Name: Sarthak Mishra

TASK 1

Python Visualization Libraries Documentation

Table of Contents

1) Introduction

Data visualization is a crucial aspect of data analysis, enabling the transformation of complex data sets into understandable and actionable insights. Python offers several libraries for creating visualizations, each with its unique features and capabilities. This guide focuses on **Matplotlib** and **Seaborn**, providing an overview of their functionalities, the types of graphs they support, practical examples, and a comparison to help you choose the right tool for your projects.

2) Library Overview

Matplotlib

Matplotlib is one of the most popular and versatile plotting libraries in Python. It provides a comprehensive API for creating static, animated, and interactive visualizations. Matplotlib is highly customizable, making it suitable for creating complex and detailed plots.

Key Features:

- Extensive range of plot types.
- Highly customizable styles and layouts.
- Supports integration with various GUI toolkits.
- Foundation for other libraries like Seaborn.

Typical Use Cases:

- Creating publication-quality figures.
- Customizing plots with fine-grained control.
- Building complex multi-plot layouts.

Seaborn

Seaborn is built on top of Matplotlib and provides a high-level interface for creating attractive and informative statistical graphics. It simplifies the process of creating complex visualizations and is particularly well-suited for statistical analysis.

Key Features:

- Simplified syntax for complex plots.
- Enhanced default aesthetics.
- Built-in support for statistical functions.
- Integration with pandas data structures.

Typical Use Cases:

- Visualizing statistical relationships.
- Creating aesthetically pleasing plots with minimal code.
- Exploring data distributions and relationships.

3) Graph Types

Matplotlib Graphs

1) Line Plot

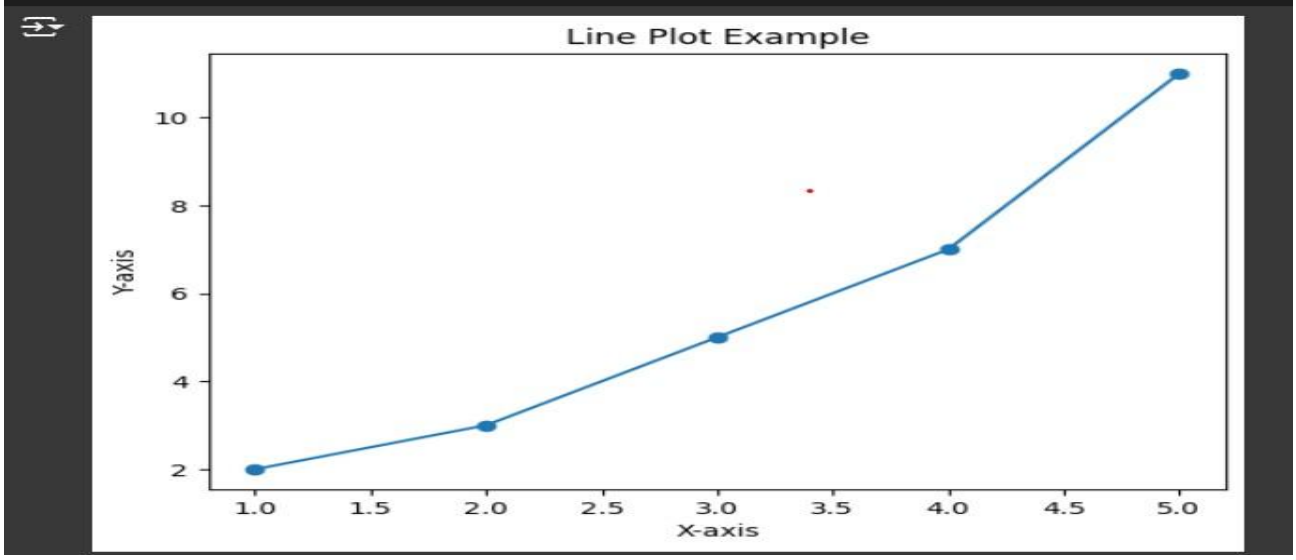
Description: Displays information as a series of data points connected by straight line segments. Useful for showing trends over time.

Use Case: Tracking changes over periods, such as stock prices or temperature variations.

Example:

Code & Output

```
import matplotlib.pyplot as plt

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

# Create line plot
plt.plot(x, y, marker='o')
plt.title('Line Plot Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



2)Scatter Plot

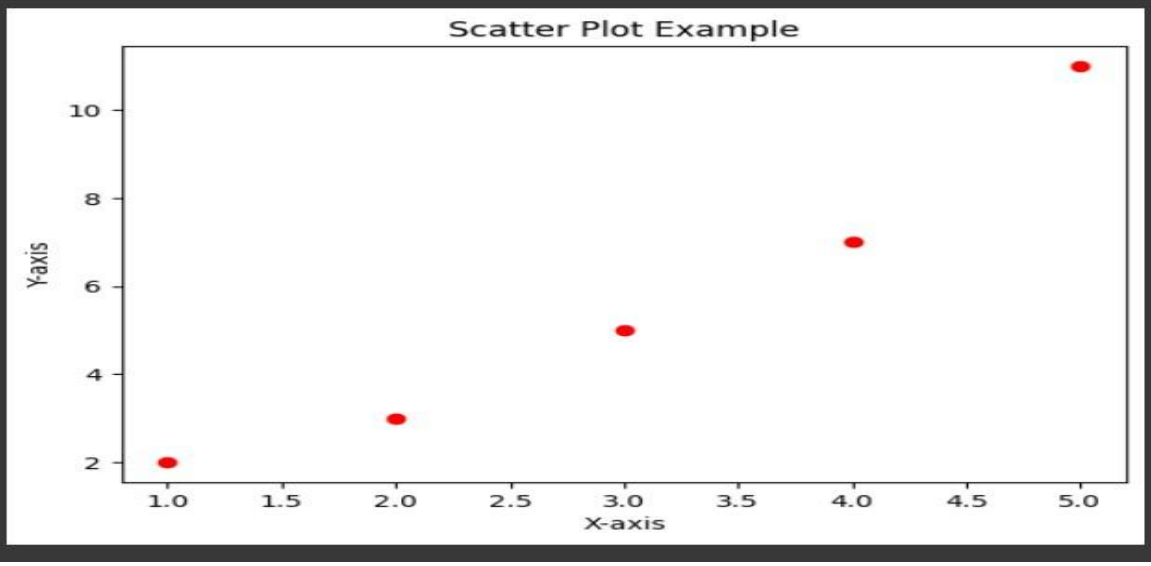
Description: Uses Cartesian coordinates to display values for two variables for a set of data. Ideal for observing relationships between variables.

Use Case: Identifying correlations, clusters, and outliers in data.

Example:

Code & Output :

```
import matplotlib.pyplot as plt

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

# Create scatter plot
plt.scatter(x, y, color='red')
plt.title('Scatter Plot Example')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```



3) Bar Chart

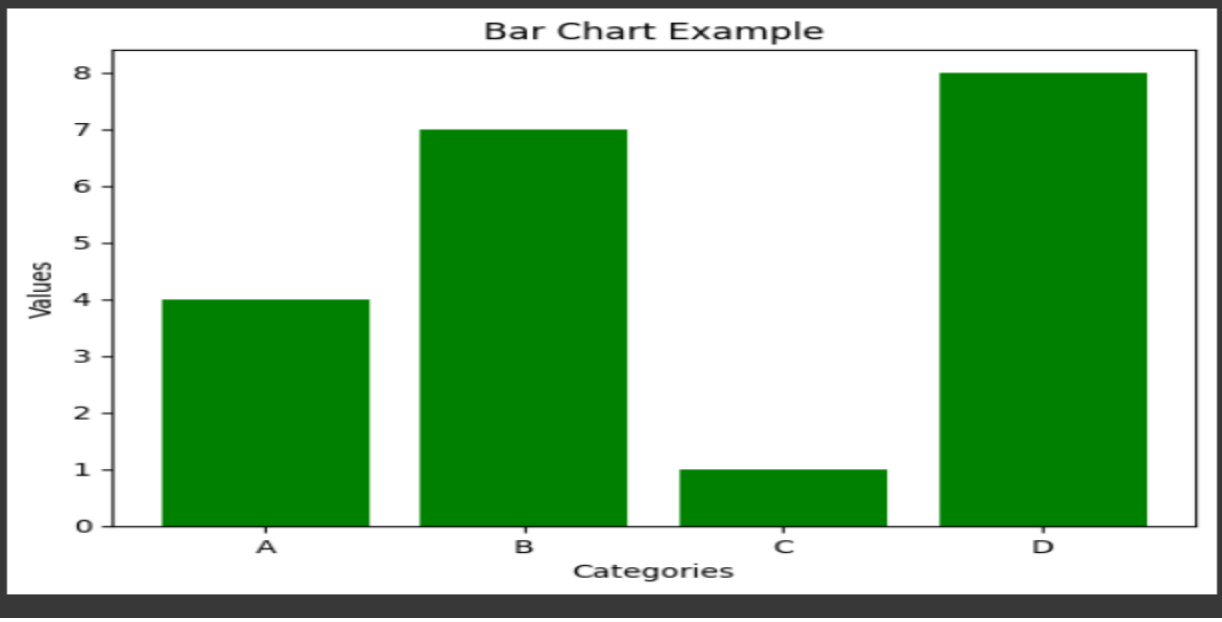
Description: Represents categorical data with rectangular bars. The length of each bar is proportional to the value it represents.

Use Case: Comparing different categories or groups.

Example:

Code & Output :

```
import matplotlib.pyplot as plt

# Sample data
categories = ['A', 'B', 'C', 'D']
values = [4, 7, 1, 8]

# Create bar chart
plt.bar(categories, values, color='green')
plt.title('Bar Chart Example')
plt.xlabel('Categories')
plt.ylabel('Values')
plt.show()
```



4) Histogram

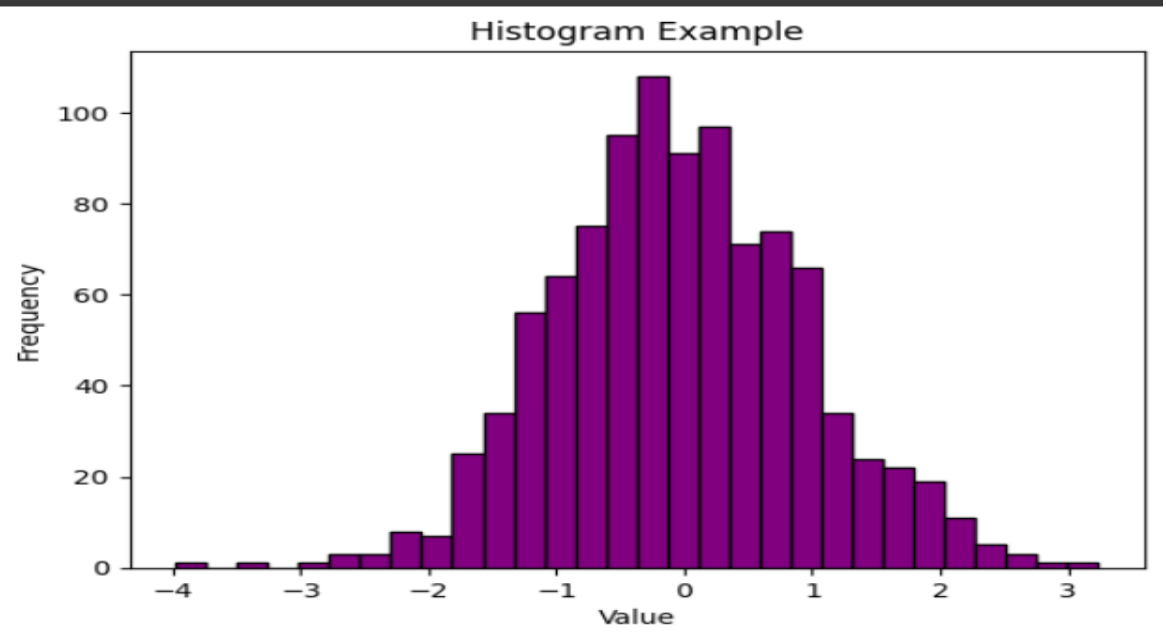
Description: Shows the distribution of a dataset by grouping data into bins and displaying the frequency of data points in each bin.

Use Case: Understanding the distribution and spread of data.

Example:

Code & Output :

```
import matplotlib.pyplot as plt
import numpy as np

# Sample data
data = np.random.randn(1000)

# Create histogram
plt.hist(data, bins=30, color='purple', edgecolor='black')
plt.title('Histogram Example')
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()
```



5) Pie Chart

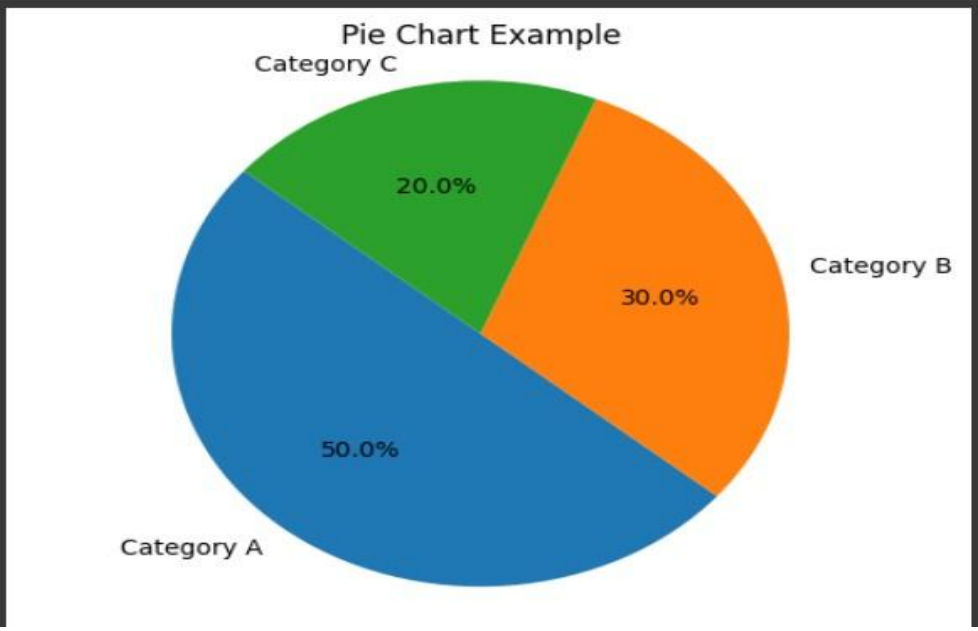
Description: Displays data as slices of a circle, where each slice represents a proportion of the whole.

Use Case: Showing percentage or proportional data.

Example:

Code & Output :

```
import matplotlib.pyplot as plt

# Sample data
labels = ['Category A', 'Category B', 'Category C']
sizes = [50, 30, 20]

# Create pie chart
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140)
plt.title('Pie Chart Example')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



Seaborn Graphs

1) Line Plot

Description: Similar to Matplotlib's line plot but with additional features for statistical analysis.

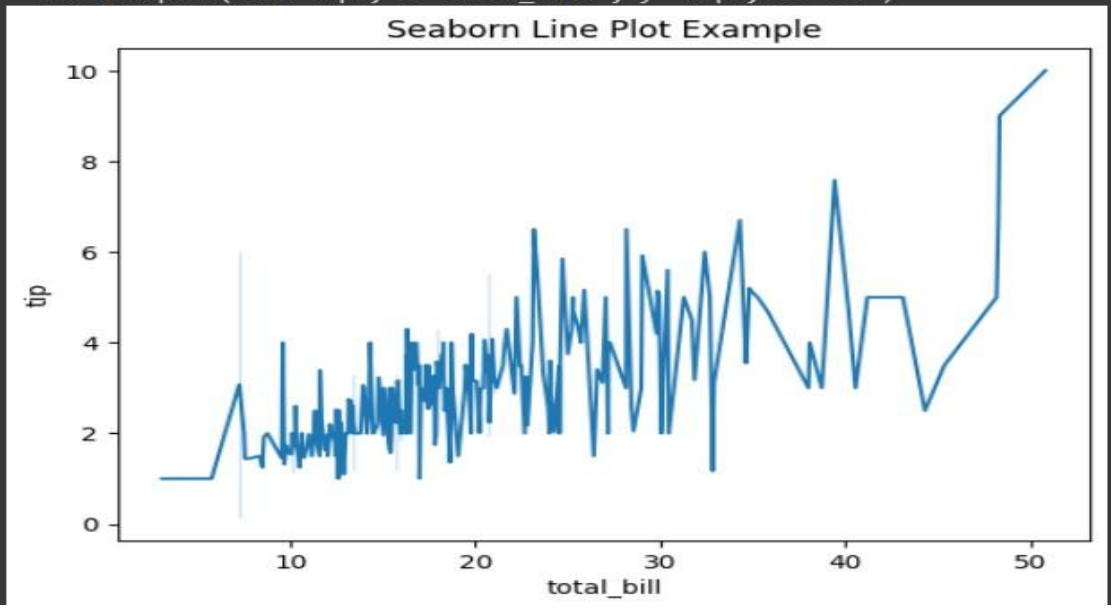
Use Case: Visualizing trends with confidence intervals.

Example:

Code & Output :

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

# Sample data
tips = sns.load_dataset('tips')

# Create line plot
sns.lineplot(data=tips, x='total_bill', y='tip', ci='sd')
plt.title('Seaborn Line Plot Example')
plt.show()
```

```
<ipython-input-6-983460d2c165>:8: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect

  sns.lineplot(data=tips, x='total_bill', y='tip', ci='sd')
```



2) Scatter Plot

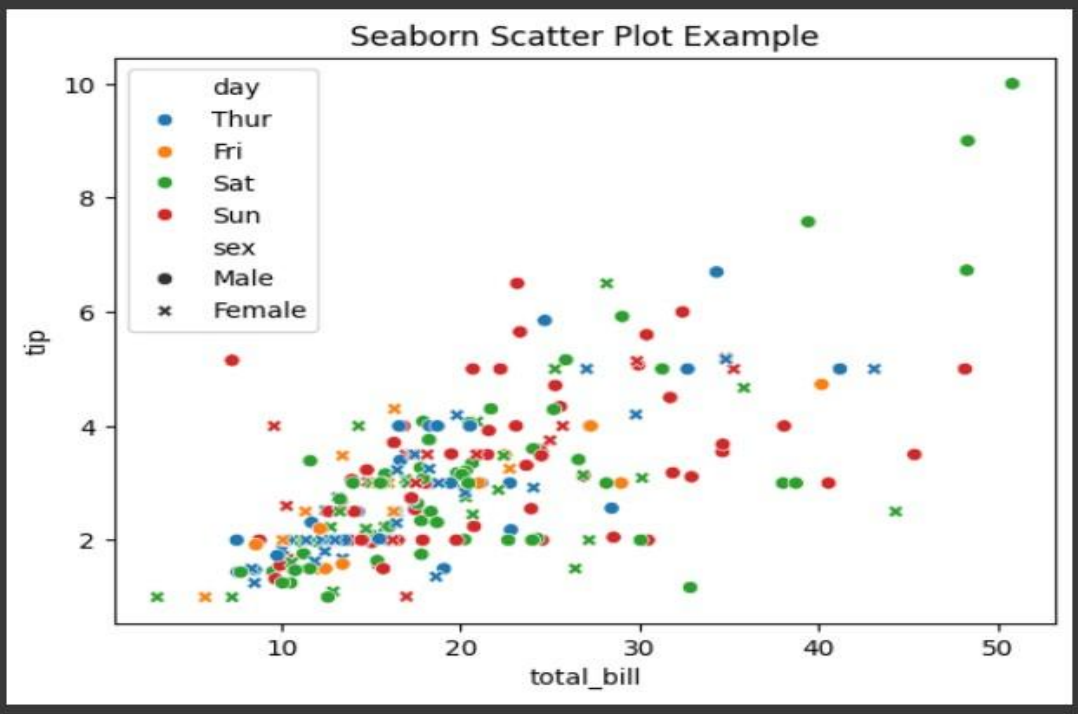
Description: Enhanced scatter plots with support for hue, size, and style.

Use Case: Exploring relationships with additional categorical variables.

Example:

Code & Output :

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

# Sample data
tips = sns.load_dataset('tips')

# Create scatter plot
sns.scatterplot(data=tips, x='total_bill', y='tip', hue='day', style='sex')
plt.title('Seaborn Scatter Plot Example')
plt.show()
```



3) Bar Plot

Description: Simplified bar charts with automatic aggregation and error bars.

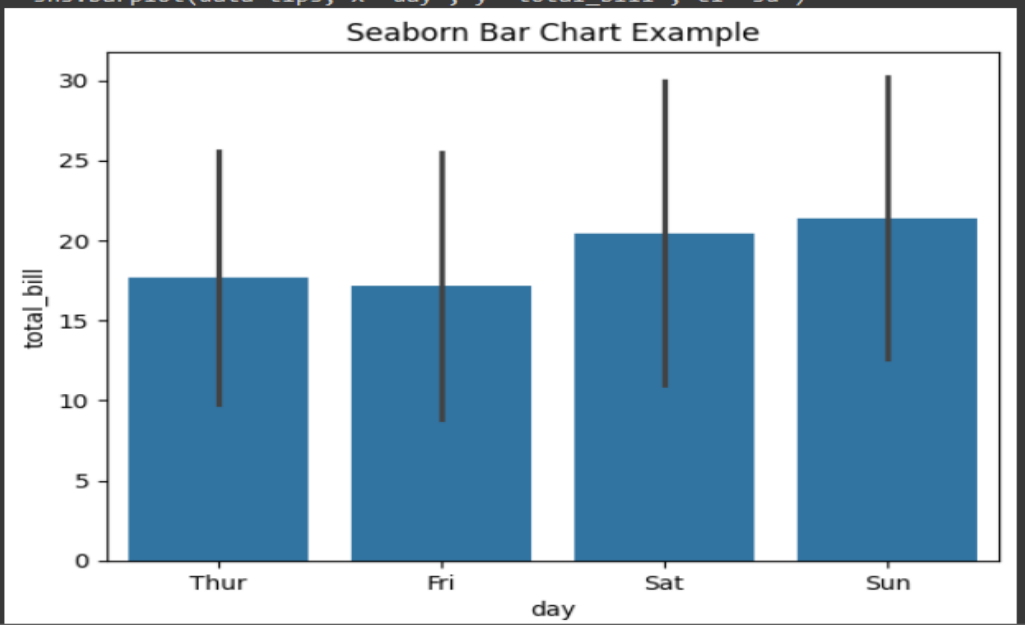
Use Case: Comparing categorical data with built-in statistical summaries.

Example:

Code & Output :

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

# Sample data
tips = sns.load_dataset('tips')

# Create bar chart
sns.barplot(data=tips, x='day', y='total_bill', ci='sd')
plt.title('Seaborn Bar Chart Example')
plt.show()
```

```
<ipython-input-8-a235af8358a1>:8: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar='sd'` for the same effect.

  sns.barplot(data=tips, x='day', y='total_bill', ci='sd')
```



4) Histogram

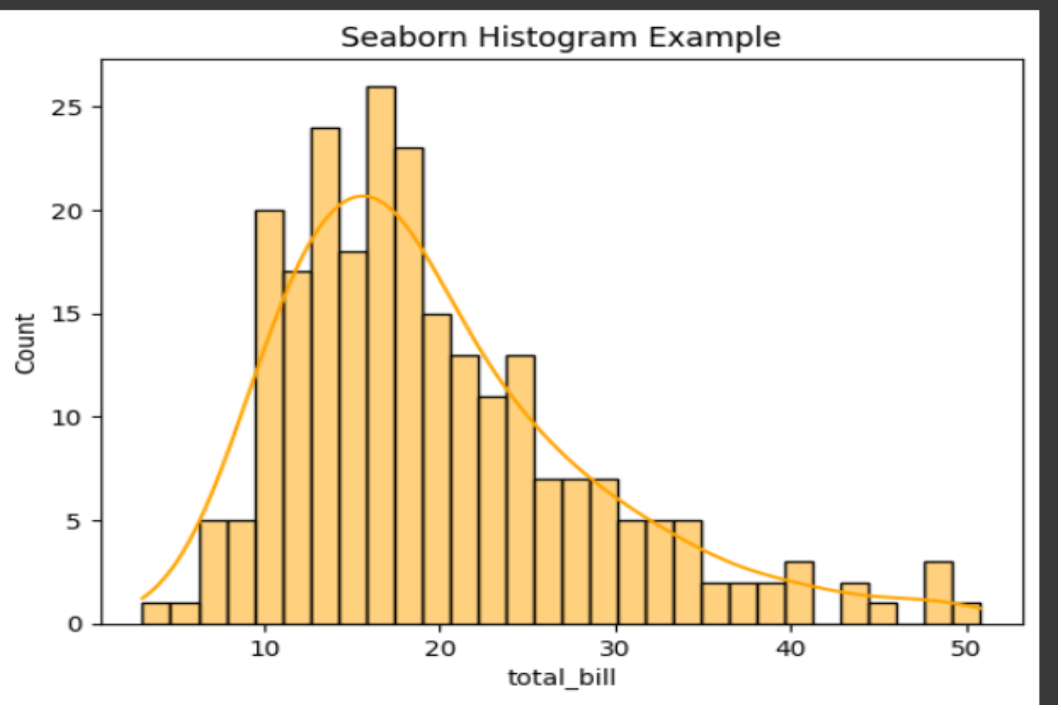
Description: Similar to Matplotlib's histogram but with better aesthetics and automatic handling of bins.

Use Case: Visualizing the distribution of data with kernel density estimates.

Example:

Code & Output :

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

# Sample data
tips = sns.load_dataset('tips')

# Create histogram
sns.histplot(data=tips, x='total_bill', bins=30, kde=True, color='orange')
plt.title('Seaborn Histogram Example')
plt.show()
```



5) Box Plot

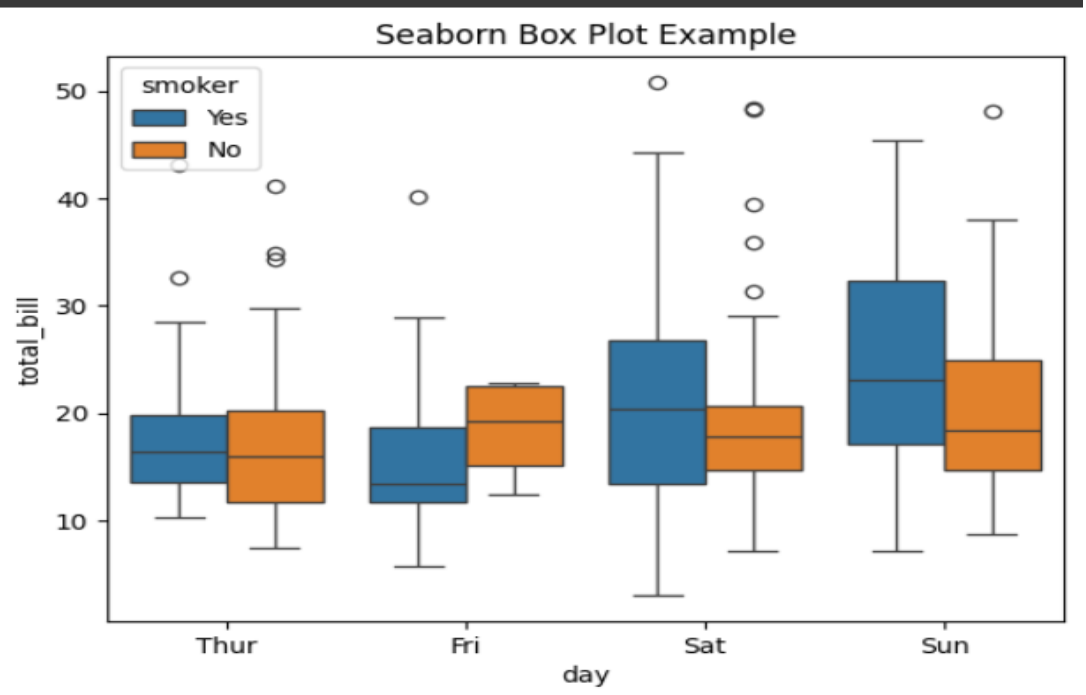
Description: Displays the distribution of data based on a five-number summary: minimum, first quartile, median, third quartile, and maximum.

Use Case: Identifying outliers and understanding the spread of data.

Example:

Code & Output :

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

# Sample data
tips = sns.load_dataset('tips')

# Create box plot
sns.boxplot(data=tips, x='day', y='total_bill', hue='smoker')
plt.title('Seaborn Box Plot Example')
plt.show()
```



4) Graph Comparison

Feature	Matplotlib	Seaborn
Ease of Use	Steeper learning curve due to extensive customization.	Higher-level interface, easier for quick plots.
Customization	Highly customizable with fine-grained control.	Limited customization compared to Matplotlib.
Aesthetics	Basic default styles, customizable.	Enhanced default aesthetics out-of-the-box.
Interactivity	Supports interactive plots with additional libraries.	Primarily for static plots, some interactive features.
Statistical Support	Basic support; requires manual implementation.	Built-in statistical functions and visualizations.
Performance	Efficient for large datasets with proper optimization.	May be slower with very large datasets.
Integration	Foundation for many other visualization libraries.	Integrates seamlessly with pandas and Matplotlib.
Typical Use Cases	Detailed, publication-quality figures.	Quick exploratory data analysis and statistical plots.

Strengths:

- **Matplotlib:**
  - Unparalleled customization capabilities.
  - Suitable for creating complex and highly tailored visualizations.
  - Extensive community support and documentation.
- **Seaborn:**
  - Simplifies the creation of aesthetically pleasing and informative statistical plots.
  - Reduces the amount of code needed for common visualization tasks.
  - Excellent for exploratory data analysis.

Weaknesses:

- **Matplotlib:**
  - Can be verbose and complex for simple plots.
  - Requires more effort to achieve modern aesthetics.
- **Seaborn:**
  - Less flexible for highly customized visualizations.
  - Primarily designed for statistical data, which may limit its use in other contexts.

5) Resources

Matplotlib: Matplotlib Quick Start Guide

Seaborn: Seaborn Tutorial Introduction

Plotly: Plotly Python Distplot

Bokeh: Bokeh User Guide - Basic

Pandas: Pandas User Guide

Conclusion

Both Matplotlib and Seaborn are powerful tools for data visualization in Python. Matplotlib offers extensive customization and is ideal for creating detailed and complex plots, while Seaborn provides a higher-level interface with enhanced aesthetics, making it perfect for quick and informative statistical visualizations. Understanding the strengths and limitations of each library will help you choose the right tool for your specific data visualization needs.