          97397.0
7               IKEA          96363.0
8         Mitsubhisi          36175.0
9             Nestle          96774.0
10              Nike          97804.0
11     Penguin Books          97140.0
12             Pepsi         162151.0
13      Random House          97113.0
14           Samsung          98752.0
15              Sony          97932.0
16           Unknown           1580.0
17         Whirepool          40064.0
18              Zara          98287.0
```

Rename columns

In [35]:
```
# Rename columns
df_renamed = df.rename(columns={'Total_Amount': 'Total_Spent', 'Total_Purchases': 'Purchases_Made'})

# Display the renamed columns
print(df_renamed[['Total_Spent', 'Purchases_Made']].head())
```

```
   Total_Spent  Purchases_Made
0   324.086270             3.0
1   806.707815             2.0
2  1063.432799             3.0
3  2466.854021             7.0
4   248.553049             2.0
```

Use iloc to select the 3rd, 7th, and 10th rows and specific columns

In [36]:
```
# Use iloc to select the 3rd, 7th, and 10th rows and specific columns
selected_rows = df.iloc[[2, 6, 9], [df.columns.get_loc('Customer_ID'), df.columns.get_loc('Age'), df.columns.get_loc('Income')]]

# Display the selected rows
print(selected_rows)
```

```
   Customer_ID   Age  Income
2        30192  48.0     Low
6        97285  29.0     Low
9        31878  25.0  Medium
```

Group by Gender and count the number of unique customers

In [37]:
```
# Group by Gender and count the number of unique customers
customer_count_by_gender = df.groupby('Gender')['Customer_ID'].nunique().reset_index()

# Display the customer count by gender
print(customer_count_by_gender)
```

```
    Gender  Customer_ID
0   Female        64400
1     Male        78598
2  Unknown          317
```

Group by 'State' and calculate max and min of Total_Amount for each state

In [38]:
```
# Group by 'State' and calculate max and min of Total_Amount for each state
max_min_total_by_state = df.groupby('State')['Total_Amount'].agg(['max', 'min']).reset_index()

# Sort by max Total_Amount in descending order and display the top 5 states
top_states_by_max_total = max_min_total_by_state.sort_values(by='max', ascending=False).head(5)

# Display the top 5 states by max Total_Amount
print("Top 5 states by maximum Total_Amount:\n", top_states_by_max_total)
```

```
Top 5 states by maximum Total_Amount:
       State          max        min
9    England  4999.625796  10.092966
17    Kansas  4999.340097  11.392996
39   Ontario  4999.171428  10.003750
20     Maine  4998.723479  10.304530
4     Berlin  4998.603558  10.063269
```

Export Dataset

In [39]:
```
# Export the DataFrame to a CSV file
df.to_csv('cleaned_retail_data.csv', index=False)

# Confirm the export
print("Data exported successfully to 'cleaned_retail_data.csv'.")
```

```
Data exported successfully to 'cleaned_retail_data.csv'.
```

In [ ]:

In [ ]: