



Retails Sales Analysis

Required Library

```
In [5]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Load Data

```
In [6]: Retails = pd.read_csv(r"C:\Users\dell\OneDrive\Desktop\Retails_Store\retail_st
```

```
In [7]: # Read sample 15 rows
Retails.sample(15)
```

Out[7]:

	Transaction ID	Customer ID	Category	Item	Price Per Unit	Quantity	Total Spent
2733	TXN_1512224	CUST_21	Electric household essentials	Item_2_EHE	6.5	8.0	52.0
11592	TXN_7790991	CUST_25	Milk Products	Item_4_MILK	9.5	5.0	47.5
10155	TXN_3478143	CUST_08	Patisserie	Item_20_PAT	33.5	9.0	301.5
8368	TXN_6380866	CUST_22	Computers and electric accessories	NaN	41.0	NaN	NaN
4876	TXN_8182879	CUST_10	Electric household essentials	Item_15_EHE	26.0	5.0	130.0
12446	TXN_9422597	CUST_04	Milk Products	Item_2_MILK	6.5	9.0	58.5
11974	TXN_8011504	CUST_20	Patisserie	Item_17_PAT	29.0	8.0	232.0
5069	TXN_2587564	CUST_24	Food	Item_18_FOOD	30.5	7.0	213.5
11580	TXN_9702142	CUST_20	Electric household essentials	Item_11_EHE	20.0	10.0	200.0
4433	TXN_5509504	CUST_02	Furniture	NaN	NaN	4.0	74.0
1723	TXN_2486150	CUST_23	Furniture	Item_16_FUR	27.5	9.0	247.5
4548	TXN_7102763	CUST_12	Butchers	Item_12_BUT	21.5	9.0	193.5
5803	TXN_2560468	CUST_14	Patisserie	Item_20_PAT	33.5	1.0	33.5
871	TXN_5044125	CUST_02	Furniture	Item_25_FUR	41.0	7.0	287.0
12499	TXN_5602350	CUST_25	Food	Item_9_FOOD	17.0	7.0	119.0

Data Cleaning & Preparation

```
In [8]: # Shape of Data
Retails.shape
```

Out[8]: (12575, 11)

```
In [9]: # Information of Data
```

```
Retails.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12575 entries, 0 to 12574
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   Transaction ID         12575 non-null  object
1   Customer ID           12575 non-null  object
2   Category              12575 non-null  object
3   Item                  11362 non-null  object
4   Price Per Unit        11966 non-null  float64
5   Quantity              11971 non-null  float64
6   Total Spent           11971 non-null  float64
7   Payment Method        12575 non-null  object
8   Location              12575 non-null  object
9   Transaction Date      12575 non-null  object
10  Discount Applied      8376 non-null   object
dtypes: float64(3), object(8)
memory usage: 1.1+ MB
```

```
In [10]: # summary statistics
Retails.describe()
```

```
Out[10]:
```

	Price Per Unit	Quantity	Total Spent
count	11966.000000	11971.000000	11971.000000
mean	23.365912	5.536380	129.652577
std	10.743519	2.857883	94.750697
min	5.000000	1.000000	5.000000
25%	14.000000	3.000000	51.000000
50%	23.000000	6.000000	108.500000
75%	33.500000	8.000000	192.000000
max	41.000000	10.000000	410.000000

```
In [11]: for col in Retails.columns:
          print(Retails[col].value_counts())
          print("----"*50)
```

Transaction ID
TXN_6867343 1
TXN_3731986 1
TXN_9303719 1
TXN_9458126 1
TXN_4575373 1

TXN_9347481 1
TXN_4009414 1
TXN_5306010 1
TXN_5167298 1
TXN_2407494 1

Name: count, Length: 12575, dtype: int64

Customer ID

CUST_05 544
CUST_24 543
CUST_13 534
CUST_08 533
CUST_09 519
CUST_15 519
CUST_16 515
CUST_23 513
CUST_20 507
CUST_18 507
CUST_01 507
CUST_11 503
CUST_10 501
CUST_22 501
CUST_21 498
CUST_12 498
CUST_07 491
CUST_02 488
CUST_17 487
CUST_19 487
CUST_14 484
CUST_06 481
CUST_25 476
CUST_04 474
CUST_03 465

Name: count, dtype: int64

Category

Electric household essentials 1591
Furniture 1591
Food 1588
Milk Products 1584
Butchers 1568
Beverages 1567
Computers and electric accessories 1558
Patisserie 1528

Name: count, dtype: int64

```
-----  
-----  
Item  
Item_2_BEV      126  
Item_25_FUR     113  
Item_11_FUR     110  
Item_16_MILK    109  
Item_1_MILK     109
```

```
      ...  
Item_5_BEV       7  
Item_13_BEV      7  
Item_13_FUR      7  
Item_21_PAT      6  
Item_3_EHE       5
```

Name: count, Length: 200, dtype: int64

```
-----  
-----
```

Price Per Unit

```
33.5    678  
20.0    634  
21.5    630  
41.0    593  
29.0    554  
15.5    554  
38.0    542  
6.5     527  
12.5    522  
23.0    508  
26.0    507  
27.5    497  
11.0    497  
32.0    466  
36.5    451  
24.5    437  
14.0    421  
5.0     419  
9.5     414  
39.5    387  
8.0     380  
35.0    371  
30.5    355  
17.0    335  
18.5    287
```

Name: count, dtype: int64

```
-----  
-----
```

Quantity

```
10.0    1232  
5.0     1228  
7.0     1227  
8.0     1226  
3.0     1224  
6.0     1220  
2.0     1164
```

```
4.0      1155
9.0      1148
1.0      1147
Name: count, dtype: int64
```

Total Spent

```
40.0      140
80.0      124
25.0      122
140.0     118
20.0      108
```

```
...
129.5      25
85.0       24
274.5      24
37.0       22
166.5      17
```

```
Name: count, Length: 227, dtype: int64
```

Payment Method

```
Cash          4310
Digital Wallet 4144
Credit Card   4121
```

```
Name: count, dtype: int64
```

Location

```
Online      6354
In-store    6221
```

```
Name: count, dtype: int64
```

Transaction Date

```
30-05-2022    26
17-07-2023    24
16-03-2024    22
12-06-2023    22
23-01-2022    21
```

```
..
16-09-2022     3
19-12-2022     2
20-10-2022     2
17-03-2023     2
06-06-2023     1
```

```
Name: count, Length: 1114, dtype: int64
```

Discount Applied

```
True      4219
False     4157
```

```
Name: count, dtype: int64
```

```
In [12]: for col in Retails.columns:
          print(Retails[col].unique())
          print("----"*50)
```

['TXN_6867343' 'TXN_3731986' 'TXN_9303719' ... 'TXN_5306010' 'TXN_5167298'
'TXN_2407494']

['CUST_09' 'CUST_22' 'CUST_02' 'CUST_06' 'CUST_05' 'CUST_07' 'CUST_21'
'CUST_23' 'CUST_25' 'CUST_14' 'CUST_15' 'CUST_17' 'CUST_01' 'CUST_10'
'CUST_04' 'CUST_13' 'CUST_18' 'CUST_08' 'CUST_20' 'CUST_12' 'CUST_11'
'CUST_19' 'CUST_16' 'CUST_24' 'CUST_03']

['Patisserie' 'Milk Products' 'Butchers' 'Beverages' 'Food' 'Furniture'
'Electric household essentials' 'Computers and electric accessories']

['Item_10_PAT' 'Item_17_MILK' 'Item_12_BUT' 'Item_16_BEV' 'Item_6_FOOD'
nan 'Item_1_FOOD' 'Item_16_FUR' 'Item_22_BUT' 'Item_3_BUT' 'Item_2_FOOD'
'Item_24_PAT' 'Item_16_MILK' 'Item_17_PAT' 'Item_13_EHE' 'Item_7_BEV'
'Item_4_EHE' 'Item_10_FOOD' 'Item_14_FUR' 'Item_20_BUT' 'Item_25_FUR'
'Item_14_FOOD' 'Item_22_PAT' 'Item_11_FOOD' 'Item_6_PAT' 'Item_21_EHE'
'Item_25_BEV' 'Item_23_FOOD' 'Item_10_FUR' 'Item_11_BEV' 'Item_23_BUT'
'Item_22_BEV' 'Item_10_EHE' 'Item_24_BUT' 'Item_8_BEV' 'Item_3_FOOD'
'Item_12_FOOD' 'Item_16_CEA' 'Item_11_PAT' 'Item_16_BUT' 'Item_5_CEA'
'Item_19_MILK' 'Item_23_FUR' 'Item_7_FUR' 'Item_15_CEA' 'Item_6_MILK'
'Item_24_CEA' 'Item_22_CEA' 'Item_22_FOOD' 'Item_2_BUT' 'Item_14_PAT'
'Item_12_PAT' 'Item_18_FOOD' 'Item_1_PAT' 'Item_4_BEV' 'Item_22_FUR'
'Item_7_PAT' 'Item_20_CEA' 'Item_20_FOOD' 'Item_11_FUR' 'Item_25_PAT'
'Item_7_FOOD' 'Item_21_FUR' 'Item_24_FUR' 'Item_8_MILK' 'Item_4_FOOD'
'Item_14_BEV' 'Item_4_PAT' 'Item_4_MILK' 'Item_7_CEA' 'Item_6_EHE'
'Item_21_BUT' 'Item_16_PAT' 'Item_25_CEA' 'Item_8_BUT' 'Item_10_CEA'
'Item_5_FUR' 'Item_9_FOOD' 'Item_21_CEA' 'Item_8_CEA' 'Item_8_EHE'
'Item_23_MILK' 'Item_23_BEV' 'Item_19_BEV' 'Item_20_BEV' 'Item_24_FOOD'
'Item_21_MILK' 'Item_6_BEV' 'Item_1_MILK' 'Item_24_MILK' 'Item_2_CEA'
'Item_18_BUT' 'Item_1_FUR' 'Item_3_MILK' 'Item_11_MILK' 'Item_13_CEA'
'Item_6_CEA' 'Item_2_FUR' 'Item_21_BEV' 'Item_8_FUR' 'Item_13_BUT'
'Item_2_BEV' 'Item_7_EHE' 'Item_14_CEA' 'Item_19_EHE' 'Item_18_CEA'
'Item_11_CEA' 'Item_17_FUR' 'Item_15_MILK' 'Item_20_EHE' 'Item_16_EHE'
'Item_23_EHE' 'Item_7_BUT' 'Item_1_EHE' 'Item_19_CEA' 'Item_25_FOOD'
'Item_12_EHE' 'Item_22_EHE' 'Item_13_PAT' 'Item_17_EHE' 'Item_25_BUT'
'Item_4_CEA' 'Item_2_MILK' 'Item_1_BUT' 'Item_12_BEV' 'Item_5_FOOD'
'Item_25_EHE' 'Item_9_CEA' 'Item_1_CEA' 'Item_15_FUR' 'Item_15_PAT'
'Item_5_EHE' 'Item_1_BEV' 'Item_17_BUT' 'Item_3_BEV' 'Item_13_FOOD'
'Item_11_EHE' 'Item_9_MILK' 'Item_17_FOOD' 'Item_20_PAT' 'Item_9_PAT'
'Item_10_BEV' 'Item_17_CEA' 'Item_8_PAT' 'Item_13_MILK' 'Item_5_BUT'
'Item_22_MILK' 'Item_4_FUR' 'Item_17_BEV' 'Item_19_PAT' 'Item_2_PAT'
'Item_14_BUT' 'Item_20_FUR' 'Item_6_BUT' 'Item_9_FUR' 'Item_12_CEA'
'Item_15_EHE' 'Item_5_PAT' 'Item_18_MILK' 'Item_6_FUR' 'Item_24_BEV'
'Item_14_MILK' 'Item_12_FUR' 'Item_18_BEV' 'Item_23_CEA' 'Item_24_EHE'
'Item_2_EHE' 'Item_23_PAT' 'Item_15_FOOD' 'Item_8_FOOD' 'Item_15_BEV'
'Item_20_MILK' 'Item_9_EHE' 'Item_11_BUT' 'Item_18_EHE' 'Item_5_MILK'
'Item_3_FUR' 'Item_19_BUT' 'Item_3_CEA' 'Item_19_FUR' 'Item_7_MILK'
'Item_9_BEV' 'Item_10_BUT' 'Item_18_FUR' 'Item_25_MILK' 'Item_4_BUT'
'Item_15_BUT' 'Item_21_PAT' 'Item_21_FOOD' 'Item_13_BEV' 'Item_5_BEV'
'Item_3_PAT' 'Item_13_FUR' 'Item_18_PAT' 'Item_12_MILK' 'Item_10_MILK'
'Item_16_FOOD' 'Item_19_FOOD' 'Item_14_EHE' 'Item_3_EHE' 'Item_9_BUT']

```
[18.5 29. 21.5 27.5 12.5 nan 5. 33.5 36.5 8. 6.5 39.5 24.5 23.
35. 14. 9.5 41. 20. 38. 15.5 11. 32. 26. 30.5 17. ]
```

```
[10. 9. 2. 7. 8. nan 1. 3. 6. 4. 5.]
```

```
[185. 261. 43. 247.5 87.5 200. 40. nan 27.5 109.5 72. 52.
45.5 237. 55. 232. 275. 23. 126. 105. 66.5 18.5 49. 134.
410. 245. 182.5 100. 196. 315. 287. 76. 92.5 190. 255.5 276.5
46.5 8. 107.5 165. 80. 192.5 335. 66. 215. 96. 42. 11.
234. 316. 82.5 180. 365. 39. 172. 122. 30. 84. 320. 219.
67. 146. 290. 70. 160. 82. 14. 355.5 124. 28.5 47.5 193.5
38. 12.5 140. 120. 183. 305. 41. 155. 19.5 33. 108.5 119.
280. 62. 32. 380. 304. 139.5 114. 192. 167.5 88. 395. 158.
25. 50. 32.5 15. 33.5 24. 111. 46. 36.5 62.5 161. 26.
98. 85. 228. 91.5 93. 35. 208. 100.5 9.5 10. 138. 288.
60. 123. 73. 197.5 69. 61. 116. 55.5 6.5 152.5 118.5 74.
205. 150.5 110. 19. 48. 175. 369. 37.5 65. 170. 77.5 129.5
15.5 24.5 147. 21.5 31. 99. 128. 58.5 266. 29. 268. 64.
115. 64.5 260. 207. 224. 328. 92. 244. 39.5 16. 328.5 182.
44. 129. 301.5 57. 5. 122.5 220. 112.5 68. 292. 28. 342.
102. 77. 148. 125. 210. 137.5 95. 136. 145. 153. 79. 78.
51. 256. 152. 230. 274.5 184. 171.5 20. 58. 213.5 203. 104.
201. 234.5 87. 56. 86. 30.5 85.5 174. 22. 34. 156. 350.
75. 246. 220.5 166.5 45. 112. 164. 130. 37. 73.5 13. 17. ]
```

```
['Digital Wallet' 'Credit Card' 'Cash']
```

```
['Online' 'In-store']
```

```
['08-04-2024' '23-07-2023' '05-10-2022' ... '17-03-2023' '29-02-2024'
'16-09-2022']
```

```
[True False nan]
```

```
In [13]: # duplicate rows summary
print('Duplicate Rows Count:', Retails[Retails.duplicated()].shape[0])
```

Duplicate Rows Count: 0

```
In [14]: # Missing Values
Null_Summary = pd.DataFrame({"Num_of_Null":Retails.isna().sum(),"Percentage of
Null_Summary
```

Out[14]:

	Num_of_Null	Percentage of Null
Transaction ID	0	0.00
Customer ID	0	0.00
Category	0	0.00
Item	1213	9.65
Price Per Unit	609	4.84
Quantity	604	4.80
Total Spent	604	4.80
Payment Method	0	0.00
Location	0	0.00
Transaction Date	0	0.00
Discount Applied	4199	33.39

```
In [15]: # Make a copy to keep original data safe
Retails_clean = Retails.copy()
```

```
In [16]: # Rename Columns
Retails_clean.columns = Retails_clean.columns.str.replace(" ", "_")
```

```
In [17]: # Change boolean to String
Retails_clean["Discount_Applied"] = (
    Retails_clean["Discount_Applied"]
    .fillna("Unknown")
    .astype(str)
)
```

```
In [18]: # Handling Categorical Missing values
Cat_columns = Retails_clean.select_dtypes("object")
for col in Cat_columns:
    Retails_clean[col].fillna(Retails_clean[col].mode()[0],inplace=True)
```

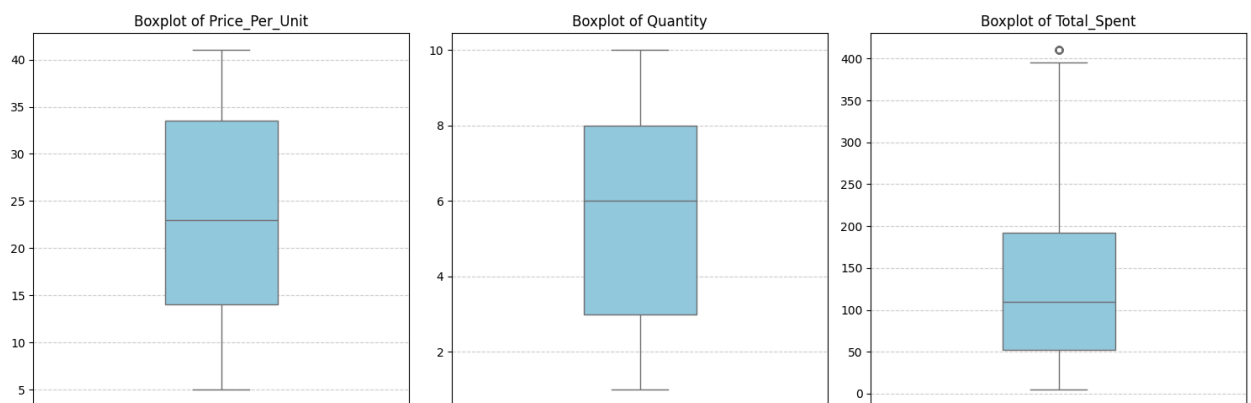
```
In [19]: # Handling Numerical Missing Values
Retails_clean["Quantity"].fillna(Retails_clean["Quantity"].median(),inplace=True)
Retails_clean["Price_Per_Unit"].fillna(
    Retails_clean["Total_Spent"]/Retails_clean["Quantity"],inplace=True)
Retails_clean["Total_Spent"].fillna(
    Retails_clean["Quantity"]*Retails_clean["Price_Per_Unit"],inplace=True
)
```

```
In [20]: # Handing Data column
Retails_clean["Transaction_Date"] = pd.to_datetime(Retails_clean["Transaction_
```

```
In [21]: Retails_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12575 entries, 0 to 12574
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   Transaction_ID         12575 non-null  object
1   Customer_ID            12575 non-null  object
2   Category               12575 non-null  object
3   Item                   12575 non-null  object
4   Price_Per_Unit         12575 non-null  float64
5   Quantity               12575 non-null  float64
6   Total_Spent            12575 non-null  float64
7   Payment_Method         12575 non-null  object
8   Location               12575 non-null  object
9   Transaction_Date       12575 non-null  datetime64[ns]
10  Discount_Applied       12575 non-null  object
dtypes: datetime64[ns](1), float64(3), object(7)
memory usage: 1.1+ MB
```

```
In [22]: # Check Outlier
cols = ["Price_Per_Unit", "Quantity", "Total_Spent"]
plt.figure(figsize=(15,5))
for i, col in enumerate(cols,1):
    plt.subplot(1,3,i)
    sns.boxplot(y=Retails_clean[col], color='skyblue', width=0.3)
    plt.title(f'Boxplot of {col}', fontsize=12)
    plt.ylabel('')
    plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



Export Clean Data in DataBase

```
In [23]: import urllib.parse
from sqlalchemy import create_engine
import time
username = "root"
raw_password = "ANku7970@!#"

```

```

password = urllib.parse.quote_plus(raw_password)

host = "localhost"
port = 3306
database = "Retail_Analysis"

engine = create_engine(f"mysql+pymysql://{username}:{password}@{host}:{port}/{database}")

try:
    print("Attempting to connect to database...")
    for x in range(1,4):
        print(x)
        time.sleep(1)
    Retails_clean.to_sql("Retails_clean",engine,
        if_exists="replace",index=False)
    print("Connected ✅ Data Loaded Successfully")

except Exception as e:
    print("Not Connected❌")
    print("Reason",e)

```

Attempting to connect to database...

1

2

3

Connected ✅ Data Loaded Successfully

Exploratory Data Analysis & Visualization

In [24]: *# Set theme for Visualization*
`sns.set_theme(style="dark")`

In [25]: `Retails_clean.describe()`

Out[25]:

	Price_Per_Unit	Quantity	Total_Spent	Transaction_Date
count	12575.000000	12575.000000	12575.000000	12575
mean	23.369304	5.558648	130.208111	2023-07-15 00:59:05.320079360
min	5.000000	1.000000	5.000000	2022-01-01 00:00:00
25%	14.000000	3.000000	52.000000	2022-10-03 00:00:00
50%	23.000000	6.000000	110.000000	2023-07-16 00:00:00
75%	33.500000	8.000000	192.000000	2024-04-24 00:00:00
max	41.000000	10.000000	410.000000	2025-12-01 00:00:00
std	10.748728	2.790160	93.580667	NaN

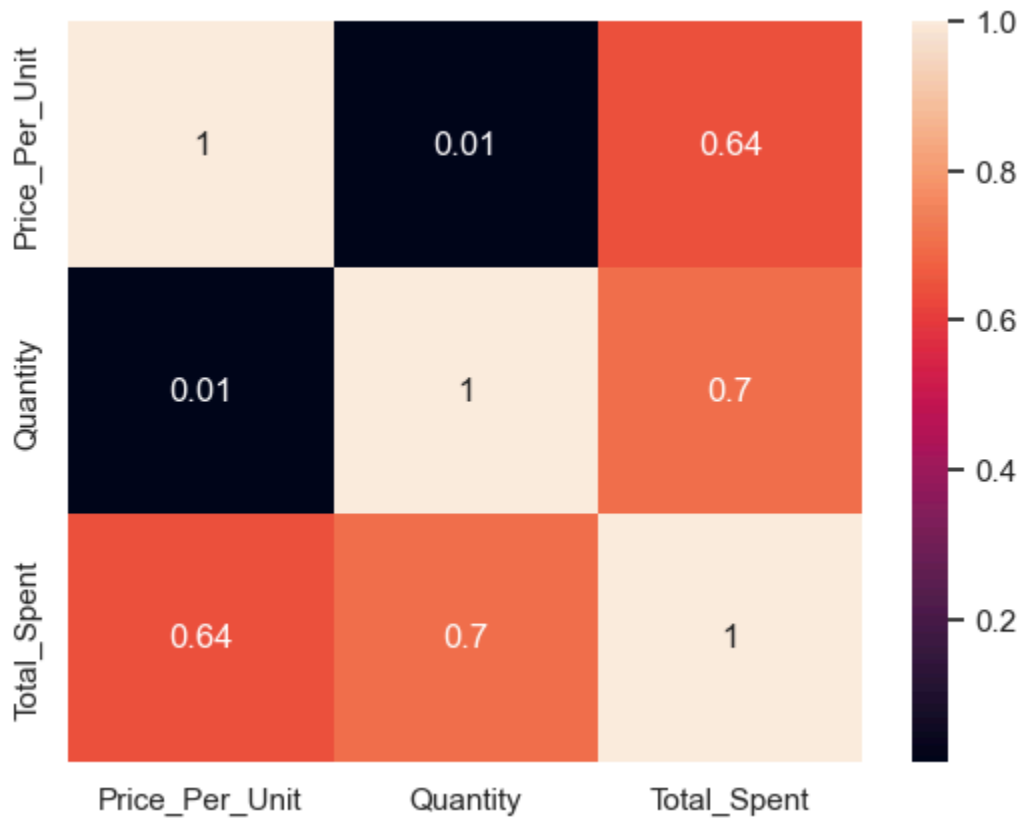
In [26]: `Retails_clean.describe(include="object")`

Out[26]:

	Transaction_ID	Customer_ID	Category	Item	Payment_Method	L
count	12575	12575	12575	12575	12575	
unique	12575	25	8	200	3	
top	TXN_6867343	CUST_05	Electric household essentials	Item_2_BEV	Cash	
freq	1	544	1591	1339	4310	

In [27]: *# Correlation with numerical column*
 Num_Col = Retails_clean.select_dtypes("float64")
 sns.heatmap(Num_Col.corr(),annot=True)

Out[27]: <Axes: >



Insights

1. Price_Per_Unit vs Quantity (Correlation ≈ 0.01)

- There is almost no correlation between price per unit and quantity purchased.
- This indicates that customers do not significantly change the quantity

they buy based on unit price.

- Buying behavior appears price-independent in terms of quantity.

2. Price_Per_Unit vs Total_Spent (Correlation ≈ 0.64)

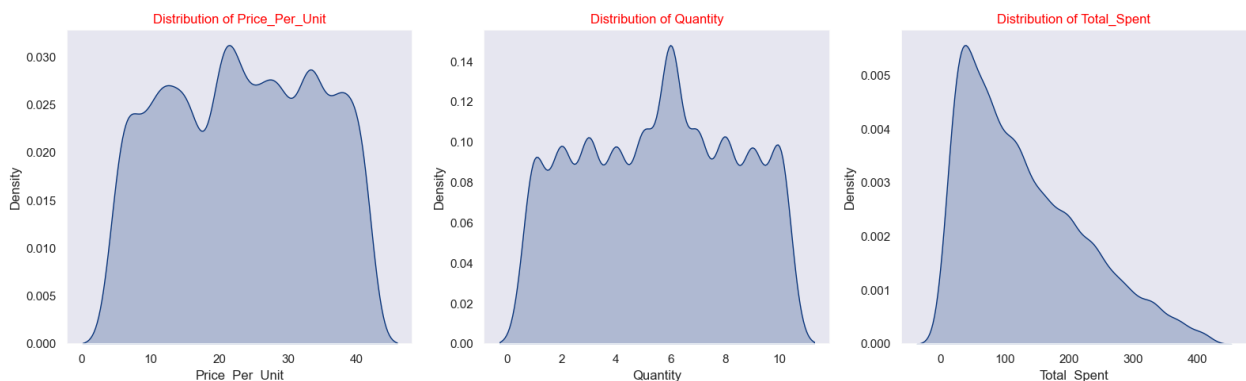
- A moderate positive correlation exists between price per unit and total spending.
- Higher-priced items naturally contribute to higher total transaction value.
- Unit price plays an important role in driving revenue.

3. Quantity vs Total_Spent (Correlation ≈ 0.70)

- Quantity has a strong positive relationship with total spending.
- As the number of items purchased increases, total sales increase significantly.
- Quantity is a key driver of revenue growth.

Distribution of Numerical Columns

```
In [28]: plt.figure(figsize=(16,5))
for x,col in enumerate(Num_Col):
    plt.subplot(1,3,x+1)
    sns.kdeplot(x = col, fill=True,data= Retails_clean,color="#09347c")
    plt.title(f"Distribution of {col}",color = "Red")
plt.tight_layout()
plt.show()
```



Insights:

1 Distribution of Price_Per_Unit

- Prices are fairly evenly distributed across the range.
- No extreme skewness or abnormal spikes are observed.

- This suggests a balanced product pricing strategy across categories.

2. Distribution of Quantity

- Quantity shows a central concentration around mid-range values (around 5–6 units).
- Extremely low or very high quantities are less frequent.
- This indicates typical customer purchases involve moderate quantities.

3. Distribution of Total_Spent

- The distribution is right-skewed.
- Most transactions have low to medium total spending, while a few high-value transactions exist.
- These high-value transactions create a long tail in the distribution.

General Analysis

- What is the total revenue generated?
- What is the total number of transactions?
- What is the average revenue per transaction?
- What is the total quantity sold?

```
In [29]: Total_Revenue = np.sum(Retails_clean["Total_Spent"])
Total_Transaction = Retails_clean["Transaction_ID"].count()
Avg_Revenue_Per_Trans = Total_Revenue/Total_Transaction
Total_QTY = np.sum(Retails_clean["Quantity"])

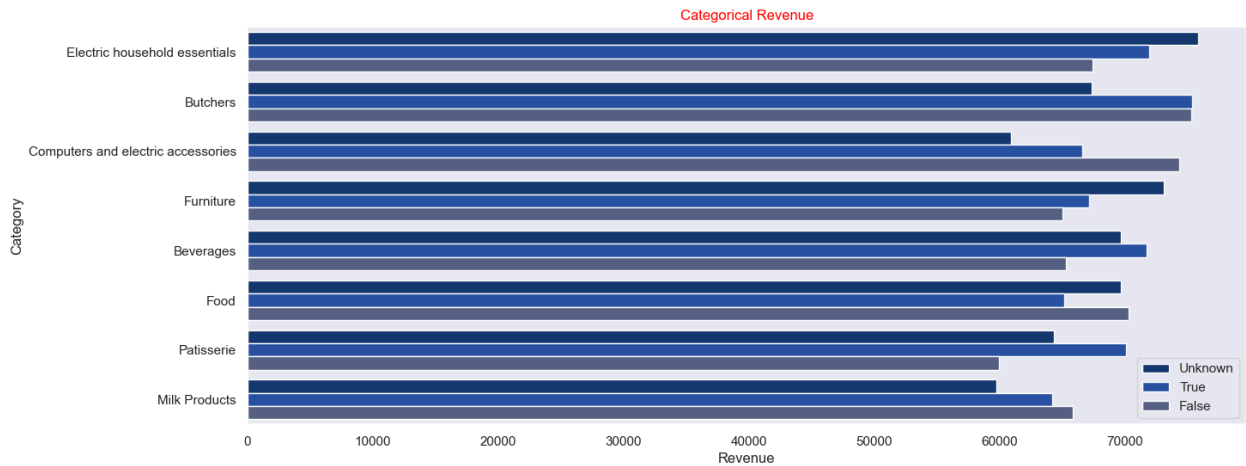
print("-----General Analysis-----")
print(f"Total Revenue: {Total_Revenue}")
print(f"Total Transaction: {Total_Transaction}")
print(f"Avg Revenue Per Transaction: {round(Avg_Revenue_Per_Trans,2)}")
print(f"Total Quantity: {Total_QTY}")
print("-----")
```

```
-----General Analysis-----
Total Revenue: 1637367.0
Total Transaction: 12575
Avg Revenue Per Transaction: 130.21
Total Quantity: 69900.0
-----
```

Categorical Analysis

```
In [30]: Cate_Revenue = pd.read_sql_query("Select Category, Discount_Applied, sum(Total
from Retails_Clean group by Category , Discount_Applied order by Revenue desc
plt.figure(figsize=(15,6))
```

```
sns.barplot(y = "Category", x = "Revenue", hue="Discount_Applied", data=Cate_Re
plt.title("Categorical Revenue",color = "Red")
plt.legend()
plt.show()
```

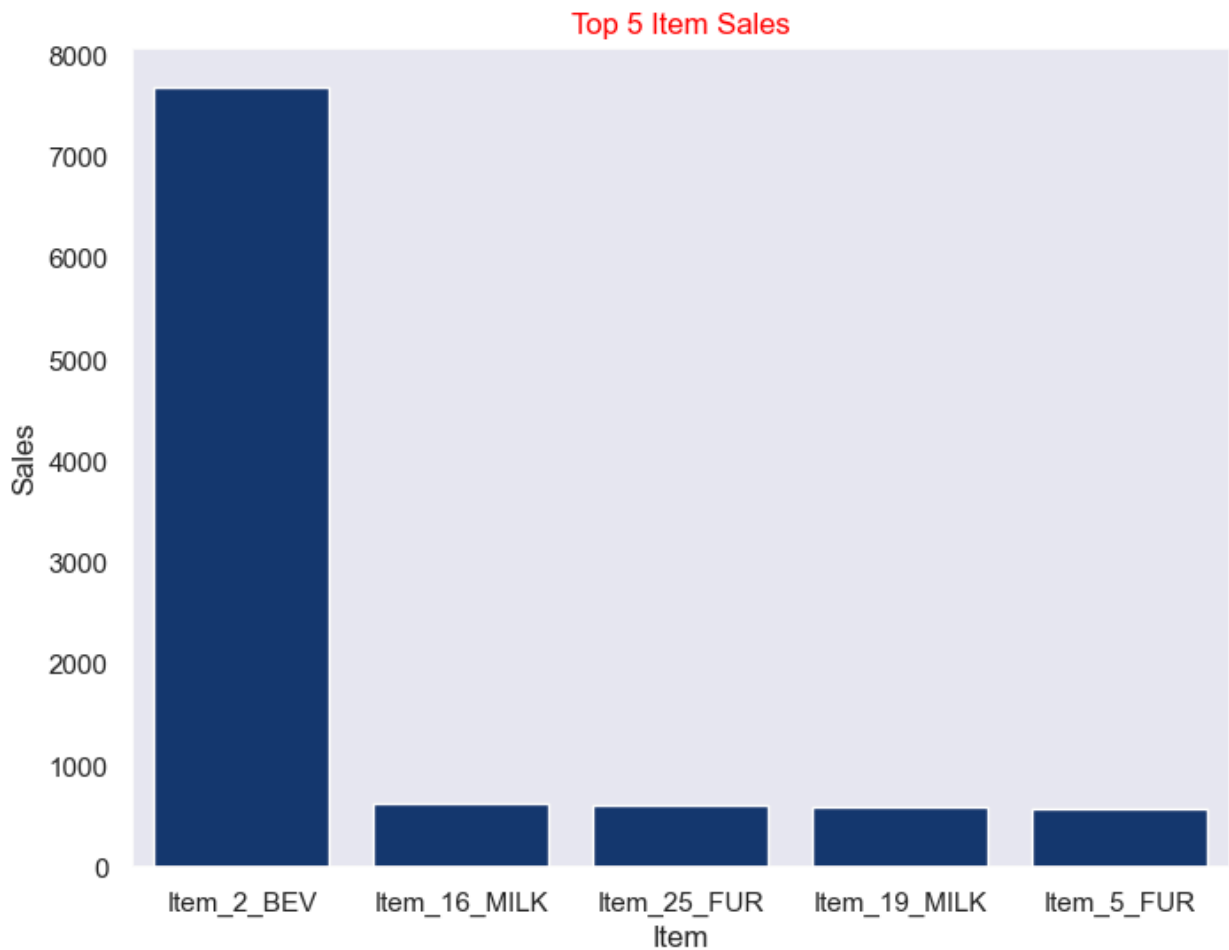


Insights

- Electric household essentials, Butchers, and Beverages generate the highest revenue across categories.
- Discount-applied sales generally show higher or comparable revenue, indicating discounts help boost overall spending.
- Categories like Milk Products and Patisserie contribute moderate but stable revenue, showing consistent demand.

Item Analysis

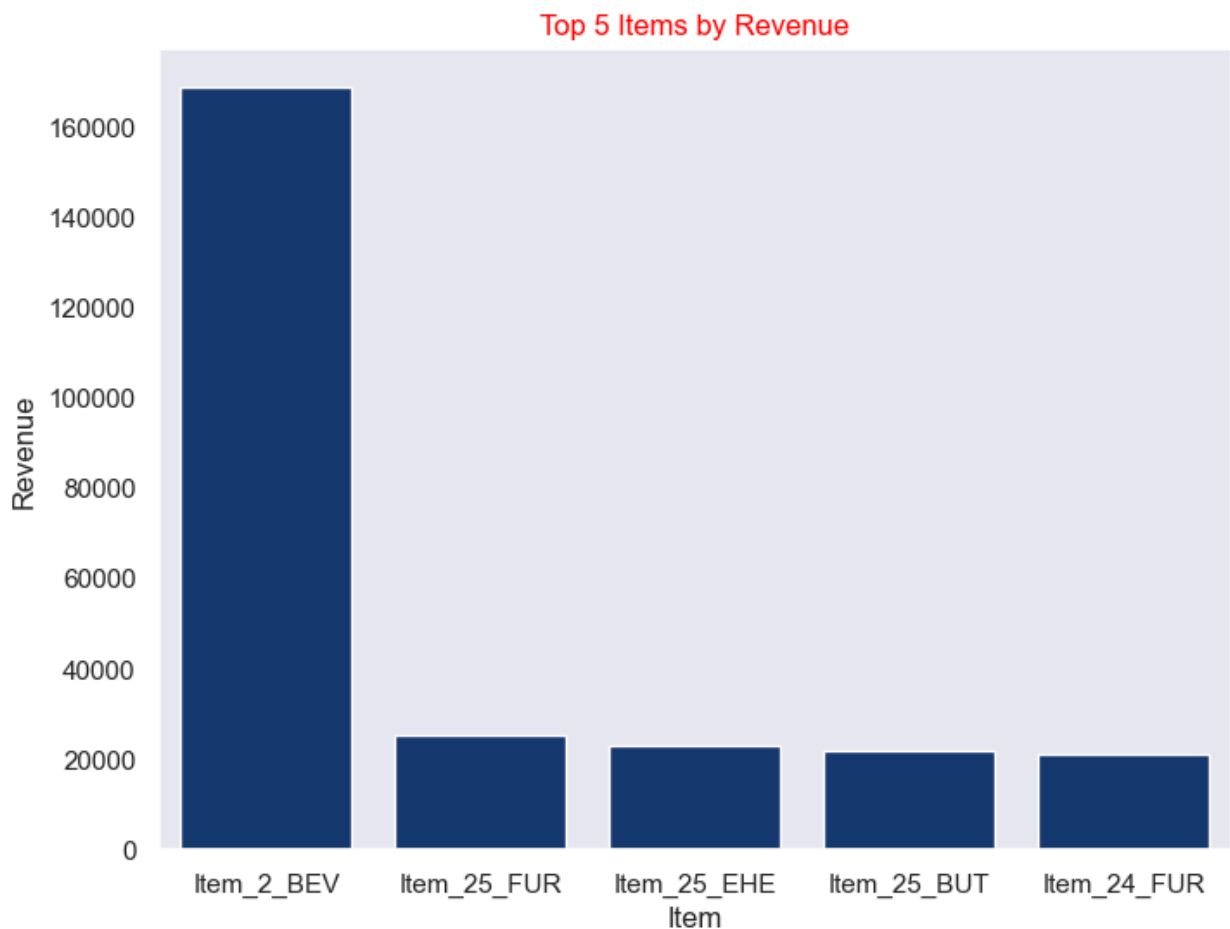
```
In [31]: Item_Sales = pd.read_sql("Select Item, sum(Quantity) as Sales" \
" from Retails_Clean group by Item order by Sales desc limit 5",engine)
plt.figure(figsize=(8,6))
sns.barplot(x = "Item", y = "Sales", data=Item_Sales, color="#09347c")
plt.title("Top 5 Item Sales",color = "Red")
plt.show()
```



Insights

- Item_2_BEV is the top-selling product by quantity, showing very high customer demand.
- There is a large gap between Item_2_BEV and other top items, indicating sales concentration on one product.
- Beverage and Milk categories dominate the top sales, highlighting daily-use items as key revenue drivers

```
In [32]: Item_Revenue =pd.read_sql(" Select Item , sum(Total_Spent) as Revenue from Ret
plt.figure(figsize=(8,6))
sns.barplot(x = "Item", y = "Revenue", data=Item_Revenue,color="#09347c")
plt.title("Top 5 Items by Revenue", color = "Red")
plt.show()
```



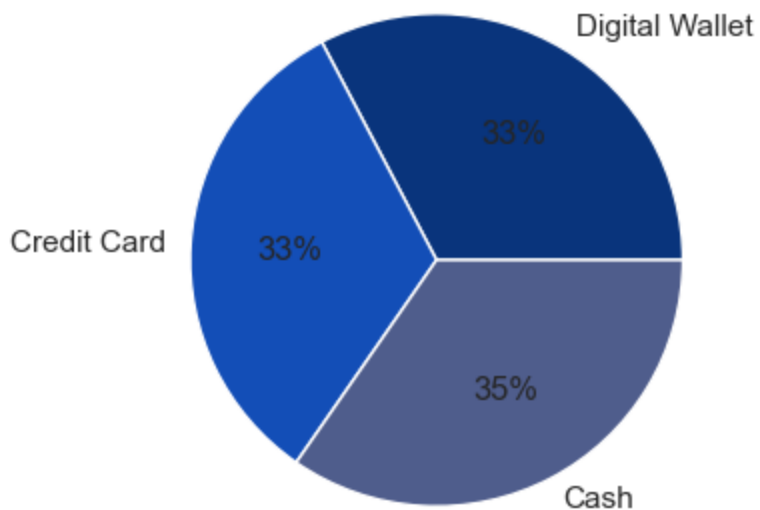
Insights

- Item_2_BEV generates the highest revenue by a large margin, making it the top revenue-driving product.
- The remaining top items contribute significantly lower but similar revenue, indicating revenue concentration on one key item.
- This suggests that high demand combined with strong pricing/volume makes Item_2_BEV the most valuable product.

Payment Methods

```
In [33]: Payment_Revenue = pd.read_sql_query("Select Payment_Method, sum(Total_Spent) "
      "as Revenue from Retails_Clean group by Payment_Method", engine)
      plt.figure(figsize=(6,4))
      plt.pie(Payment_Revenue["Revenue"], labels= Payment_Revenue["Payment_Method"],
      autopct= "%1.0f%%", colors= ["#09347c", "#134eb7", "#4f5d8c"])
      plt.title("Revenue by Payment Methods ", color = "red")
      plt.show()
```

Revenue by Payment Methods



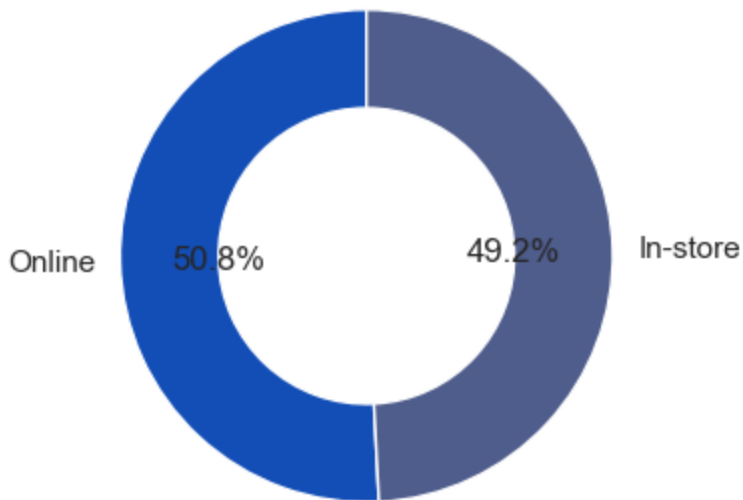
Insights

- Cash contributes the highest share of total revenue, making it the most preferred payment method.
- Credit Card and Digital Wallet revenues are almost equal, showing balanced adoption of digital payments.
- Despite digital options, cash remains dominant in overall customer transactions.

Location Revenue

```
In [34]: Location_Revenue = pd.read_sql_query("Select Location, sum(Total_Spent) as Revenue  
from Retails_clean group by Location",engine)  
plt.figure(figsize=(7,4))  
plt.title("Revenue by Location", color="Red")  
plt.pie(Location_Revenue["Revenue"], labels=Location_Revenue["Location"],  
        colors=["#134eb7", "#4f5d8c"], autopct="%1.1f%%",  
        wedgeprops={"width":0.4}, startangle=90)  
plt.show()
```

Revenue by Location

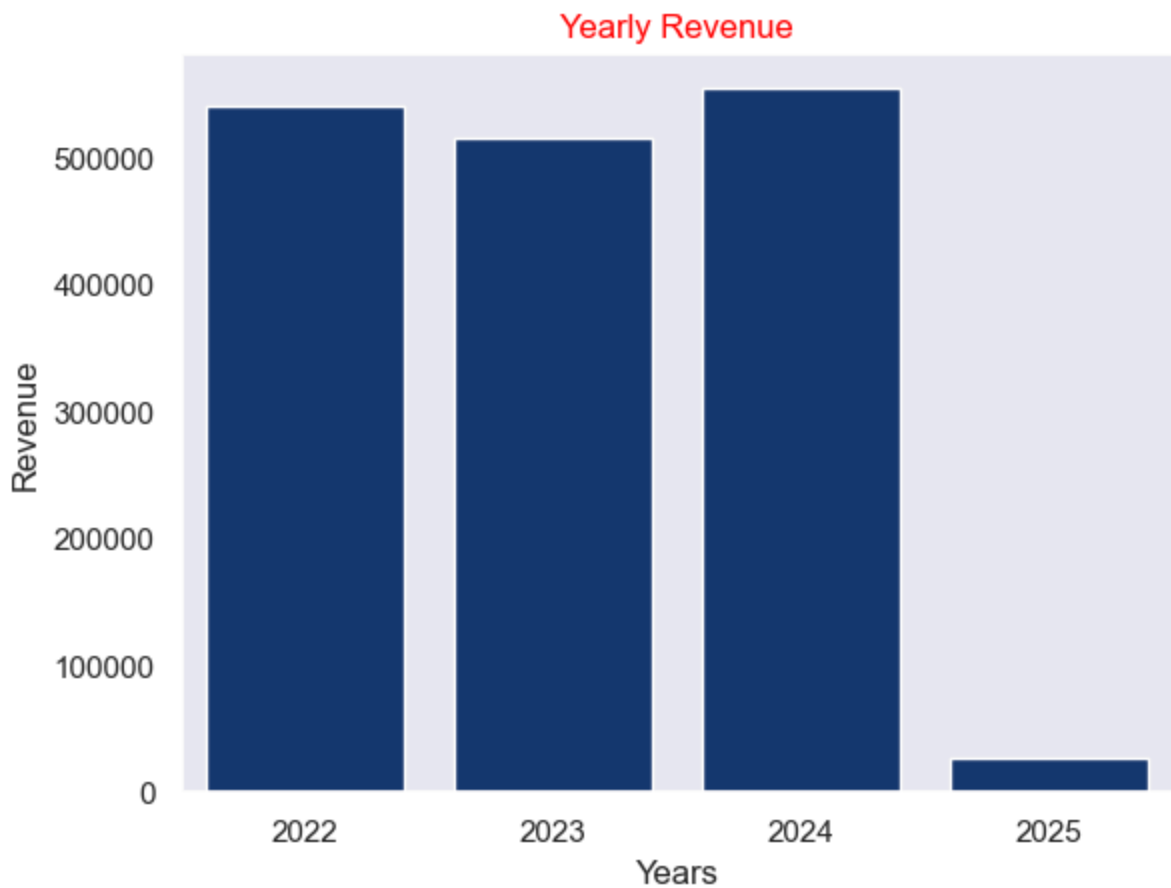


Insights

- Online sales generate slightly higher revenue than in-store, though both contribute almost equally.
- Even this small difference is important, as it shows a growing shift toward online purchasing.
- Maintaining in-store performance while strengthening online channels can maximize total revenue.

Yearly Revenue

```
In [35]: Year_Rev = pd.read_sql_query("Select Year(Transaction_Date) as Years , " \
    "sum(Total_Spent) as Revenue from Retails_Clean group by Years", engine)
sns.barplot(x = "Years",y="Revenue",data=Year_Rev,color="#09347c")
plt.title("Yearly Revenue", color = "Red")
plt.show()
```



Insights

- 2024 records the highest revenue, showing peak business performance.
- Revenue in 2022 and 2023 remains strong and stable, indicating consistent growth.
- 2025 revenue is low as it likely represents partial or ongoing year data.

Monthly Revenue

```
In [36]: Monthly_Rev = pd.read_sql_query("Select MonthName(Transaction_Date) as Months,
sum(Total_Spent) as Revenue from Retails_Clean group by Months", engine)
plt.figure(figsize=(13,7))
sns.lineplot(x = "Months", y = "Revenue", data = Monthly_Rev, color = "#09347c")
plt.title("Monthly Revenue", color = "Red")
plt.show()
```

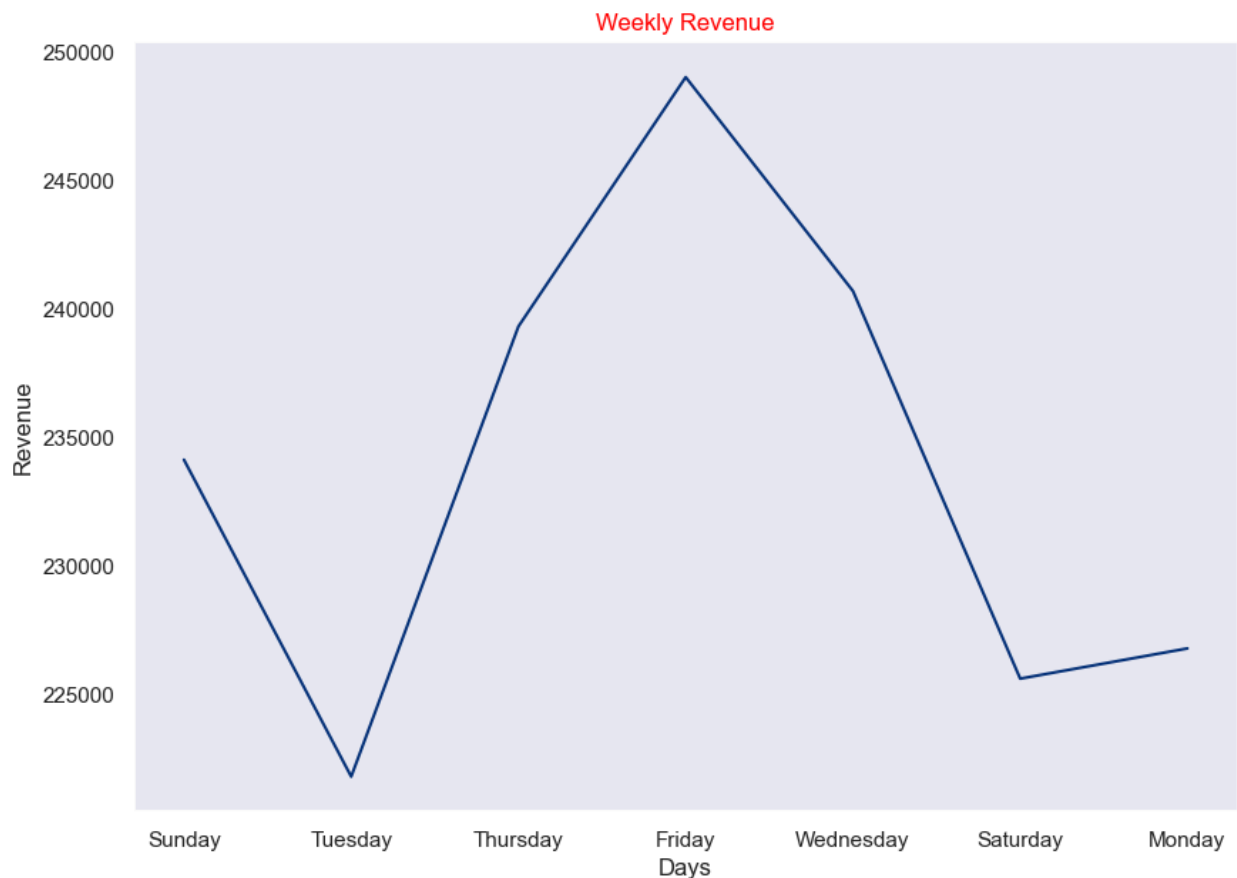


Isights

- January records the highest revenue, indicating peak monthly performance.
- Revenue fluctuates across months, showing seasonal variation in customer spending.
- Lower revenue in some months suggests opportunities for targeted promotions during off-peak periods.

Weekly Revenue

```
In [37]: Weekly_Rev = pd.read_sql_query("Select DayName(Transaction_Date) as Days, " \
    "sum(Total_Spent) as Revenue from Retails_Clean group by Days", engine)
plt.figure(figsize=(10,7))
sns.lineplot(x = "Days",y = "Revenue", data= Weekly_Rev, color = "#09347c" )
plt.title("Weekly Revenue", color = "Red")
plt.show()
```

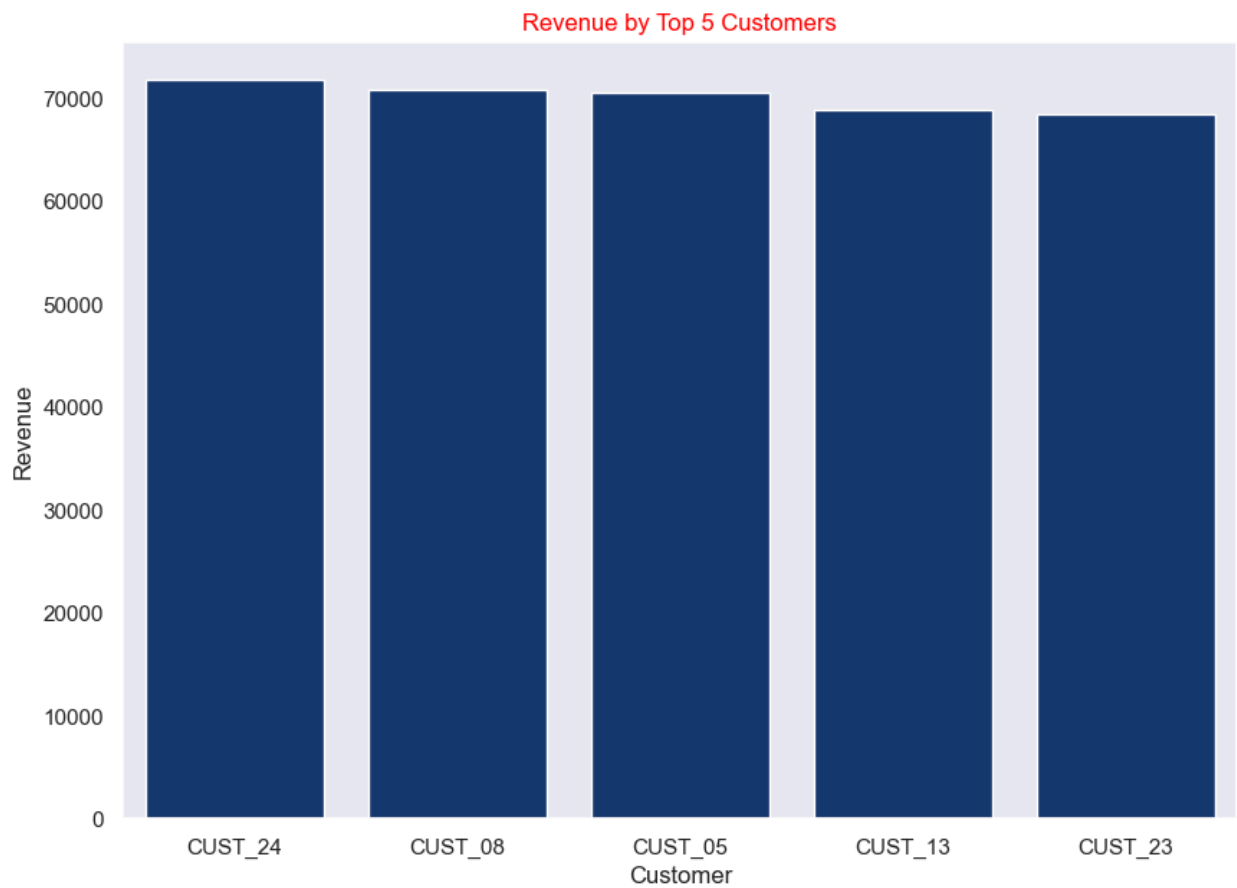


Insights

- Friday records the highest revenue, making it the strongest sales day of the week.
- Mid-week (Thursday-Friday) revenue is generally higher, indicating increased customer activity before weekends.
- Lower revenue on some days suggests scope for weekday-focused promotions.

Customer Analysis

```
In [38]: Customer = pd.read_sql_query("Select Customer_Id as Customer,sum(Total_Spent)
    " from Retails_clean group by Customer order by Revenue Desc limit 5",engine)
plt.figure(figsize=(10,7))
sns.barplot(x = "Customer", y = "Revenue", data= Customer, color="#09347c")
plt.title("Revenue by Top 5 Customers", color = "Red")
plt.show()
```



Insights

- CUST_24 generates the highest revenue among the top 5 customers.
- Other top customers contribute similar but slightly lower revenue, showing a balanced high-value customer base.
- Retaining customers like CUST_24 is critical for sustaining overall revenue.