**Part A 1**

**Step 1: Import libraries**

import pandas as pd

# Step 2: Load the dataset

df = pd.read\_csv('Retail\_Transactions\_2000.csv') # Replace with your actual file path

# Step 3: Check dataset size (rows and columns)

print("🔹 Dataset Shape (rows, columns):", df.shape)

# Step 4: View basic info: data types, non-null counts

print("\n🔹 Dataset Info:")

print(df.info())

# Step 5: View column names and first few rows

print("\n🔹 Column Names:", df.columns.tolist())

print("\n🔹 First 5 Rows:")

print(df.head())

# Step 6: Check for missing values

print("\n🔹 Missing Values per Column:")

print(df.isnull().sum())

# Step 7: Check for duplicate rows

duplicate\_count = df.duplicated().sum()

print(f"\n🔹 Number of Duplicate Rows: {duplicate\_count}")

# Step 8: Check for inconsistent values (e.g., in Gender column if exists)

if 'Gender' in df.columns:

print("\n🔹 Unique values in 'Gender' column:")

print(df['Gender'].unique())

# Optional: Check for columns with only one unique value (not useful for analysis)

print("\n🔹 Columns with one unique value:")

print([col for col in df.columns if df[col].nunique() == 1])

output:

Dataset Shape (rows, columns): (2000, 11)

🔹 Dataset Info:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 2000 entries, 0 to 1999

Data columns (total 11 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 TransactionID 2000 non-null object

1 CustomerID 2000 non-null object

2 Gender 2000 non-null object

3 Age 2000 non-null int64

4 City 2000 non-null object

5 ProductCategory 2000 non-null object

6 Quantity 2000 non-null int64

7 Price 2000 non-null int64

8 PurchaseDate 2000 non-null object

9 PaymentMode 2000 non-null object

10 TotalAmount 2000 non-null int64

dtypes: int64(4), object(7)

memory usage: 172.0+ KB

None

🔹 Column Names: ['TransactionID', 'CustomerID', 'Gender', 'Age', 'City', 'ProductCategory', 'Quantity', 'Price', 'PurchaseDate', 'PaymentMode', 'TotalAmount']

🔹 First 5 Rows:

TransactionID CustomerID Gender Age City ProductCategory Quantity \

0 T00001 C2824 Female 33 Bengaluru Fashion 3

1 T00002 C1409 Other 20 Pune Books 5

2 T00003 C5506 Other 47 Pune Furniture 1

3 T00004 C5012 Other 21 Hyderabad Groceries 5

4 T00005 C4657 Female 41 Chennai Sports 1

Price PurchaseDate PaymentMode TotalAmount

0 4479 2023-03-15 Cash 13437

1 1051 2023-01-22 Card 5255

2 111 2023-04-18 Wallet 111

3 2946 2023-08-09 Cash 14730

4 3123 2023-09-23 Wallet 3123

🔹 Missing Values per Column:

TransactionID 0

CustomerID 0

Gender 0

Age 0

City 0

ProductCategory 0

Quantity 0

Price 0

PurchaseDate 0

PaymentMode 0

TotalAmount 0

dtype: int64

🔹 Number of Duplicate Rows: 0

🔹 Unique values in 'Gender' column:

['Female' 'Other' 'Male']

🔹 Columns with one unique value:

[]

PART A 2

No missing values

Part A 3

# Remove duplicate transactions

initial\_rows = df.shape[0]

df.drop\_duplicates(inplace=True)

rows\_after\_dropping\_duplicates = df.shape[0]

print(f"🔹 Removed {initial\_rows - rows\_after\_dropping\_duplicates} duplicate rows.")

OUTPUT:

Removed 0 duplicate rows.

# Standardize categorical values in 'Gender' column

if 'Gender' in df.columns:

    df['Gender'] = df['Gender'].str.title()

    print("\n🔹 Standardized 'Gender' column.")

    print("Unique values in 'Gender' after standardization:", df['Gender'].unique())

OUTPUT:

Standardized 'Gender' column.

Unique values in 'Gender' after standardization: ['Female' 'Other' 'Male']

# Correct negative or zero values in Quantity or Price

initial\_rows = df.shape[0]

df = df[(df['Quantity'] > 0) & (df['Price'] > 0)]

rows\_after\_dropping\_invalid\_values = df.shape[0]

print(f"\n🔹 Removed {initial\_rows - rows\_after\_dropping\_invalid\_values} rows with non-positive Quantity or Price.")

OUTPUT:

Removed 0 rows with non-positive Quantity or Price.

PART A 4

# Derive TotalAmount if missing (not applicable in this dataset, but good practice)

# In this dataset, TotalAmount = Quantity \* Price, so we can verify or create if missing

if 'TotalAmount' not in df.columns or df['TotalAmount'].isnull().any():

df['TotalAmount'] = df['Quantity'] \* df['Price']

print("\n🔹 Derived 'TotalAmount' column.")

else:

# Verify TotalAmount calculation if it exists and is not missing

if not (df['TotalAmount'] == df['Quantity'] \* df['Price']).all():

print("\n⚠️ Warning: 'TotalAmount' does not match 'Quantity' \* 'Price'. Recalculating 'TotalAmount'.")

df['TotalAmount'] = df['Quantity'] \* df['Price']

else:

print("\n🔹 'TotalAmount' column is already present and consistent.")

# Display the first few rows to show the new/verified TotalAmount

print(df[['Quantity', 'Price', 'TotalAmount']].head())

OUTPUT:

'TotalAmount' column is already present and consistent.

Quantity Price TotalAmount

0 3 4479 13437

1 5 1051 5255

2 1 111 111

3 5 2946 14730

4 1 3123 3123

# Convert 'PurchaseDate' to datetime objects

df['PurchaseDate'] = pd.to\_datetime(df['PurchaseDate'])

# Extract Month and DayOfWeek

df['PurchaseMonth'] = df['PurchaseDate'].dt.month

df['PurchaseDayOfWeek'] = df['PurchaseDate'].dt.day\_name()

print("\n🔹 Extracted 'PurchaseMonth' and 'PurchaseDayOfWeek'.")

print(df[['PurchaseDate', 'PurchaseMonth', 'PurchaseDayOfWeek']].head())

OUTPUT:

Extracted 'PurchaseMonth' and 'PurchaseDayOfWeek'.

PurchaseDate PurchaseMonth PurchaseDayOfWeek

0 2023-03-15 3 Wednesday

1 2023-01-22 1 Sunday

2 2023-04-18 4 Tuesday

3 2023-08-09 8 Wednesday

4 2023-09-23 9 Saturday

# Create an AgeGroup column

bins = [0, 18, 25, 40, 60, df['Age'].max()]

labels = ['0-18', '19-25', '26-40', '41-60', '60+']

df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels, right=True, include\_lowest=True)

print("\n🔹 Created 'AgeGroup' column.")

print(df[['Age', 'AgeGr

OUTPUT:

Created 'AgeGroup' column.

Age AgeGroup

0 33 26-40

1 20 19-25

2 47 41-60

3 21 19-25

4 41 41-60

PART A 5

Select categorical columns for encoding

categorical\_cols = ['Gender', 'City', 'ProductCategory', 'PaymentMode']

# Apply one-hot encoding

df\_encoded = pd.get\_dummies(df, columns=categorical\_cols, drop\_first=True)

print("\n🔹 Applied One-Hot Encoding.")

print("Shape after encoding:", df\_encoded.shape)

print("First 5 rows of encoded data:")

print(df\_encoded.head())

OUTPUT:

Applied One-Hot Encoding.

Shape after encoding: (2000, 30)

First 5 rows of encoded data:

TransactionID CustomerID Age Quantity Price PurchaseDate TotalAmount \

0 T00001 C2824 33 3 4479 2023-03-15 13437

1 T00002 C1409 20 5 1051 2023-01-22 5255

2 T00003 C5506 47 1 111 2023-04-18 111

3 T00004 C5012 21 5 2946 2023-08-09 14730

4 T00005 C4657 41 1 3123 2023-09-23 3123

PurchaseMonth PurchaseDayOfWeek AgeGroup ... City\_Pune \

0 3 Wednesday 26-40 ... False

1 1 Sunday 19-25 ... True

2 4 Tuesday 41-60 ... True

3 8 Wednesday 19-25 ... False

4 9 Saturday 41-60 ... False

ProductCategory\_Books ProductCategory\_Electronics \

0 False False

1 True False

2 False False

3 False False

4 False False

ProductCategory\_Fashion ProductCategory\_Furniture \

0 True False

1 False False

2 False True

3 False False

4 False False

ProductCategory\_Groceries ProductCategory\_Sports PaymentMode\_Cash \

0 False False True

1 False False False

2 False False False

3 True False True

4 False True False

PaymentMode\_UPI PaymentMode\_Wallet

0 False False

1 False False

2 False True

3 False False

4 False True

[5 rows x 30 columns]

0s

from sklearn.preprocessing import StandardScaler

# Select numerical columns for normalization

numerical\_cols = ['Age', 'Price', 'Quantity', 'TotalAmount']

# Initialize StandardScaler

scaler = StandardScaler()

# Apply standardization

df\_encoded[numerical\_cols] = scaler.fit\_transform(df\_encoded[numerical\_cols])

print("\n🔹 Applied Standardization to Numerical Columns.")

print("First 5 rows of normalized numerical columns:")

print(df\_encoded[numerical\_cols].head())

OUTPUT:

Applied Standardization to Numerical Columns.

First 5 rows of normalized numerical columns:

Age Price Quantity TotalAmount

0 -0.769570 1.402371 0.030458 1.074680

1 -1.626186 -1.022646 1.447099 -0.359843

2 0.152939 -1.687615 -1.386183 -1.261723

3 -1.560293 0.317905 1.447099 1.301377

4 -0.242422 0.443117 -1.386183 -0.733639

PART B 1

import matplotlib.pyplot as plt

import seaborn as sns

# Age distribution

plt.figure(figsize=(10, 6))

sns.histplot(df['Age'], bins=20, kde=True)

plt.title('Distribution of Customer Age')

plt.xlabel('Age')

plt.ylabel('Frequency')

plt.show()

# Gender distribution

plt.figure(figsize=(6, 6))

df['Gender'].value\_counts().plot(kind='pie', autopct='%1.1f%%', startangle=90, colors=['skyblue', 'lightcoral', 'lightgreen'])

plt.title('Distribution of Customer Gender')

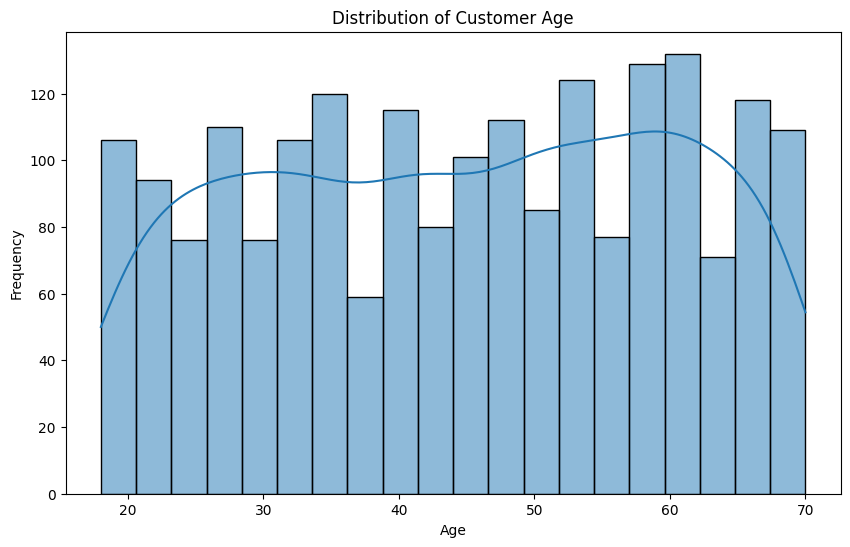
plt.ylabel('') # Hide the default 'Gender' label on the y-axis for a cleaner pie chart

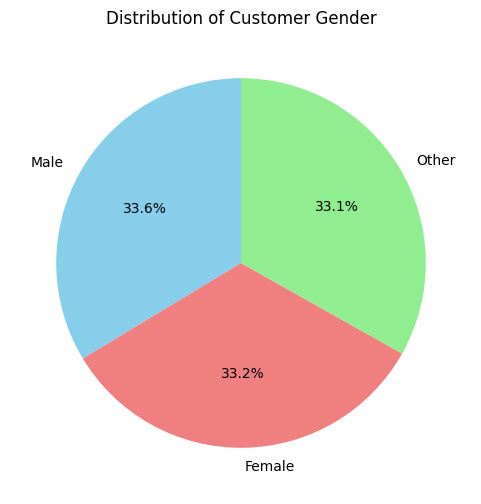
plt.show()

# Customers by city (Top 10)

print("\n🔹 Top 10 Cities by Customer Count:")

print(df['City'].value\_counts().head(10))





🔹 Top 10 Cities by Customer Count:

City

Ahmedabad 222

Bengaluru 217

Lucknow 213

Mumbai 202

Delhi 199

Jaipur 196

Kolkata 196

Chennai 187

Hyderabad 185

Pune 183

Name: count, dtype: int64

PART B 2

import matplotlib.pyplot as plt

import seaborn as sns

# Total sales by product category

sales\_by\_category = df.groupby('ProductCategory')['TotalAmount'].sum().sort\_values(ascending=False)

print("\n🔹 Total Sales by Product Category:")

print(sales\_by\_category)

plt.figure(figsize=(12, 7))

sns.barplot(x=sales\_by\_category.index, y=sales\_by\_category.values, palette='viridis')

plt.title('Total Sales by Product Category')

plt.xlabel('Product Category')

plt.ylabel('Total Sales Amount')

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.show()

Total Sales by Product Category:

ProductCategory

Furniture 2307895

Beauty 2193537

Books 2159914

Electronics 2067687

Sports 2022630

Groceries 1945573

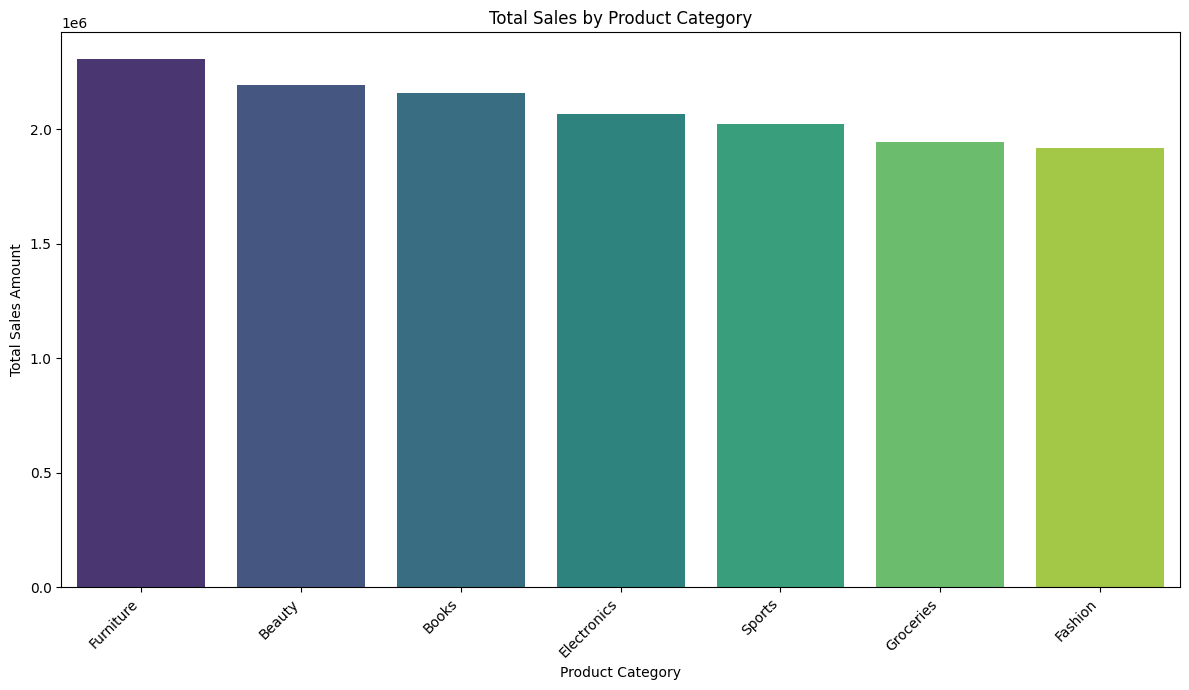
Fashion 1917593

Name: TotalAmount, dtype: int64

/tmp/ipython-input-3394873381.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

sns.barplot(x=sales\_by\_category.index, y=sales\_by\_category.values, palette='viridis')



# Monthly sales trend

# Ensure PurchaseDate is datetime type

df['PurchaseDate'] = pd.to\_datetime(df['PurchaseDate'])

monthly\_sales = df.set\_index('PurchaseDate').resample('M')['TotalAmount'].sum()

print("\n🔹 Monthly Sales Trend:")

print(monthly\_sales)

plt.figure(figsize=(12, 6))

monthly\_sales.plot(kind='line', marker='o')

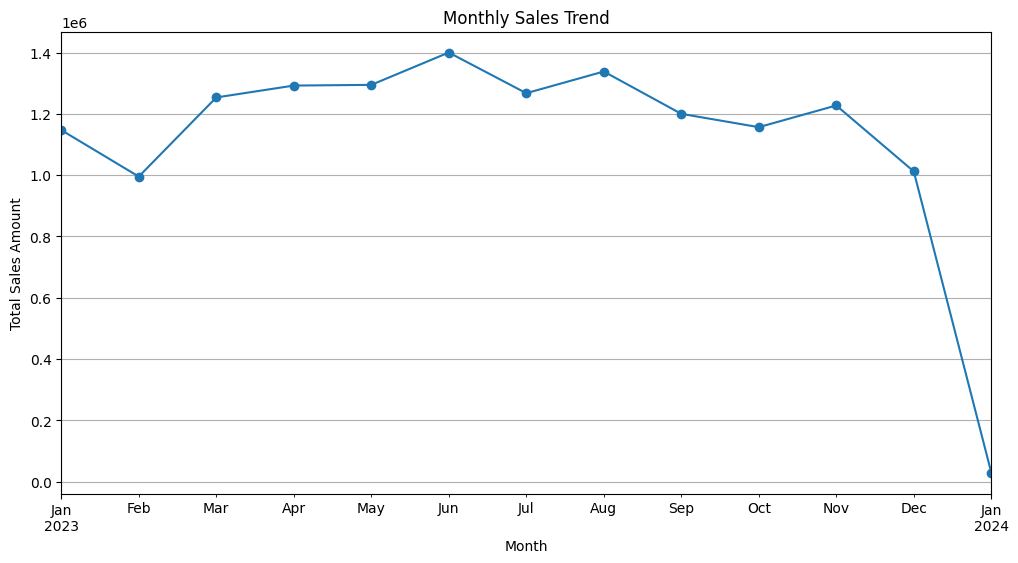
plt.title('Monthly Sales Trend')

plt.xlabel('Month')

plt.ylabel('Total Sales Amount')

plt.grid(True)

plt.show()



# Payment mode usage

payment\_mode\_counts = df['PaymentMode'].value\_counts()

print("\n🔹 Payment Mode Usage:")

print(payment\_mode\_counts)

plt.figure(figsize=(7, 7))

payment\_mode\_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90, colors=['gold', 'lightcoral', 'skyblue'])

plt.title('Distribution of Payment Mode Usage')

plt.ylabel('') # Hide the default 'PaymentMode' label on the y-axis

plt.show()

Payment Mode Usage:

PaymentMode

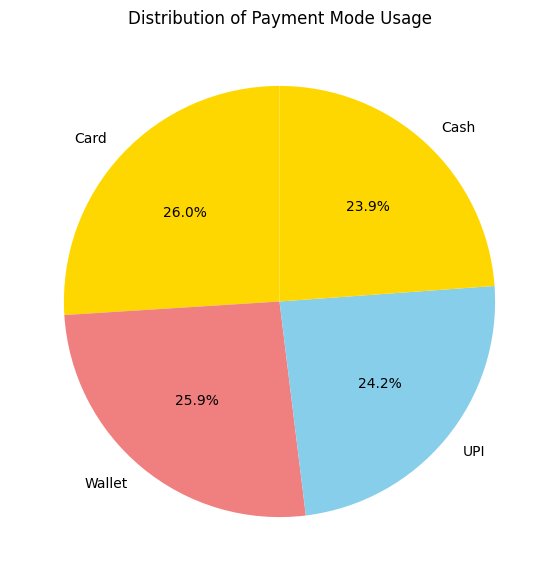
Card 520

Wallet 519

UPI 484

Cash 477

Name: count, dtype: int64



PART B 3

# Average spend per customer by age group

average\_spend\_by\_age\_group = df.groupby('AgeGroup')['TotalAmount'].mean().sort\_values(ascending=False)

print("\n🔹 Average Spend per Customer by Age Group:")

print(average\_spend\_by\_age\_group)

plt.figure(figsize=(10, 6))

sns.barplot(x=average\_spend\_by\_age\_group.index, y=average\_spend\_by\_age\_group.values, palette='viridis')

plt.title('Average Spend per Customer by Age Group')

plt.xlabel('Age Group')

plt.ylabel('Average Total Amount Spent')

plt.show()

OUTPUT:

🔹 Average Spend per Customer by Age Group:

AgeGroup

0-18 8917.521739

19-25 7427.500000

26-40 7341.934066

41-60 7232.245259

60+ 7149.602067

Name: TotalAmount, dtype: float64

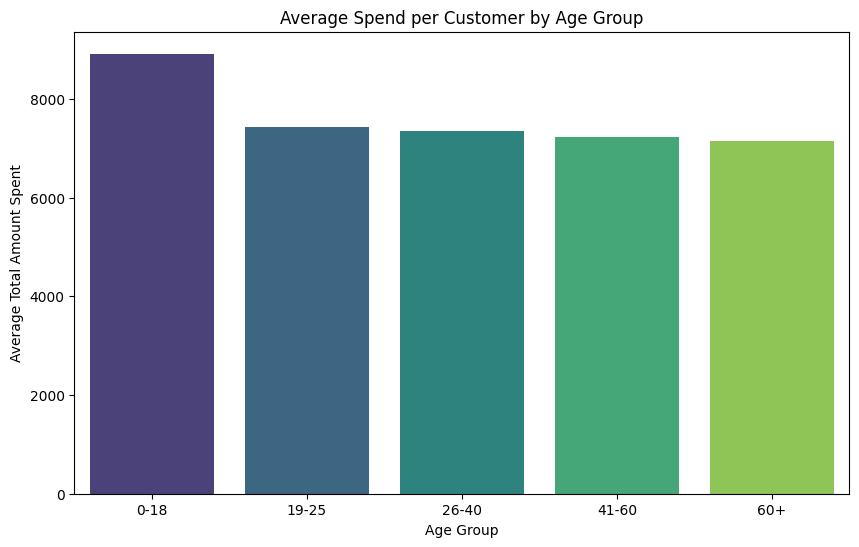
/tmp/ipython-input-1959794725.py:2: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.

average\_spend\_by\_age\_group = df.groupby('AgeGroup')['TotalAmount'].mean().sort\_values(ascending=False)

/tmp/ipython-input-1959794725.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

sns.barplot(x=average\_spend\_by\_age\_group.index, y=average\_spend\_by\_age\_group.values, palette='viridis')



City-wise revenue contribution

city\_revenue = df.groupby('City')['TotalAmount'].sum().sort\_values(ascending=False)

print("\n🔹 City-wise Revenue Contribution:")

print(city\_revenue)

plt.figure(figsize=(12, 7))

sns.barplot(x=city\_revenue.index, y=city\_revenue.values, palette='viridis')

plt.title('City-wise Revenue Contribution')

plt.xlabel('City')

plt.ylabel('Total Revenue')

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.show()

OUTP[UT:

City-wise Revenue Contribution:

City

Lucknow 1726697

Ahmedabad 1587809

Delhi 1540166

Bengaluru 1526529

Kolkata 1510583

Chennai 1448730

Hyderabad 1335722

Mumbai 1326521

Jaipur 1321035

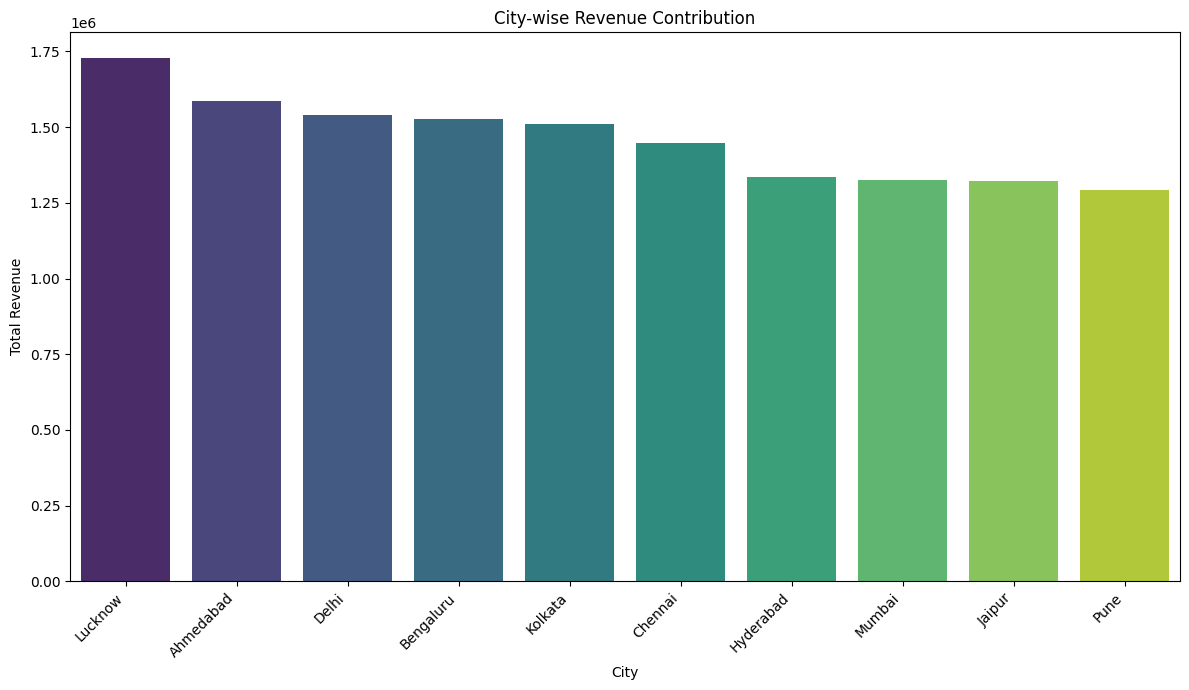
Pune 1291037

Name: TotalAmount, dtype: int64

/tmp/ipython-input-3280112501.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

sns.barplot(x=city\_revenue.index, y=city\_revenue.values, palette='viridis')



# Heatmap of product category vs payment mode

# Create a pivot table for the heatmap

heatmap\_data = df.pivot\_table(index='ProductCategory', columns='PaymentMode', values='TotalAmount', aggfunc='sum')

print("\n🔹 Heatmap Data (Total Amount by Product Category and Payment Mode):")

print(heatmap\_data)

plt.figure(figsize=(10, 7))

sns.heatmap(heatmap\_data, annot=True, fmt=".1f", cmap="YlGnBu")

plt.title('Heatmap of Total Sales Amount by Product Category and Payment Mode')

plt.xlabel('Payment Mode')

plt.ylabel('Product Category')

plt.show()

Heatmap Data (Total Amount by Product Category and Payment Mode):

PaymentMode Card Cash UPI Wallet

ProductCategory

Beauty 532068 487461 567460 606548

Books 551446 529041 623227 456200

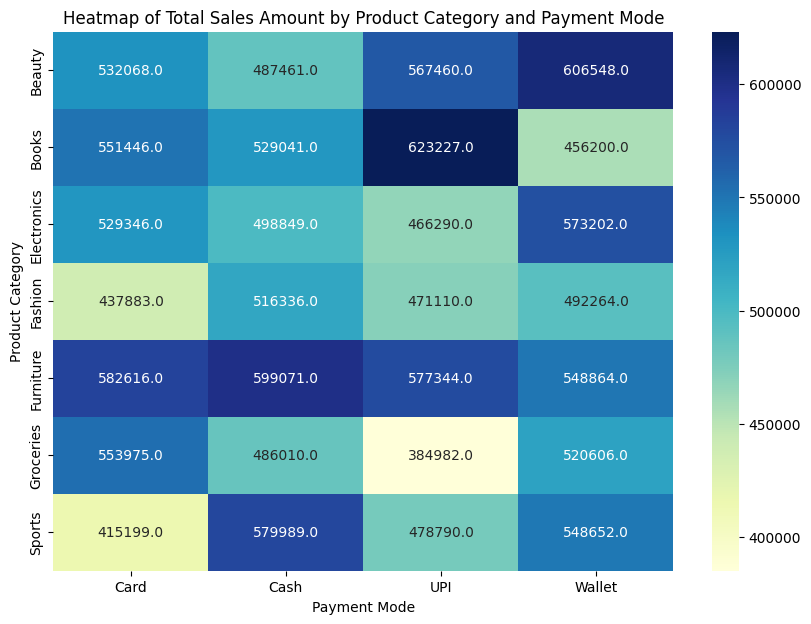
Electronics 529346 498849 466290 573202

Fashion 437883 516336 471110 492264

Furniture 582616 599071 577344 548864

Groceries 553975 486010 384982 520606

Sports 415199 579989 478790 548652



PART B 4

