Coffee Shop Store

The main objective of this project is to analyze retail sales data to gain actionable insights that will enhance the performance of the Coffee Shop

Recommended Analysis

1. How do sales vary by day of the week and hour of the day?
2. Are there any peak times for sales activity?
3. What is the total sales revenue for each month?
4. How do sales vary across different store locations?
5. what is the average price/order per person?
6. Which products are the best selling in terms of quantity and revenue?
7. How do sales vary by product category and type?

In [2]:

```python
import warnings
warnings.filterwarnings("ignore")
```

In [4]:

```python
import pandas as pd
shop_data= pd.read_csv("coffee shop sales -final.csv")
print(shop_data)
```

```
        transaction_id transaction_date transaction_time  store_id  \
0               114301       06/01/2023      11:33:29 AM         3
1               115405       06/02/2023      11:18:24 AM         3
2               115478       06/02/2023      12:02:45 PM         3
3               116288       06/02/2023      07:39:47 PM         3
4               116714       06/03/2023      12:24:57 PM         3
...                ...              ...              ...       ...
149111          129465       06/14/2023      08:34:10 AM         5
149112          133523       06/17/2023      09:55:47 AM         8
149113          133674       06/17/2023      10:41:11 AM         8
149114          133744       06/17/2023      11:18:31 AM         8
149115          149043       06/30/2023      11:18:31 AM         8

         store_location  product_id  transaction_qty  unit_price  \
0               Astoria          45                1        3.00
1               Astoria          45                1        3.00
2               Astoria          45                1        3.00
3               Astoria          45                1        3.00
4               Astoria          45                1        3.00
...                 ...         ...              ...         ...
149111  Lower Manhattan          41                4        4.25
149112   Hell's Kitchen           8                8       45.00
149113   Hell's Kitchen           8                8       45.00
149114   Hell's Kitchen           8                8       45.00
149115   Hell's Kitchen           8                8       45.00

       product_category       product_type product_detail         Size  \
0                   Tea  Brewed herbal tea     Peppermint        Large
1                   Tea  Brewed herbal tea     Peppermint        Large
2                   Tea  Brewed herbal tea     Peppermint        Large
3                   Tea  Brewed herbal tea     Peppermint        Large
4                   Tea  Brewed herbal tea     Peppermint        Large
...                 ...                ...            ...          ...
149111           Coffee   Barista Espresso     Cappuccino        Large
149112     Coffee beans      Premium Beans      Civet Cat  Not Defined
149113     Coffee beans      Premium Beans      Civet Cat  Not Defined
149114     Coffee beans      Premium Beans      Civet Cat  Not Defined
149115     Coffee beans      Premium Beans      Civet Cat  Not Defined

        Total_bill Month Name   Day Name  Hour  Day of Week  Month
0              3.0       June   Thursday    11            4      6
1              3.0       June     Friday    11            5      6
2              3.0       June     Friday    12            5      6
3              3.0       June     Friday    19            5      6
4              3.0       June   Saturday    12            6      6
...            ...        ...        ...   ...          ...    ...
149111        17.0       June  Wednesday     8            3      6
149112       360.0       June   Saturday     9            6      6
149113       360.0       June   Saturday    10            6      6
149114       360.0       June   Saturday    11            6      6
149115       360.0       June     Friday    11            5      6

[149116 rows x 18 columns]
```

```
In [5]: shop_data.columns
```

```
Out[5]: Index(['transaction_id', 'transaction_date', 'transaction_time', 'store_i
        d',
               'store_location', 'product_id', 'transaction_qty', 'unit_price',
               'product_category', 'product_type', 'product_detail', 'Size',
               'Total_bill', 'Month Name', 'Day Name', 'Hour', 'Day of Week', 'Mont
        h'],
              dtype='object')
```

The dataset we are working with contains the following columns:

1. transaction_id: A unique identifier for each transaction, typically used to distinguish between different sales or transactions. This is usually an alphanumeric or numeric code.

2. transaction_date: The date when the transaction occurred. This would typically be in a YYYY-MM-DD format or any other standard date format. It records the specific day of the sale.

3. transaction_time: The time when the transaction occurred, typically in HH:MM:SS format. This column tracks the exact time of day when the transaction was made.

4. store_id: A unique identifier for the store where the transaction took place. This helps to track which store processed the transaction, especially useful in multi-store setups.

5. store_location: This column provides information about the geographical location of the store, such as the city, region, or address. It helps understand where the sale took place geographically.

6. product_id: A unique identifier for each product sold during the transaction. It can be a numeric or alphanumeric code, used to reference specific products in the inventory.

7. transaction_qty: The quantity of the product sold during the transaction. This shows how many units of a particular product were sold in a single transaction.

8. unit_price: The price per unit of the product at the time of the transaction. This indicates the selling price of one unit of the product.

9. product_category: A broad category that groups similar products together. For example, in a coffee shop, categories might include Beverages, Snacks, or Bakery Items.

10. product_type: A more specific classification within the product_category. For instance, under the Beverages category, the product_type could be Coffee, Tea, or Juice.

11. product_detail: Additional information about the product, such as a detailed description. It could include product features like flavors, variations, or special ingredients.

12. Size: The size of the product, especially relevant for products like beverages or clothing. For example, in a coffee shop, Size might refer to Small, Medium, or Large cups.

13. Total_bill: The total amount billed for the transaction. This is the sum of the cost of all products sold in the transaction, typically calculated as transaction_qty * unit_price (before taxes or discounts).

14. Month Name: The name of the month during which the transaction took place (e.g., January, February). It provides a readable format of the month extracted from the transaction_date.

15. Day Name: The name of the day on which the transaction occurred (e.g., Monday, Tuesday). This is often derived from the transaction_date and helps analyze sales trends by days of the week.

16. Hour: The hour when the transaction occurred, extracted from the transaction_time column. This helps in understanding the time-of-day patterns in sales (e.g., peak sales hours).

17. Day of Week: A numeric value representing the day of the week when the transaction occurred (e.g., 1 for Monday, 7 for Sunday). This is useful for analyzing data programmatically.

18. Month:A numeric representation of the month in which the transaction took place (e.g., 1 for January, 12 for December). This is a simplified version of the Month Name column for easier sorting and calculations.

Understand the Data Types:

1. Numeric Columns: transaction_qty, unit_price, Total_bill, Hour, Day of Week, Month
2. Categorical Columns: store_id, store_location, product_category, product_type, product_detail, Size, Month Name, Day Name
3. Date/Time Columns: transaction_date, transaction_time
4. Identifiers: transaction_id, product_id

In [6]:

```
# Let's have a look at whether the data containsa any null values or not:
shop_data.isnull().sum()
```

Out[6]:

```
transaction_id      0
transaction_date    0
transaction_time    0
store_id            0
store_location      0
product_id          0
transaction_qty     0
unit_price          0
product_category    0
product_type        0
product_detail      0
Size                0
Total_bill          0
Month Name          0
Day Name            0
Hour                0
Day of Week         0
Month               0
dtype: int64
```

So, the data does not contains any null values.

In [8]:

```
shop_data.dtypes
```

Out[8]:

```
transaction_id        int64
transaction_date     object
transaction_time     object
store_id              int64
store_location       object
product_id            int64
transaction_qty       int64
unit_price          float64
product_category     object
product_type         object
product_detail       object
Size                 object
Total_bill          float64
Month Name           object
Day Name             object
Hour                  int64
Day of Week           int64
Month                 int64
dtype: object
```

Statistical Summary of Numeric Columns:

```
    Use describe() to get a statisticsl summary(mean, median,
min,max,quartiles) for the numeric columns
```

In [49]:

```
shop_data[['transaction_qty','unit_price','Total_bill','Hour','Day of Week',
```

Out[49]:

	transaction_qty	unit_price	Total_bill	Hour	Day of
count	149116.000000	149116.000000	149116.000000	149116.000000	149116.0
mean	1.438276	3.382219	4.686367	11.735790	2.9
std	0.542509	2.658723	4.227099	3.764662	1.9
min	1.000000	0.800000	0.800000	6.000000	0.0
25%	1.000000	2.500000	3.000000	9.000000	1.0
50%	1.000000	3.000000	3.750000	11.000000	3.0
75%	2.000000	3.750000	6.000000	15.000000	5.0
max	8.000000	45.000000	360.000000	20.000000	6.0

Key Insights:

1. Most transactions involve small quantities (1 or 2 items), and the average price per item is affordable, with the majority priced around 3.00.
2. Sales occur most frequently around midday, with peak activity between 9 AM and 3 PM.
3. Transactions are fairly distributed across different days of the week and the first half of the year.

In [19]:

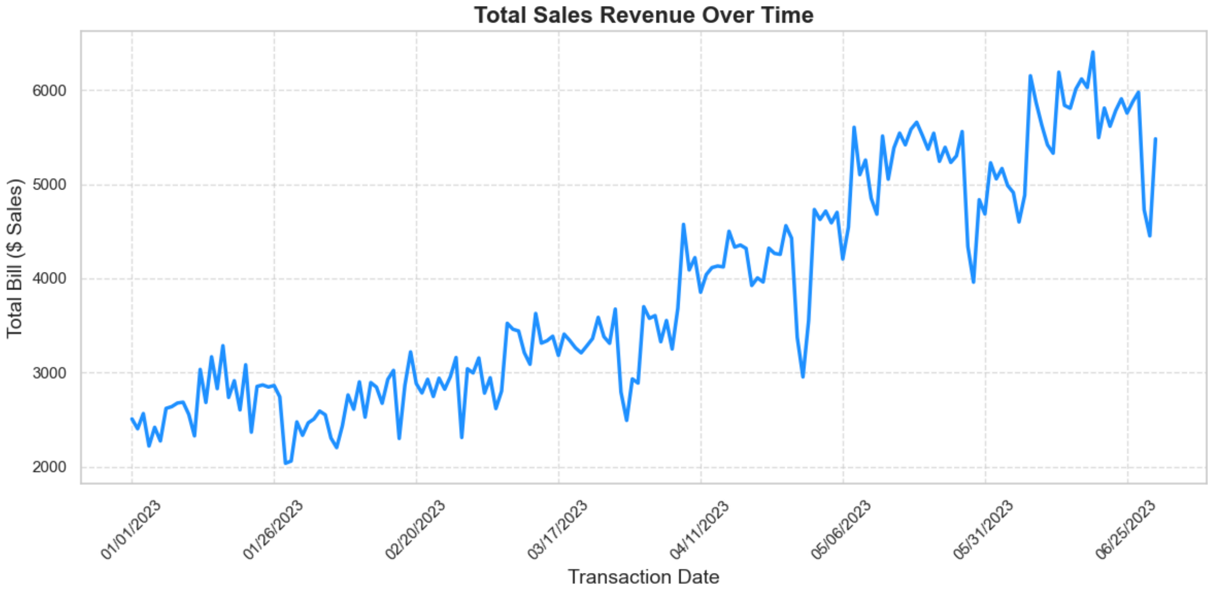
```
#These are identifiers, so no statistical analysis is necessary.
#COunt of unique transactions and unique products
transaction_id_unique= shop_data['transaction_id'].nunique()
print(transaction_id_unique)
product_id_unique= shop_data['product_id'].nunique()
print(product_id_unique)
```

```
149116
80
```

In [30]:

```
#Plot the trend of sales over time (using Total_bill) to check seasonality a
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style for seaborn for a more visually appealing plot
sns.set(style="whitegrid")

# Group the data by transaction_date and sum the Total_bill
shop_data.groupby('transaction_date')['Total_bill'].sum().plot(figsize=(12,

# Adding a title and labels for better interpretation
plt.title('Total Sales Revenue Over Time', fontsize=16, fontweight='bold')
plt.xlabel('Transaction Date', fontsize=14)
plt.ylabel('Total Bill ($ Sales)', fontsize=14)

# Rotate x-ticks for better readability (especially if dates are dense)
plt.xticks(rotation=45)

# Add gridlines to make the plot easier to read
```

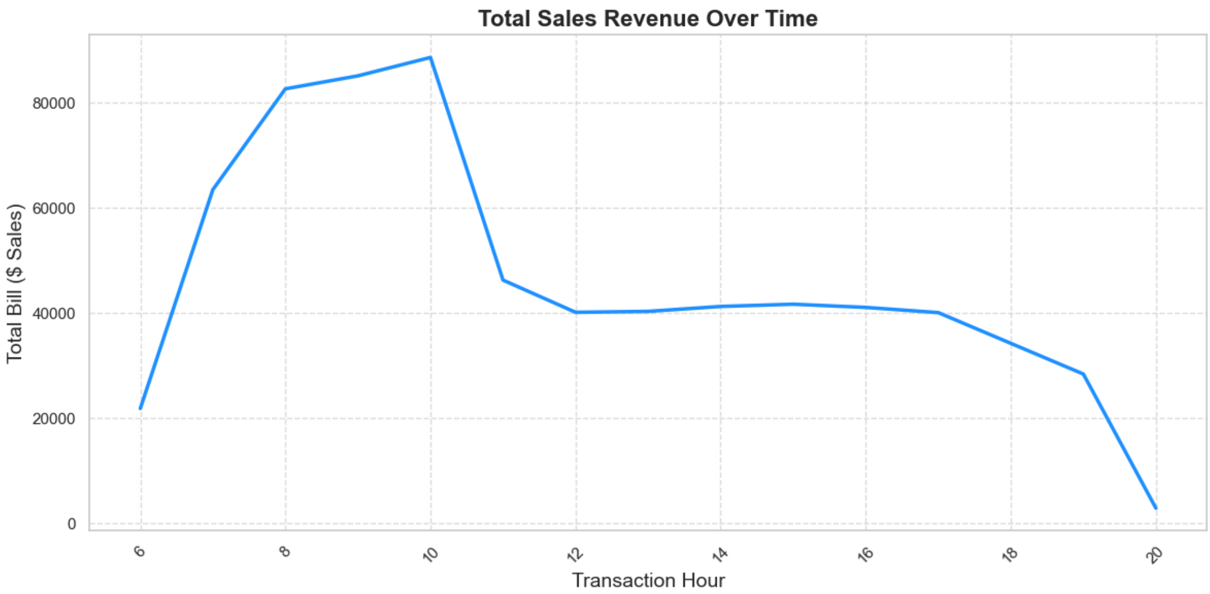
```
plt.grid(True, linestyle='--', alpha=0.7)

# Show the plot
plt.tight_layout()
plt.show()
```



We can observe that the total bills over the transaction dates are rapidly increasing with respect to date.

In [31]:

```
#Hourly analysis: plot sales trends by hour of the day(transaction_time)

# Set the style for seaborn for a more visually appealing plot
sns.set(style="whitegrid")

# Group the data by transaction_date and sum the Total_bill
shop_data.groupby('Hour')['Total_bill'].sum().plot(figsize=(12, 6), color='c

# Adding a title and labels for better interpretation
plt.title('Total Sales Revenue Over Time', fontsize=16, fontweight='bold')
plt.xlabel('Transaction Hour', fontsize=14)
plt.ylabel('Total Bill ($ Sales)', fontsize=14)

# Rotate x-ticks for better readability (especially if dates are dense)
plt.xticks(rotation=45)

# Add gridlines to make the plot easier to read
plt.grid(True, linestyle='--', alpha=0.7)

# Show the plot
plt.tight_layout()
plt.show()
```



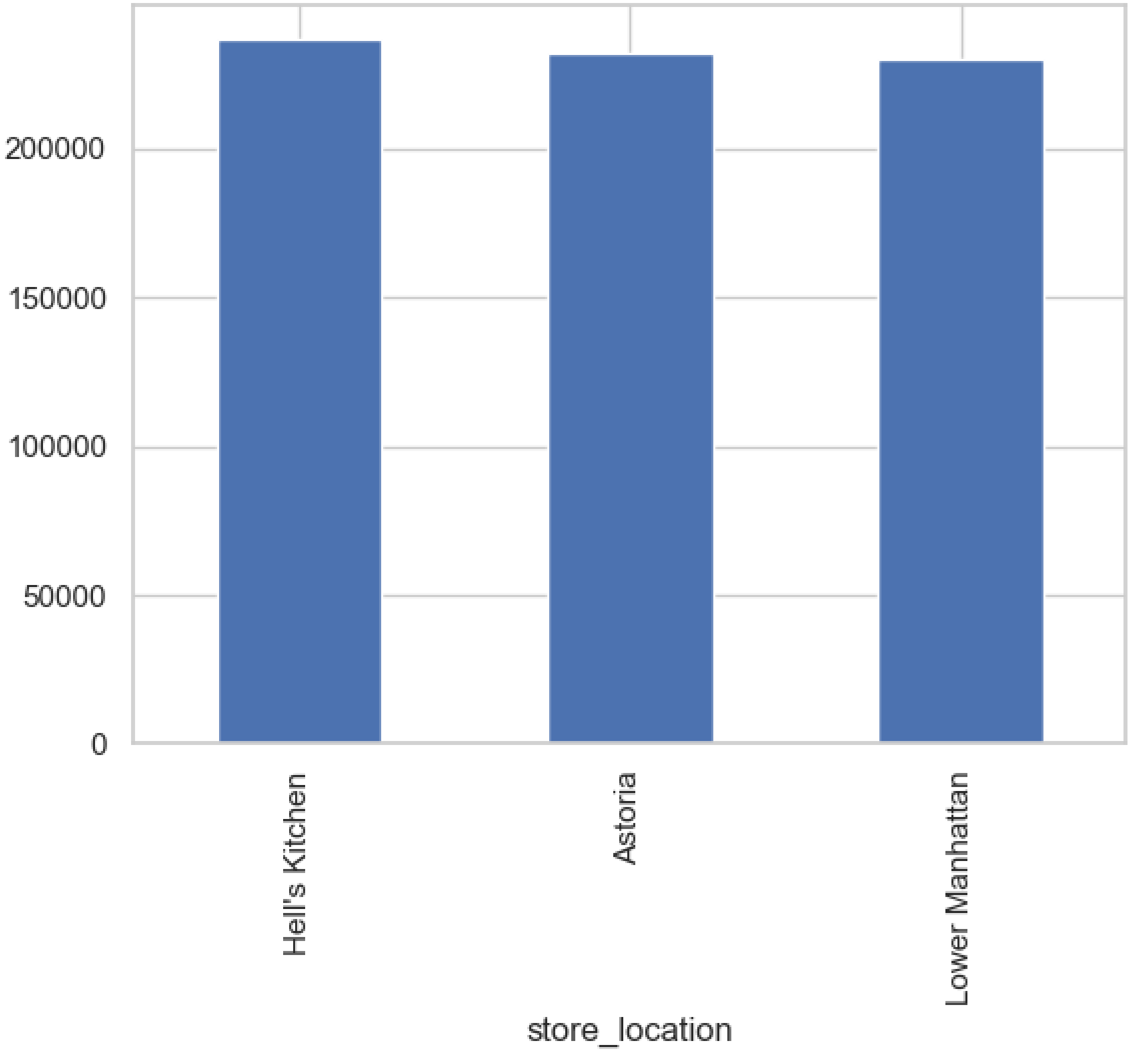
We can observe that the peak shopping hours occur between 7 AM and 11 AM. However, after 11 AM, sales start to gradually decline, and there is a noticeable sharp drop-off after 7 PM

```
In [36]: # Count of transactions per store to identify the busiest locations.
         shop_data['store_id'].value_counts()
```

```
Out[36]: store_id
         8    50735
         3    50599
         5    47782
         Name: count, dtype: int64
```

```
In [39]: # Revenue by store: Compare the total revenue across stores to find the top-
         shop_data.groupby('store_location')['Total_bill'].sum().sort_values(ascendir
```

```
Out[39]: <Axes: xlabel='store_location'>
```

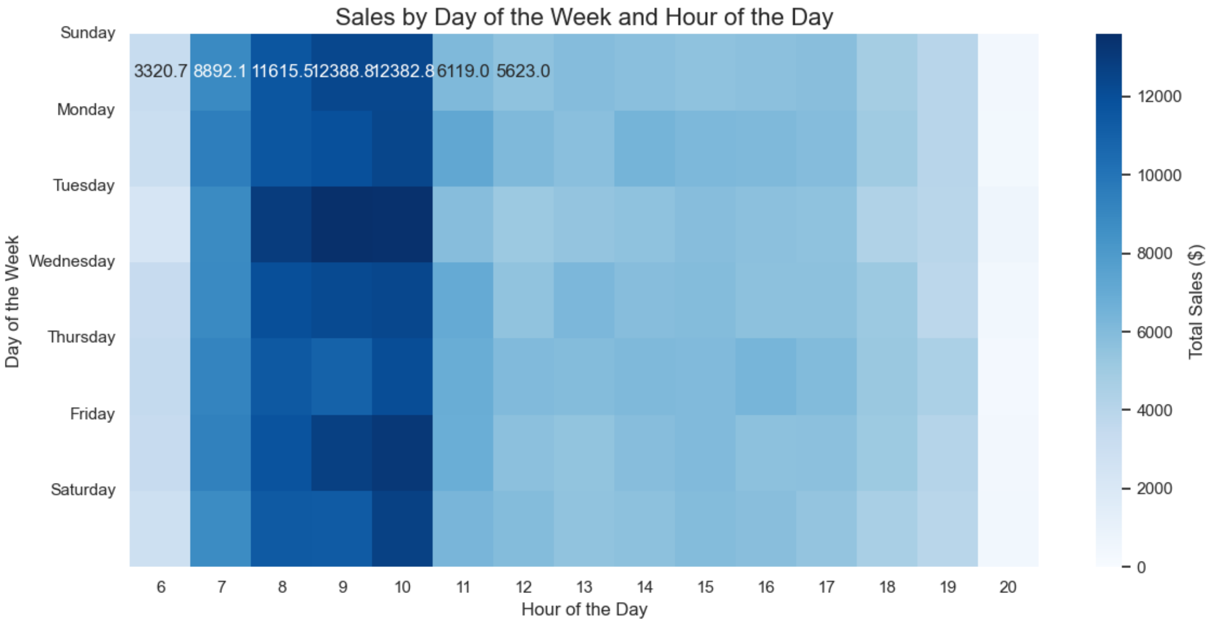


1. How do sales vary by day of the week and hour of the day?

We’ll use a heatmap to visualize the variation in sales by day of the week and hour of the day.

In [46]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Group data by day of the week and hour, summing the total bill
sales_by_day_hour = shop_data.groupby(['Day of Week', 'Hour'])['Total_bill']

# Pivot for the heatmap (correct syntax using keyword arguments)
sales_pivot = sales_by_day_hour.pivot(index='Day of Week', columns='Hour', v

# Create a list to map day numbers to names if 'Day of Week' is numeric (e.g
day_labels = ['Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Frida

# Plot heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(sales_pivot, cmap='Blues', annot=True, fmt=".1f", cbar_kws={'lab
# Set the title and axis labels
plt.title('Sales by Day of the Week and Hour of the Day', fontsize=16)
plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Day of the Week', fontsize=12)

# Use the day labels if needed
plt.yticks(ticks=range(len(day_labels)), labels=day_labels, rotation=0)

# Show the heatmap
plt.tight_layout()
plt.show()
```



Key Insights:

1.Peak Sales Hours:

- Morning Peaks: The heatmap likely shows high sales volumes in the morning, particularly between 7 AM and 11 AM. This suggests that your business experiences a strong customer influx during these hours.
- Afternoon and Evening Trends: Sales may show a dip or continue to be moderate after the peak morning hours, with another possible peak in the early evening.

2.Day of the Week Patterns:

- Weekday vs. Weekend: Sales patterns may differ between weekdays and weekends. For instance, you might observe higher sales on weekends or specific days, indicating popular shopping days.
- Consistency: Some days might exhibit more consistent sales throughout the day compared to others, helping you understand which days are less variable in terms of sales.

3.Low Sales Periods:

- Late Hours Decline: Sales are likely to decline sharply after certain hours, such as after 10 AM or in the late evening (7 PM onwards). This information can help in optimizing staff schedules and store hours.

4.High Sales Correlation:

- Hourly Correlation: If certain hours across different days consistently show higher sales, it could suggest specific times when promotions or marketing efforts are particularly effective.

5.Anomalies or Outliers:

- Unexpected Trends: Look for any anomalies or outliers in the heatmap where sales might be unexpectedly high or low. This could indicate special events, promotions, or other factors influencing sales.

6.Operational Adjustments:

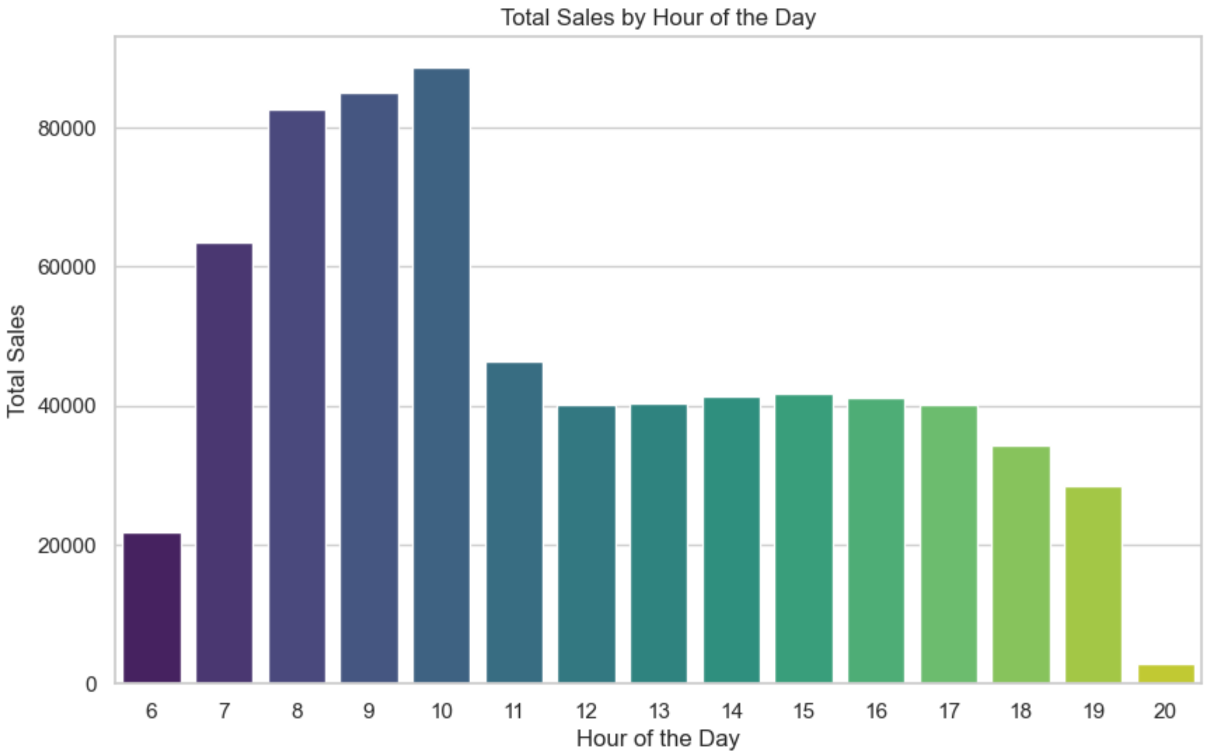
- Staffing and Inventory: Use these insights to adjust staffing levels and inventory management according to peak and off-peak hours. For example, ensuring more staff during peak hours can enhance customer service and efficiency.

2.Are there any peak times for sales activity?

To visualize peak times, we can plot the total sales for each hour of the day.

In [47]:

```
# Group sales by hour
sales_by_hour = shop_data.groupby('Hour')['Total_bill'].sum().reset_index()

# Plot sales by hour
plt.figure(figsize=(10, 6))
sns.barplot(x='Hour', y='Total_bill', data=sales_by_hour, palette='viridis')
plt.title('Total Sales by Hour of the Day')
plt.ylabel('Total Sales')
plt.xlabel('Hour of the Day')
plt.show()
```



- Peak Hours: If the plot shows a significant increase in total sales between 7 AM and 11 AM, it indicates that this is a key time for sales activity. Your business may experience high customer traffic during these hours.
- Low Sales: If sales are consistently low early in the late at night, you might consider adjusting store hours or implementing strategies to increase customer engagement during these times.

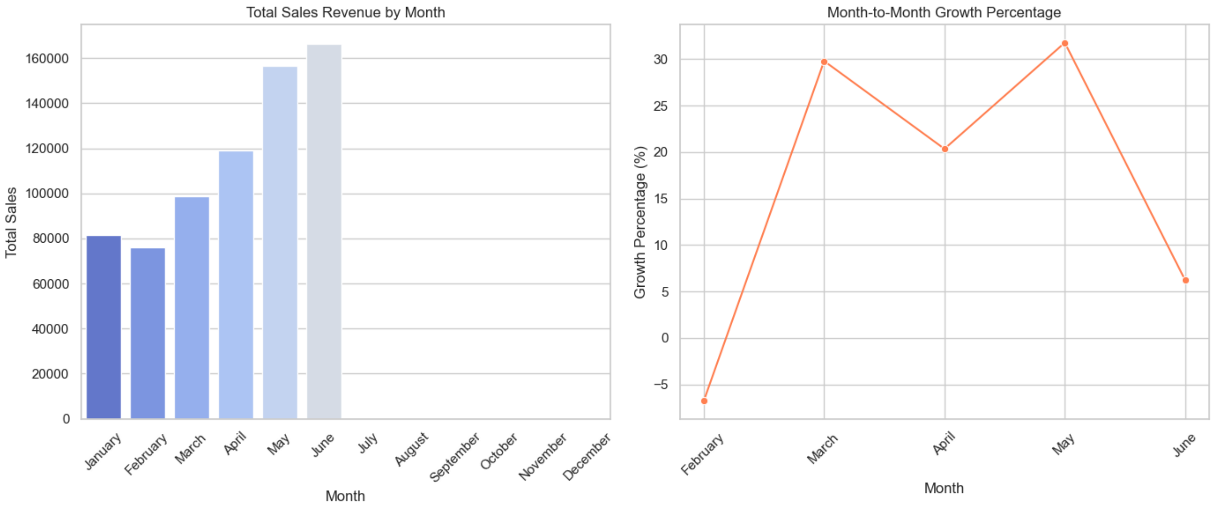
3. What is the total sales revenue for each month?

- We'll use a bar chart to show the total sales revenue for each month.

In [52]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Group sales by month name
sales_by_month = shop_data.groupby('Month Name')['Total_bill'].sum().reset_i

# Sort month order if needed
month_order = ['January', 'February', 'March', 'April', 'May', 'June', 'July
sales_by_month['Month Name'] = pd.Categorical(sales_by_month['Month Name'],
sales_by_month = sales_by_month.sort_values('Month Name')

# Calculate growth percentage month by month
sales_by_month['Growth Percentage'] = sales_by_month['Total_bill'].pct_chang

# Plot total sales by month
plt.figure(figsize=(14, 6))
```

```
# Plot sales by month
plt.subplot(1, 2, 1)
sns.barplot(x='Month Name', y='Total_bill', data=sales_by_month, palette='co
plt.title('Total Sales Revenue by Month')
plt.ylabel('Total Sales')
plt.xlabel('Month')
plt.xticks(rotation=45)

# Plot growth percentage
plt.subplot(1, 2, 2)
sns.lineplot(x='Month Name', y='Growth Percentage', data=sales_by_month, mar
plt.title('Month-to-Month Growth Percentage')
plt.ylabel('Growth Percentage (%)')
plt.xlabel('Month')
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```



- Positive Growth Trend: The plot showing continuous growth from January to June suggests a strong positive trend. This could indicate successful business operations, effective marketing strategies, or increased customer demand.
- Further Analysis: To fully understand this trend, you might consider analyzing factors such as specific marketing activities, product launches, and customer feedback during these months.

Overall, the continuous growth trend is a positive indicator of your business’s performance and can help guide future strategies to sustain and enhance this upward trajectory.

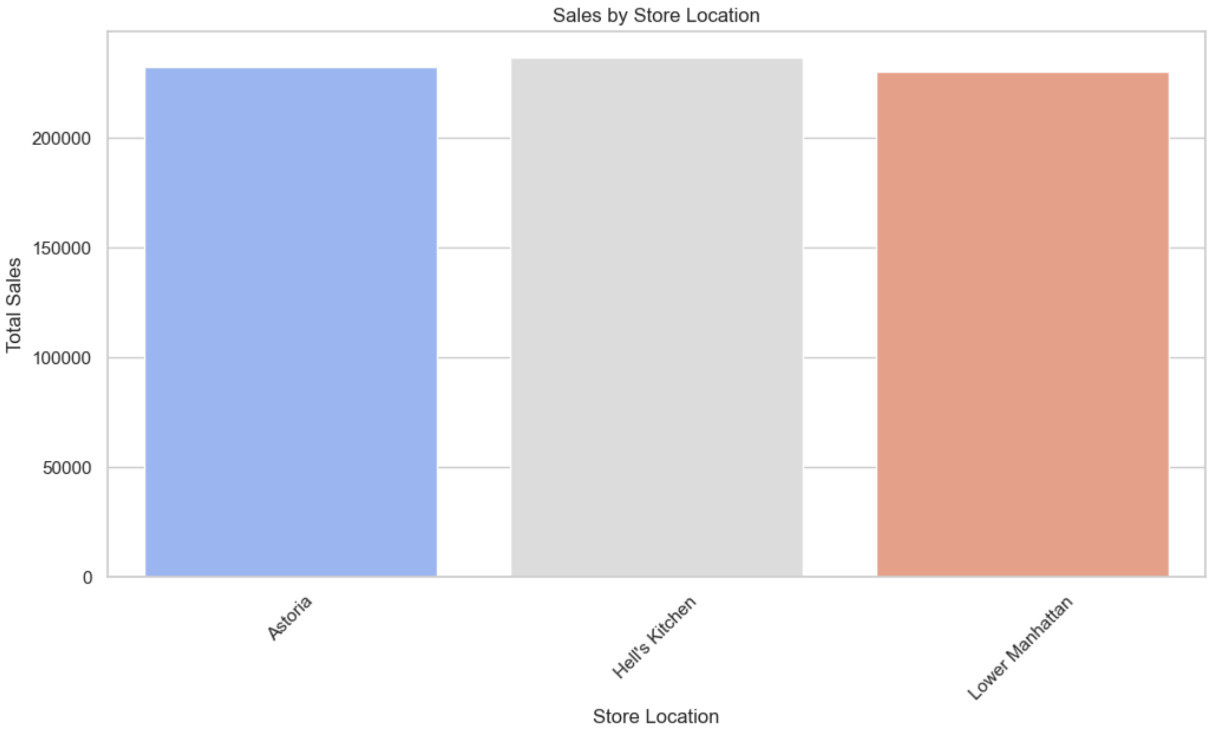
4.How do sales vary across different store locations?

```
Here, we’ll use a bar chart to show sales per store location.
```

In [53]:

```
# Group sales by store location
```

```
sales_by_location = shop_data.groupby('store_location')['Total_bill'].sum().

# Plot sales by location
plt.figure(figsize=(12, 6))
sns.barplot(x='store_location', y='Total_bill', data=sales_by_location, pale
plt.title('Sales by Store Location')
plt.ylabel('Total Sales')
plt.xlabel('Store Location')
plt.xticks(rotation=45)
plt.show()
```



- What is the price per order in the Coffee Shop?
- What is the Total Revenue of the Coffee Shop?
- What is the total days of working Coffee Shaop?
- What is the Revenue of the coffee Shop per day?
- What is the Average footfall per day in the Coffee shop?
- To find Order per person in the coffee shop?

In [126…

```
# * What is the average price per order in the Coffee Shop?
avg_price_per_order = shop_data['Total_bill'].mean()
print(f"Average price per order: ${avg_price_per_order:.2f}")

# * What is the Total Revenue of the Coffee Shop?
total_revenue= shop_data['Total_bill'].sum()
print(f"The Total Revenue of the coffee shop: ${total_revenue:.2f}")

# * What is the total days of working Coffee Shop?
total_working_days= len(shop_data['transaction_date'].unique())
print(f"The Total Working days of Working Coffee Shop: {total_working_days}"

# * What is the Revenue of the coffee Shop per day?
```

```
avg_rev= total_revenue/total_working_days
print(f"The Average Revenue of Coffee Shop per day : ${avg_rev:.2f}")

# * What is the Average footfall per day in the Coffee shop?
total_transaction= shop_data['transaction_id'].count()
avg_footfall= total_transaction/total_working_days
print(f"The Average Footfall per day in the Coffee Shop: {avg_footfall:.2f}"

 #* TO find Order per person in the coffee shop?
order_per_person = shop_data['transaction_qty'].mean()
print(f"The Order per Person in the Coffee Shop: ${order_per_person:.2f}")
```

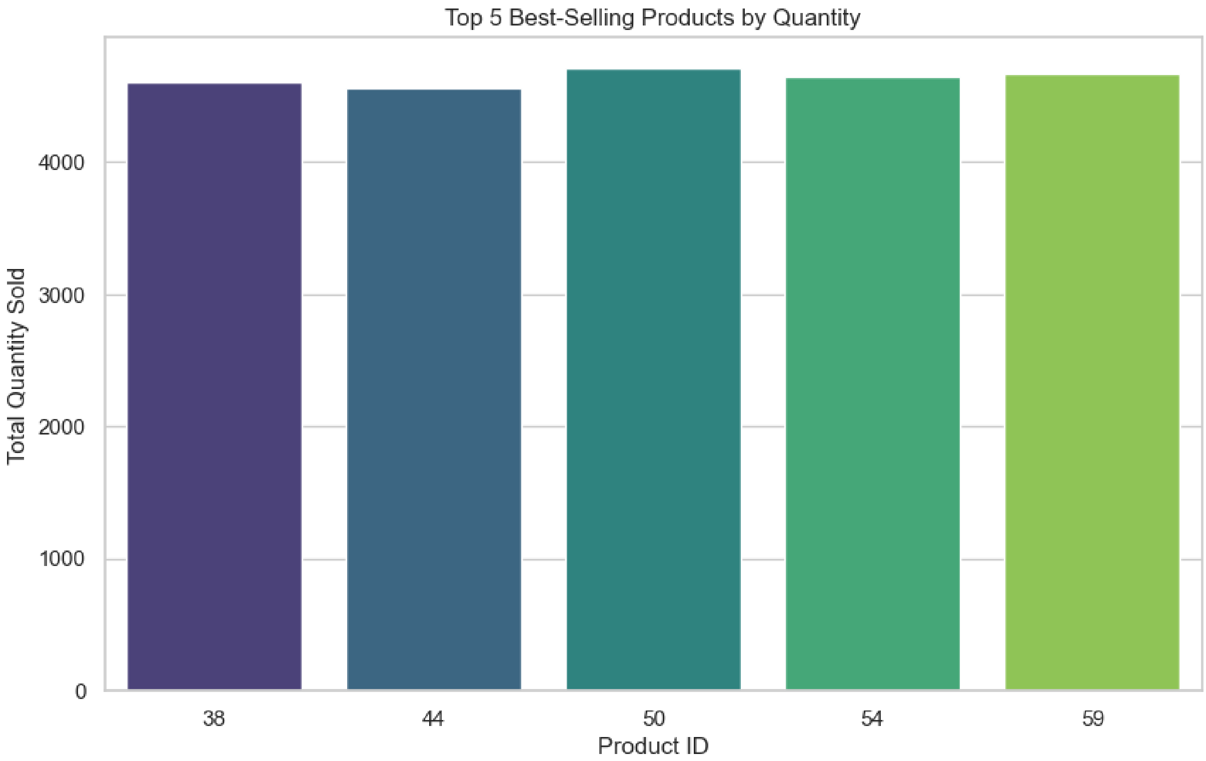
```
Average price per order: $4.69
The Total Revenue of the coffee shop: $698812.33
The Total Working days of Working Coffee Shop: 181
The Average Revenue of Coffee Shop per day : $3860.84
The Average Footfall per day in the Coffee Shop: 823.85
The Order per Person in the Coffee Shop: $1.44
```

6. Which products are the best selling in terms of quantity and revenue?

   We can use bar charts to display both best-selling products by quantity and revenue.

In [56]:

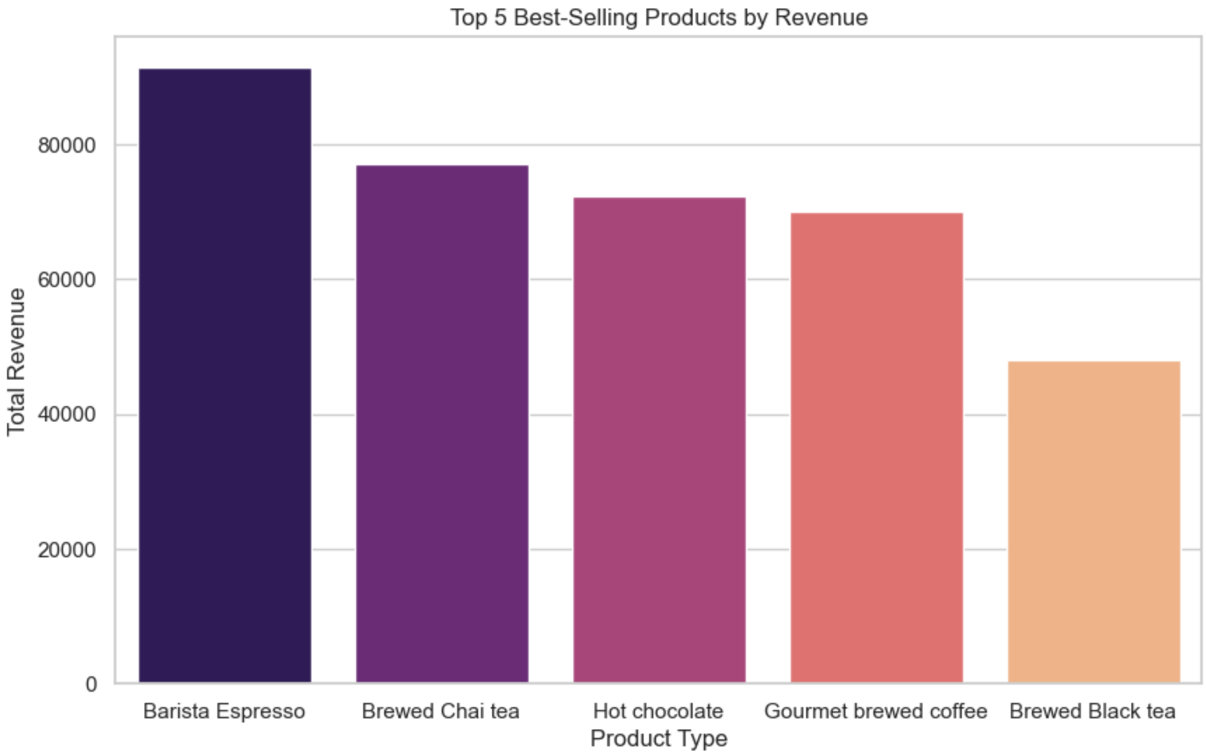
```
# Group by product_id and sum the quantities sold
best_selling_qty = shop_data.groupby('product_id')['transaction_qty'].sum().

# Plot the top 5 products by quantity
plt.figure(figsize=(10, 6))
sns.barplot(x='product_id', y='transaction_qty', data=best_selling_qty, pale
plt.title('Top 5 Best-Selling Products by Quantity')
plt.ylabel('Total Quantity Sold')
plt.xlabel('Product ID')
plt.show()
```



Best-Selling Products by Revenue:

In [67]:

```
# Group by product_id and sum the revenue generated
best_selling_revenue = shop_data.groupby('product_type')['Total_bill'].sum()

# Plot the top 5 products by revenue
plt.figure(figsize=(10, 6))
sns.barplot(x='product_type', y='Total_bill', data=best_selling_revenue, pal
plt.title('Top 5 Best-Selling Products by Revenue')
plt.ylabel('Total Revenue')
plt.xlabel('Product Type')
plt.show()
```

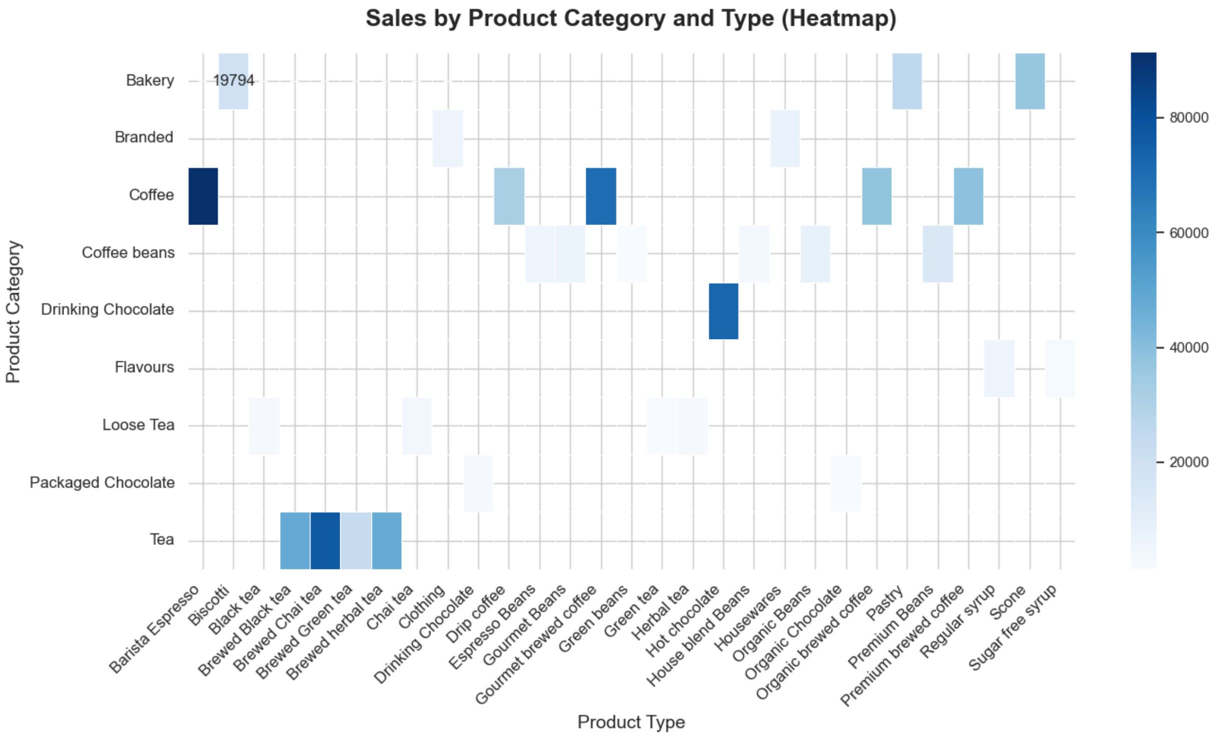


7.How do sales vary by product category and type?

Here we’ll use a HeatMap, Stacked bar plot and Pie Chart to show the variation in sales by both product category and product type.

In [108…

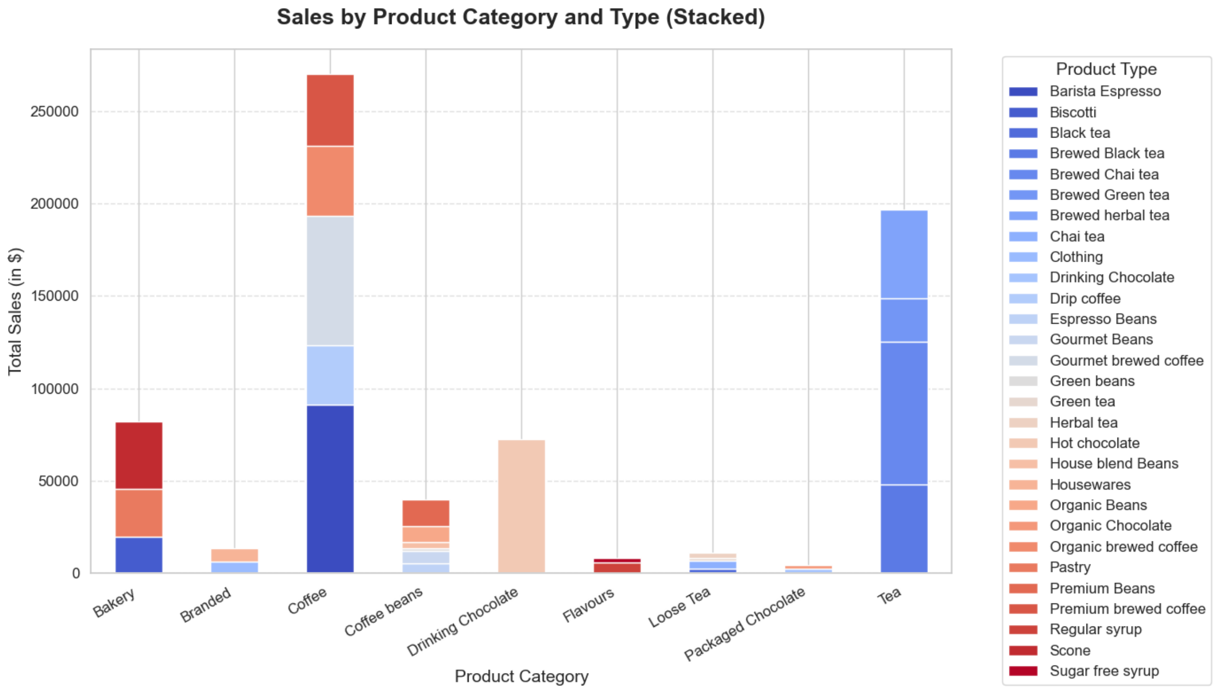
```
# Pivot the data for heatmap
sales_pivot = sales_by_category_type.pivot(index='product_category', columns

# Plot heatmap
plt.figure(figsize=(14, 8))
sns.heatmap(sales_pivot, annot=True, fmt=".0f", cmap='Blues', linewidths=.5)

plt.title('Sales by Product Category and Type (Heatmap)', fontsize=18, fontw
plt.ylabel('Product Category', fontsize=14)
plt.xlabel('Product Type', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.yticks(fontsize=12)

plt.tight_layout()
plt.show()
```



In [106…

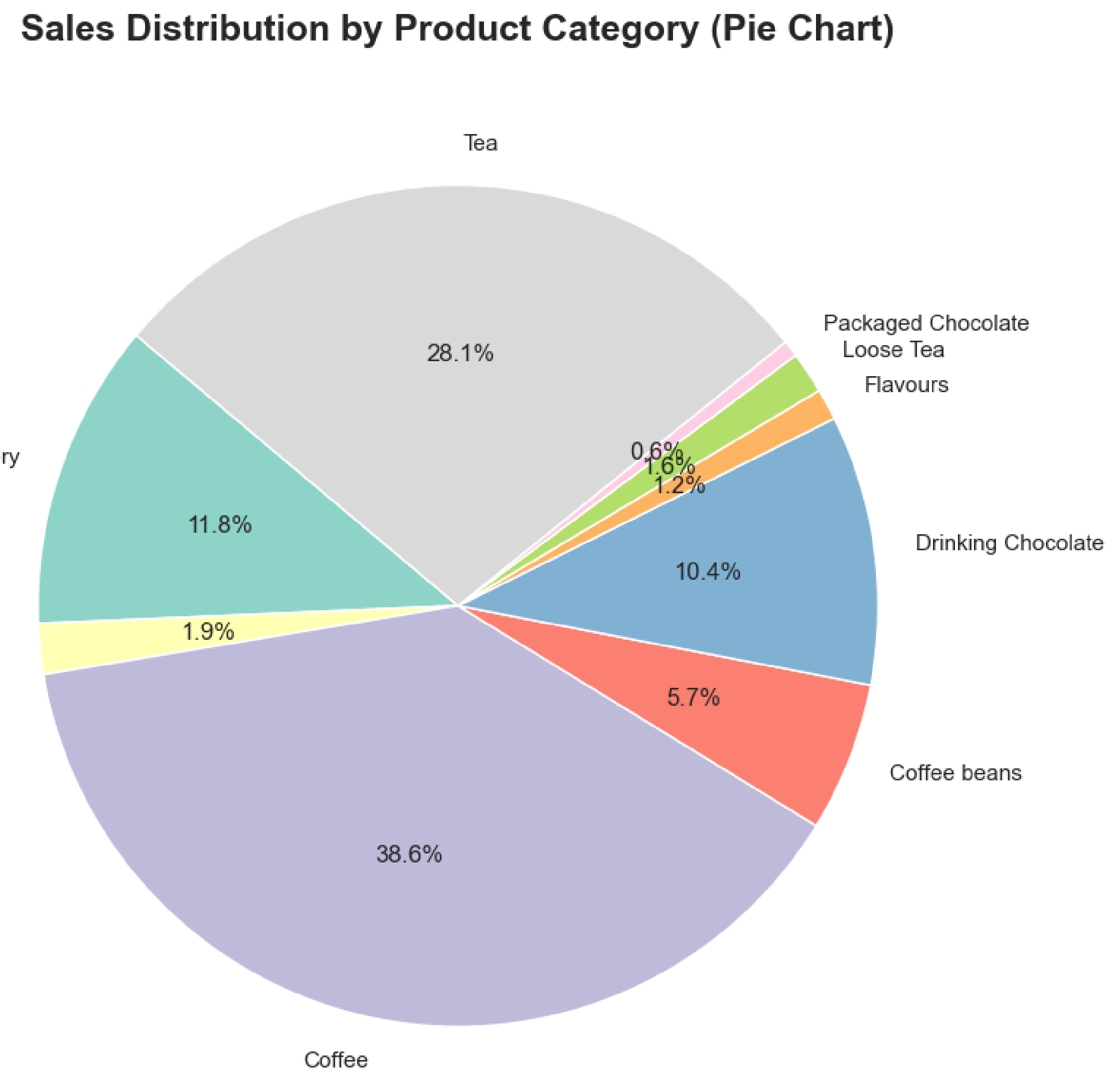
```
plt.figure(figsize=(14, 8))

# Create stacked bar plot
sales_pivot = sales_by_category_type.pivot(index='product_category', columns
sales_pivot.plot(kind='bar', stacked=True, colormap='coolwarm', figsize=(14,

plt.title('Sales by Product Category and Type (Stacked)', fontsize=18, fontw
plt.ylabel('Total Sales (in $)', fontsize=14)
plt.xlabel('Product Category', fontsize=14)
plt.xticks(rotation=30, ha='right', fontsize=12)
plt.yticks(fontsize=12)

# Adjust the legend position and style
plt.legend(title='Product Type', bbox_to_anchor=(1.05, 1), loc='upper left',

# Add gridlines for y-axis
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
<Figure size 1400x800 with 0 Axes>
```



In [107…

```
# Group sales by product category
sales_by_category = shop_data.groupby('product_category')['Total_bill'].sum(

# Plot pie chart
plt.figure(figsize=(10, 8))
plt.pie(sales_by_category['Total_bill'], labels=sales_by_category['product_c

plt.title('Sales Distribution by Product Category (Pie Chart)', fontsize=18,
plt.tight_layout()
plt.show()
```


Sales Distribution by Product Category (Pie Chart)
Tea
28.1%
Packaged Chocolate
Loose Tea
Flavours
0.6%
1.6%
1.2%
Bakery
11.8%
Drinking Chocolate
10.4%
Branded
1.9%
5.7%
Coffee beans
38.6%
Coffee

```
In [ ]:
```