

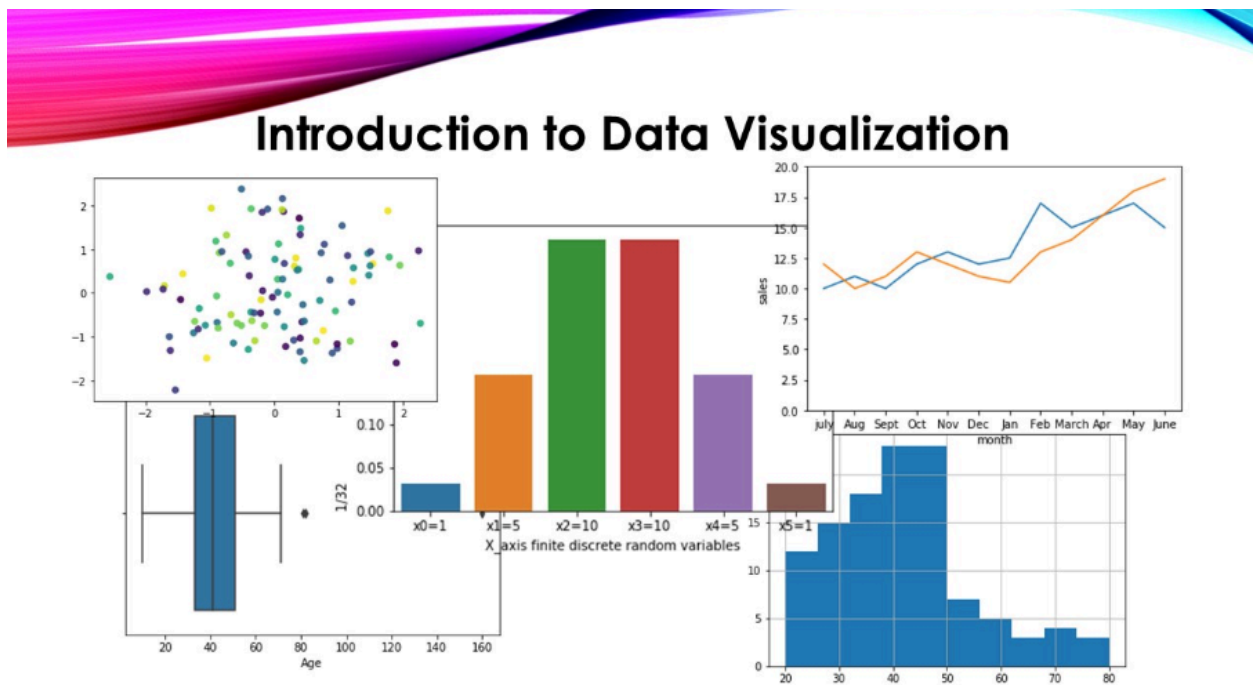
✓ Data Visualization

Agenda

1. Introduction to Data Visualization
2. Libraries for Data Visualization
 - Matplotlib
 - Seaborn
 - Plotly
3. Types of Data Visualization
 - Basic Charts
 - Intermediate Charts
4. Advanced Data Visualization Techniques
 - Pair Plots
 - Facet Grids
 - 3D Plotting
 - Geographical Plots

✓ Introduction to Data Visualization

- Data visualization is the graphical representation of data. It's a powerful tool for understanding complex data sets and communicating findings effectively. By using visual elements like charts, graphs, and maps, data visualization tools provide an accessible way to see and understand trends, outliers, and patterns in data.
- Importance of Data Visualization:
 - Simplifies complex data.
 - Helps in quick decision-making.
 - Enhances understanding of data.
 - Facilitates storytelling with data.



- Why Visualization is used?
 - Data visualization is used to transform complex datasets into easily understandable visual formats, such as charts, graphs, and maps, enhancing clarity and insight. It simplifies the interpretation of large volumes of data by revealing patterns, trends, and outliers that might be obscured in raw data. This process aids in effective data exploration, allowing users to interact with the data dynamically, explore relationships, and generate insights that inform decision-making. By presenting information visually, data visualization supports communication by making complex concepts more accessible to a broader audience, including those without technical expertise. It facilitates comparison of different data points, highlights key metrics and performance indicators, and reveals relationships and correlations between variables.

✓ Libraries for Data Visualization

✓ 1. Matplotlib

- Matplotlib is an easy-to-use, low-level data visualization library that is built on NumPy arrays. It consists of various plots like scatter plot, line plot, histogram, etc. Matplotlib provides a lot of flexibility.
- To install this type the below command in the terminal.

```
pip install matplotlib
```

```

➞ Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: numpy>=1.20 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from

```

- After installing Matplotlib, let's see the most commonly used plots using this library.

1. Scatter Plot

- Scatter plots are used to observe relationships between variables and uses dots to represent the relationship between them. The scatter() method in the matplotlib library is used to draw a scatter plot.

```

import pandas as pd
import matplotlib.pyplot as plt

# reading the database
data = pd.read_csv("tips.csv")

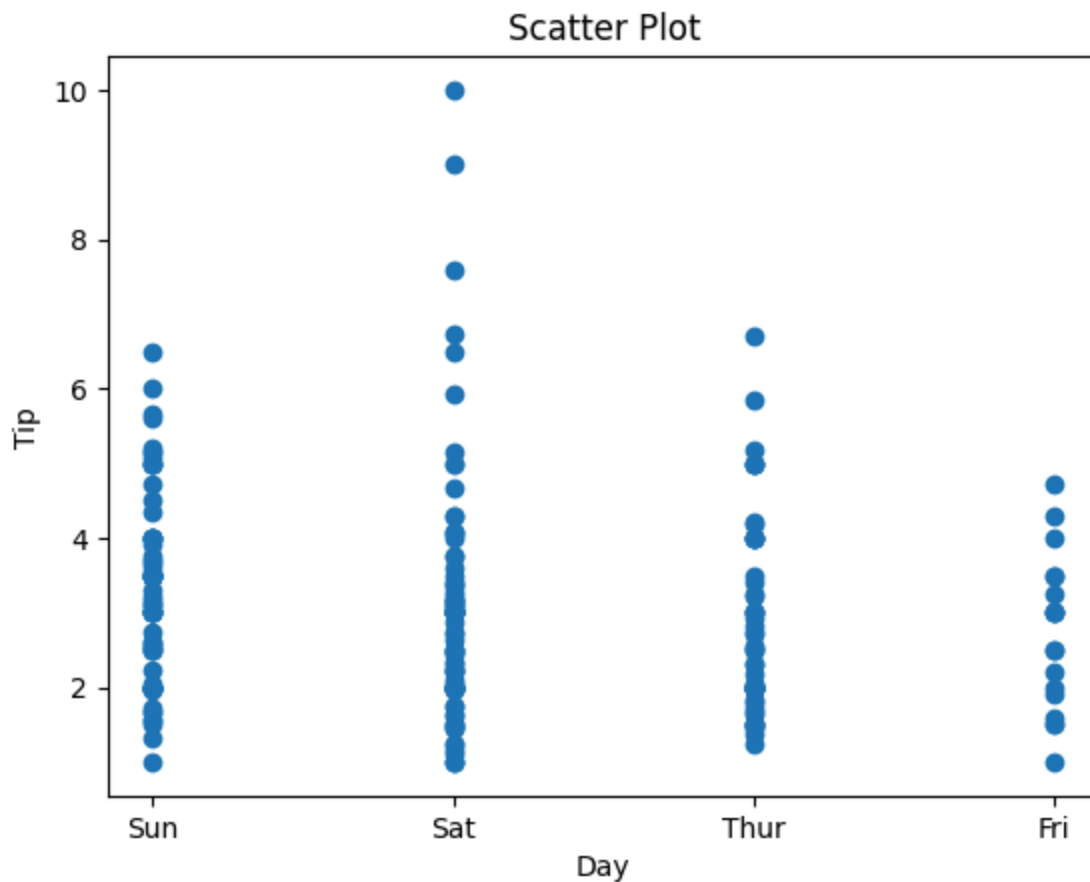
# Scatter plot with day against tip
plt.scatter(data['day'], data['tip'])

# Adding Title to the Plot
plt.title("Scatter Plot")

# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')

plt.show()

```



- This graph can be more meaningful if we can add colors and also change the size of the points. We can do this by using the `c` and `s` parameter respectively of the scatter function. We can also show the color bar using the `colorbar()` method.

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

# reading the database
data = pd.read_csv("tips.csv")

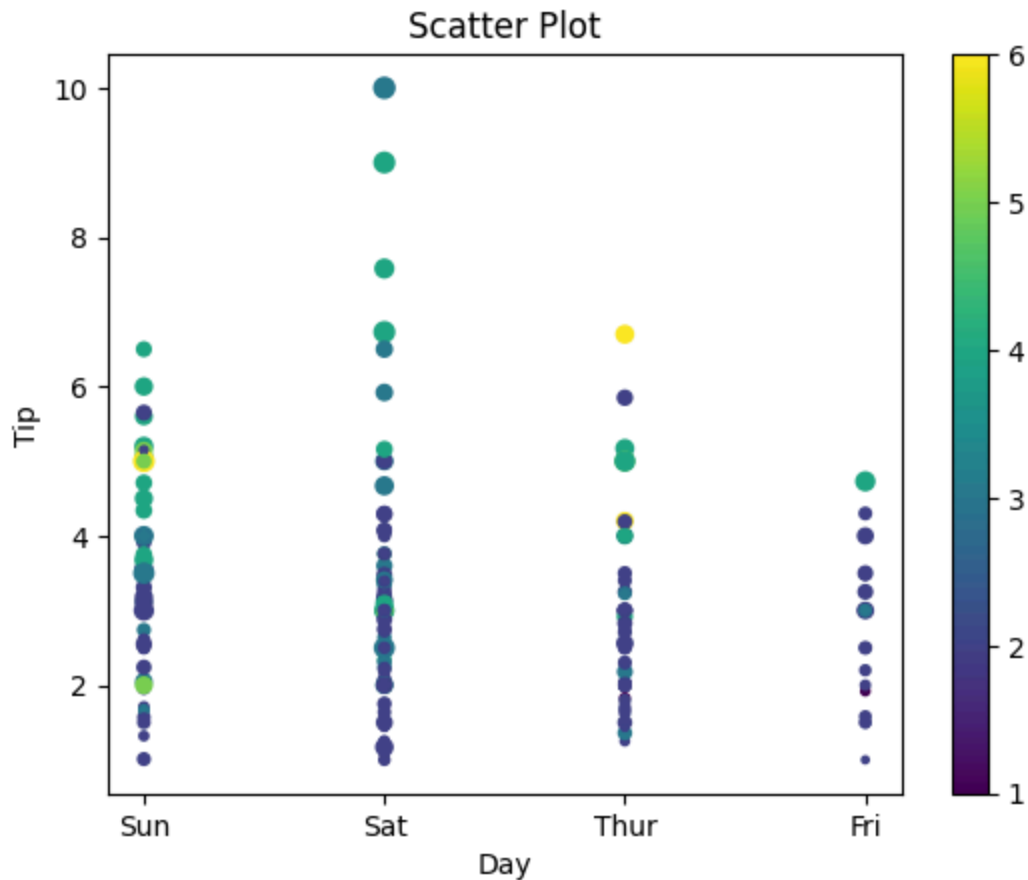
# Scatter plot with day against tip
plt.scatter(data['day'], data['tip'], c=data['size'],
            s=data['total_bill'])

# Adding Title to the Plot
plt.title("Scatter Plot")

# Setting the X and Y labels
plt.xlabel('Day')
plt.ylabel('Tip')

plt.colorbar()
```

```
plt.show()
```



2. Line Chart

- Line Chart is used to represent a relationship between two data X and Y on a different axis. It is plotted using the plot() function. Let's see the below example.

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

# reading the database
data = pd.read_csv("tips.csv")

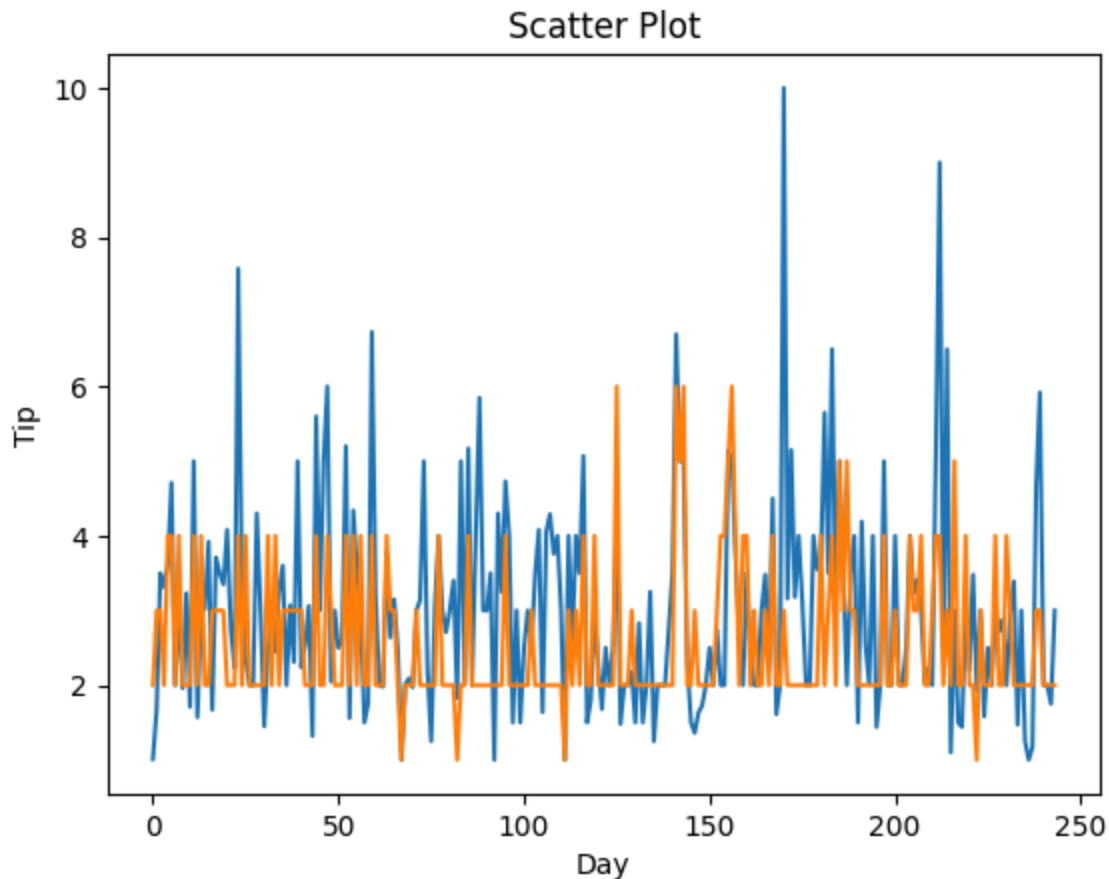
# Scatter plot with day against tip
plt.plot(data['tip'])
plt.plot(data['size'])

# Adding Title to the Plot
plt.title("Scatter Plot")

# Setting the X and Y labels
plt.xlabel('Day')
```

```
plt.ylabel('Tip')
```

```
plt.show()
```



3. Histogram

- A histogram is basically used to represent data in the form of some groups. It is a type of bar plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency. The `hist()` function is used to compute and create a histogram. In histogram, if we pass categorical data then it will automatically compute the frequency of that data i.e. how often each value occurred.

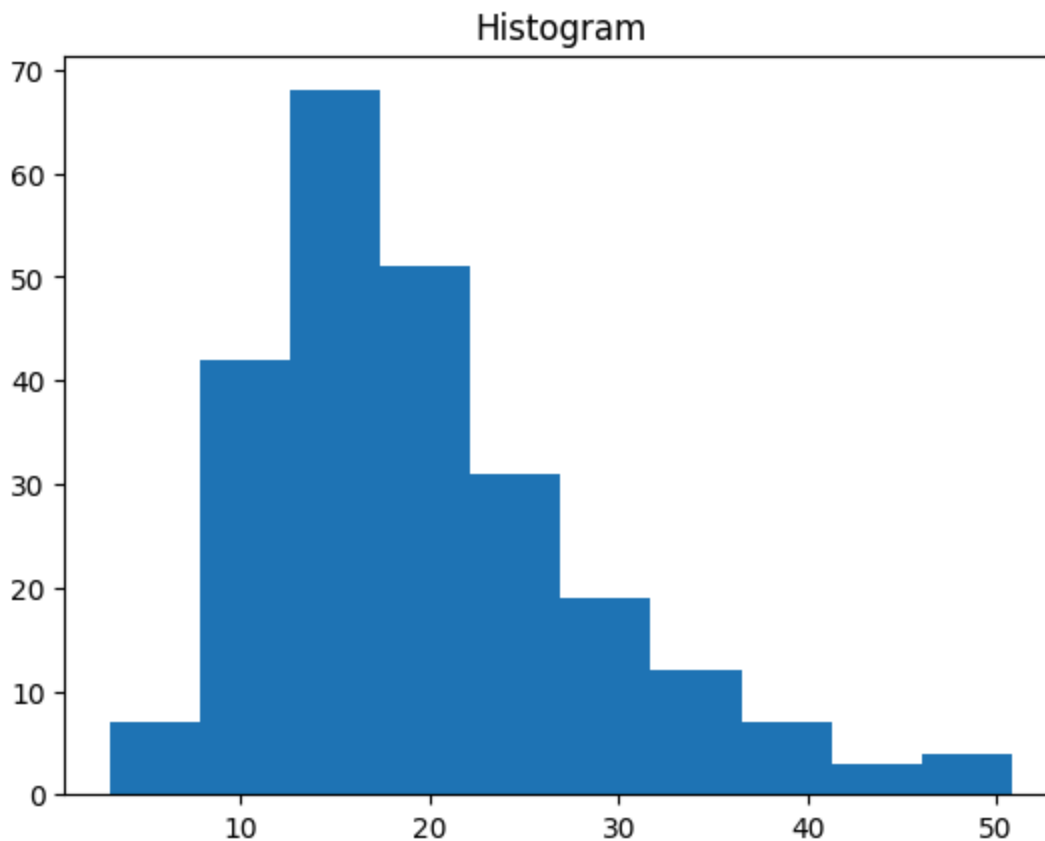
```
import pandas as pd
import matplotlib.pyplot as plt
```

```
# reading the database
data = pd.read_csv("tips.csv")
```

```
# histogram of total_bills
plt.hist(data['total_bill'])
```

```
plt.title("Histogram")
```

```
# Adding the legends
plt.show()
```



✓ Seaborn

- Seaborn is a high-level interface built on top of the Matplotlib. It provides beautiful design styles and color palettes to make more attractive graphs.
- Seaborn is built on the top of Matplotlib, therefore it can be used with the Matplotlib as well. Using both Matplotlib and Seaborn together is a very simple process. We just have to invoke the Seaborn Plotting function as normal, and then we can use Matplotlib's customization function.
- To install seaborn type the below command in the terminal.

```
pip install seaborn
```



```
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /usr/local/lib/python3.10/dist
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (
```

```
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
```

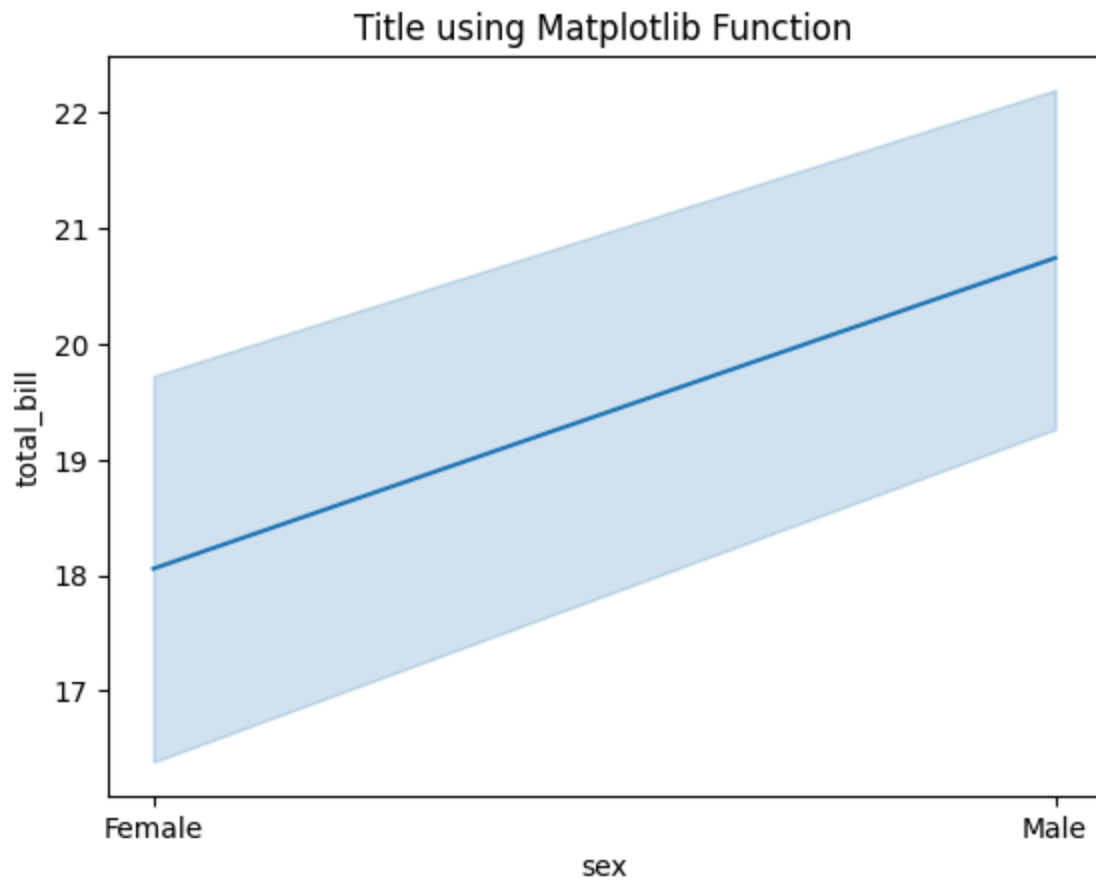
```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

# draw lineplot
sns.lineplot(x="sex", y="total_bill", data=data)

# setting the title using Matplotlib
plt.title('Title using Matplotlib Function')

plt.show()
```

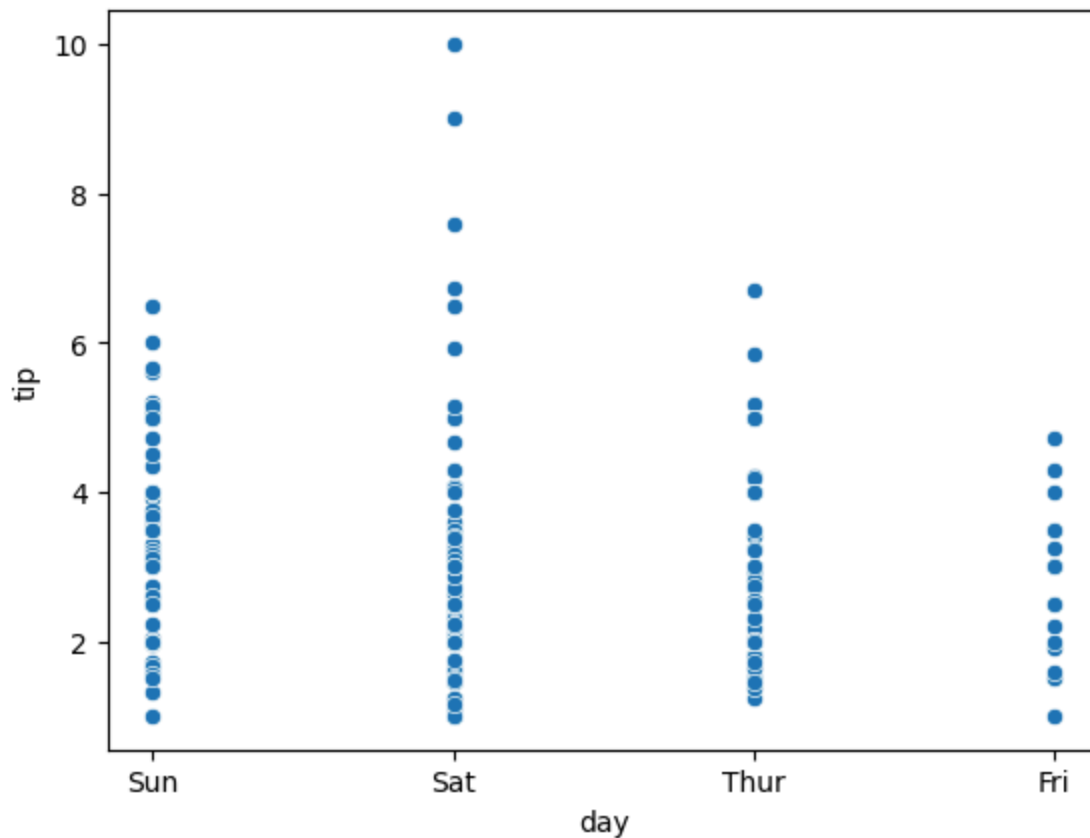
1. Scatter Plot

- Scatter plot is plotted using the `scatterplot()` method. This is similar to Matplotlib, but additional argument `data` is required.

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

sns.scatterplot(x='day', y='tip', data=data,)
plt.show()
```

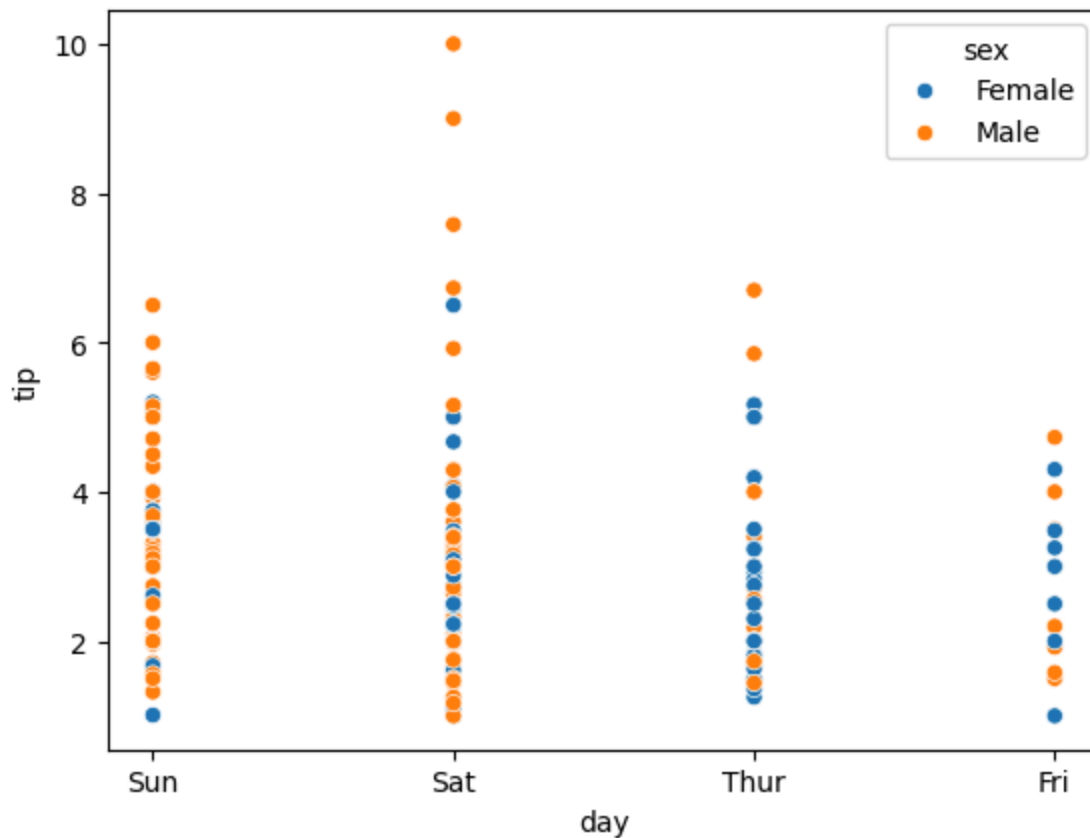


- You will find that while using Matplotlib it will a lot difficult if you want to color each point of this plot according to the sex. But in scatter plot it can be done with the help of hue argument.

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

sns.scatterplot(x='day', y='tip', data=data,
                hue='sex')
plt.show()
```



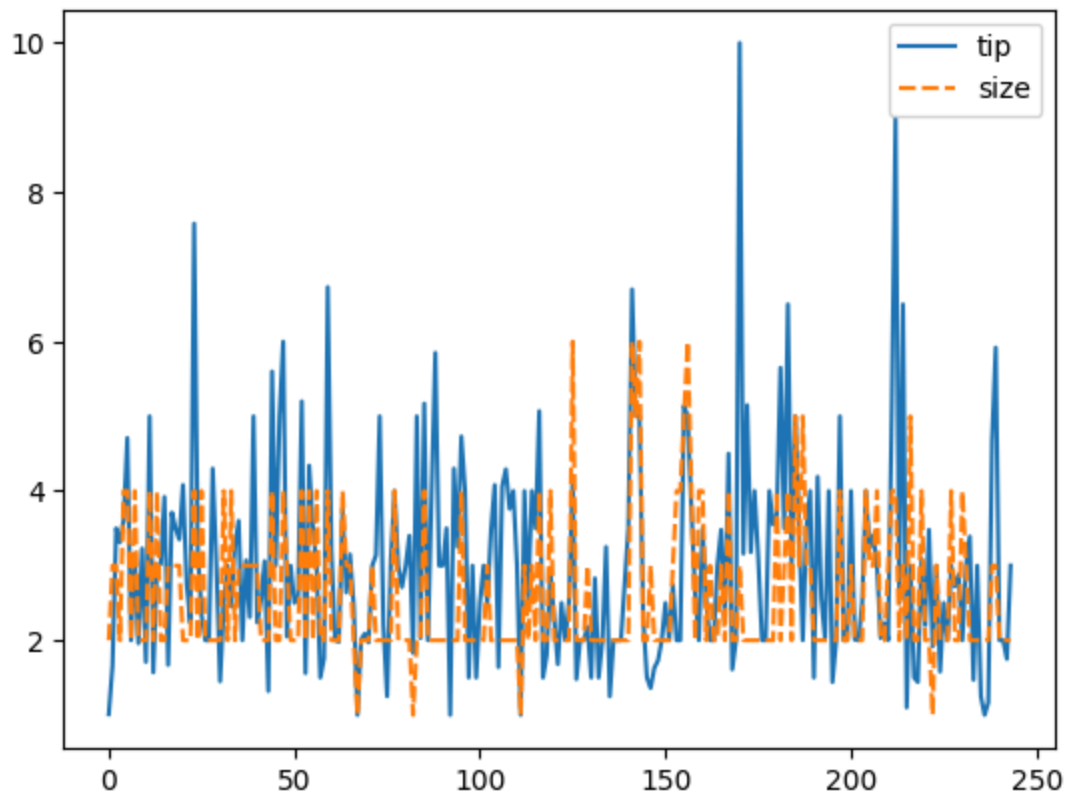
2. Line Plot

- Line Plot in Seaborn plotted using the `lineplot()` method. In this, we can pass only the data argument also.

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

# using only data attribute
sns.lineplot(data=data.drop(['total_bill'], axis=1))
plt.show()
```



3. Bar Plot

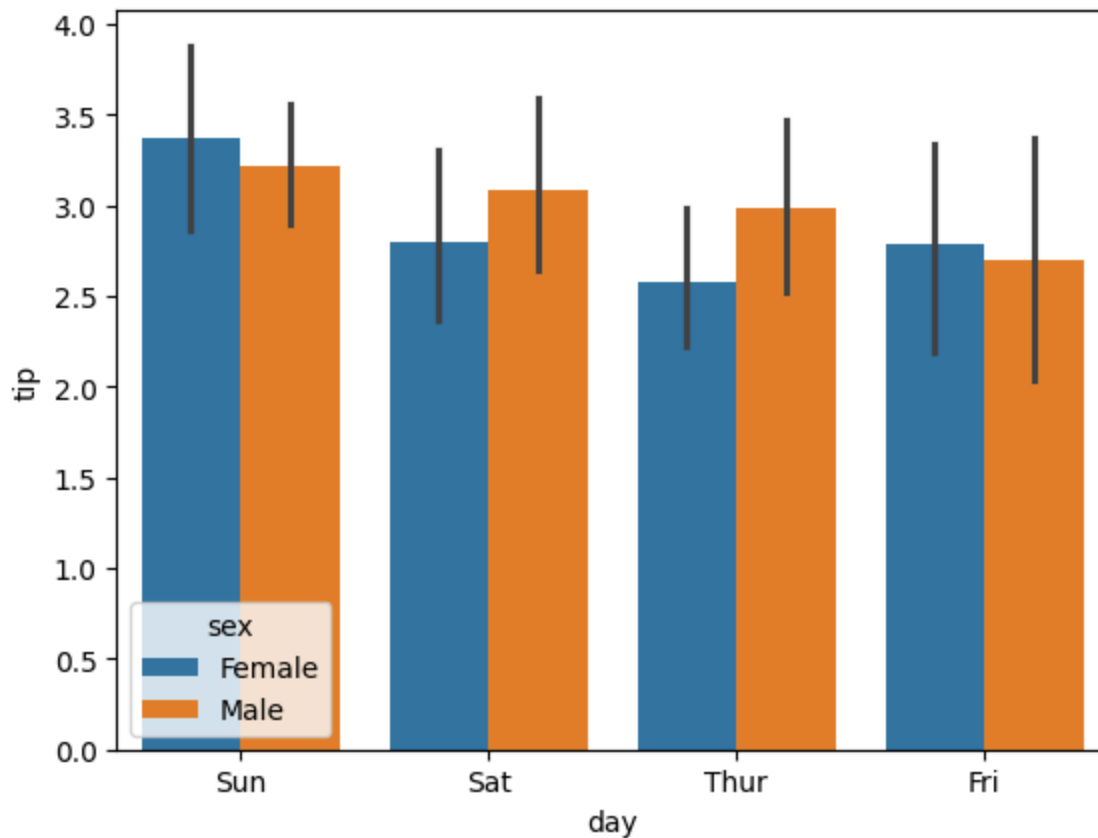
- Bar Plot in Seaborn can be created using the `barplot()` method.

```
# importing packages
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

sns.barplot(x='day', y='tip', data=data,
            hue='sex')

plt.show()
```



✓ Plotly

- This is the last library of our list and you might be wondering why plotly. Here's why –
- Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in numerous data points.
 - It allows more customization.
 - It makes the graph visually more attractive.
 - To install it type the below command in the terminal.

```
pip install plotly
```



```
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (5.15.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (fr
```

1. Scatter Plot

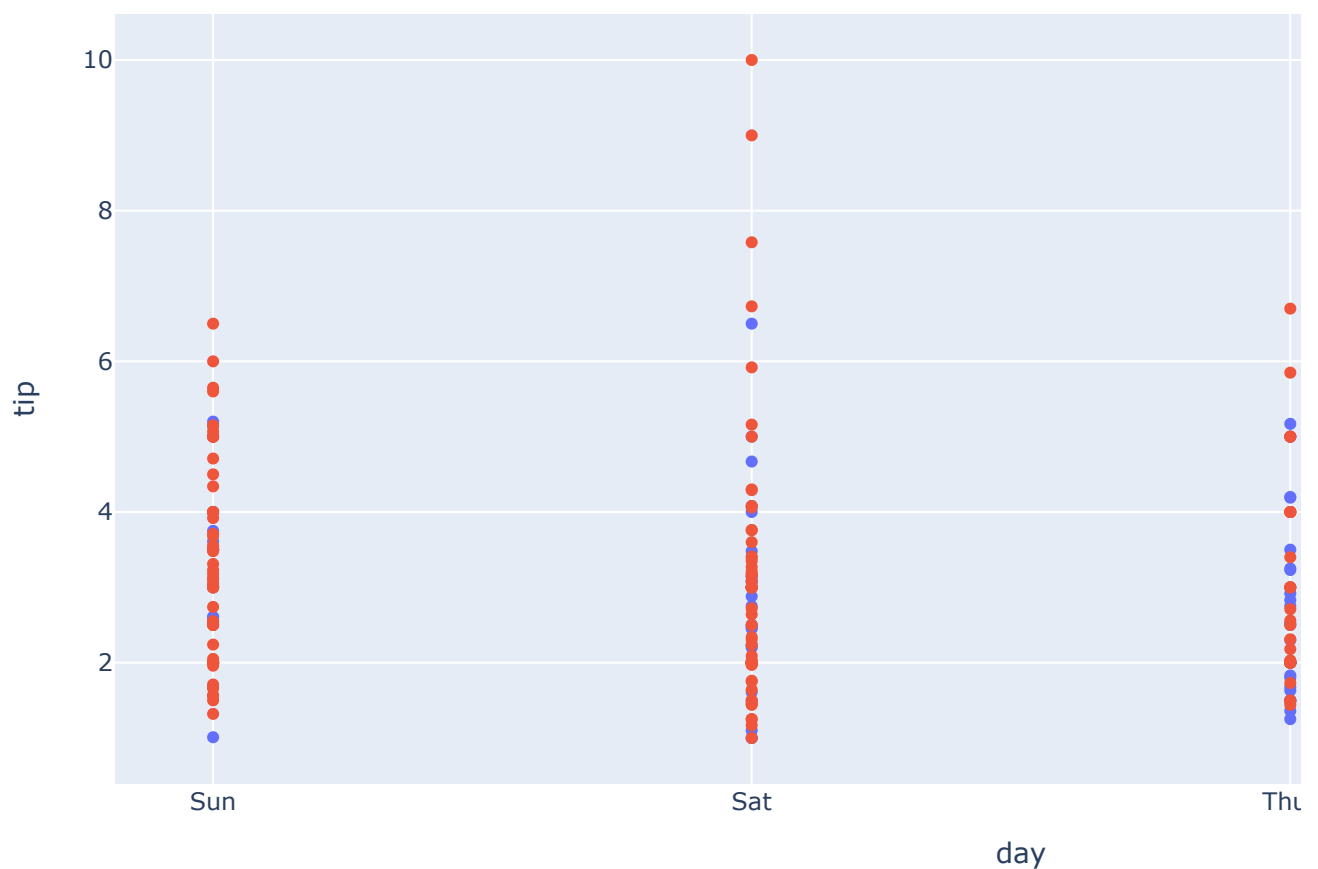
- Scatter plot in Plotly can be created using the `scatter()` method of `plotly.express`. Like Seaborn, an extra data argument is also required here.

```
import plotly.express as px
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

# plotting the scatter chart
fig = px.scatter(data, x="day", y="tip", color='sex')

# showing the plot
fig.show()
```



2. Line Chart

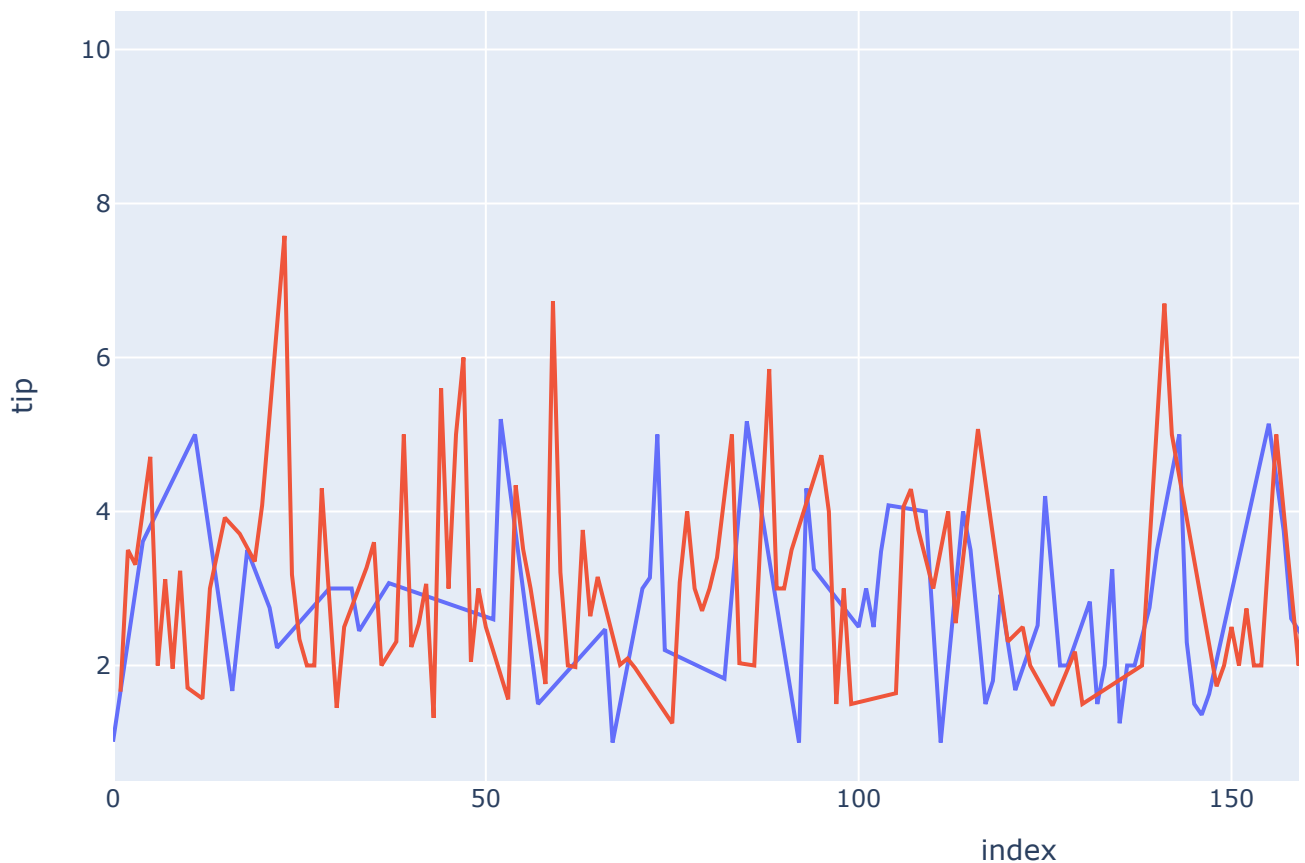
- Line plot in Plotly is much accessible and illustrious annexation to plotly which manage a variety of types of data and assemble easy-to-style statistic. With `px.line` each data position is represented as a vertex

```
import plotly.express as px
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

# plotting the scatter chart
fig = px.line(data, y='tip', color='sex')

# showing the plot
fig.show()
```



3. Bar Chart

- Bar Chart in Plotly can be created using the `bar()` method of `plotly.express` class.

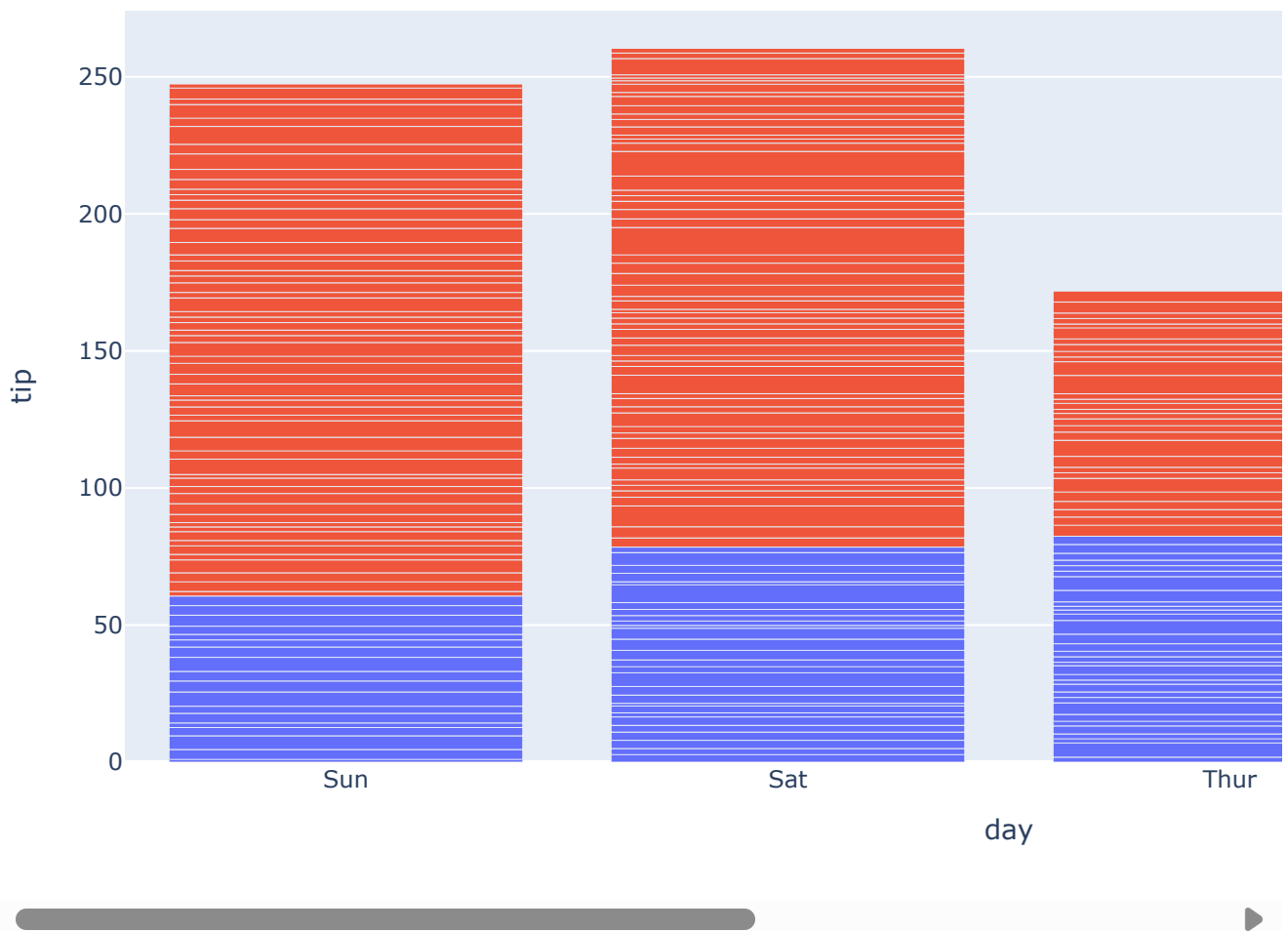
```
import plotly.express as px
import pandas as pd
```

```
# reading the database
```

```
data = pd.read_csv("tips.csv")

# plotting the scatter chart
fig = px.bar(data, x='day', y='tip', color='sex')

# showing the plot
fig.show()
```



✓ Adding interaction

- Plotly also provides various interactions. Let's discuss a few of them.

1. Creating Dropdown Menu: A drop-down menu is a part of the menu-button which is displayed on a screen all the time. Every menu button is associated with a Menu widget that can display the choices for that menu button when clicked on it. In plotly, there are 4 possible methods to modify the charts by using updatemenu method.

- restyle: modify data or data attributes

- `relayout`: modify layout attributes
- `update`: modify data and layout attributes
- `animate`: start or pause an animation

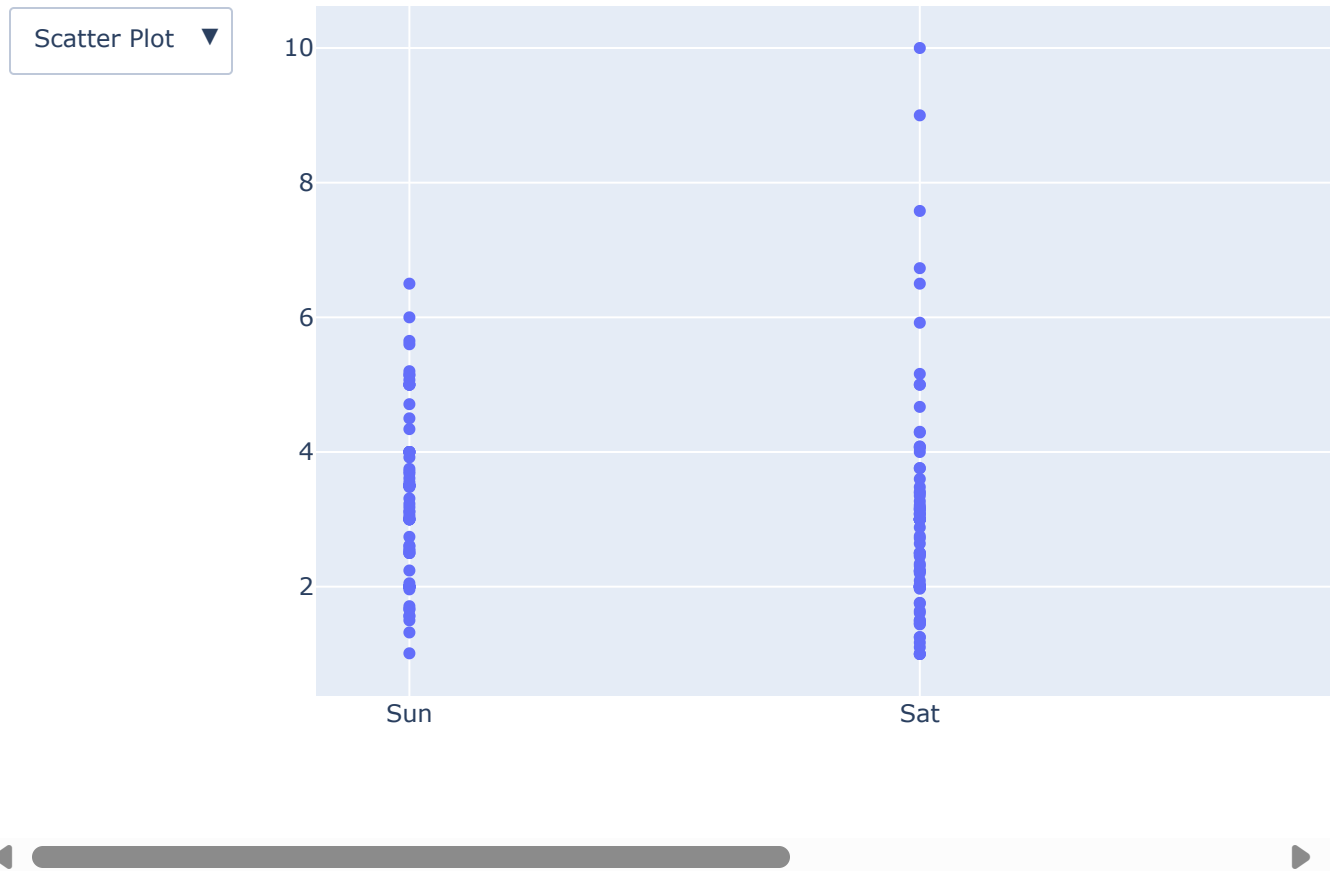
```
import plotly.graph_objects as px
import pandas as pd

# reading the database
data = pd.read_csv("tips.csv")

plot = px.Figure(data=[px.Scatter(
    x=data['day'],
    y=data['tip'],
    mode='markers',)
])

# Add dropdown
plot.update_layout(
    updatemenus=[
        dict(
            buttons=list([
                dict(
                    args=["type", "scatter"],
                    label="Scatter Plot",
                    method="restyle"
                ),
                dict(
                    args=["type", "bar"],
                    label="Bar Chart",
                    method="restyle"
                )
            ]),
            direction="down",
        ),
    ],
)

plot.show()
```



2. Creating Sliders and Selectors

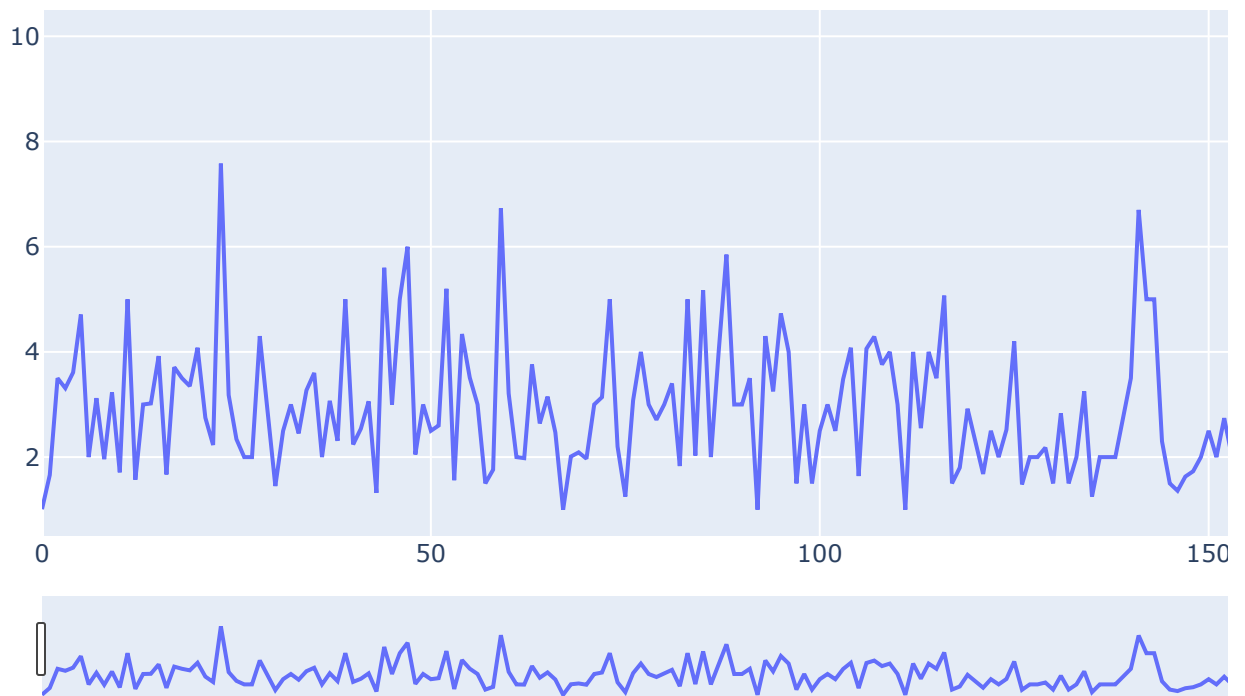
- In plotly, the range slider is a custom range-type input control. It allows selecting a value or a range of values between a specified minimum and maximum range. And the range selector is a tool for selecting ranges to display within the chart. It provides buttons to select pre-configured ranges in the chart. It also provides input boxes where the minimum and maximum dates can be manually input.

```
import plotly.graph_objects as px
import pandas as pd
```

```
# reading the database
data = pd.read_csv("tips.csv")
```

```
plot = px.Figure(data=[px.Scatter(
    y=data['tip'],
    mode='lines',)
])
```

```
plot.update_layout(  
    xaxis=dict(  
        rangeselector=dict(  
            buttons=list([  
                dict(count=1,  
                    step="day",  
                    stepmode="backward"),  
            ]) )  
    ),  
    rangeslider=dict(  
        visible=True  
    ),  
)  
)  
  
plot.show()
```



✓ Types of Data Visualization

✓ Basic Charts

✓ 1. Bar Chart

- A bar chart is a common type of data visualization used to represent categorical data with rectangular bars. The length or height of each bar is proportional to the value it represents, making it easy to compare different categories at a glance.
- Key Elements of a Bar Chart:
 - Categories (x-axis): The different groups or categories being compared.
 - Values (y-axis): The numerical values corresponding to each category.
 - Bars: Rectangles representing the values for each category. Bars can be plotted either vertically (vertical bar chart) or horizontally (horizontal bar chart).

```
# Creating a bar chart using matplotlib
```

```
import matplotlib.pyplot as plt
```

```
# Data
```

```
categories = ['A', 'B', 'C', 'D', 'E']
```

```
values = [4, 7, 1, 8, 5]
```

```
# Creating the bar chart
```

```
plt.bar(categories, values)
```

```
# Adding title and labels
```

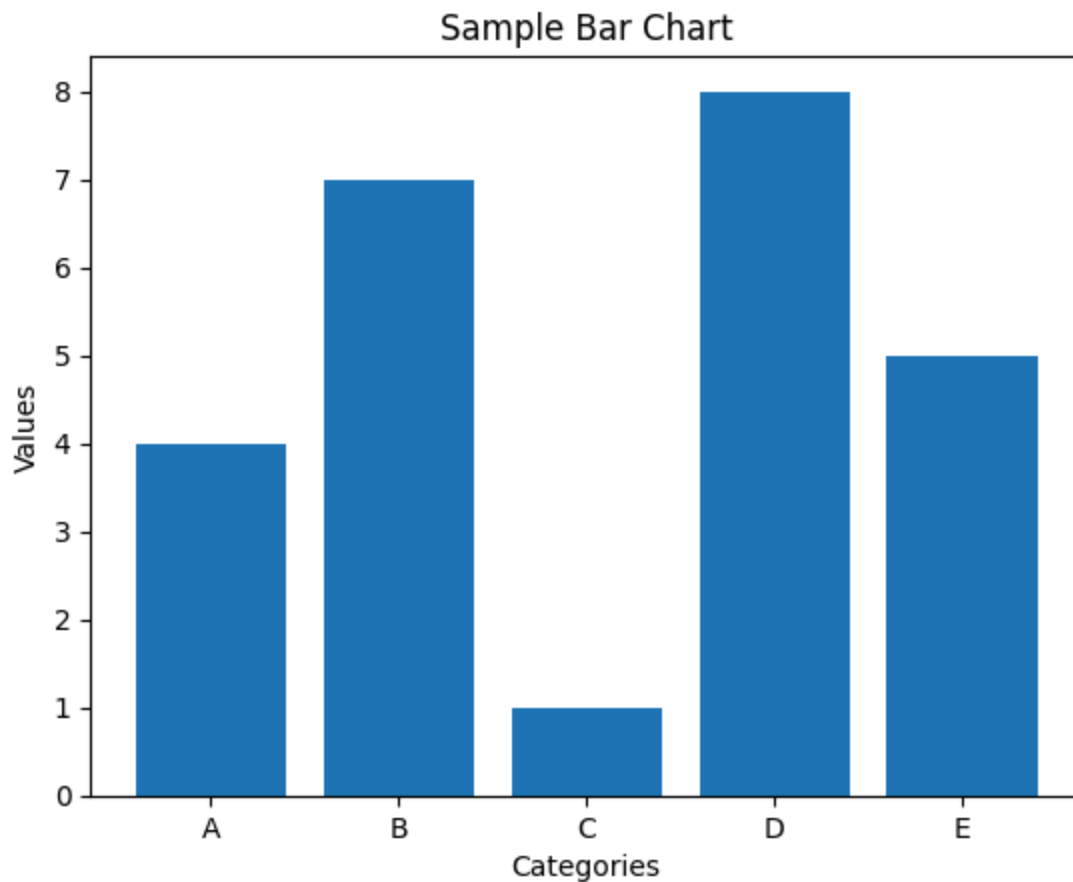
```
plt.title('Sample Bar Chart')
```

```
plt.xlabel('Categories')
```

```
plt.ylabel('Values')
```

```
# Displaying the chart
```

```
plt.show()
```



✓ 2. Histogram

- A histogram is a type of data visualization that displays the distribution of a dataset by grouping data into bins (or intervals) and counting the number of observations that fall into each bin. Unlike a bar chart, which is used for categorical data, a histogram is used for continuous data.
- Key Elements of a Histogram:
 - Bins (x-axis): Intervals into which the entire range of the data is divided. Each bin represents a range of values.
 - Frequency (y-axis): The number of observations that fall into each bin.

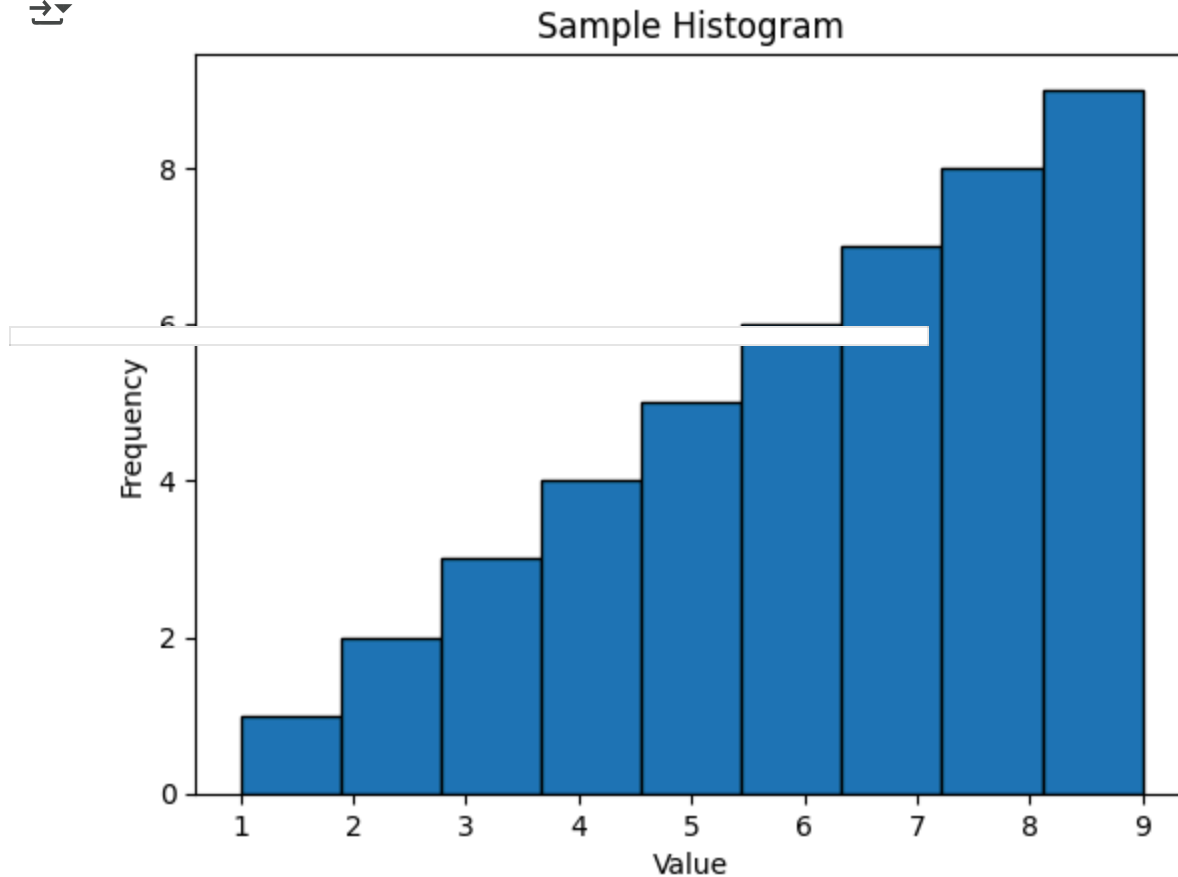
```
# Creating a Histogram in Python using Matplotlib
import matplotlib.pyplot as plt
```

```
# Sample data
data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 7, 7, 7, 7, 7, 7, 7, 7]
```

```
# Creating the histogram
plt.hist(data, bins=9, edgecolor='black')
```

```
# Adding title and labels
plt.title('Sample Histogram')
plt.xlabel('Value')
plt.ylabel('Frequency')

# Displaying the histogram
plt.show()
```



✓ 3. Line Chart

- A line chart is a type of data visualization that displays information as a series of data points called 'markers' connected by straight line segments. It is often used to visualize data points over a period of time or any continuous sequence, making it ideal for showing trends, patterns, or changes.
- Key Elements of a Line Chart:
 - X-axis: Represents the independent variable, often time or another continuous variable.
 - Y-axis: Represents the dependent variable, or the value corresponding to each point on the x-axis.
 - Line: The main visual element connecting data points, showing the trend or pattern over time.

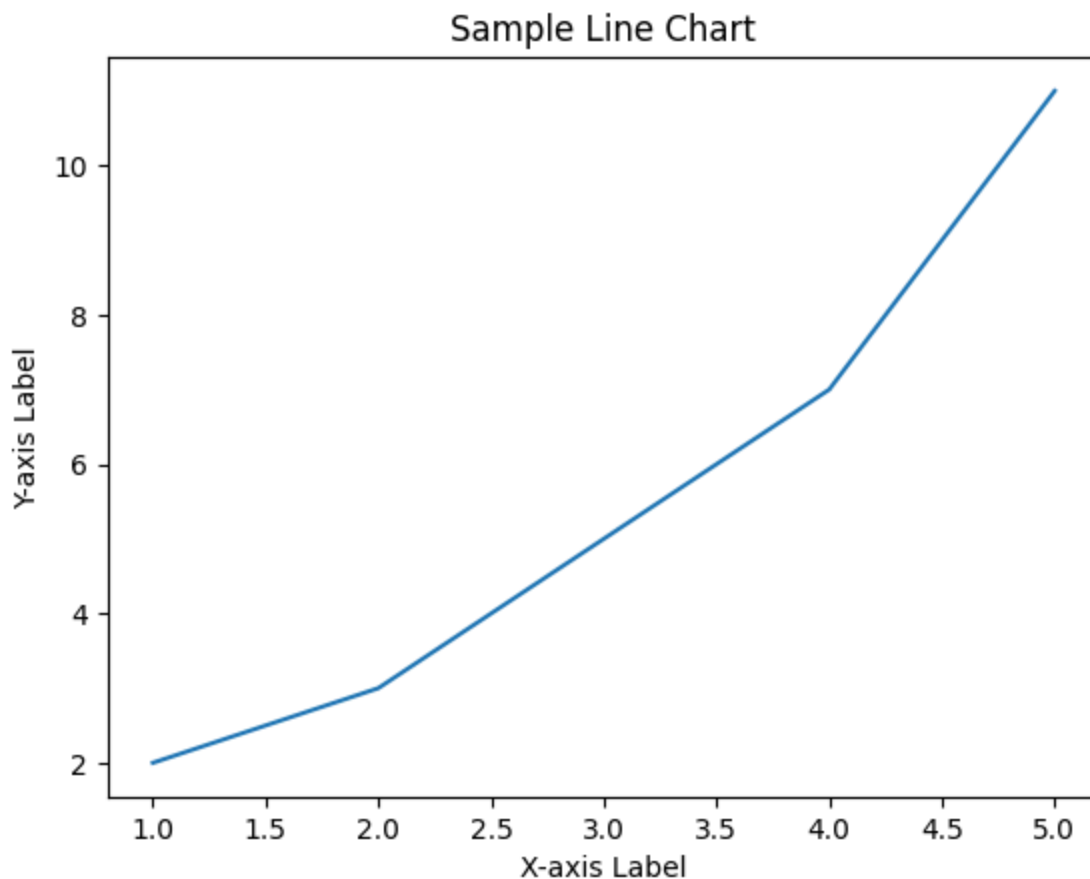
```
# Creating a Line Chart in Python using Matplotlib
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [2, 3, 5, 7, 11]

# Creating the line chart
plt.plot(x, y)

# Adding title and labels
plt.title('Sample Line Chart')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')

# Displaying the chart
plt.show()
```



✓ 4. Pie Chart

- A pie chart is a circular statistical graphic that is divided into slices to illustrate numerical proportions. Each slice of the pie represents a category's contribution to the whole, with the size of each slice corresponding to its proportion relative to the total sum.

- Key Elements of a Pie Chart:
 - Sectors (Slices): Each slice represents a category or group within the dataset.
 - Proportions: The size of each slice is proportional to the quantity it represents, typically as a percentage of the total.

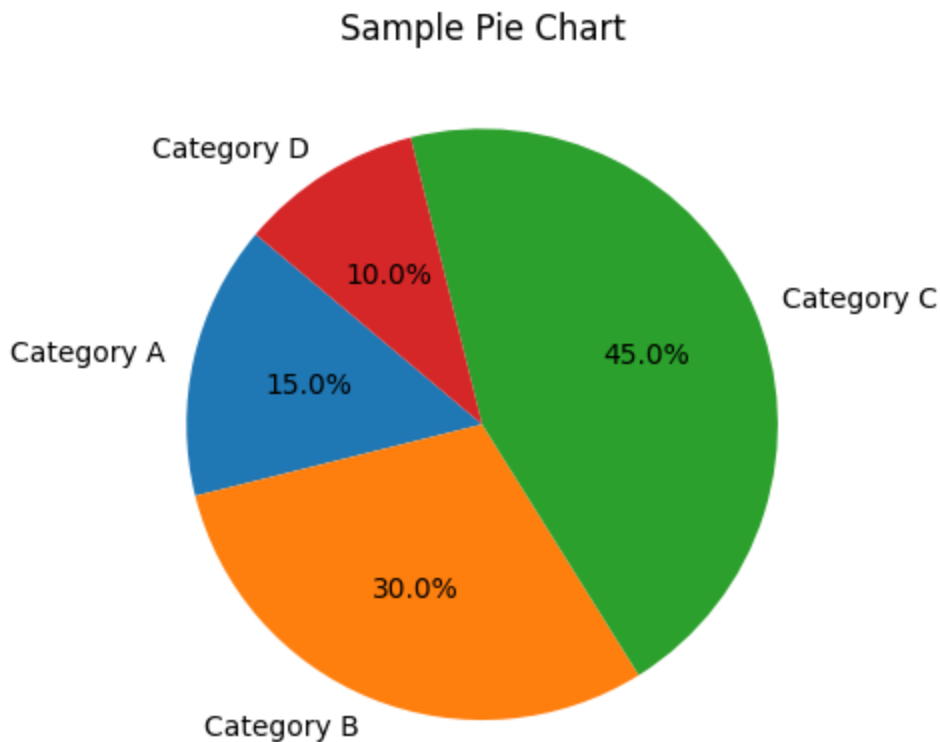
```
# Creating a Pie Chart in Python using Matplotlib
import matplotlib.pyplot as plt

# Data
labels = ['Category A', 'Category B', 'Category C', 'Category D']
sizes = [15, 30, 45, 10]

# Creating the pie chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)

# Adding title
plt.title('Sample Pie Chart')

# Displaying the chart
plt.show()
```



✓ Intermediate Charts

✓ 1. Scatter Plot

- A scatter plot is a type of data visualization that displays values for two variables as a collection of points. Each point represents an observation in the dataset, with its position determined by the values of the two variables. Scatter plots are used to identify relationships, trends, and correlations between the variables.
- Key Elements of a Scatter Plot:
 - X-axis: Represents the independent variable.
 - Y-axis: Represents the dependent variable.
 - Points: Each point on the scatter plot corresponds to an observation in the dataset.

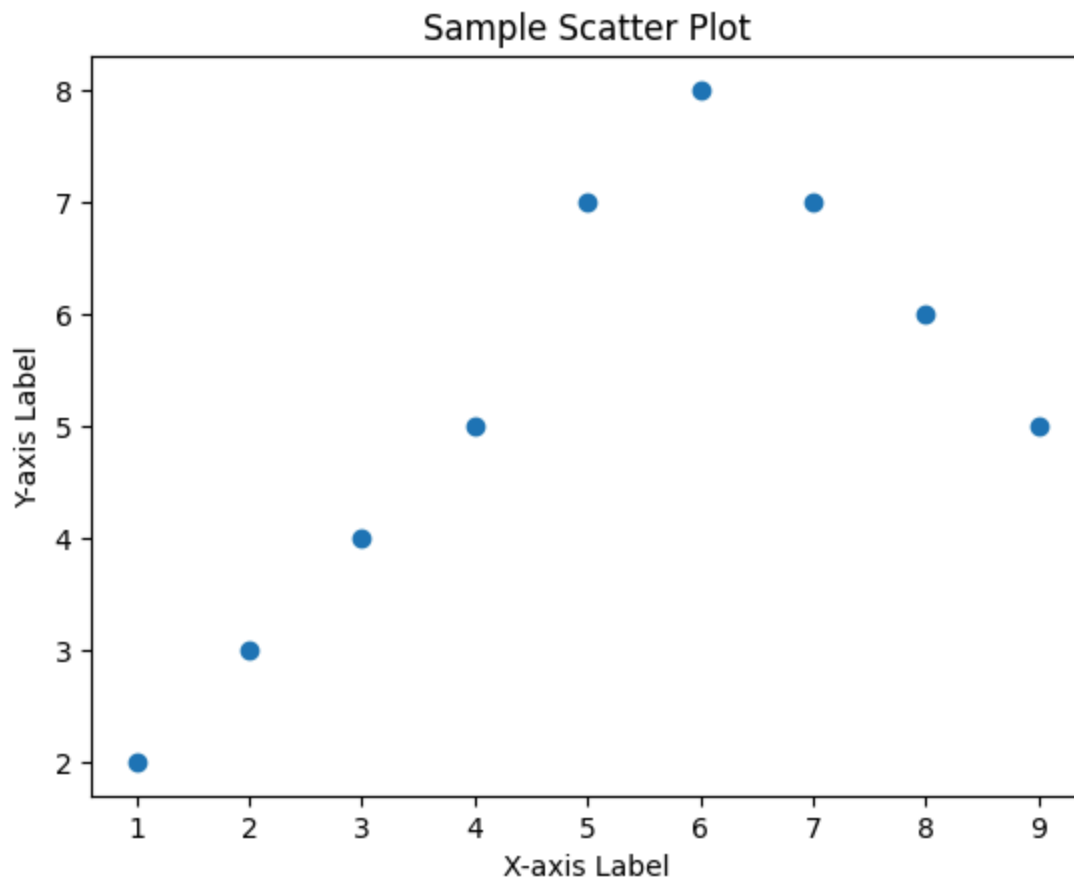
```
# Creating a Scatter Plot in Python using Matplotlib
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
y = [2, 3, 4, 5, 7, 8, 7, 6, 5]

# Creating the scatter plot
plt.scatter(x, y)

# Adding title and labels
plt.title('Sample Scatter Plot')
plt.xlabel('X-axis Label')
plt.ylabel('Y-axis Label')

# Displaying the plot
plt.show()
```



✓ 2. Box Plot

- A box plot (also known as a whisker plot) is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum. Box plots are useful for identifying outliers and understanding the spread and skewness of the data.
- Key Elements of a Box Plot:
 - Median (Q2): The middle value of the dataset, represented by a line inside the box.
 - First Quartile (Q1): The median of the lower half of the data, marking the 25th percentile.
 - Third Quartile (Q3): The median of the upper half of the data, marking the 75th percentile.
 - Interquartile Range (IQR): The range between the first and third quartiles ($IQR = Q3 - Q1$), representing the middle 50% of the data.
 - Whiskers: Lines extending from the box to the minimum and maximum values within 1.5 times the IQR from Q1 and Q3, respectively.
 - Outliers: Data points outside 1.5 times the IQR from the quartiles, often plotted as individual points.

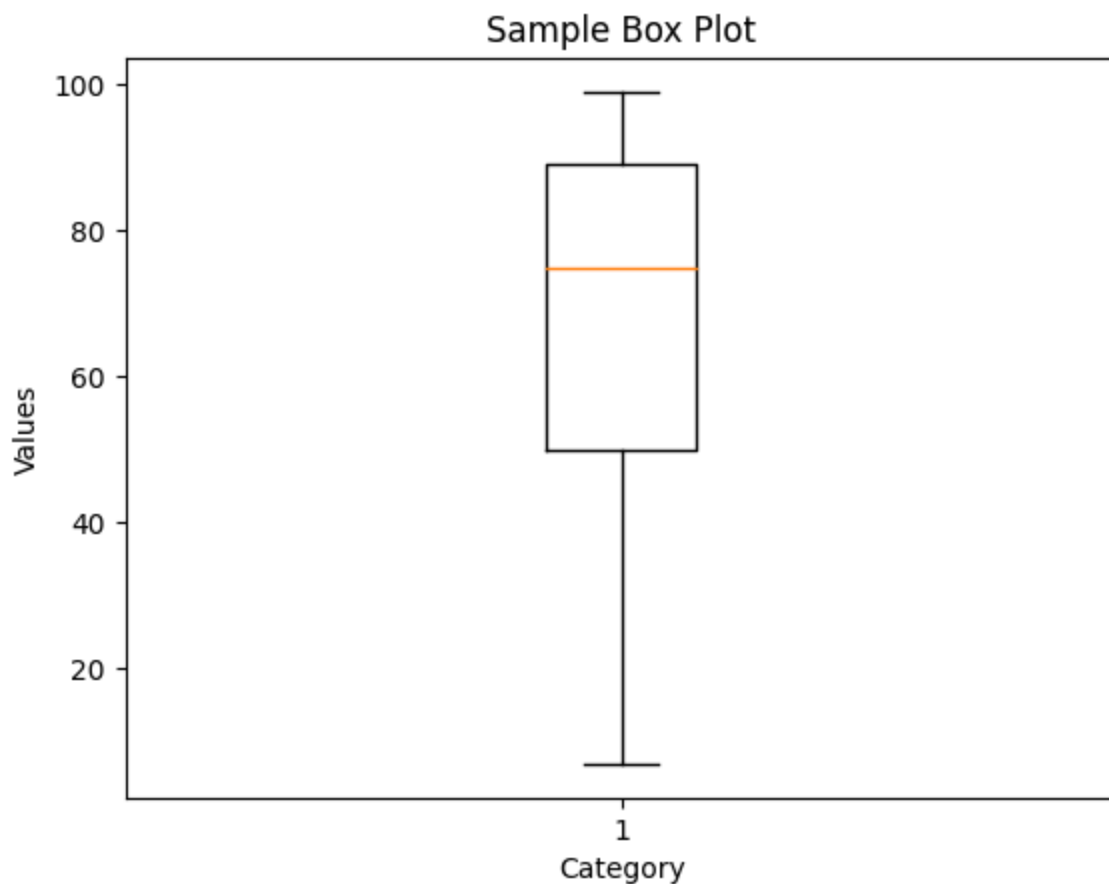
```
# Creating a Box Plot in Python using Matplotlib
import matplotlib.pyplot as plt

# Sample data
data = [7, 15, 36, 39, 46, 49, 50, 56, 60, 61, 68, 69, 75, 78, 79, 84, 85, 88, 89, 90, 91, 95]

# Creating the box plot
plt.boxplot(data)

# Adding title and labels
plt.title('Sample Box Plot')
plt.xlabel('Category')
plt.ylabel('Values')

# Displaying the plot
plt.show()
```



✓ 3. Heatmap

- A heatmap is a graphical representation of data where individual values are represented as colors. Heatmaps are particularly useful for visualizing the magnitude of values across a matrix, identifying patterns, correlations, and outliers in data.

- Key Elements of a Heatmap:
 - Color Scale: Represents the range of values in the data. Different colors or shades indicate different magnitudes.
 - Axis Labels: Typically, heatmaps have labels along both axes to indicate the categories or variables being compared.
 - Data Matrix: The actual data is presented in a grid format, where each cell corresponds to a value and its color indicates its magnitude.

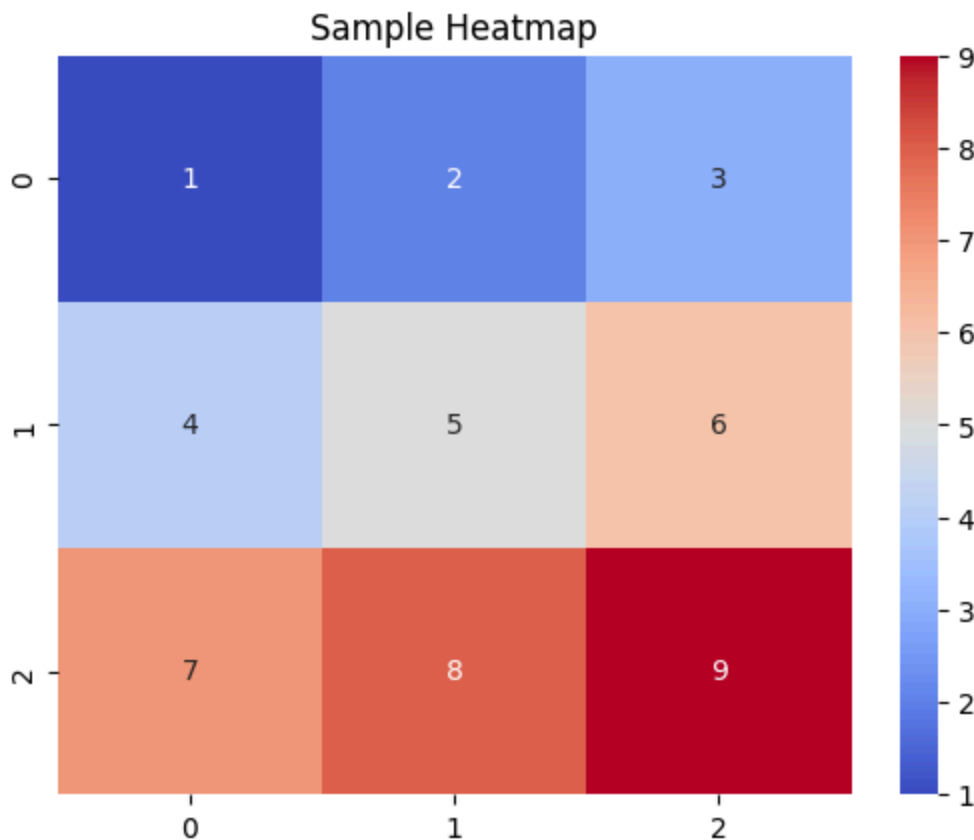
```
# Creating a Heatmap in Python using Seaborn
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data
data = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]

# Creating the heatmap
sns.heatmap(data, annot=True, cmap='coolwarm')

# Adding title
plt.title('Sample Heatmap')

# Displaying the plot
plt.show()
```




✓ 4. Violin Plot

- A violin plot is a data visualization technique that combines aspects of a box plot and a kernel density plot. It provides a more comprehensive view of the distribution of the data by showing the density of the data at different values. Violin plots are particularly useful for comparing multiple distributions.
- Key Components of a Violin Plot:
 - Density Plot (The Violin Shape): The violin plot displays the kernel density estimation of the data on each side. The width of the plot at a given value indicates the density or the frequency of data points around that value. The shape of the violin plot reflects the distribution of the data. For example, if the plot is wider at a certain value, it indicates a higher density of data points around that value.
 - Central Dot/Line (Median or Mean): Some violin plots include a central line or dot to represent the median or mean of the data.
 - Inner Box Plot: The violin plot can include an embedded box plot within the "violin." The box plot represents the interquartile range (IQR), median, and potential outliers.
 - Outliers: Outliers may be displayed as individual points outside the range of the "violin."

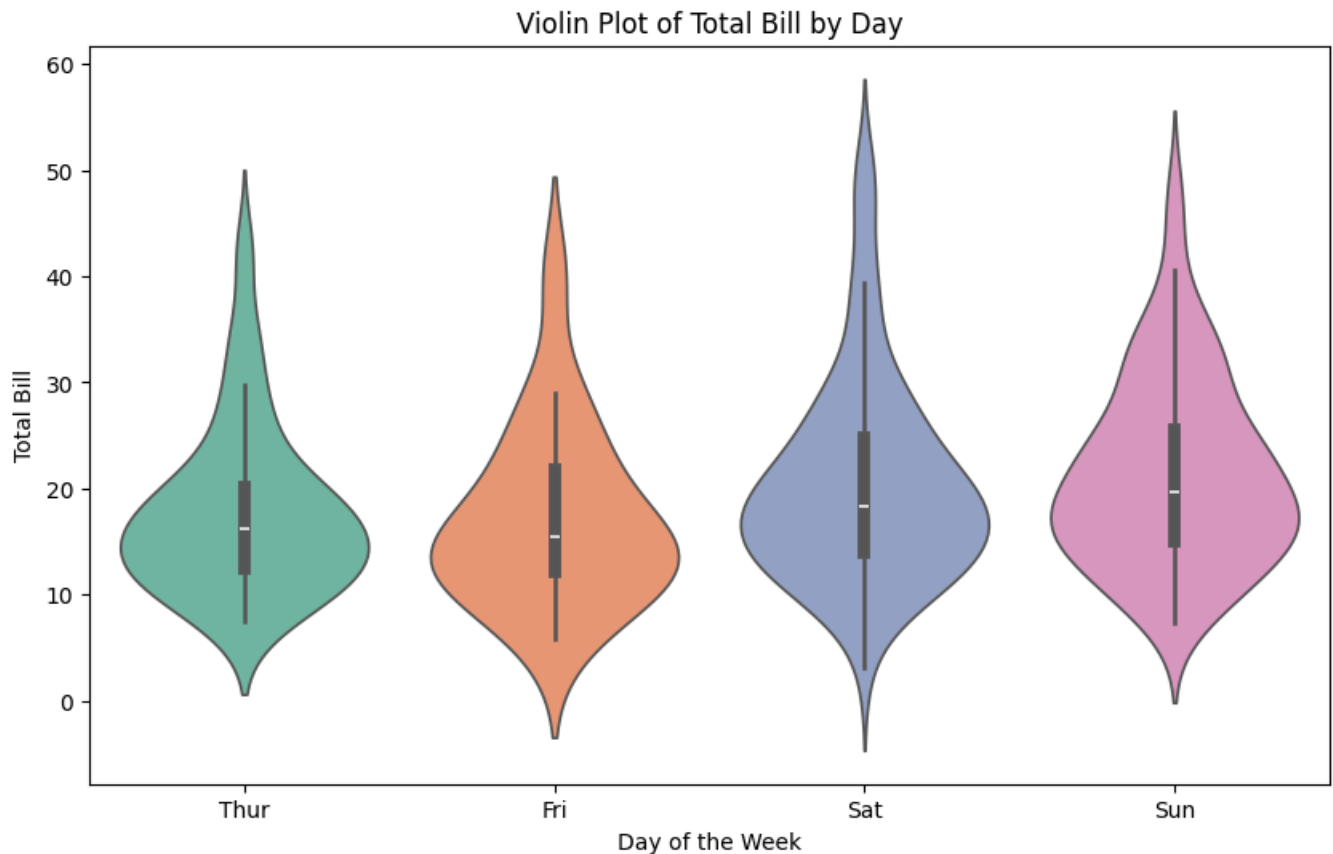
```
# Creating a Violin Plot in Python using Seaborn
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Example dataset
tips = sns.load_dataset('tips')

# Creating a violin plot
plt.figure(figsize=(10, 6))
sns.violinplot(x='day', y='total_bill', data=tips, inner='box', palette='Set2')
plt.title('Violin Plot of Total Bill by Day')
plt.xlabel('Day of the Week')
plt.ylabel('Total Bill')
plt.show()
```

 <ipython-input-26-82061ad069fe>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.



✓ Advanced Data Visualization Techniques

✓ Pair Plots

- Pair plots (also known as scatterplot matrices) are a type of data visualization used to visualize relationships between multiple variables in a dataset. They create a grid of scatter plots for each pair of variables, allowing for the examination of pairwise relationships and distributions within the data.
- Key Elements of a Pair Plot:

- Diagonal Plots: The diagonal typically shows the distribution of each individual variable, often using histograms or density plots.
- Off-Diagonal Plots: Scatter plots of each variable pair, showing the relationship between the two variables.
- Axes: Both x and y axes are labeled to indicate the variables being compared.

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd

# Load sample data
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

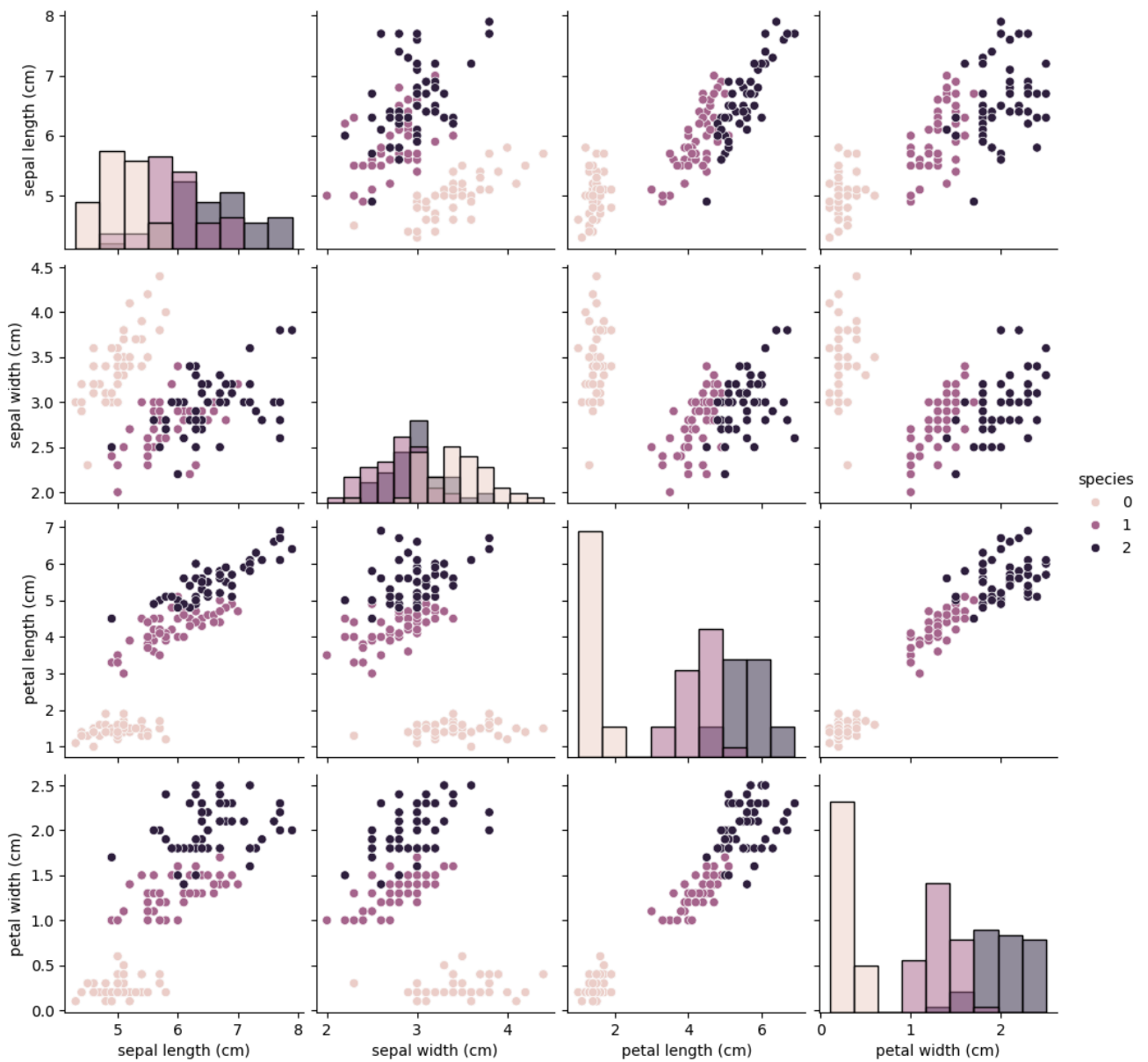
# Creating the pair plot
sns.pairplot(df, hue='species', diag_kind='hist')

# Adding title
plt.suptitle('Pair Plot of Iris Dataset', y=1.02)

# Displaying the plot
plt.show()
```




Pair Plot of Iris Dataset



▼ Facet Grids

- Facet grids are a powerful visualization tool used to create multiple subplots (or "facets") that display different subsets of data or variations of a plot within the same figure. They allow you to compare and analyze multiple variables or categories in a systematic way.
- Key Elements of a Facet Grid:
 - Facets: Multiple subplots arranged in a grid format, where each subplot represents a subset of the data based on categorical variables.
 - Axes: Each facet has its own set of axes, and plots are organized by rows and columns.
 - Mapping: Data is mapped to facets based on one or more categorical variables, allowing comparison across different levels.

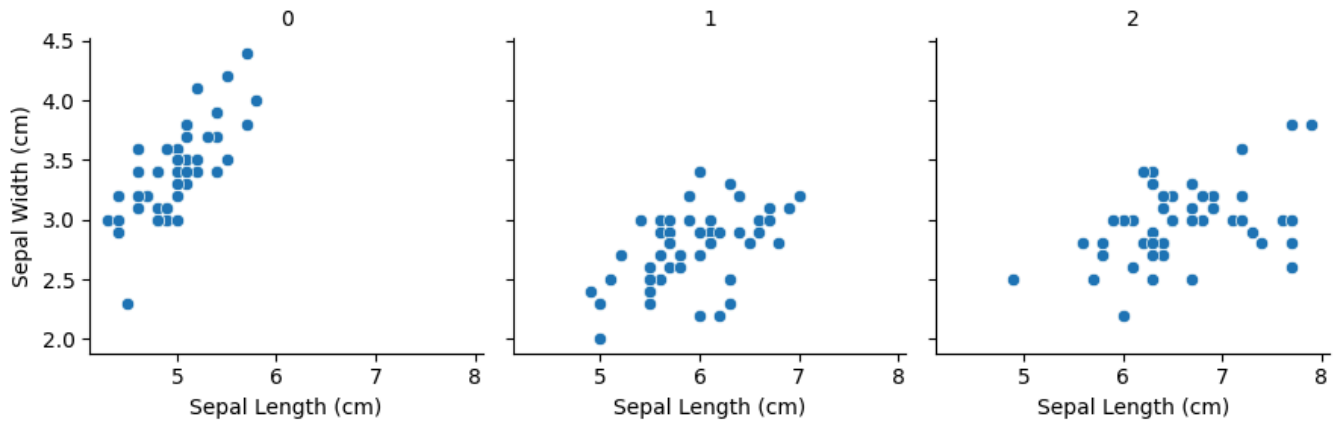
```
# Creating a Facet Grid in Python using Seaborn
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
import pandas as pd

# Load sample data
iris = load_iris()
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['species'] = iris.target

# Creating the facet grid
g = sns.FacetGrid(df, col='species', margin_titles=True)
g.map_dataframe(sns.scatterplot, x='sepal length (cm)', y='sepal width (cm)')

# Adding titles and labels
g.set_axis_labels('Sepal Length (cm)', 'Sepal Width (cm)')
g.set_titles(col_template="{col_name}")

# Displaying the plot
plt.show()
```



✓ 3D Plotting

- 3D plotting allows for the visualization of data in three dimensions, adding depth to the representation and enabling a more comprehensive understanding of the relationships between variables. In Python, several libraries facilitate 3D plotting, with Matplotlib and Plotly being among the most commonly used.
- Libraries for 3D Plotting:
 - Matplotlib: Provides basic 3D plotting capabilities through its `mpl_toolkits.mplot3d` module.
 - Plotly: Offers interactive 3D plots with extensive customization options.

```
import plotly.graph_objects as go
import numpy as np

# Generate sample data
np.random.seed(0)
x = np.random.rand(100)
y = np.random.rand(100)
z = np.random.rand(100)

# Create a 3D scatter plot
fig = go.Figure(data=[go.Scatter3d(
    x=x, y=y, z=z,
    mode='markers',
    marker=dict(size=5, color=z, colorscale='Viridis', opacity=0.8
))]

# Adding titles and labels
fig.update_layout(
```

```
title='3D Scatter Plot',
scene=dict(
    xaxis_title='X axis',
    yaxis_title='Y axis',
    zaxis_title='Z axis'
)
)

# Display the plot
```