coffee-shop-analysis

February 14, 2025

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1 Reading and Analyzing Excel Data with Pandas

[2]:
```
sales = pd.read_excel('Coffee Shop Sales.xlsx') # read the data from the excel␣
 ↪file, and store it in a variable called sales


print("="*40)
print("  Coffee Shop Sales Data Overview  ")
print("="*40)
print(sales.shape) # checking the shape of the data, to know the number of rows␣
 ↪and columns
print(f"number of rows :  {sales.shape[0]} and number of columns : {sales.
 ↪shape[1]}") # print the number of rows and columns in the data
print(f"number of duplicated rows : {sales.duplicated().sum()})") # check for␣
 ↪duplicated rows in the data and print the number of duplicated rows in the␣
 ↪data
print(f"Nams of columns : {sales.columns}") # print the names of the columns in␣
 ↪the data set
```

```
========================================
  Coffee Shop Sales Data Overview
========================================
(149116, 11)
number of rows :  149116 and number of columns : 11
number of duplicated rows : 0)
Nams of columns : Index(['transaction_id', 'transaction_date',
'transaction_time',
       'transaction_qty', 'store_id', 'store_location', 'product_id',
       'unit_price', 'product_category', 'product_type', 'product_detail'],
      dtype='object')
```

[3]:
```
sales.head() # print the first 5 rows of the data set
sales.dtypes # check the data types of the columns in the data set and if the␣
 ↪data types are not correct, we can change them to the correct data type
# with help of astype() function and to_datetime() , to_numeric() functions,␣
 ↪to_oject() functions as well as to_string() functions
```

[3]:
```
transaction_id              int64
transaction_date    datetime64[ns]
transaction_time            object
transaction_qty              int64
store_id                     int64
store_location              object
product_id                   int64
unit_price                 float64
product_category            object
product_type                object
product_detail              object
dtype: object
```

[4]:
```
sales["transaction_time"]=pd.to_datetime(sales["transaction_time"],format="%H:
 ↪%M:%S")
# convert the transaction_time column to datetime data type using to_datetime()␣
 ↪function and format the time to hours, minutes and seconds using format␣
 ↪parameter in the function
```

[6]:
```
sales.dtypes # checking the data types of the columns in the data set to see if␣
 ↪the data type of the transaction_time column has been changed to datetime␣
 ↪data type
```

[6]:
```
transaction_id              int64
transaction_date    datetime64[ns]
transaction_time    datetime64[ns]
transaction_qty              int64
store_id                     int64
store_location              object
product_id                   int64
unit_price                 float64
product_category            object
product_type                object
product_detail              object
dtype: object
```

[7]:
```
# calculation of total sales
sales["total_sales"]=sales["unit_price"]*sales["transaction_qty"]
sales
```

[7]:
```
        transaction_id transaction_date    transaction_time  transaction_qty  \
0                    1       2023-01-01 1900-01-01 07:06:11                2
1                    2       2023-01-01 1900-01-01 07:08:56                2
2                    3       2023-01-01 1900-01-01 07:14:04                2
3                    4       2023-01-01 1900-01-01 07:20:24                1
4                    5       2023-01-01 1900-01-01 07:22:41                2
…                  …                 …                   …              …
149111          149452       2023-06-30 1900-01-01 20:18:41                2
149112          149453       2023-06-30 1900-01-01 20:25:10                2
149113          149454       2023-06-30 1900-01-01 20:31:34                1
149114          149455       2023-06-30 1900-01-01 20:57:19                1
149115          149456       2023-06-30 1900-01-01 20:57:19                2

        store_id   store_location  product_id  unit_price    product_category  \
0              5  Lower Manhattan          32        3.00              Coffee
1              5  Lower Manhattan          57        3.10                 Tea
2              5  Lower Manhattan          59        4.50  Drinking Chocolate
3              5  Lower Manhattan          22        2.00              Coffee
4              5  Lower Manhattan          57        3.10                 Tea
…            …                …         …           …                 …
149111         8   Hell's Kitchen          44        2.50                 Tea
149112         8   Hell's Kitchen          49        3.00                 Tea
149113         8   Hell's Kitchen          45        3.00                 Tea
149114         8   Hell's Kitchen          40        3.75              Coffee
149115         8   Hell's Kitchen          64        0.80            Flavours

                product_type              product_detail  total_sales
0       Gourmet brewed coffee                 Ethiopia Rg         6.00
1             Brewed Chai tea     Spicy Eye Opener Chai Lg         6.20
2               Hot chocolate            Dark chocolate Lg         9.00
3                Drip coffee  Our Old Time Diner Blend Sm         2.00
4             Brewed Chai tea     Spicy Eye Opener Chai Lg         6.20
…                         …                           …          …
149111      Brewed herbal tea               Peppermint Rg         5.00
149112       Brewed Black tea         English Breakfast Lg         6.00
149113      Brewed herbal tea               Peppermint Lg         3.00
149114       Barista Espresso                  Cappuccino         3.75
149115          Regular syrup              Hazelnut syrup         1.60

[149116 rows x 12 columns]
```

[8]:
```
# making one month column of month names
 # insert syntax is sales.insert(index, column_name, value)

sales.insert(2,"month",sales["transaction_date"].dt.strftime("%B"))
sales.head()
```

[8]:
```
   transaction_id transaction_date    month    transaction_time  \
0               1       2023-01-01  January 1900-01-01 07:06:11
1               2       2023-01-01  January 1900-01-01 07:08:56
2               3       2023-01-01  January 1900-01-01 07:14:04
3               4       2023-01-01  January 1900-01-01 07:20:24
4               5       2023-01-01  January 1900-01-01 07:22:41

   transaction_qty  store_id   store_location  product_id  unit_price  \
0                2         5  Lower Manhattan          32         3.0
1                2         5  Lower Manhattan          57         3.1
2                2         5  Lower Manhattan          59         4.5
3                1         5  Lower Manhattan          22         2.0
4                2         5  Lower Manhattan          57         3.1

     product_category           product_type               product_detail  \
0              Coffee  Gourmet brewed coffee                 Ethiopia Rg
1                 Tea        Brewed Chai tea     Spicy Eye Opener Chai Lg
2  Drinking Chocolate          Hot chocolate            Dark chocolate Lg
3              Coffee            Drip coffee  Our Old Time Diner Blend Sm
4                 Tea        Brewed Chai tea     Spicy Eye Opener Chai Lg

   total_sales
0          6.0
1          6.2
2          9.0
3          2.0
4          6.2
```

[9]:
```
# extraing and making a column of time
# insert syntax is sales.insert(index, column_name, value)

sales.insert(4,"time",sales["transaction_time"].dt.time)
```

[10]:
```
# making a column of day of week
# https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior

sales.insert(5,"day_of_week",sales["transaction_date"].dt.strftime("%A"))
sales.head()
```

[10]:
```
   transaction_id transaction_date    month    transaction_time      time  \
0               1       2023-01-01  January 1900-01-01 07:06:11  07:06:11
1               2       2023-01-01  January 1900-01-01 07:08:56  07:08:56
2               3       2023-01-01  January 1900-01-01 07:14:04  07:14:04
3               4       2023-01-01  January 1900-01-01 07:20:24  07:20:24
4               5       2023-01-01  January 1900-01-01 07:22:41  07:22:41

  day_of_week  transaction_qty  store_id   store_location  product_id  \
```

```
0      Sunday                2         5  Lower Manhattan          32
1      Sunday                2         5  Lower Manhattan          57
2      Sunday                2         5  Lower Manhattan          59
3      Sunday                1         5  Lower Manhattan          22
4      Sunday                2         5  Lower Manhattan          57

   unit_price    product_category           product_type  \
0         3.0              Coffee  Gourmet brewed coffee
1         3.1                 Tea        Brewed Chai tea
2         4.5  Drinking Chocolate          Hot chocolate
3         2.0              Coffee            Drip coffee
4         3.1                 Tea        Brewed Chai tea

                product_detail  total_sales
0                 Ethiopia Rg          6.0
1     Spicy Eye Opener Chai Lg          6.2
2            Dark chocolate Lg          9.0
3  Our Old Time Diner Blend Sm          2.0
4     Spicy Eye Opener Chai Lg          6.2
```

2 Matplotlib Overview

Matplotlib is a powerful plotting library in Python used for data visualization. It provides various tools to create static, animated, and interactive visualizations. It is widely used in data science, machine learning, and scientific computing.

2.1 Key Features of Matplotlib

- **Supports Multiple Plot Types**: Line plots, bar charts, histograms, scatter plots, etc.
- **Highly Customizable**: Control over colors, labels, grids, and styles.
- **Object-Oriented & State-Based Interfaces**:
  - **Pyplot Interface** (`matplotlib.pyplot`): Simplifies plotting like MATLAB.
  - **Object-Oriented Interface**: Provides more control by creating figure and axis objects.
- **Supports Multiple Backends**: Works with Jupyter Notebooks, GUI applications, and scripts.
- **Integration with Other Libraries**: Works well with NumPy, Pandas, Seaborn, and SciPy.

2.2 Installing Matplotlib

To install Matplotlib, run:
“‘bash pip install matplotlib

Pie Chart

What is a Pie Chart?

A **Pie Chart** is a circular statistical graphic that is divided into slices to illustrate numerical proportions. Each slice represents a category, and the **arc length** of each slice is proportional to the quantity it represents.

It is called a **pie chart** because it looks like a sliced pie, but there are variations in how it can be presented.

###

Where is a Pie Chart Used?

Percentage or Proportional Data: Pie charts are ideal for visualizing part-to-whole relationships.

Small Number of Categories: Best for datasets with **6 categories or fewer**.

Business & Marketing: Market share, budget distribution, or customer demographics.

Education

When NOT to Use a Pie Chart?
- When there are **too many categories** (makes the chart cluttered).
- If the **categories are not mutually exclusive** (values overlap).
- If exact comparisons are needed (Bar charts are often better).

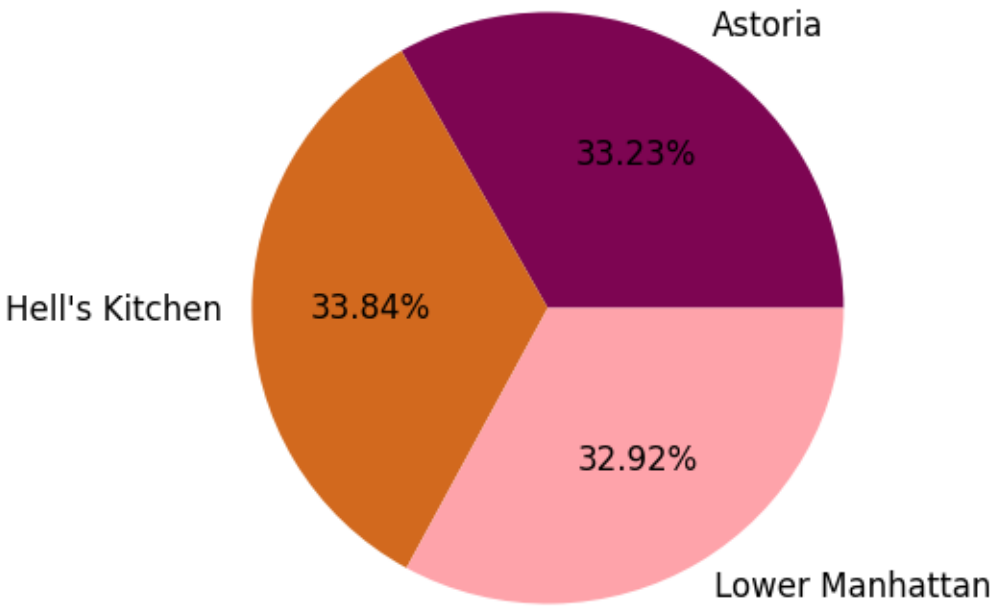
2.2.1 Problem statement

- Finding the Overall Total sales percentage or weightage of Store location
- *Question can be –* : You have a dataset containing sales transactions with columns such as "store_location" and "total_sales". Write a Python script to calculate the total sales for each store location and visualize the sales distribution

using a pie chart. Ensure the pie chart includes percentage labels and custom colors (red, yellow, green).

[11]:
```
# finding the total sales as per store location
store_sales=pd.DataFrame(sales.groupby("store_location")["total_sales"].sum()).
 ↪reset_index()

print(store_sales)
# ploting the pie chart with the help of matplotlib
plt.figure(figsize=(10,5))
plt.pie(store_sales["total_sales"],
        labels=store_sales["store_location"],
        autopct='%1.2f%%',
        colors=["#7D0552",'#D2691E','#FEA3AA'],
        textprops={'fontsize': 12})
plt.show()
```

```
    store_location  total_sales
0          Astoria    232243.91
1   Hell's Kitchen    236511.17
2  Lower Manhattan    230057.25
```



Line Chart

###

What is a Line Chart?
A **Line Chart** is a type of data visualization that displays information as a series of data points connected by a continuous line. It is commonly used to show **trends over time** or continuous data relationships.

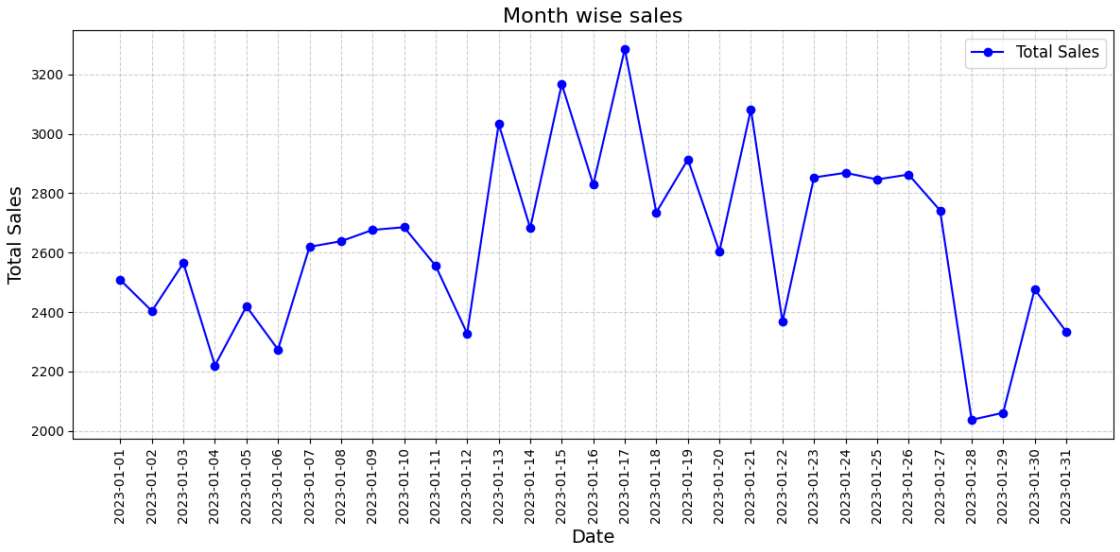
Why Use a Line Chart?

Shows Trends Over Time: Best for visualizing changes and patterns over a period.
Easy to Interpret: The slope and direction of the line make it intuitive.
Effective for Comparisons: Multiple lines can be plotted to compare different

2.2.2 Probleam statement 1

- *Question :* Write a Python script that takes a user input for a month and then filters sales data to display a trend of total sales per day for that month. The dataset contains a "transaction_date" column with timestamps and a "total_sales" column. Use a line chart to visualize the trend, ensuring that dates are properly formatted on the x-axis.

[12]:
```
#print(sales["product_category"].unique())
# date wise sales using input user for month year

month=input("Enter the month : ") # thi is the input from user for month so␣
 ↪that we can get the sales of that month


#trend_sales= sales[sales["transaction_date"].dt.month == int(month)]
date_trend_sale=pd.DataFrame(sales[sales["transaction_date"].dt.month ==␣
 ↪int(month)].
                             groupby(sales["transaction_date"].dt.
 ↪date)["total_sales"].sum()).reset_index()
date_trend_sale["transaction_date"]=pd.
 ↪to_datetime(date_trend_sale["transaction_date"])

# above code is for getting the sales of the month which is input by the user␣
 ↪and then we are grouping the sales as per date and then we are making the␣
 ↪date as datetime format for plotting the graph

# ploting line chart
plt.figure(figsize=(12, 6))
plt.plot(date_trend_sale['transaction_date'], date_trend_sale['total_sales'],
         marker='o', linestyle='-', color='blue', label='Total Sales')
plt.title('Month wise sales', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.xticks(date_trend_sale['transaction_date'],␣
 ↪labels=date_trend_sale['transaction_date'].dt.strftime('%Y-%m-%d'),␣
 ↪rotation=90)

plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
```

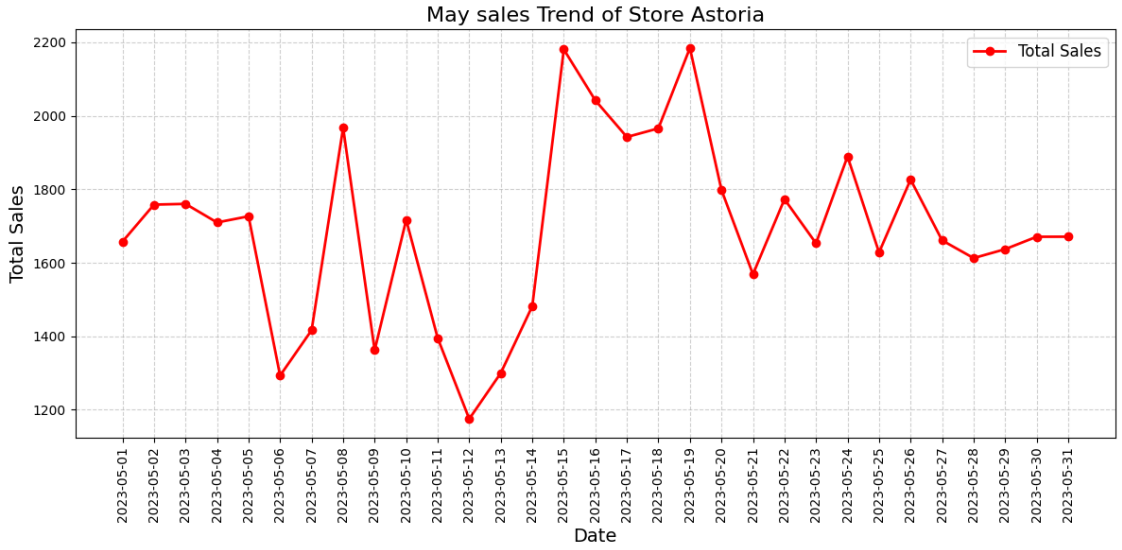


2.2.3 Probleam statement 2

- *Question :* Write a Python script that prompts the user to enter a month and a store location. Using a dataset containing "transaction_date" and "total_sales", filter the data to calculate total daily sales for the selected month and location. Then, visualize the sales trend using a line chart, ensuring proper date formatting on the x-axis.

[13]:
```
month=input("Enter the month ['January', 'February', 'March', 'April', 'May',␣
 ↪'June']: ")

location=input("Enter the location ('Lower Manhattan , Hell's Kitchen ,␣
 ↪Astoria) :")



date_trend_sale=pd.DataFrame(sales[
    (sales["transaction_date"].dt.strftime("%B") == month) &
    (sales["store_location"]==location)
    ].groupby(sales["transaction_date"].dt.date)["total_sales"].sum()).
 ↪reset_index()

date_trend_sale["transaction_date"]=pd.
 ↪to_datetime(date_trend_sale["transaction_date"])

# ploting line chart
plt.figure(figsize=(12, 6))
plt.plot(date_trend_sale['transaction_date'], date_trend_sale['total_sales'],
         marker='o', linestyle='-', linewidth=2 ,color='red', label='Total␣
 ↪Sales')
```

```
plt.title(f'{month} sales Trend of Store {location}', fontsize=16)
plt.  xlabel('Date', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.xticks(date_trend_sale['transaction_date'],
           labels=date_trend_sale['transaction_date'].dt.strftime('%Y-%m-%d'),
           rotation=90)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
```



2.3 Bar Chart

2.3.1 What is a Bar Chart?

A **Bar Chart** is a graphical representation of data using rectangular bars of different heights or lengths. The height (or length) of each bar is proportional to the value it represents.

Bar charts can be plotted **vertically** or **horizontally** and are commonly used for **categorical data**.

2.3.2 Where is a Bar Chart Used?

Comparing Different Categories: Used when data is divided into distinct categories.
Business & Finance: Comparing revenue across products or sales in different regions.
Marketing & Research: Customer preferences, survey responses.
Education & Science: Student performance, experiment results.

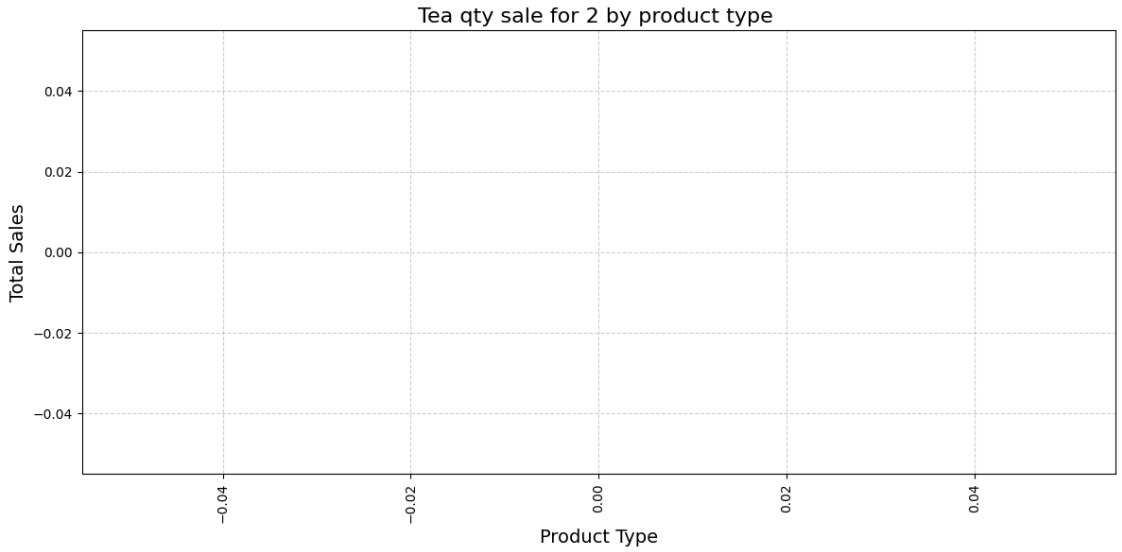
When NOT to Use a Bar Chart?

- When displaying **continuous data** (use a **Line Chart** instead).
- If **data categories are too many**, making it cluttered (consider a grouped or stacked bar chart).

2.3.3 Probleam Statemnt 1

- *Quesiton :* Write a Python script that prompts the user to enter a month and a product category. Using a dataset with "transaction_date", "product_category", and "transaction_qty", filter the data to compute the total quantity sold for each product type within the selected month and category. Visualize the results using a bar chart, ensuring proper labeling and formatting.

[14]:
```
month=input("Enter the month : ")
product_category=input("Enter the product_category ,'Coffee', 'Tea', 'Drinking␣
 ↪Chocolate', 'Bakery', 'Flavours','Loose Tea', 'Coffee beans', 'Packaged␣
 ↪Chocolate', 'Branded': ")


qty_order_pt =pd.DataFrame(sales[sales["transaction_date"].dt.month ==␣
 ↪int(month) &
      (sales["product_category"]==product_category)].
 ↪groupby("product_type")["transaction_qty"].sum()).reset_index()

qty_order_pt

# ploting the bar chart
plt.figure(figsize=(12, 6))
plt.bar_label(plt.bar(qty_order_pt['product_type'],
                      qty_order_pt['transaction_qty'],
                      color='Green'))

plt.title(f'{product_category} qty sale for {month} by product type ',␣
 ↪fontsize=16)
plt.xlabel('Product Type', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.xticks(rotation=90)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



[15]:
```
sales.head()
```

[15]:

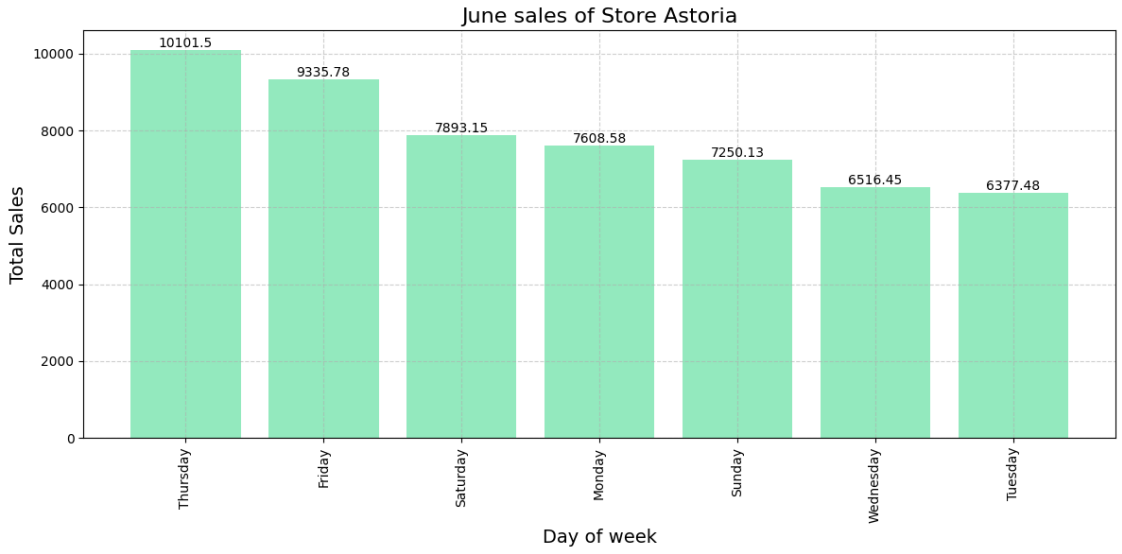
	transaction_id	transaction_date	month	transaction_time	time	day_of_week	transaction_qty	store_id	store_location	product_id	unit_price	product_category	product_type	product_detail	total_sales
0	1	2023-01-01	January	1900-01-01 07:06:11	07:06:11	Sunday	2	5	Lower Manhattan	32	3.0	Coffee	Gourmet brewed coffee	Ethiopia Rg	6.0
1	2	2023-01-01	January	1900-01-01 07:08:56	07:08:56	Sunday	2	5	Lower Manhattan	57	3.1	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	6.2
2	3	2023-01-01	January	1900-01-01 07:14:04	07:14:04	Sunday	2	5	Lower Manhattan	59	4.5	Drinking Chocolate	Hot chocolate	Dark chocolate Lg	9.0
3	4	2023-01-01	January	1900-01-01 07:20:24	07:20:24	Sunday	1	5	Lower Manhattan	22	2.0	Coffee	Drip coffee	Our Old Time Diner Blend Sm	2.0
4	5	2023-01-01	January	1900-01-01 07:22:41	07:22:41	Sunday	2	5	Lower Manhattan	57	3.1	Tea	Brewed Chai tea		

```
4     Spicy Eye Opener Chai Lg          6.2
```

2.3.4 Probleam Statement 2

- **Question :** Find which dayof week has highest sales for each store location and month

[16]:
```
month=input("Enter the month ['January', 'February', 'March', 'April', 'May',␣
 ↪'June']: ")
location=input("Enter the location ('Lower Manhattan , Hell's Kitchen ,␣
 ↪Astoria) :")


week_sales=sales[(sales["transaction_date"].dt.strftime("%B")==month) &
      (sales["store_location"]==location)].
 ↪groupby("day_of_week")["total_sales"].sum().reset_index().
 ↪sort_values("total_sales",ascending=False)


# ploting the bar chart
plt.figure(figsize=(12, 6))
plt.bar_label(plt.bar(week_sales['day_of_week'],
                      week_sales['total_sales'],
                      color='#93E9BE'))
plt.title(f'{month} sales of Store {location}', fontsize=16)
plt.xlabel('Day of week', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.xticks(rotation=90)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



[17]:
```
# What is the most popular time of day for transactions?

time_sales=sales.groupby("time")["transaction_id"].count().reset_index().
 ↪sort_values("transaction_id",ascending=False).head(10)
time_sales
```

[17]:

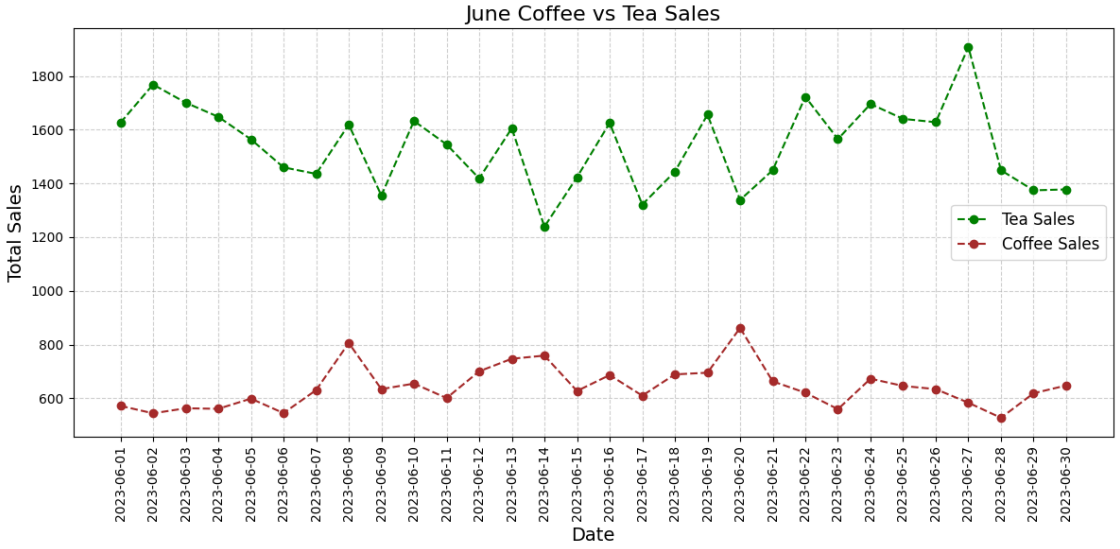
	time	transaction_id
6857	09:31:15	41
3723	08:15:41	40
3868	08:19:08	38
11794	11:40:03	36
8524	10:11:25	36
9474	10:34:04	36
7428	09:44:57	35
8398	10:08:24	35
7716	09:52:03	35
3406	08:08:13	35

2.3.5 Questions :

- *What is the trend of coffee vs. tea sales over time?*

[18]:
```
# What is the trend of coffee vs. tea sales over time?
month=input("Enter the month ['January', 'February', 'March', 'April', 'May',␣
 ↪'June']: ")

tea_data = sales[(sales["product_category"] == "Tea") &
                 (sales["transaction_date"].dt.strftime("%B") == month)].
 ↪groupby("transaction_date")["total_sales"].sum().reset_index()

coffee_data = sales[(sales["product_category"] == "Bakery") &
                    (sales["transaction_date"].dt.strftime("%B") == month)].
 ↪groupby("transaction_date")["total_sales"].sum().reset_index()

plt.figure(figsize=(12, 6))
plt.plot(tea_data['transaction_date'], tea_data['total_sales'], marker='o',␣
 ↪linestyle='--', color='green', label='Tea Sales')
plt.plot(coffee_data['transaction_date'], coffee_data['total_sales'],␣
 ↪marker='o', linestyle='--', color='brown', label='Coffee Sales')

plt.title(f'{month} Coffee vs Tea Sales', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
```

```
plt.xticks(tea_data['transaction_date'], labels=tea_data['transaction_date'].dt.
  ↪strftime('%Y-%m-%d'), rotation=90)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
```

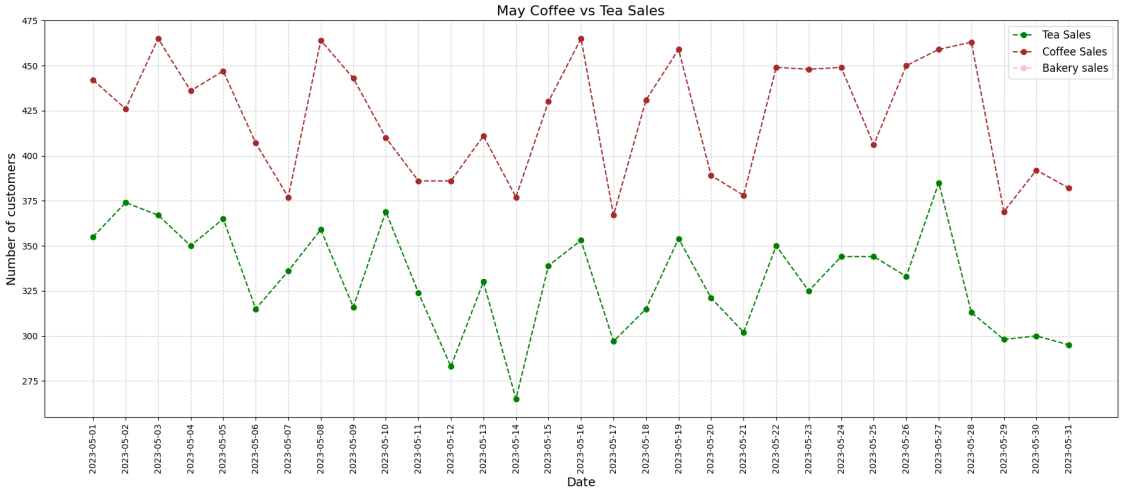


[19]:
```
sales["product_category"].unique()
```

[19]:
```
array(['Coffee', 'Tea', 'Drinking Chocolate', 'Bakery', 'Flavours',
       'Loose Tea', 'Coffee beans', 'Packaged Chocolate', 'Branded'],
      dtype=object)
```

[20]:
```
# What is the trend of coffee vs. tea sales over time?
month=input("Enter the month ['January', 'February', 'March', 'April', 'May',␣
  ↪'June']: ")

tea_data = sales[(sales["product_category"] == "Tea") &
                 (sales["transaction_date"].dt.strftime("%B") == month)].
  ↪groupby("transaction_date")["transaction_id"].count().reset_index()

coffee_data = sales[(sales["product_category"] == "Coffee") &
                    (sales["transaction_date"].dt.strftime("%B") == month)].
  ↪groupby("transaction_date")["transaction_id"].count().reset_index()

bakery_data= sales[(sales["product_category"] == "") &
                    (sales["transaction_date"].dt.strftime("%B") == month)].
  ↪groupby("transaction_date")["transaction_id"].count().reset_index()
```

```
plt.figure(figsize=(18, 8))
plt.plot(tea_data['transaction_date'], tea_data['transaction_id'], marker='o',␣
 ↪linestyle='--', color='green', label='Tea Sales')
plt.plot(coffee_data['transaction_date'], coffee_data['transaction_id'],␣
 ↪marker='o', linestyle='--', color='brown', label='Coffee Sales')
plt.plot(bakery_data['transaction_date'], bakery_data['transaction_id'],␣
 ↪marker='o', linestyle='--', color='pink', label='Bakery sales')
plt.title(f'{month} Coffee vs Tea Sales', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Number of customers', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.xticks(tea_data['transaction_date'], labels=tea_data['transaction_date'].dt.
 ↪strftime('%Y-%m-%d'), rotation=90)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()

print(bakery_data)
```




```
Empty DataFrame
Columns: [transaction_date, transaction_id]
Index: []
```

[21]:
```
sales["product_category"].nunique()
sales["product_category"].unique()
```

[21]:
```
array(['Coffee', 'Tea', 'Drinking Chocolate', 'Bakery', 'Flavours',
       'Loose Tea', 'Coffee beans', 'Packaged Chocolate', 'Branded'],
```

```
      dtype=object)
```

2.4 Scatter Plot

2.4.1 What is a Scatter Plot?

A **Scatter Plot** is a type of data visualization that uses dots to represent values for two different variables. The **position of each dot** on the horizontal and vertical axis indicates values for an individual data point.

Scatter plots are useful for detecting **correlations, trends, and outliers** in datasets.

2.4.2 Where is a Scatter Plot Used?

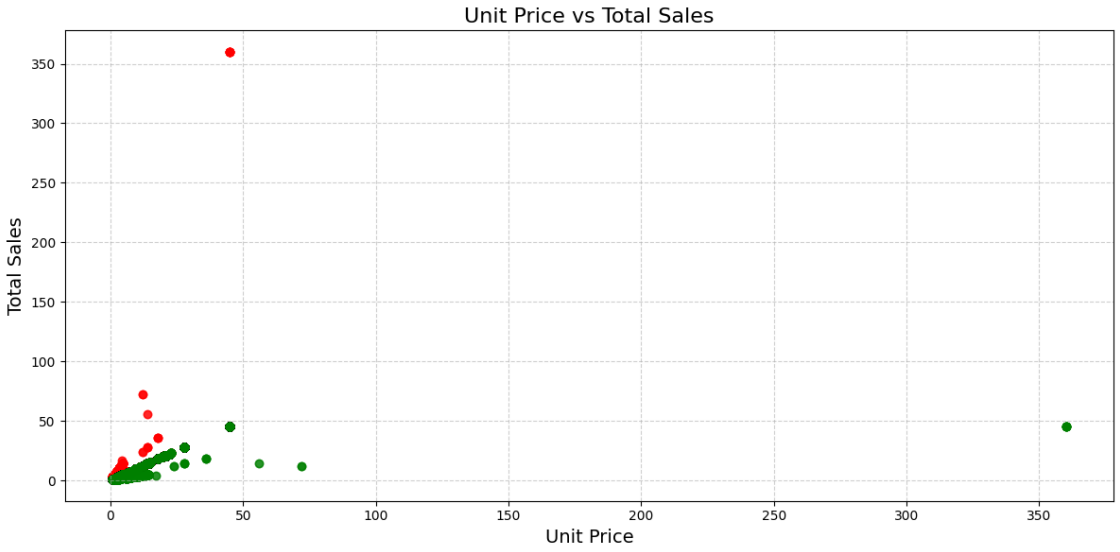
Finding Relationships Between Two Variables: Example: Height vs. Weight.
Scientific & Statistical Analysis: Identifying patterns in experiments.
Business & Finance: Visualizing customer spending vs. income.
Machine Learning: Identifying clusters and outliers in data.

When NOT to Use a Scatter Plot?
- If you only have **one variable** (use a **Histogram or Bar Chart** instead).
- If **data points overlap too much**, making trends hard to see.

2.4.3 Probleam Statement

- *A retail chain wants to analyze whether increasing the price of products has a direct impact on total sales. By examining the relationship between unit price and total sales, they aim to determine if premium-priced products generate more revenue or if lower-priced products contribute more due to higher sales volume.*

[22]:
```
plt.figure(figsize=(12, 6))
plt.scatter(sales['unit_price'], sales['total_sales'], color='red', alpha=0.6,␣
 ↪label="Unit Price")

# Plot total sales (Green)
plt.scatter(sales['total_sales'], sales['unit_price'], color='green', alpha=0.
 ↪6, label="Total Sales")

plt.title('Unit Price vs Total Sales', fontsize=16)
plt.xlabel('Unit Price', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



2.5 Donut Chart

2.5.1 What is a Donut Chart?

A **Donut Chart** is a variation of a **Pie Chart** with a hollow center. It represents data in a circular format, where each segment shows the proportion of a category.

The **main difference** between a Donut Chart and a Pie Chart is that the center is removed, making it visually distinct.

2.5.2 Where is a Donut Chart Used?

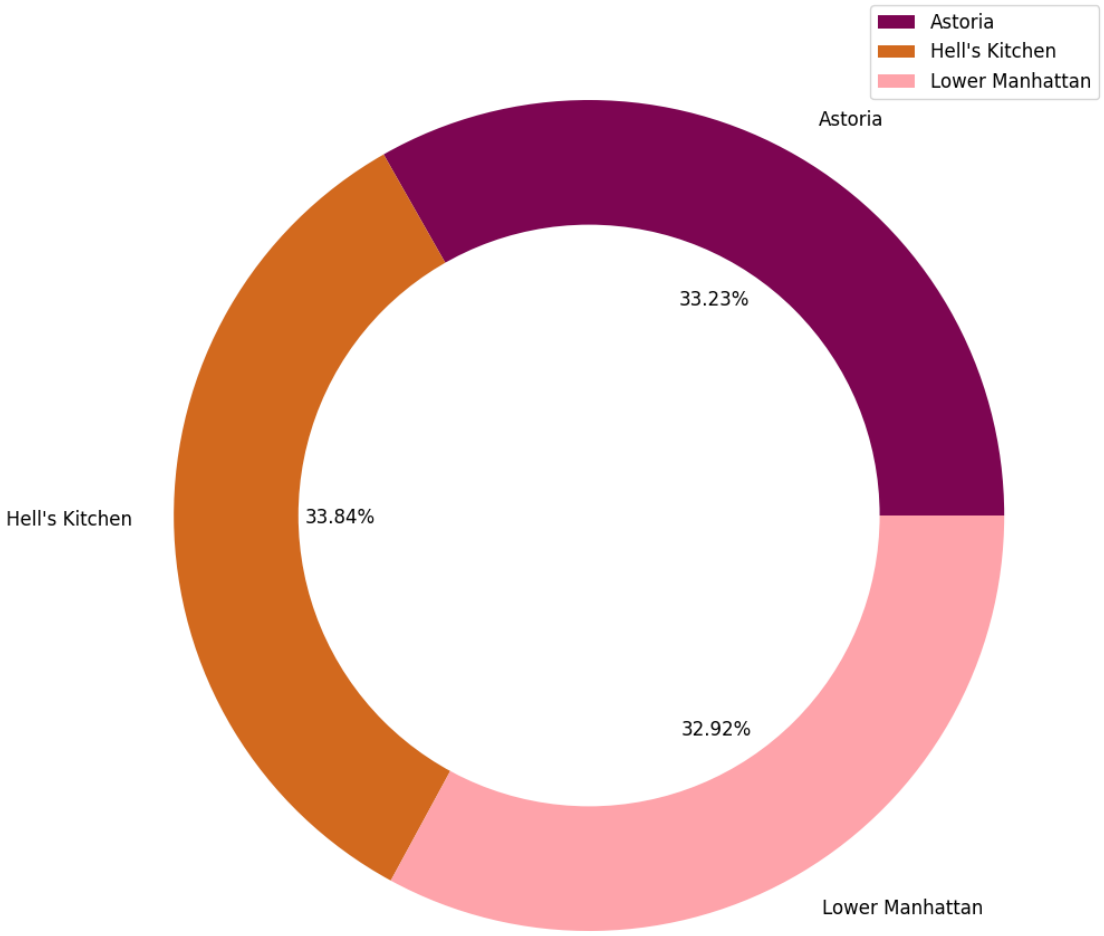
Proportional Data Visualization: Just like a Pie Chart, it shows the **part-to-whole relationship** of categories.
Business & Finance: Market share, expense breakdown, budget allocation.
Marketing & Research: Customer segmentation, sales distribution.
Better Readability: The hollow center makes it easier to compare slices compared to a full Pie Chart.

When NOT to Use a Donut Chart?
- When **precise numerical comparison** is required (Bar Charts are better).
- If there are **too many categories** (hard to differentiate).

---

[23]:
```
# donut chart
store_sales=pd.DataFrame(sales.groupby("store_location")["total_sales"].sum()).
  ↪reset_index()
plt.figure(figsize=(12,14))
plt.pie(store_sales["total_sales"],
```

```
        labels=store_sales["store_location"],
        autopct='%1.2f%%',
        colors=["#7D0552",'#D2691E','#FEA3AA'],
        textprops={'fontsize': 12})
plt.gca().add_artist(plt.Circle((0,0),0.70,fc='white'))
plt.legend(fontsize=12, loc='upper right')
plt.show()
```



2.6 Filled Area Chart

2.6.1 What is a Filled Area Chart?

A **Filled Area Chart** is a variation of a Line Chart where the area below the line is filled with color. It is useful for visualizing the magnitude of a variable over time or comparing multiple datasets.

It helps **highlight trends and differences** in datasets by filling the space beneath the lines.

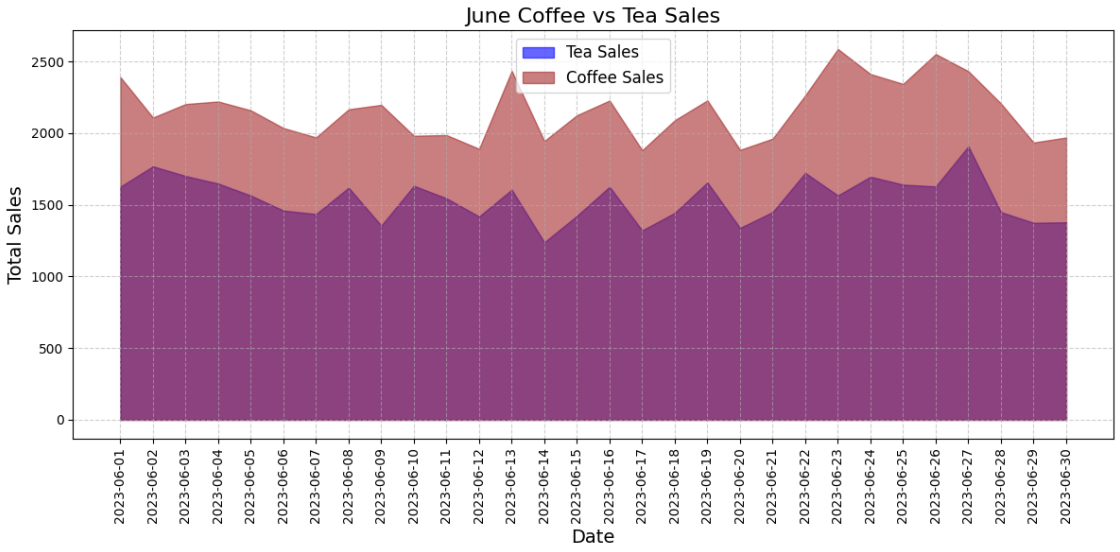
2.6.2 Where is a Filled Area Chart Used?

Trend Analysis: Best for showing changes over time (e.g., stock prices, temperature variations).
Comparing Multiple Series: Helps compare overlapping data (e.g., revenue vs. expenses).
Business & Finance: Visualizing income, costs, and profits.
Weather & Environment: Rainfall, temperature fluctuations, energy consumption.

When NOT to Use a Filled Area Chart?
- If **precise comparisons** are needed (use **Bar Charts** instead).
- If there are **too many overlapping areas**, making it cluttered.

---

[24]:
```
# filled area chart
month=input("Enter the month ['January', 'February', 'March', 'April', 'May',␣
 ↪'June']: ")

tea_data = sales[(sales["product_category"] == "Tea") &
                    (sales["transaction_date"].dt.strftime("%B") == month)].
 ↪groupby("transaction_date")["total_sales"].sum().reset_index()

coffee_data = sales[(sales["product_category"] == "Coffee") &
                    (sales["transaction_date"].dt.strftime("%B") == month)].
 ↪groupby("transaction_date")["total_sales"].sum().reset_index()

plt.figure(figsize=(12, 6))
# Fill area means fill the area between the line and x-axis with color
plt.fill_between(tea_data['transaction_date'], tea_data['total_sales'],␣
 ↪color='blue', alpha=0.6, label='Tea Sales')
plt.fill_between(coffee_data['transaction_date'], coffee_data['total_sales'],␣
 ↪color='brown', alpha=0.6, label='Coffee Sales')
plt.title(f'{month} Coffee vs Tea Sales', fontsize=16)
plt.xlabel('Date', fontsize=14)
plt.ylabel('Total Sales', fontsize=14)
plt.grid(True, linestyle='--', alpha=0.6)
plt.xticks(tea_data['transaction_date'], labels=tea_data['transaction_date'].dt.
 ↪strftime('%Y-%m-%d'), rotation=90)
plt.legend(fontsize=12)
plt.tight_layout()
plt.show()
```



2.7 Box Plot (Whisker Plot) in Matplotlib

2.7.1 What is a Box Plot?

A **Box Plot**, also known as a **Whisker Plot**, is a graphical representation of a dataset's **distribution, variability, and outliers**. It provides a summary of data using five key statistics:

1. **Minimum (Lower Whisker)** – The smallest data point, excluding outliers.
2. **First Quartile (Q1, 25th Percentile)** – The median of the lower half of the data.
3. **Median (Q2, 50th Percentile)** – The middle value of the dataset.
4. **Third Quartile (Q3, 75th Percentile)** – The median of the upper half of the data.
5. **Maximum (Upper Whisker)** – The largest data point, excluding outliers.

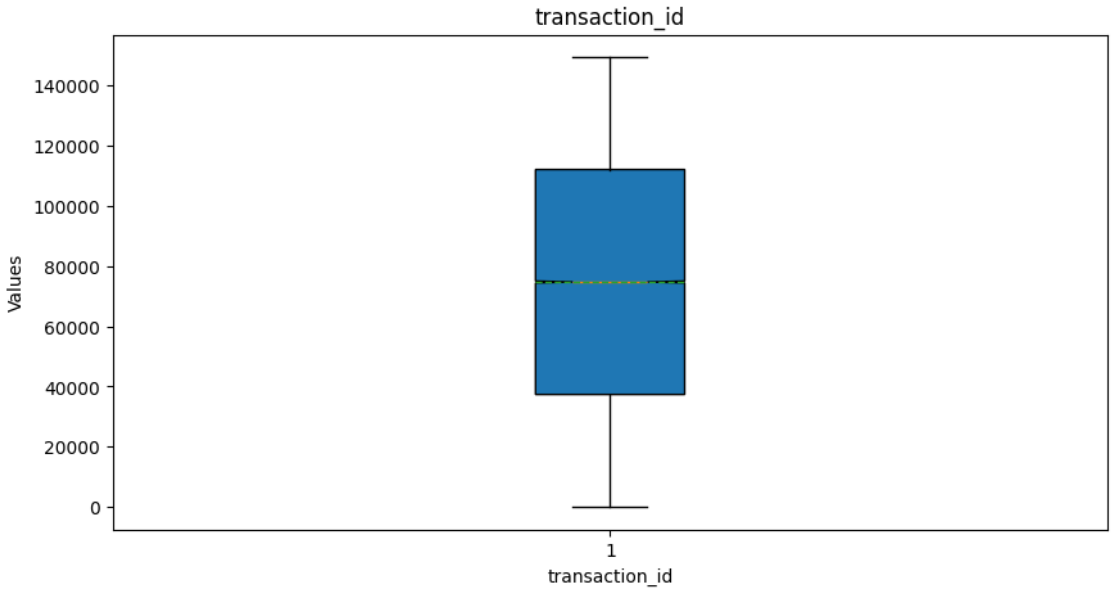
Outliers are data points outside the whiskers and are usually represented by individual dots.

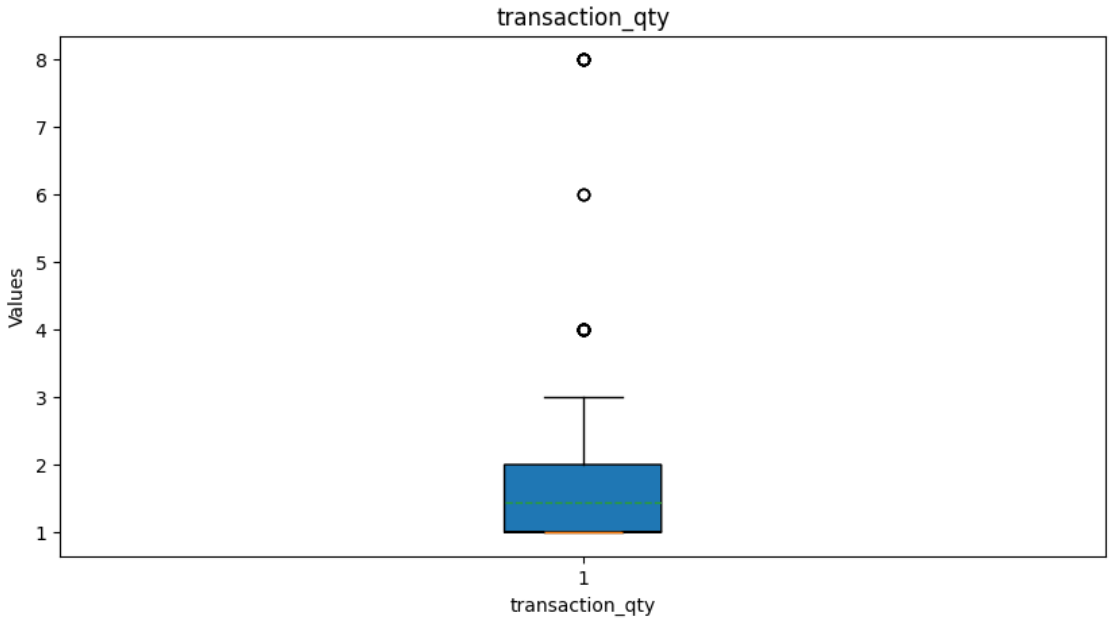
2.7.2 Where is a Box Plot Used?

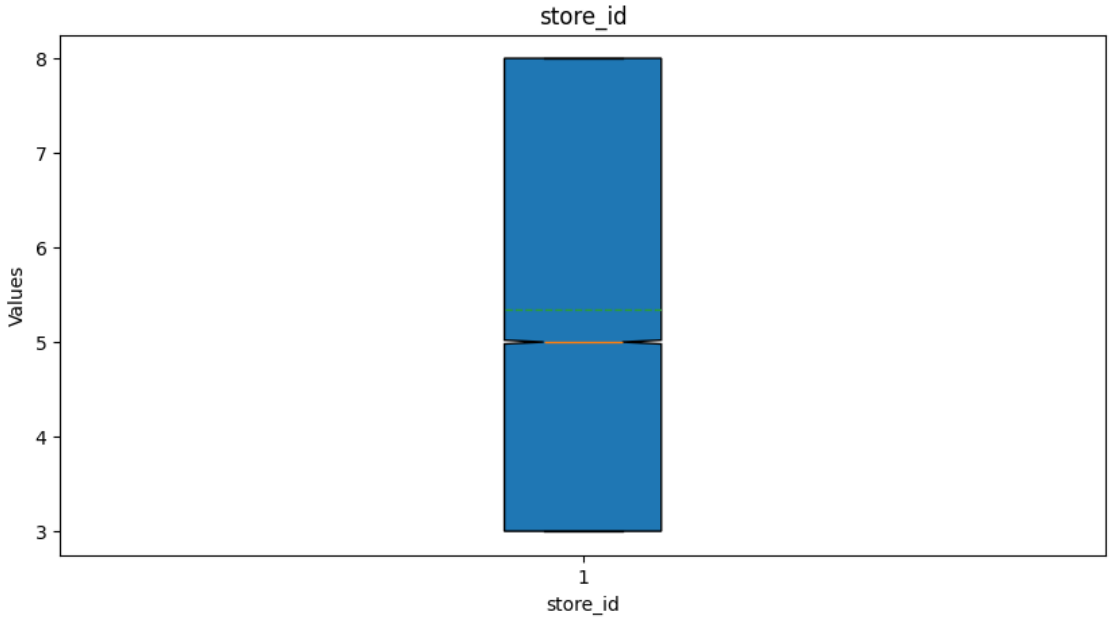
Identifying Outliers: Detect extreme values in data.
Comparing Distributions: Compare multiple datasets side by side.
Data Spread & Skewness: Understand how data is distributed.
Business & Finance: Analyzing sales, revenue, and stock prices.
Science & Healthcare: Measuring patient response times, test scores.

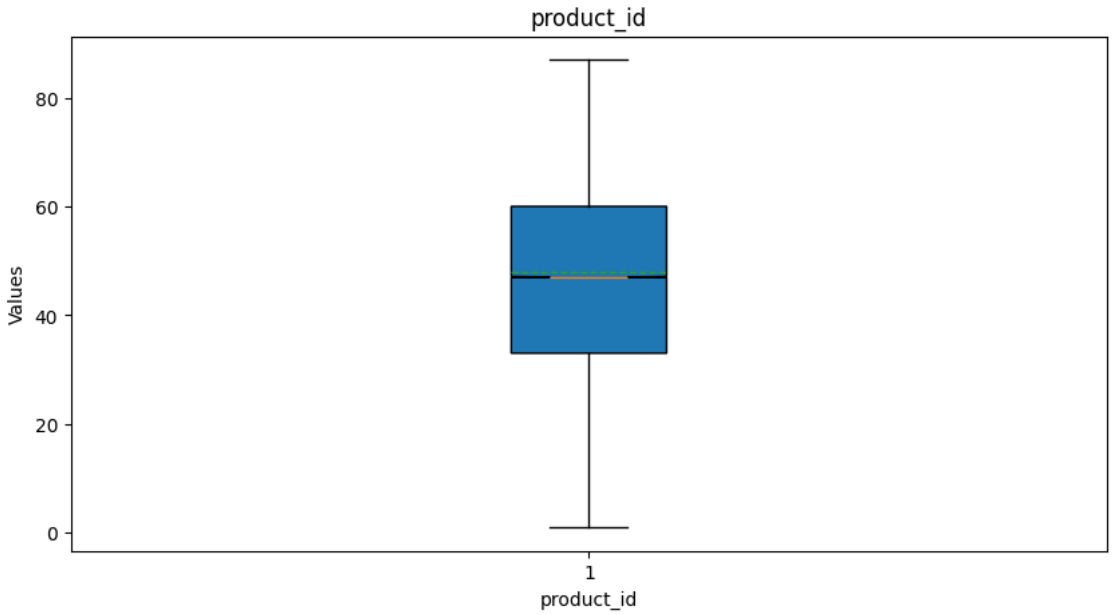
When NOT to Use a Box Plot?
- If you need to **see exact values** (use a Histogram instead).
- If you are **only dealing with a small dataset** (a simple summary might be enough).

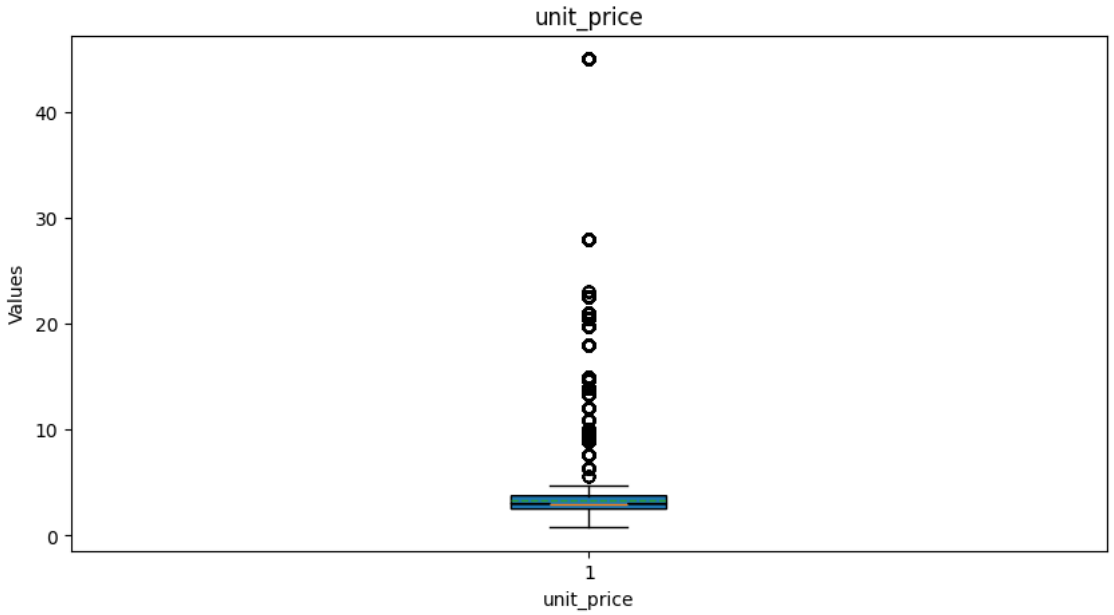
[25]:
```
# box plot
for i in sales.select_dtypes(include="number").columns: # loop through the␣
 ↪columns in the data set which are of numeric data type and plot the box plot␣
 ↪for each column
    plt.figure(figsize=(10,5))
    plt.
 ↪boxplot(sales[i],patch_artist=True,notch=True,showmeans=True,meanline=True)
    # plot the box plot for each column ,patch_artist=True is used to fill the␣
 ↪box plot with color,
    # notch=True is used to show the confidence interval,
    # showmeans=True is used to show the mean of the data,
    # meanline=True is used to show the mean line in the box plot
    plt.title(i)
    plt.xlabel(i)
    plt.ylabel("Values")
    plt.show()
```




transaction_qty
Values
8
7
6
5
4
3
2
1
1
transaction_qty


store_id
Values
8
7
6
5
4
3
1
store_id


product_id
Values
0
20
40
60
80
1
product_id


unit_price
Values
0
10
20
30
40
1
unit_price