Week 17 GRADED MINI PROJECT

Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) in Python involves using libraries like pandas, matplotlib, and seaborn to understand the structure, patterns, and relationships within a dataset before modeling.

Project Objective

Perform a comprehensive analysis on this retail dataset to help answer key business questions. Your project should be organized into the following tasks:

Task 1: Data Preparation

- Read the CSV file.
- Parse and convert the Date column into appropriate format.
- Extract additional useful information like Year, Month, or DayOfWeek from the Date.
- Clean and preprocess the data if required.

In [1]:
```
!pip install pandas
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.ticker as mtick
import os
```

```
Requirement already satisfied: pandas in c:\users\91704\anaconda3\pkgs\new folder\li
b\site-packages (2.2.3)
Requirement already satisfied: numpy>=1.26.0 in c:\users\91704\anaconda3\pkgs\new fo
lder\lib\site-packages (from pandas) (2.1.3)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\91704\anaconda3\pk
gs\new folder\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\91704\anaconda3\pkgs\new fol
der\lib\site-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\91704\anaconda3\pkgs\new f
older\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\91704\anaconda3\pkgs\new folder
\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

In [2]:
```
#Task 1
# Read the CSV file
df = pd.read_csv("Retail_Transactions_Dataset.csv")
print(f"Successfully loaded : \n \n {df}. Initial rows:")
print(df.head(5))
```

```
print("\nInitial DataFrame Info: \n \n")
df.info()
```

```
Successfully loaded :

         Transaction_ID                 Date      Customer_Name  \
0           1000000000  2022-01-21 06:27:29       Stacey Price
1           1000000001  2023-03-01 13:01:21   Michelle Carlson
2           1000000002  2024-03-21 15:37:04        Lisa Graves
3           1000000003  2020-10-31 09:59:47  Mrs. Patricia May
4           1000000004  2020-12-10 00:59:59     Susan Mitchell
...                ...                  ...                ...
999995      1000999995  2023-03-27 06:12:10      Lisa Gonzalez
999996      1000999996  2022-05-19 05:13:58       Emily Graham
999997      1000999997  2021-09-03 13:59:39   Cynthia Anderson
999998      1000999998  2023-10-17 05:50:40  Michael Rodriguez
999999      1000999999  2020-06-15 11:58:49     Jennifer Davis

                                                  Product  Total_Items  \
0            ['Ketchup', 'Shaving Cream', 'Light Bulbs']            3
1       ['Ice Cream', 'Milk', 'Olive Oil', 'Bread', 'P...            2
2                                          ['Spinach']            6
3                                ['Tissues', 'Mustard']            1
4                                       ['Dish Soap']           10
...                                               ...          ...
999995  ['Pickles', 'Carrots', 'Peanut Butter', 'Spong...            1
999996                                      ['Cereal']            8
999997                                  ['Trash Bags']            3
999998            ['Diapers', 'Coffee', 'Coffee', 'Mop']            3
999999                      ['Trash Cans', 'Mop', 'Jam']            8

        Total_Cost  Payment_Method           City         Store_Type  \
0            71.65  Mobile Payment    Los Angeles     Warehouse Club
1            25.93            Cash  San Francisco    Specialty Store
2            41.49     Credit Card        Houston   Department Store
3            39.34  Mobile Payment        Chicago           Pharmacy
4            16.42      Debit Card        Houston    Specialty Store
...            ...             ...            ...                ...
999995       22.07      Debit Card    Los Angeles        Supermarket
999996       80.25            Cash        Houston     Warehouse Club
999997       60.74     Credit Card    Los Angeles  Convenience Store
999998       23.48      Debit Card  San Francisco        Supermarket
999999       44.12     Credit Card        Atlanta           Pharmacy

        Discount_Applied Customer_Category  Season                   Promotion
0                   True         Homemaker  Winter                         NaN
1                   True      Professional    Fall      BOGO (Buy One Get One)
2                   True      Professional  Winter                         NaN
3                   True         Homemaker  Spring                         NaN
4                  False       Young Adult  Winter  Discount on Selected Items
...                  ...               ...     ...                         ...
999995             False       Middle-Aged  Winter                         NaN
999996              True    Senior Citizen  Spring  Discount on Selected Items
999997             False         Homemaker  Winter                         NaN
999998              True           Retiree  Winter      BOGO (Buy One Get One)
999999             False      Professional    Fall  Discount on Selected Items

[1000000 rows x 13 columns]. Initial rows:
   Transaction_ID                 Date      Customer_Name  \
```

```
0      1000000000  2022-01-21 06:27:29       Stacey Price
1      1000000001  2023-03-01 13:01:21   Michelle Carlson
2      1000000002  2024-03-21 15:37:04        Lisa Graves
3      1000000003  2020-10-31 09:59:47  Mrs. Patricia May
4      1000000004  2020-12-10 00:59:59     Susan Mitchell

                                          Product  Total_Items  Total_Cost  \
0       ['Ketchup', 'Shaving Cream', 'Light Bulbs']            3       71.65
1  ['Ice Cream', 'Milk', 'Olive Oil', 'Bread', 'P...            2       25.93
2                                     ['Spinach']            6       41.49
3                           ['Tissues', 'Mustard']            1       39.34
4                                    ['Dish Soap']           10       16.42

   Payment_Method           City        Store_Type  Discount_Applied  \
0  Mobile Payment    Los Angeles    Warehouse Club              True
1            Cash  San Francisco   Specialty Store              True
2     Credit Card        Houston  Department Store              True
3  Mobile Payment        Chicago          Pharmacy              True
4      Debit Card        Houston   Specialty Store             False

  Customer_Category  Season                    Promotion
0         Homemaker  Winter                          NaN
1      Professional    Fall      BOGO (Buy One Get One)
2      Professional  Winter                          NaN
3         Homemaker  Spring                          NaN
4       Young Adult  Winter  Discount on Selected Items

Initial DataFrame Info:


<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 13 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   Transaction_ID     1000000 non-null  int64
 1   Date               1000000 non-null  object
 2   Customer_Name      1000000 non-null  object
 3   Product            1000000 non-null  object
 4   Total_Items        1000000 non-null  int64
 5   Total_Cost         1000000 non-null  float64
 6   Payment_Method     1000000 non-null  object
 7   City               1000000 non-null  object
 8   Store_Type         1000000 non-null  object
 9   Discount_Applied   1000000 non-null  bool
 10  Customer_Category  1000000 non-null  object
 11  Season             1000000 non-null  object
 12  Promotion          666057 non-null   object
dtypes: bool(1), float64(1), int64(2), object(9)
memory usage: 92.5+ MB
```

In [3]:
```
# Convert 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])
```

In [4]:
```
# Extract Year, Month, Day of Week
df['Year'] = df['Date'].dt.year
```

```
df['Month'] = df['Date'].dt.month
df['DayOfWeek'] = df['Date'].dt.day_name()

print("\nDataFrame after Date processing: \n ")
print(df[['Date', 'Year', 'Month', 'DayOfWeek']].head())
```

```
DataFrame after Date processing:

                 Date  Year  Month  DayOfWeek
0 2022-01-21 06:27:29  2022      1     Friday
1 2023-03-01 13:01:21  2023      3  Wednesday
2 2024-03-21 15:37:04  2024      3   Thursday
3 2020-10-31 09:59:47  2020     10   Saturday
4 2020-12-10 00:59:59  2020     12   Thursday
```

In [5]:
```
# Check for missing values and clean
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000000 entries, 0 to 999999
Data columns (total 16 columns):
 #   Column             Non-Null Count    Dtype
---  ------             --------------    -----
 0   Transaction_ID     1000000 non-null  int64
 1   Date               1000000 non-null  datetime64[ns]
 2   Customer_Name      1000000 non-null  object
 3   Product            1000000 non-null  object
 4   Total_Items        1000000 non-null  int64
 5   Total_Cost         1000000 non-null  float64
 6   Payment_Method     1000000 non-null  object
 7   City               1000000 non-null  object
 8   Store_Type         1000000 non-null  object
 9   Discount_Applied   1000000 non-null  bool
 10  Customer_Category  1000000 non-null  object
 11  Season             1000000 non-null  object
 12  Promotion          666057 non-null   object
 13  Year               1000000 non-null  int32
 14  Month              1000000 non-null  int32
 15  DayOfWeek          1000000 non-null  object
dtypes: bool(1), datetime64[ns](1), float64(1), int32(2), int64(2), object(9)
memory usage: 107.8+ MB
```

In [6]:
```
df.head()
```

Out[6]:

	Transaction_ID	Date	Customer_Name	Product	Total_Items	Total_Cost	Payment_M
0	1000000000	2022-01-21 06:27:29	Stacey Price	['Ketchup', 'Shaving Cream', 'Light Bulbs']	3	71.65	Mobile P
1	1000000001	2023-03-01 13:01:21	Michelle Carlson	['Ice Cream', 'Milk', 'Olive Oil', 'Bread', 'P...	2	25.93	
2	1000000002	2024-03-21 15:37:04	Lisa Graves	['Spinach']	6	41.49	Cred
3	1000000003	2020-10-31 09:59:47	Mrs. Patricia May	['Tissues', 'Mustard']	1	39.34	Mobile P
4	1000000004	2020-12-10 00:59:59	Susan Mitchell	['Dish Soap']	10	16.42	Deb

In [7]:
```
## Clean and preprocess data
# Check for missing values
print("\nChecking for missing values: \n")
print(df.isnull().sum())
```

```
Checking for missing values:

Transaction_ID            0
Date                      0
Customer_Name             0
Product                   0
Total_Items               0
Total_Cost                0
Payment_Method            0
City                      0
Store_Type                0
Discount_Applied          0
Customer_Category         0
Season                    0
Promotion            333943
Year                      0
Month                     0
DayOfWeek                 0
dtype: int64
```

Task 2: Basic Exploration :

- How many total transactions are there?
- How many unique customers are in the dataset?
- What are the top 5 most common products sold across all transactions?
- Which cities have the highest number of transactions?

In [8]:
```
# Total Transaction
total_transactions = df['Transaction_ID']
print(f'Total transactions: {total_transactions}')
```

```
Total transactions: 0         1000000000
1         1000000001
2         1000000002
3         1000000003
4         1000000004
             ...
999995    1000999995
999996    1000999996
999997    1000999997
999998    1000999998
999999    1000999999
Name: Transaction_ID, Length: 1000000, dtype: int64
```

In [9]:
```
## Unique customers
unique_customers = df['Customer_Name'].nunique()
print(f'Unique Customers: {unique_customers}')
```

```
Unique Customers: 329738
```

In [10]:
```
import ast

def safe_eval(val):
    if pd.isna(val):
        return [] # empty list for NaN
    if isinstance(val, list):
        return val  # already a list
    if isinstance(val, str):
        val = val.strip()
        if val.startswith('[') and val.endswith(']'):
            try:
                return ast.literal_eval(val)
            except (ValueError, SyntaxError):
                return [val]  # fallback: treat whole string as one product
        else:
            return [val]  # treat as single product name
    return [str(val)]  # last fallback
# Apply safe parsing
df['Product'] = df['Product'].apply(safe_eval)
# Explode and count
top_products = df['Product'].explode().value_counts().head(5)
print(top_products)
```

```
Product
Toothpaste    73324
Ice Cream     37094
Soap          37076
Jam           36956
Orange        36928
Name: count, dtype: int64
```

In [11]:
```
## Cities with the highest number of transactions
top_cities = df['City'].value_counts().head(5)
print('Cities with the Highest Number of Transactions: \n')
print(top_cities)
```

```
Cities with the Highest Number of Transactions:

City
Boston     100566
Dallas     100559
Seattle    100167
Chicago    100059
Houston    100050
Name: count, dtype: int64
```

Task 3: Customer Behavior Analysis

- Which customer categories spend the most on average?
- Do certain customer categories prefer specific payment methods?
- What is the average number of items bought per transaction per store type?

In [12]:
```
## Customer categories spending
avg_spending = df.groupby('Customer_Category')['Total_Cost'].mean().sort_values(asc
print('\nAverage Spending by Customer Category:')
print(avg_spending)
```

```
Average Spending by Customer Category:
Customer_Category
Teenager          52.529091
Professional      52.525762
Student           52.487994
Homemaker         52.461417
Young Adult       52.448246
Retiree           52.435589
Middle-Aged       52.411318
Senior Citizen    52.342672
Name: Total_Cost, dtype: float64
```

In [13]:
```
## Payment method preferences
payment_preferences = df.groupby('Customer_Category')['Payment_Method'].value_count
print('\nPayment Method Preferences by Customer Category:')
print(payment_preferences)
```

```
Payment Method Preferences by Customer Category:
Payment_Method         Cash  Credit Card  Debit Card  Mobile Payment
Customer_Category
Homemaker          0.250044     0.250466    0.249685        0.249805
Middle-Aged        0.249350     0.249117    0.250313        0.251220
Professional       0.250668     0.250307    0.249641        0.249384
Retiree            0.250008     0.251119    0.248265        0.250608
Senior Citizen     0.250564     0.250428    0.250388        0.248619
Student            0.250773     0.248778    0.250613        0.249836
Teenager           0.248989     0.251095    0.249938        0.249978
Young Adult        0.251451     0.248553    0.251756        0.248240
```

In [14]:
```
## Average items bought per transaction per store type
avg_items_per_store = df.groupby('Store_Type')['Total_Items'].mean().sort_values(as
print("\nAverage Number of Items Bought per Transaction per Store Type:")
print(avg_items_per_store)
```

```
Average Number of Items Bought per Transaction per Store Type:
Store_Type
Specialty Store      5.508395
Convenience Store    5.505574
Pharmacy             5.498182
Department Store     5.495547
Supermarket          5.485767
Warehouse Club       5.482233
Name: Total_Items, dtype: float64
```

Task 4: Promotion & Discount Impact

- What is the average cost of transactions where a discount was applied vs not applied?
- Compare the average number of items purchased for different promotion types.
- Which promotion type seems to be most effective in terms of increasing total cost?

In [15]:
```
## Average cost of transactions with/without discounts
avg_cost_discount = df.groupby('Discount_Applied')['Total_Cost'].mean()
print('\nAverage Cost of Transactions with and without Discounts:')
print(avg_cost_discount)
```

```
Average Cost of Transactions with and without Discounts:
Discount_Applied
False    52.423512
True     52.486915
Name: Total_Cost, dtype: float64
```

In [16]:
```
## Average items purchased for different promotion types
avg_items_promotion = df.groupby('Promotion')['Total_Items'].mean().sort_values(asc
print("\nAverage Items Purchased for Different Promotion Types:")
print(avg_items_promotion)
```

```
Average Items Purchased for Different Promotion Types:
Promotion
Discount on Selected Items    5.501248
BOGO (Buy One Get One)        5.494351
Name: Total_Items, dtype: float64
```

In [17]:
```
## Effectiveness of promotion types
promotion_effectiveness = df.groupby('Promotion')['Total_Cost'].sum().sort_values(a
print('\nTotal Revenue by Promotion Type:')
print(promotion_effectiveness)
```

```
Total Revenue by Promotion Type:
Promotion
Discount on Selected Items    17462227.94
BOGO (Buy One Get One)        17438953.65
Name: Total_Cost, dtype: float64
```

Task 5: Seasonality Trends

- Which season has the highest total revenue?
- Are there seasonal preferences for certain store types or product categories?
- Create a plot showing average spending per season.

In [18]:
```
# --- Task 5: Seasonality Trends ---
print("\n--- Task 5: Seasonality Trends ---")
# Define seasons based on months
# Winter: Dec, Jan, Feb (12, 1, 2)
# Spring: Mar, Apr, May (3, 4, 5)
# Summer: Jun, Jul, Aug (6, 7, 8)
# Fall: Sep, Oct, Nov (9, 10, 11)
def get_season(month):
    if month in [12, 1, 2]:
        return 'Winter'
    elif month in [3, 4, 5]:
        return 'Spring'
    elif month in [6, 7, 8]:
        return 'Summer'
    elif month in [9, 10, 11]:
        return 'Fall'
    return 'Unknown' # Should not happen with valid months
df['Season'] = df['Month'].apply(get_season)
```

```
--- Task 5: Seasonality Trends ---
```

In [19]:
```
# highest total revenue of season
seasonal_revenue = df.groupby('Season')['Total_Cost'].sum().sort_values(ascending=F
print("\nTotal Revenue by Season:")
print(seasonal_revenue)
```

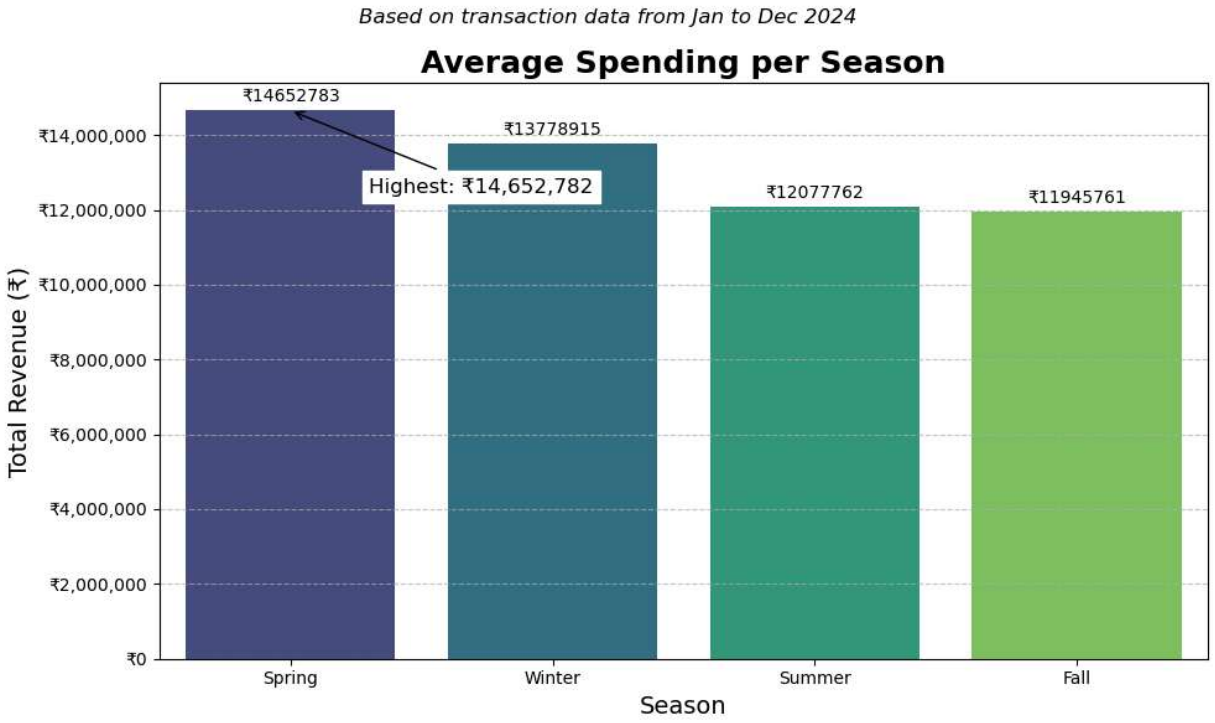
```
Total Revenue by Season:
Season
Spring    14652782.73
Winter    13778914.75
Summer    12077762.19
Fall      11945760.73
Name: Total_Cost, dtype: float64
```

In [20]:
```
## Seasonal preferences for store types/product categories
seasonal_preferences = df.groupby(['Season', 'Store_Type'], observed=True)['Total_C
print('\nSeasonal Preferences for Store Types:')
print(seasonal_preferences)
```

```
Seasonal Preferences for Store Types:
Store_Type  Convenience Store  Department Store    Pharmacy  Specialty Store  \
Season
Fall               1987200.76        1986928.25  1990334.12       1984214.54
Spring             2448609.02        2432843.96  2451323.08       2441711.02
Summer             2009498.09        2013509.03  2026540.49       1985213.90
Winter             2286593.49        2298274.33  2298481.32       2290460.76

Store_Type  Supermarket  Warehouse Club
Season
Fall         1992929.03      2004154.03
Spring       2439114.39      2439181.26
Summer       2017464.93      2025535.75
Winter       2313946.86      2291157.99
```

In [21]:

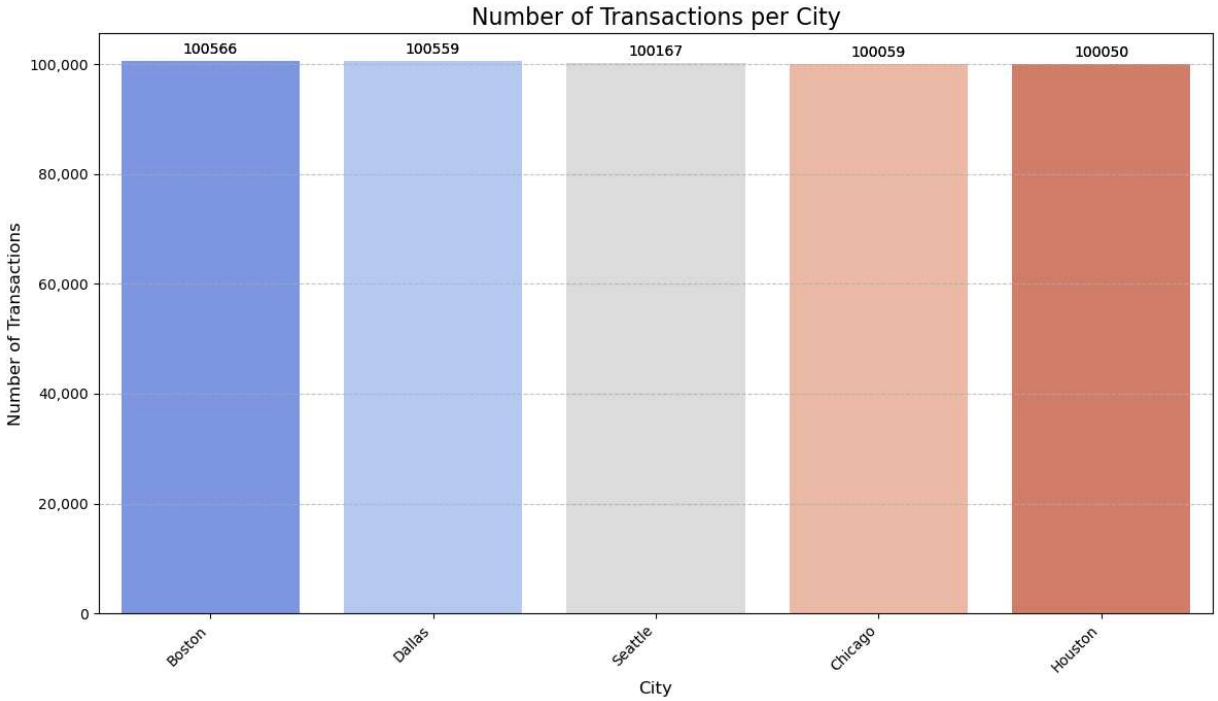
```
## Plot average spending per season
OUTPUT_DIR = 'plots'
os.makedirs(OUTPUT_DIR, exist_ok=True)

plt.figure(figsize=(10, 6))
ax = sns.barplot(x=seasonal_revenue.index,y=seasonal_revenue.values,hue=seasonal_re
plt.title('Average Spending per Season', fontsize=18, weight='bold')
plt.suptitle('Based on transaction data from Jan to Dec 2024', fontsize=12, style='
plt.xlabel('Season', fontsize=14)
plt.ylabel('Total Revenue (₹)', fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.7)
ax.yaxis.set_major_formatter(mtick.FuncFormatter(lambda x, _: f'₹{int(x):,}'))  # C
# Add value labels on top of bars
for container in ax.containers:
    ax.bar_label(container, fmt='₹%.0f', label_type='edge', padding=3)
# Annotate highest spending season
max_idx = seasonal_revenue.values.argmax()
max_season = seasonal_revenue.index[max_idx]
max_value = seasonal_revenue.values[max_idx]
ax.annotate(f'Highest: ₹{int(max_value):,}',xy=(max_idx, max_value), xytext=(max_id
plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, 'total_revenue_per_season.png'))
plt.show()
```



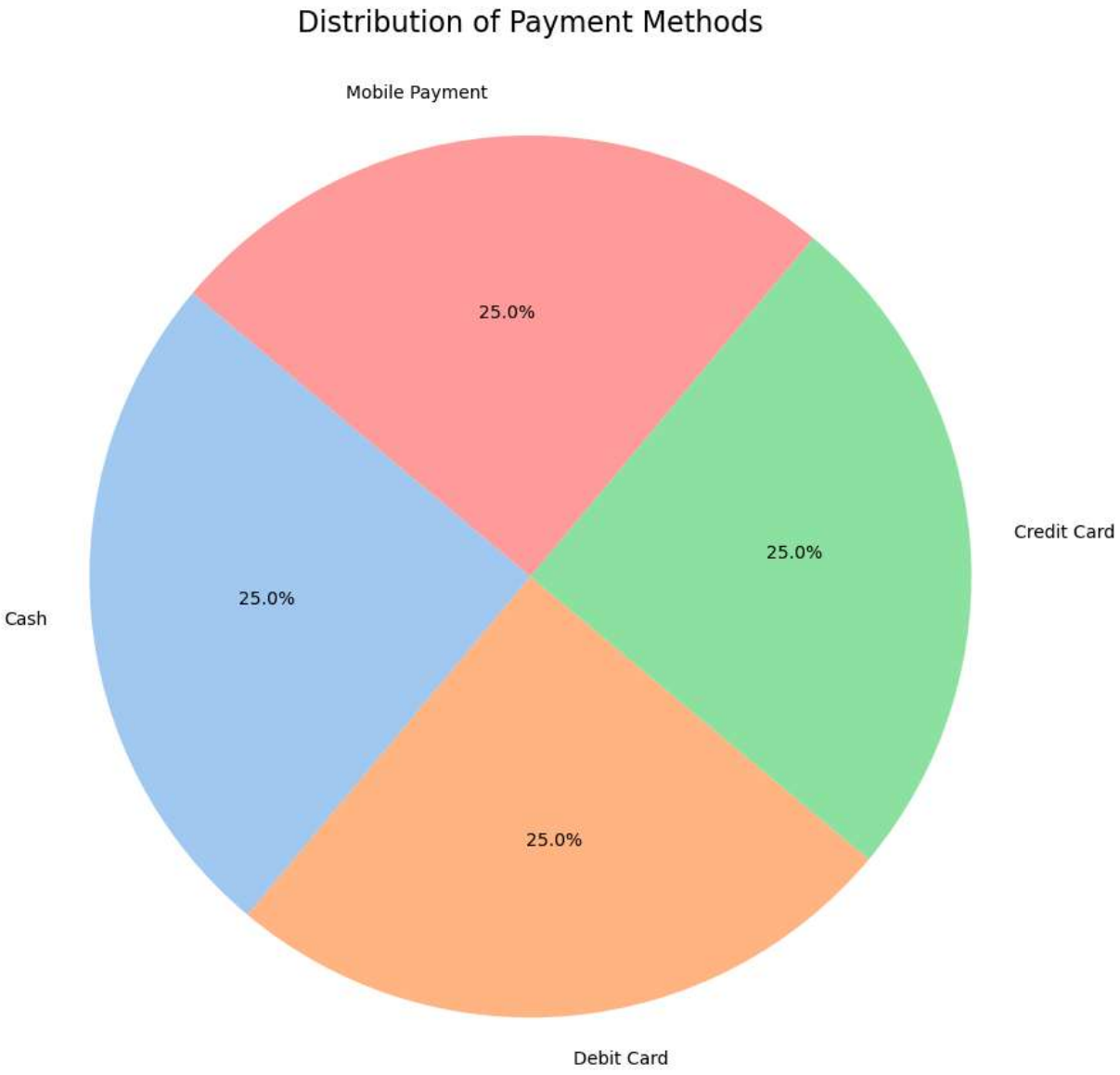
Task 6: Visualization Dashboard

- Bar plot of number of transactions per city
- Pie chart showing distribution of payment methods
- Line chart of monthly revenue trends (grouped by year if applicable)
- Heatmap or stacked bar showing revenue by season and customer category

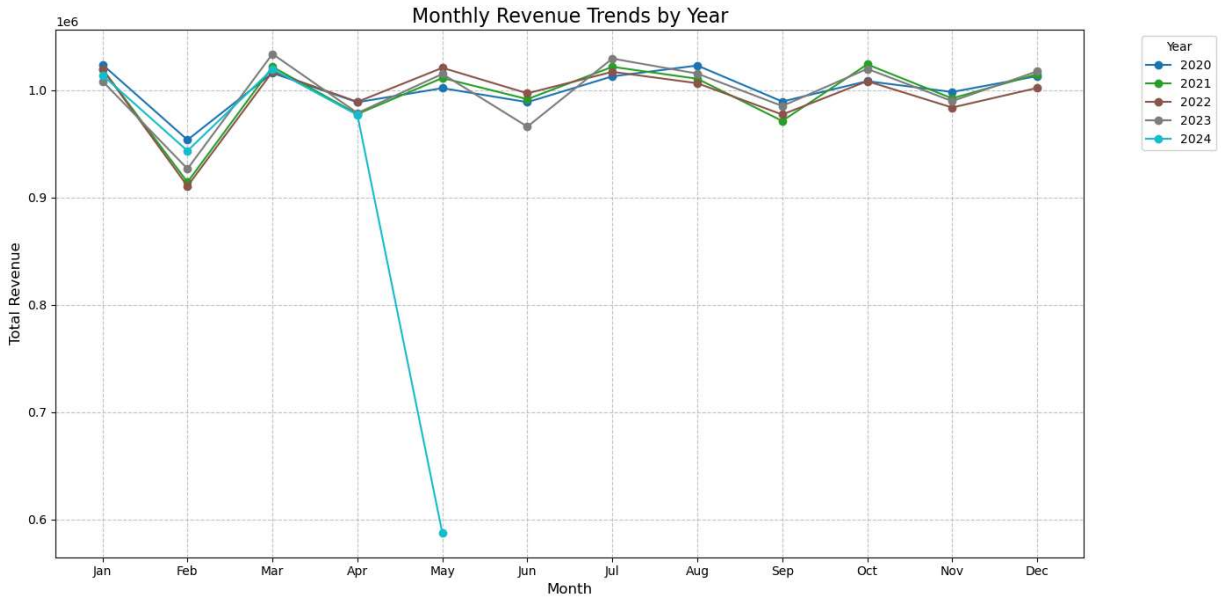
In [22]:
```
import matplotlib.ticker as mtick
## Bar plot of transactions per city
plt.figure(figsize=(12, 7))
sns.barplot(x=top_cities.index, y=top_cities.values, hue=top_cities.index, palette=
ax = sns.barplot(x=top_cities.index, y=top_cities.values, hue=top_cities.index, pal
for container in ax.containers:
    ax.bar_label(container, fmt='%.0f', label_type='edge', padding=3)
ax.yaxis.set_major_formatter(mtick.FuncFormatter(lambda x, _: f'{int(x):,}'))
plt.title('Number of Transactions per City', fontsize=16)
plt.xlabel('City', fontsize=12)
plt.ylabel('Number of Transactions', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, 'transactions_per_city_bar_plot.png'))
plt.show()
```



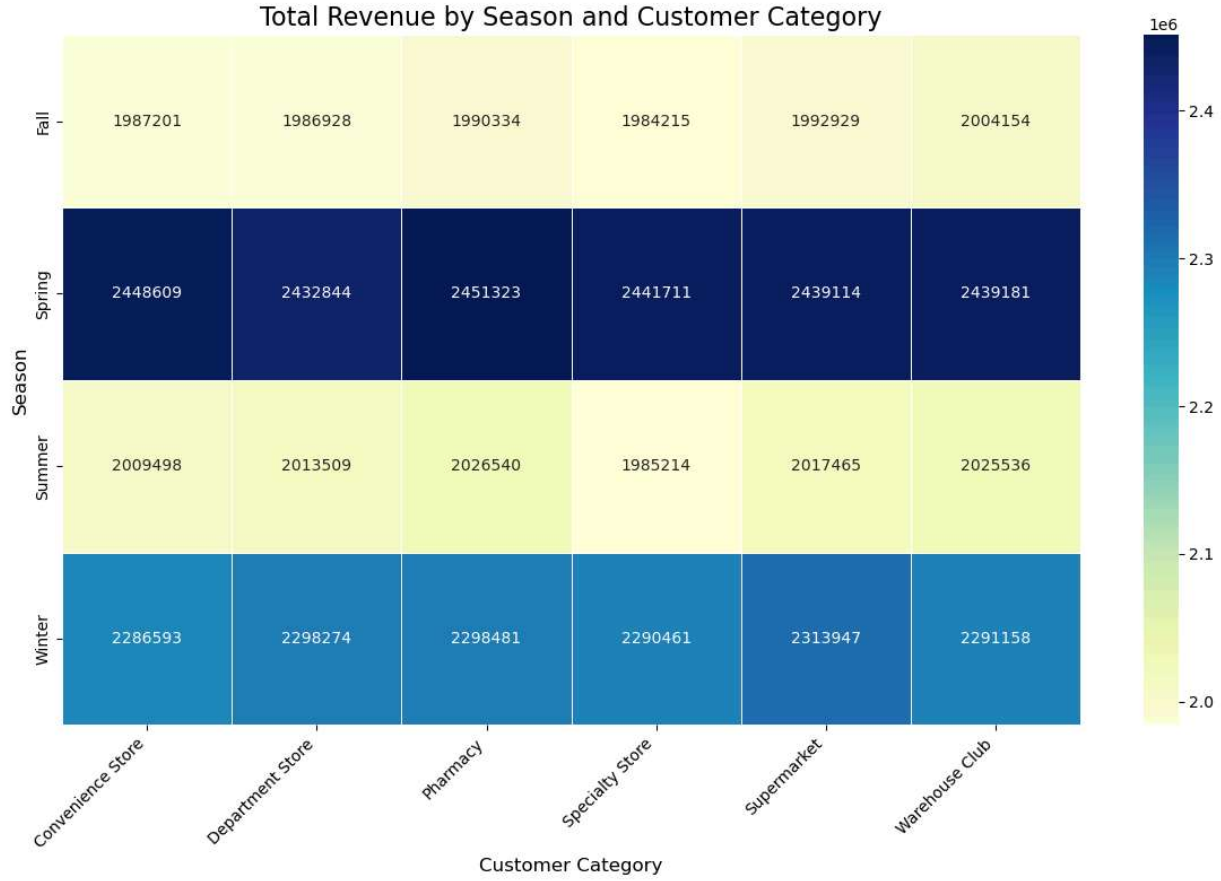
In [23]:
```
## Pie chart of payment method distribution
import os

OUTPUT_DIR = 'charts'  # or any folder name you prefer
os.makedirs(OUTPUT_DIR, exist_ok=True)
payment_distribution = df['Payment_Method'].value_counts()
plt.figure(figsize=(9, 9))
plt.pie(payment_distribution, labels=payment_distribution.index, autopct='%1.1f%%',
plt.title('Distribution of Payment Methods', fontsize=16)
plt.axis('equal')
plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, 'payment_method_distribution_pie_chart.png'))
plt.show()
```



In [24]:
```
## Line chart of monthly revenue trends
monthly_revenue = df.groupby(['Year', 'Month'])['Total_Cost'].sum().unstack(level=0
plt.figure(figsize=(14,7))
monthly_revenue.plot(kind='line',marker='o', ax=plt.gca(), colormap='tab10')
plt.title('Monthly Revenue Trends by Year', fontsize=16)
plt.xlabel('Month', fontsize=12)
plt.ylabel('Total Revenue', fontsize=12)
plt.xticks(ticks=range(1, 13), labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'J
plt.legend(title='Year', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(axis='both', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, 'monthly_revenue_trends_line_chart.png'))
plt.show()
```



In [27]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(seasonal_preferences, annot=True, fmt=".0f", cmap="YlGnBu", linewidths=
plt.title('Total Revenue by Season and Customer Category', fontsize=16)
plt.xlabel('Customer Category', fontsize=12)
plt.ylabel('Season', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.savefig(os.path.join(OUTPUT_DIR, 'revenue_season_customer_category_heatmap.png'
plt.show()
```



In [28]:
```
print(f"\nAnalysis complete. All plots saved to the '{OUTPUT_DIR}' directory.")
```

```
Analysis complete. All plots saved to the 'charts' directory.
```

In [ ]:

In [ ]:

In [ ]: