

E-Commerce Sales Data Analysis **2024**

Dataset: [Comprehensive Synthetic E-Commerce Dataset](#)

Introduction:

The dataset is a synthetic e-commerce dataset that models a realistic online marketplace by combining transaction, customer, product, and advertising data. It captures key business dimensions such as customer demographics, product categories, revenue, discounts, and marketing performance metrics. The data incorporates seasonal trends and regional variations to reflect real-world consumer behavior, making it well-suited for analyzing sales patterns, customer behavior, product performance, and marketing effectiveness.

The objective of this analysis is to uncover meaningful patterns and insights related to product performance, revenue trends, and the impact of advertising efforts. Through systematic exploration and visualization, this EDA aims to provide a foundational understanding of the dataset and highlight key drivers of business performance that can inform data-driven decision-making in e-commerce contexts.

About The Dataset:

The dataset provides detailed synthetic data representing transactions, product information, and advertising metrics in an e-commerce setting.

Column Descriptions:

The dataset is divided into 15 columns and over 1 lakh rows. The following are the columns of the dataset.

- **Transaction Data:** -
 - **Transaction_ID:** Unique identifier for each transaction.
 - **Customer_ID:** Unique identifier for the customer associated with the transaction.
 - **Product_ID:** Unique identifier for the product purchased in the transaction.
 - **Transaction_Date:** The date when the transaction occurred.

- **Product Information:** -
 - **Category:** The category to which the product belongs (e.g, Electronics, Clothing).
 - **Units_Sold:** The quantity of the product sold in the transaction.

- **Discount_Applied:** The discount percentage applied to the product during the transaction.
- **Revenue:** Total revenue generated from the transaction, calculated as $\text{Price} \times \text{Units Sold} \times (1 - \text{Discount})$.

- **Advertising Metrics:** -
 - **Clicks:** Number of ad clicks associated with the product during the time of the transaction.
 - **Impressions:** Number of ad impressions served during the campaign.
 - **Conversion_Rate:** Calculated as $\text{Clicks} / \text{Impressions}$, representing the percentage of impressions that resulted in clicks.
 - **Ad_CTR:** Click-through rate (CTR) for the advertisement, representing the effectiveness of the ad campaign.
 - **Ad_CPC:** Cost-per-click for the advertisement.
 - **Ad_Spend:** Total advertising spends for the product, calculated as $\text{Ad_CTR} \times \text{Ad_CPC} \times 1000$.

- **Regional Information:** -
 - **Region:** The geographical region where the transaction occurred (e.g North America, Europe, Asia).

Importing the libraries needed for the EDA analysis:

I have imported numpy, pandas, matplotlib, seaborn, and plotly for the EDA analysis.

- **Numpy** is used for fast numerical computation and handling arrays that support mathematical operations.
- **Pandas** is essential for loading, cleaning, manipulating and analyzing structured tabular data.
- **Matplotlib** provides basic plotting capabilities to visualize data distributions and trends.
- **Seaborn** builds on Matplotlib to create more informative and aesthetically pleasing statistical visualizations with less code.
- **Plotly** enables interactive and dynamic visualizations that allow deeper exploration of data insights.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

✓ 8.0s

```
import os
os.getcwd()
```

✓ 0.0s

Importing the dataset:

The dataset was downloaded and stored in the system as ecommerce.csv. As it is a csv file, I used `.read_csv` to read the file.

```
ecom = pd.read_csv('D:\project\project\ecommerce.csv')
```

✓ 0.4s

Handling and visualizing the data:

- Checking the first 10 rows of the data using `.head(10)`

```
ecom.head(10)
```

✓ 0.0s

- Getting the summary of the dataset using `.info()`. This displays the summary of the dataset, including the number of rows and columns, column names, data types, non-null values and memory usage.

```
ecom.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Transaction_ID         100000 non-null object
1   Customer_ID           100000 non-null object
2   Product_ID            100000 non-null object
3   Transaction_Date       100000 non-null object
4   Units_Sold             100000 non-null int64
5   Discount_Applied      100000 non-null float64
6   Revenue               100000 non-null float64
7   Clicks                100000 non-null int64
8   Impressions           100000 non-null int64
9   Conversion_Rate       100000 non-null float64
10  Category               100000 non-null object
11  Region                100000 non-null object
12  Ad_CTR                100000 non-null float64
13  Ad_CPC                100000 non-null float64
14  Ad_Spend              100000 non-null float64
dtypes: float64(6), int64(3), object(6)
memory usage: 11.4+ MB
```

- Generating the summary statistics of numerical columns with **.describe()** function. This will show the count, mean, standard deviation, minimum, maximum, and quartiles of the numerical data.

```
ecom.describe()
```

✓ 0.0s

- Finding the data type of each column with **.dtypes**. This will help identifying whether features are numerical or categorical.

```
ecom.dtypes
```

✓ 0.0s

Transaction_ID	object
Customer_ID	object
Product_ID	object
Transaction_Date	object
Units_Sold	int64
Discount_Applied	float64
Revenue	float64
Clicks	int64
Impressions	int64
Conversion_Rate	float64
Category	object
Region	object
Ad_CTR	float64
Ad_CPC	float64
Ad_Spend	float64
dtype:	object

- Finding if there are any null values in the data with **.isnull().sum()**. This will provide the count of null values in each column. As of this dataset, there are no null values in it.

```
ecom.isnull().sum()
✓ 0.0s
```

Transaction_ID	0
Customer_ID	0
Product_ID	0
Transaction_Date	0
Units_Sold	0
Discount_Applied	0
Revenue	0
Clicks	0
Impressions	0
Conversion_Rate	0
Category	0
Region	0
Ad_CTR	0
Ad_CPC	0
Ad_Spend	0
dtype: int64	

- Finding if there are any duplicated values in the data with **.duplicated()**. This will find if there are any duplicated values in the dataset. As in this dataset there are no duplicated values found.

```
ecom.duplicated()
✓ 0.0s
```

0	False
1	False
2	False
3	False
4	False
...	
99995	False
99996	False
99997	False
99998	False
99999	False
Length: 100000, dtype: bool	

- Handling the outliers in the column Clicks.

```
Q1 = ecom['Clicks'].quantile(0.25)
Q3 = ecom['Clicks'].quantile(0.75)
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
✓ 0.0s

print(lower_bound)
print(upper_bound)
✓ 0.0s

-25.5
74.5
```

- Finding the frequency of each unique value in the column Category with `.value_counts()`.
This shows there are 5 unique values in the column Category.

```
ecom['Category'].value_counts()
✓ 0.0s

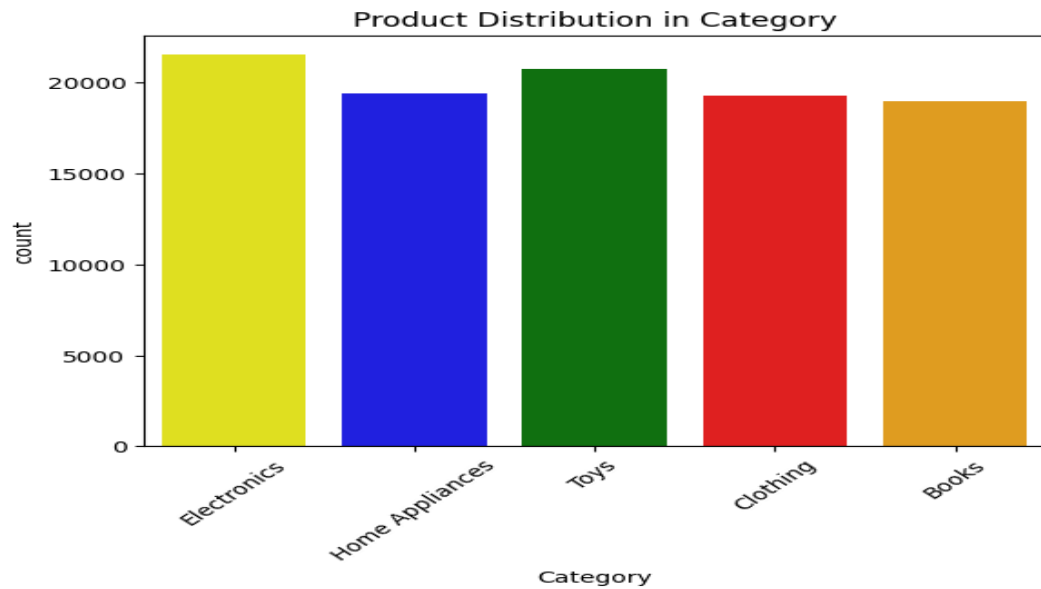
Category
Electronics      21533
Toys              20764
Home Appliances  19419
Clothing         19278
Books            19006
Name: count, dtype: int64
```

- Creating a count plot for visualizing the product category.

- Input:

```
colors = ['yellow', 'blue', 'green', 'red', 'orange']
sns.countplot(x='Category', data=ecom, palette=colors)
plt.title('Product Distribution in Category')
plt.xticks(rotation=45)
plt.show()
✓ 0.4s
```

- Output:



- Finding the regions of product distribution. `ecom['Region'].value_counts()` this will find unique values in the column Region.

```
12] ecom['Region'].value_counts()
✓ 0.0s

.. Region
Asia      33472
Europe    33265
North America 33263
Name: count, dtype: int64
```

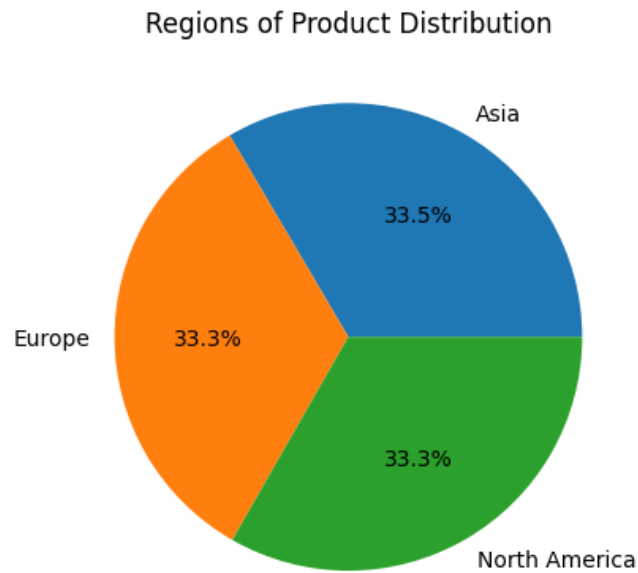
- Visualizing the regions of product distribution with a pie chart.

- Input:

```
region = ecom['Region'].value_counts()

plt.pie(region, labels=region.index, autopct='%1.1f%%')
plt.title('Regions of Product Distribution')
plt.show()
✓ 0.0s
```

- Output:



- Grouping the columns Category and Units_Sold for finding the average units sold in each category.

```
group1 = ecom.groupby('Category')['Units_Sold'].mean()
group1
```

✓ 0.0s

Category	Units_Sold
Books	116.771335
Clothing	128.198413
Electronics	139.409279
Home Appliances	122.091457
Toys	133.370208

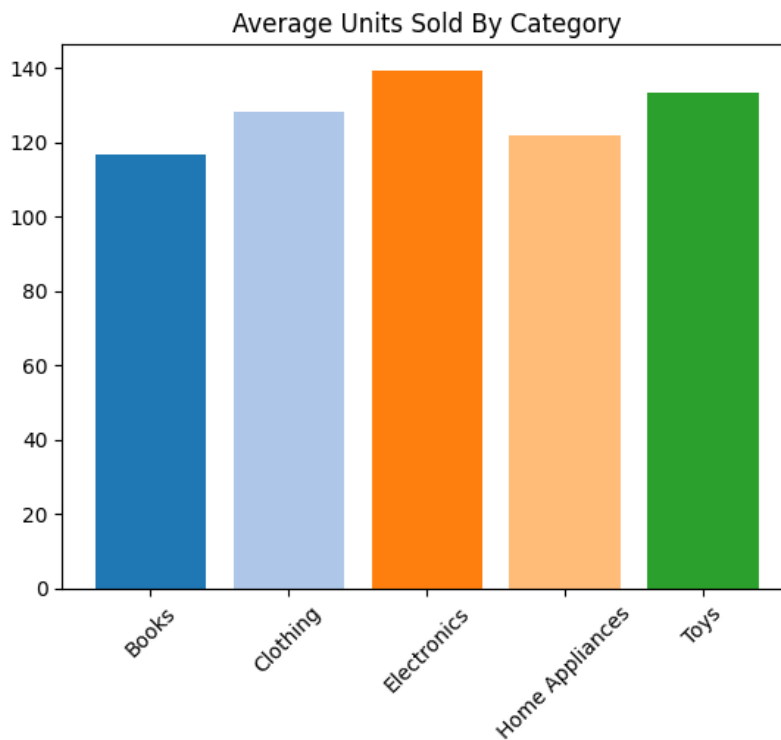
Name: Units_Sold, dtype: float64

- Visualizing the average units sold by category with a bar plot.
 - Input:

```
plt.bar(x=group1.index,height=group1.values,color=plt.cm.tab20(range(len(group1))))
plt.xticks(rotation=45)
plt.title('Average Units Sold By Category')
plt.show()
```

✓ 0.1s

- Output:



- Creating a column named Transaction_month.

```
ecom['Transaction_Date'] = pd.to_datetime(ecom['Transaction_Date'],errors='coerce')
ecom['Transaction_month'] = ecom['Transaction_Date'].dt.month_name()
ecom[['Transaction_Date', 'Transaction_month']].head()
```

✓ 0.0s

- `ecom['Transaction_Date']=pd.to_datetime(ecom['Transaction_Date'],errors='coerce')` This converts the Transaction_Date column into a proper datetime format.

- `ecom['Transaction_month'] = ecom['Transaction_Date'].dt.month_name()` This extracts the month name from each date and stores it in a new column named Transaction_month.
- `ecom[['Transaction_Date', 'Transaction_month']].head()` This displays the first five rows of the Transaction date and Transaction_mont column to verify the transaction.

```
ecom['Transaction_Date'] = pd.to_datetime(ecom['Transaction_Date'],errors='coerce')
ecom['Transaction_month'] = ecom['Transaction_Date'].dt.month_name()
ecom[['Transaction_Date', 'Transaction_month']].head()
```

✓ 0.0s

	Transaction_Date	Transaction_month
0	2024-10-06	October
1	2024-10-29	October
2	2024-04-04	April
3	2024-08-25	August
4	2024-05-05	May

- Creating a variable called month_order for organizing the month in calendar order. This ensures correct sorting and meaningful visualizations when analyzing or plotting monthly transaction trends.

```
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']

ecom['Transaction_month'] = pd.Categorical(
    ecom['Transaction_month'],
    categories=month_order,
    ordered=True
)
```

✓ 0.0s

- This will create a list that defines the correct chronological order of months.
- `ecom['Transaction_month'] = pd.Categorical(...)` this will convert the Transaction_month column into a categorical data type.
- `Categories=month_order, ordered=True`. This enforces the logical month order (Jan-Dec) instead of alphabetical order.

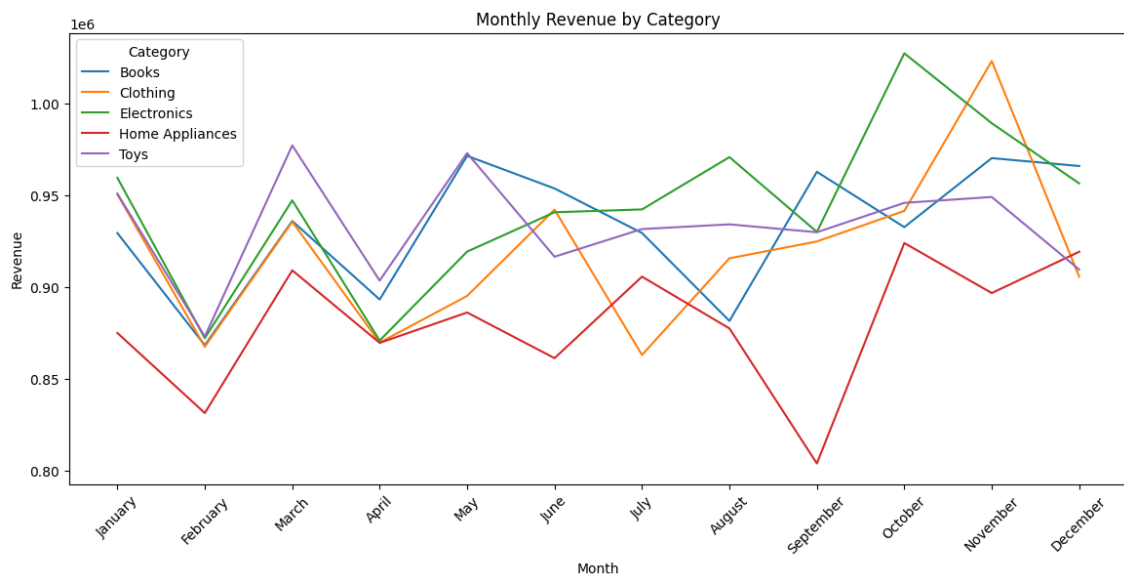
- Creating a line plot for finding the monthly revenue of each category.
 - Input

```
monthly_rev = ecom.groupby(['Transaction_month', 'Category'])['Revenue'].sum().reset_index()

plt.figure(figsize=(14,6))
sns.lineplot(data=monthly_rev, x='Transaction_month', y='Revenue', hue='Category')

plt.xticks(rotation=45)
plt.title('Monthly Revenue by Category')
plt.xlabel("Month")
plt.show()
✓ 0.3s
```

- Output:

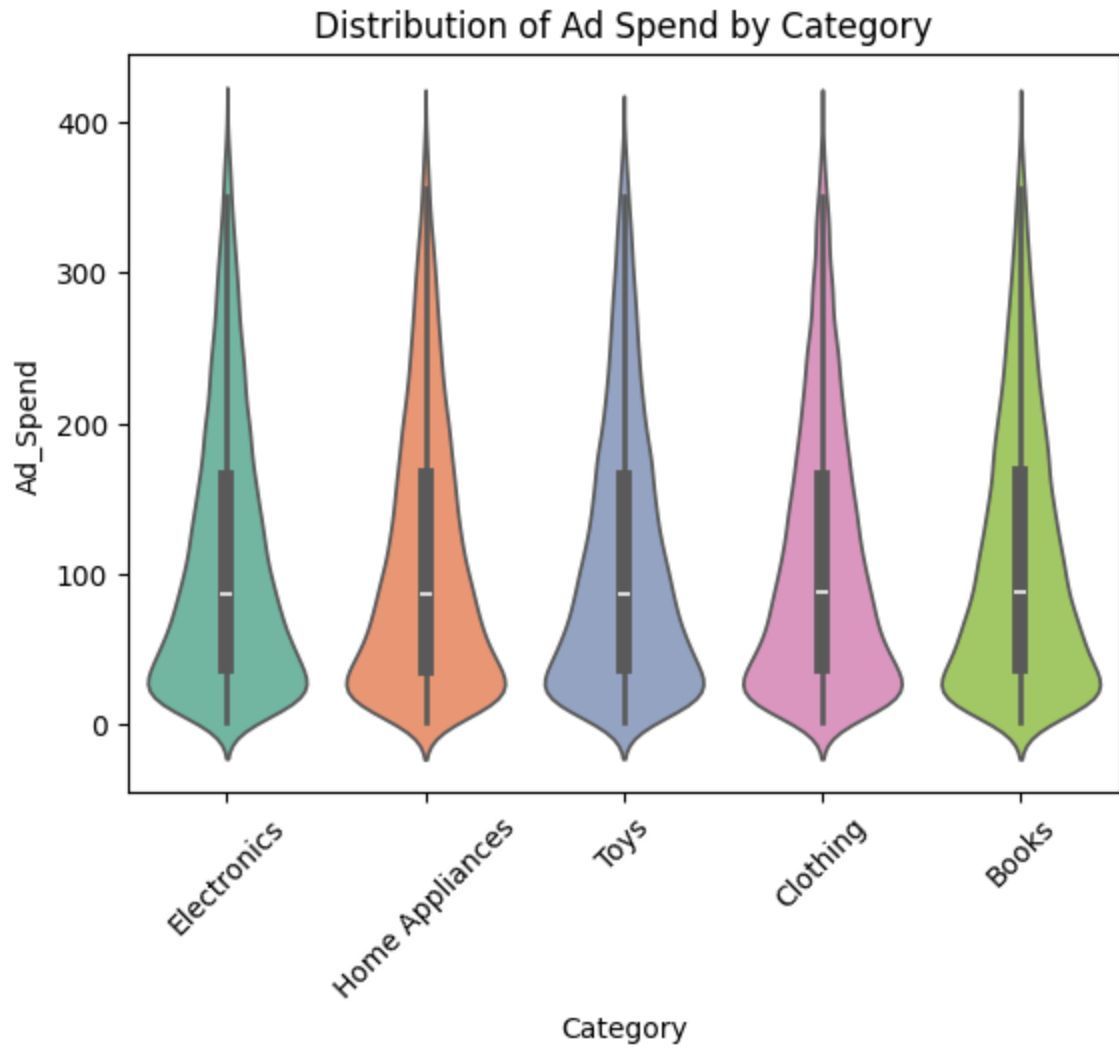


- Creating a violin plot for finding the Ad-spend on each category.

- Input:

```
sns.violinplot(x='Category', y='Ad_Spend', data=ecom, palette='Set2')
plt.title('Distribution of Ad Spend by Category')
plt.xticks(rotation=45)
plt.show()
✓ 0.4s
```

- Output:

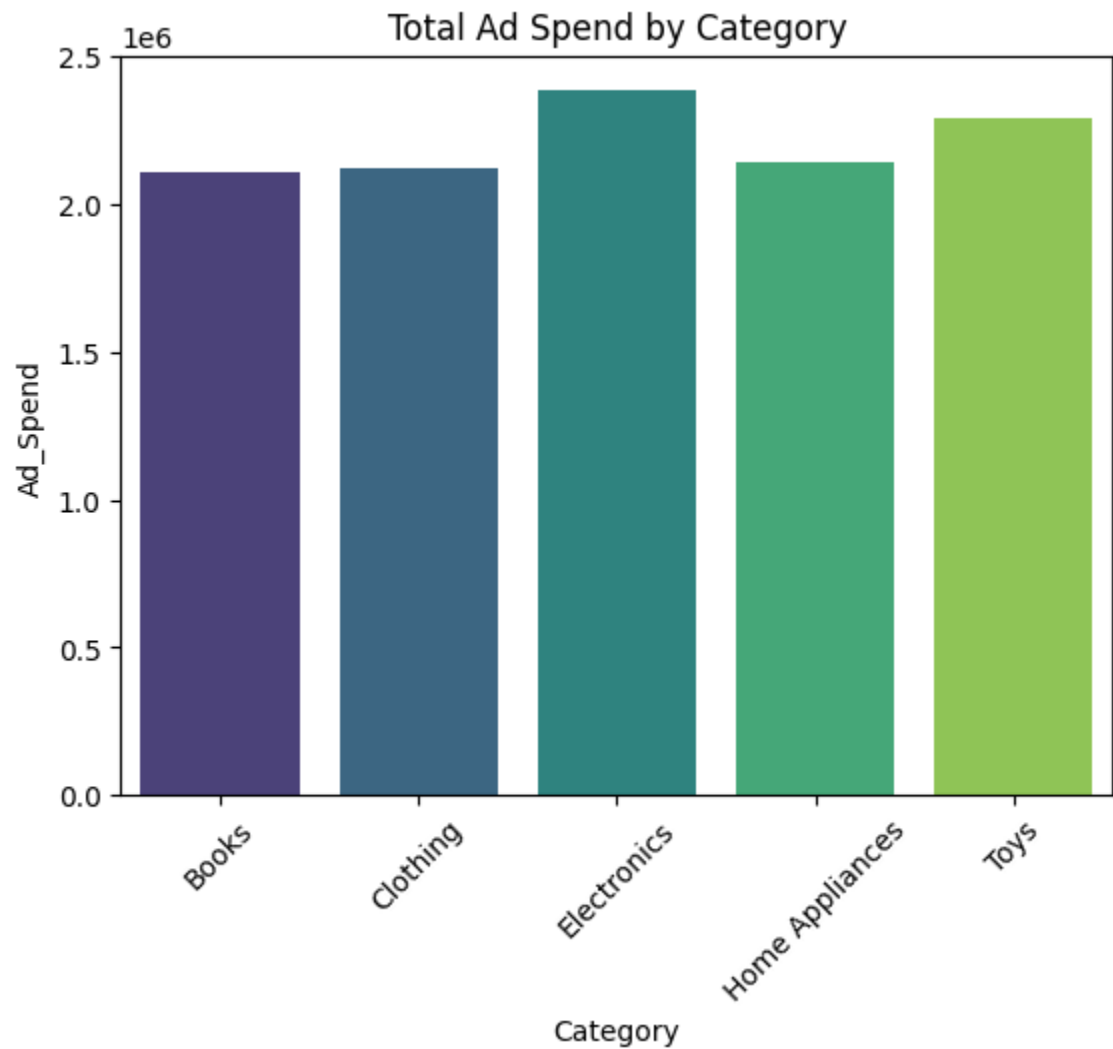


- This will show the price range of ad-spend on each categories. Most of the price range is in between 0-100.
- Creating a bar plot for finding the total amount spend on ad on each category.
 - Input:

```
total_ad = ecom.groupby('Category')['Ad_Spend'].sum().reset_index()

sns.barplot(data=total_ad, x='Category', y='Ad_Spend', palette='viridis')
plt.title("Total Ad Spend by Category")
plt.xticks(rotation=45)
plt.show()
```

- Grouped the columns category and ad_spend for better visualization.
- Output:



- This shows the total money spend on advertisement. The total amount spend on advertisement on each category is in between 20 to 25 lakhs.

- Creating a bar plot for finding the total revenue made on each category.

```
total_revenue = ecom.groupby('Category')['Revenue'].sum().reset_index().sort_values('Revenue')
total_revenue
```

✓ 0.0s

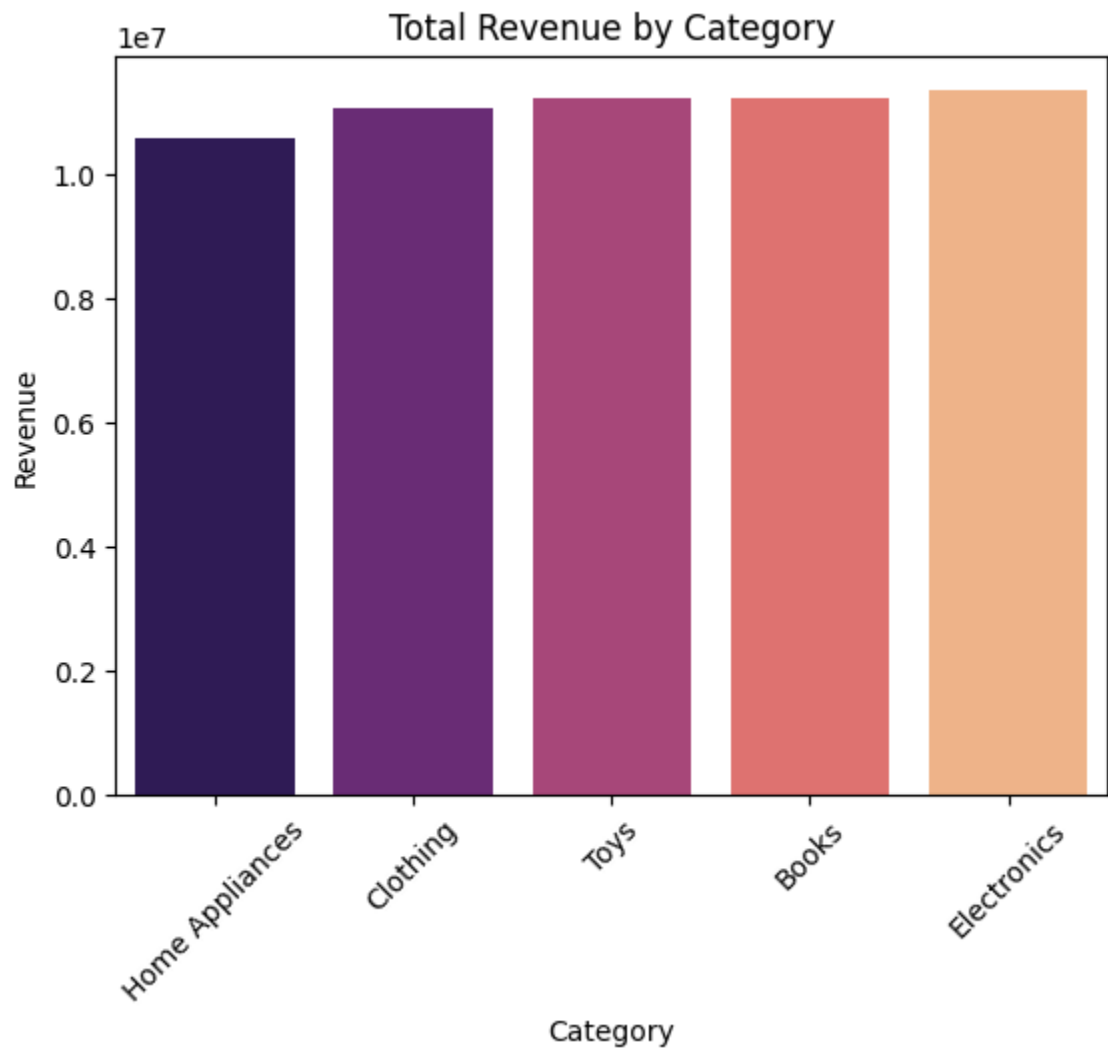
	Category	Revenue
3	Home Appliances	10559207.22
1	Clothing	11034899.26
4	Toys	11194842.46
0	Books	11195311.88
2	Electronics	11326820.25

- Grouping the columns Category and Revenue for better visualization.
- Input:

```
sns.barplot(x='Category', y='Revenue', data=total_revenue, palette='magma')  
plt.title('Total Revenue by Category')  
plt.xticks(rotation=45)  
plt.show()
```

✓ 0.4s

- Output:



- Creating a heatmap with Ad_spend, Ad_CTR, Ad_CPC, Impression, Clicks and Conversion_rate to find the correlation between them.

- Input:

```
cols = ['Ad_Spend', 'Ad_CTR', 'Ad_CPC', 'Impressions', 'Clicks', 'Conversion_Rate']

print(ecom[cols].head())

plt.figure(figsize=(10,6))
sns.heatmap(
    ecom[cols].corr(),
    annot=True,
    cmap='coolwarm',
    linewidths=0.5,
    fmt=".2f"
)
plt.title("Correlation Heatmap of Advertising Metrics")
plt.show()
```

- Output:

