ShadowFox Data Science Internship:

Name: Roopesh

Email: roopeshpoojary757@gmail.com

Beginner Guide - Python Visualization Libraries: Matplotlib & Seaborn

MATPLOTLIB

Matplotlib is a popular Python data visualization library used to create a wide variety of static, animated, and interactive graphs. It is the foundation of most Python visualizations and is highly customizable.

Uses of Matplotlib:

- *Used to create different types of graphs like line, bar, scatter, and histogram.*
- *Helps to visualize data clearly and understand patterns or trends.*
- *Useful for reports, presentations, and scientific research.*

SEABORN

Seaborn is a Python data visualization library built on top of Matplotlib.It is used to create beautiful, attractive, and statistical graphs with very simple code. Seaborn works extremely well with Pandas DataFrames, making it ideal for data analysis.

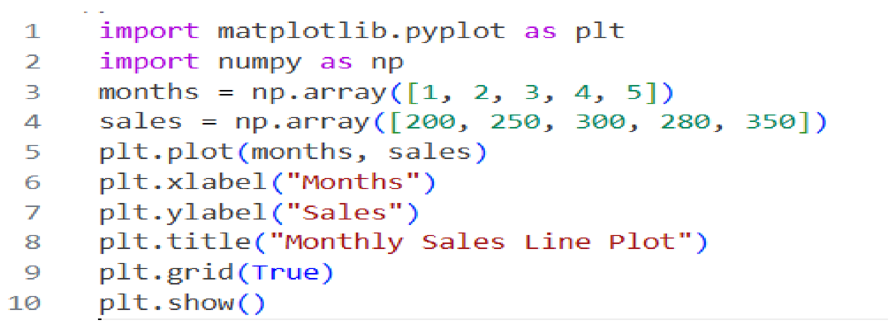
Uses of Seaborn:

- Used to create high-quality and stylish graphs automatically.
- It is specially designed for statistical visualization.
- It makes difficult plots easy to create (heatmaps, violin plots, pairplots).
- It works smoothly with datasets in rows and columns.

Types of graphs in Matplotlib:

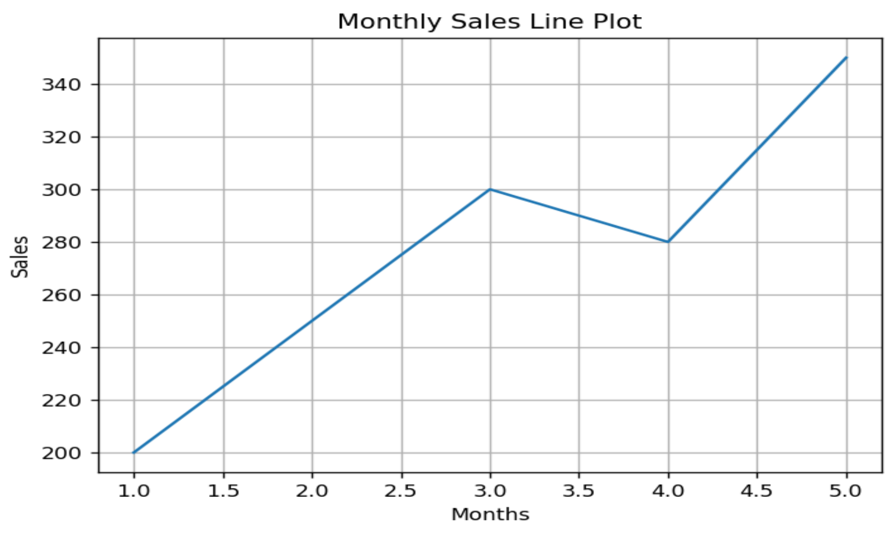
1. **Line Plot:** A line plot displays data points connected by straight lines. It is mainly used to show trends over time or continuous data.
   Use cases:
   Showing trends, Displaying stock price movement, Visualizing changes in temperature across days.

Code Example:



```python
import matplotlib.pyplot as plt
import numpy as np
months = np.array([1, 2, 3, 4, 5])
sales = np.array([200, 250, 300, 280, 350])
plt.plot(months, sales)
plt.xlabel("Months")
plt.ylabel("Sales")
plt.title("Monthly Sales Line Plot")
plt.grid(True)
plt.show()
```

Output:



Description: np.array() creates numerical arrays for months and sales. plot() draws a line connecting the sales values month-wise. xlabel() and ylabel() label the axes. grid(True) adds a background grid to the plot.

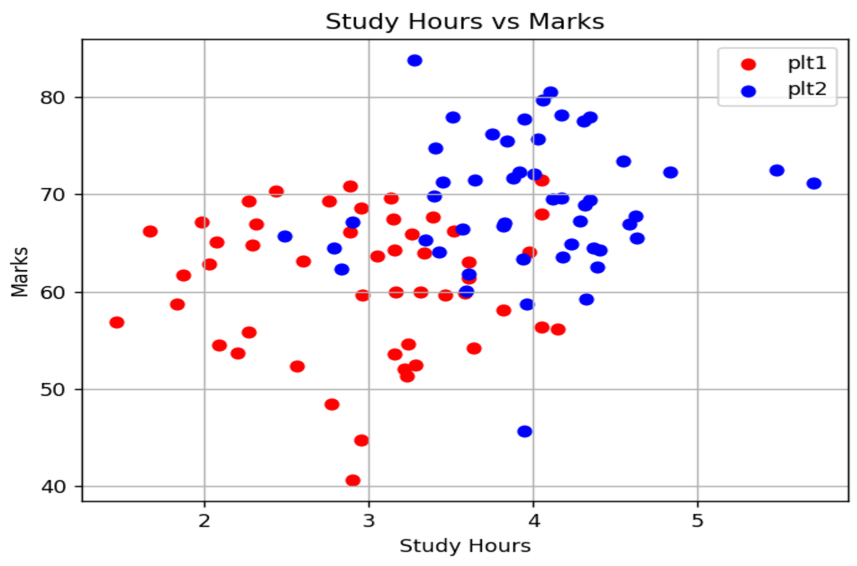
title() gives a title to the graph. show() displays the final line graph.

2. **Scatter Plot:** A scatter plot displays individual data points on a 2D graph. It is used to show relationships or correlations between two variables.
   Use cases:
   Finding correlation between height and weight, Comparing two numerical variables.

Code Example:

```
import matplotlib.pyplot as plt
import numpy as np
hours1=np.random.normal(3,0.7,50)
marks1=np.random.normal(60,8,50)
hours2=np.random.normal(4,0.6,50)
marks2=np.random.normal(70,7,50)
plt.scatter(hours1,marks1,color='red')
plt.scatter(hours2,marks2,color='blue')
plt.xlabel("Study Hours")
plt.ylabel("Marks")
plt.title("Study Hours vs Marks")
plt.legend(["plt1","plt2"])
plt.grid(True)
plt.show()
```

Output:



Description: np.random.normal() generates random values following a normal distribution for study hours and marks of two groups.

scatter() plots each group of values as scatter points (red for plt1 blue for plt2).

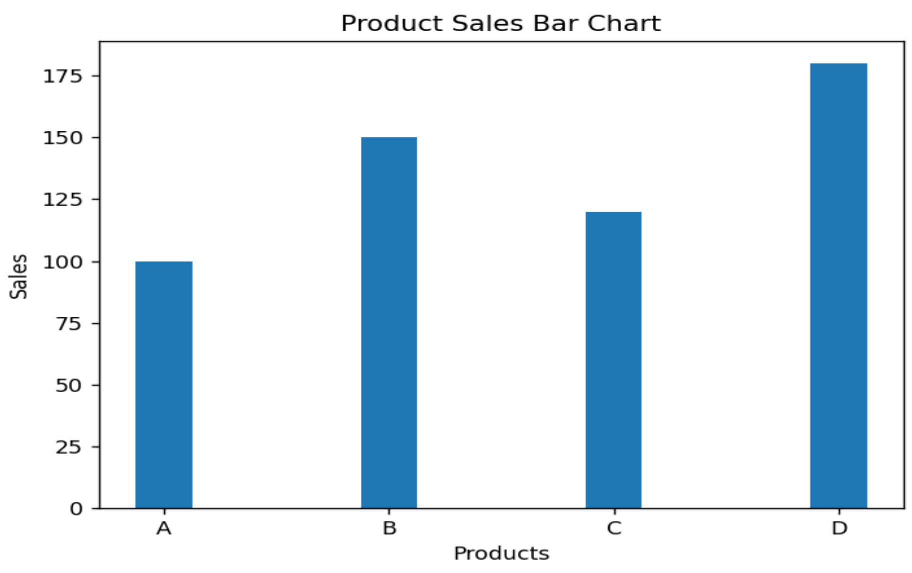
legend() displays labels for the two plotted groups.

3. **Bar Chart:** A bar chart represents data using rectangular bars. Useful for comparison between categories.
   Use Cases: Comparing sales of different products, Marks comparison between subjects.

Code Example:

```python
import matplotlib.pyplot as plt
import numpy as np
products=np.array(['A','B','C','D'])
sales=np.array([100,150,120,180])
plt.bar(products,sales, width=0.25)
plt.xlabel("Products")
plt.ylabel("Sales")
plt.title("Product Sales Bar Chart")
plt.show()
```

Output:



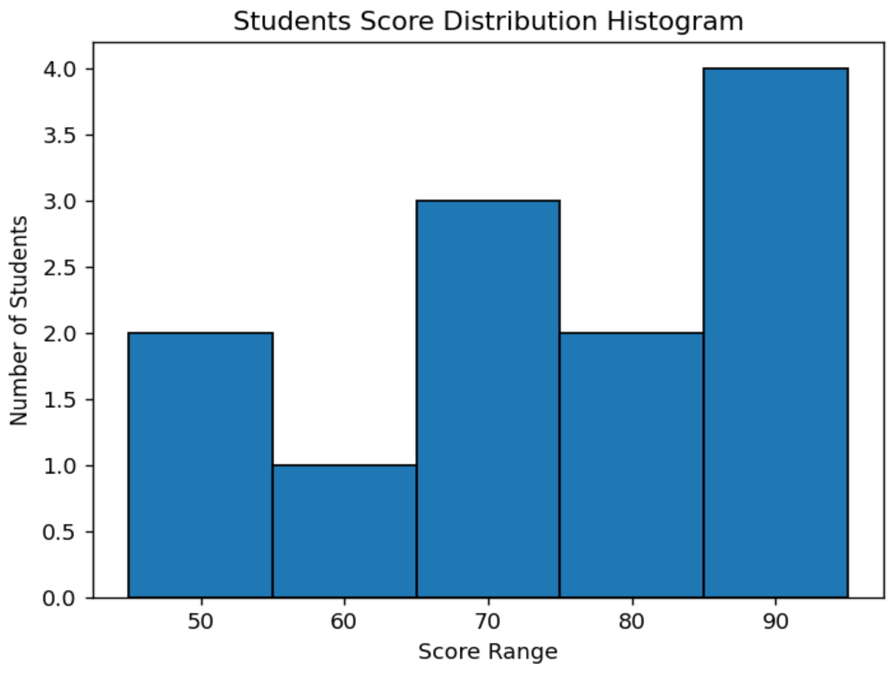
Description: bar() draws vertical bars for each product category. Height of each bar represents the sales value. Labels and title explain what the bars represent.

4. **Histogram:** A histogram shows the distribution of numerical data by grouping values into bins.
   Use Cases: Visualizing exam score distribution, Analyzing age distribution.

Code Example:

```python
import matplotlib.pyplot as plt
import numpy as np
scores=np.array([45,50,60,65,70,72,75,80,85,90,92,95])
plt.hist(scores,bins=5,edgecolor="black")
plt.xlabel("Score Range")
plt.ylabel("Number of Students")
plt.title("Students Score Distribution Histogram")
plt.show()
```

Output:



Description:

hist() groups scores into 5 equal bins and counts frequency.

Histogram shows how many students fall into each score range.

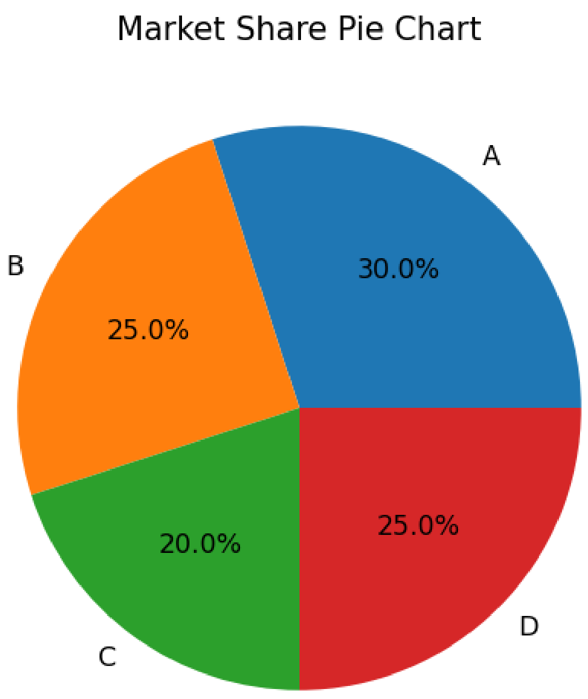
Labels and title describe the distribution.

5. **Pie Chart:** A pie chart represents a whole divided into percentage-based slices. It is used for showing proportional data.
Use Cases: Percentage of students in different streams.

Code Example:

```python
import matplotlib.pyplot as plt
import numpy as np
companies=np.array(['A','B','C','D'])
market_share=np.array([30,25,20,25])
plt.pie(market_share,labels=companies,autopct='%1.1f%%')
plt.title("Market Share Pie Chart")
plt.show()
```

Output:



Description:

pie() creates the pie chart using these values.

labels= sets the company names around the chart.

autopct='%1.1f%%' displays percentages on each slice.

plt.title() adds a heading.

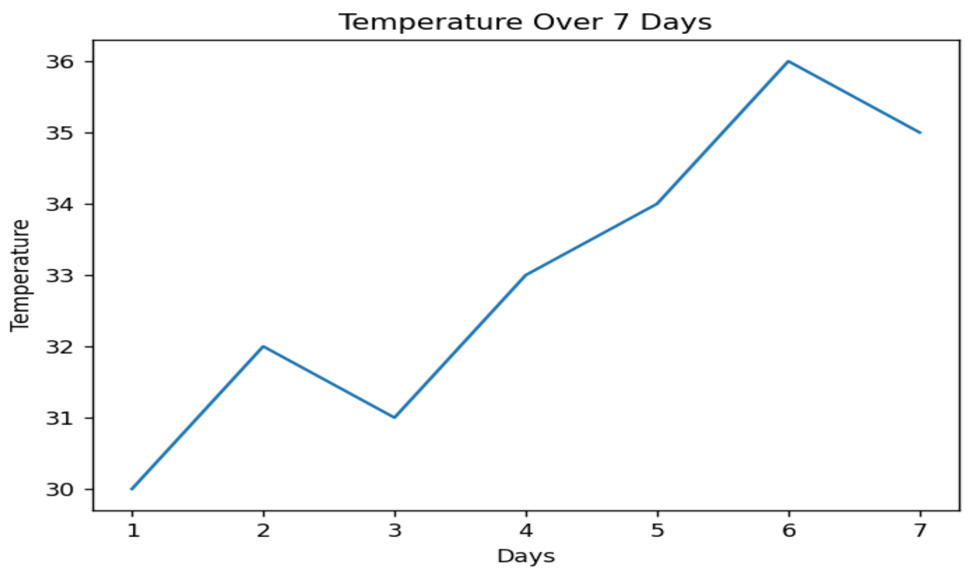
Types of graphs in Seaborn:

1. **Line Plot:** A line plot shows trends or continuous data using connected lines.
   Seaborn automatically adds statistical styling and smoothing.
   Use Cases: Showing trends over time (sales, temperature), Plotting continuous data.

Code Example:

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
days=np.array([1,2,3,4,5,6,7])
temperature=np.array([30,32,31,33,34,36,35])
sns.lineplot(x=days,y=temperature)
plt.xlabel("Days")
plt.ylabel("Temperature")
plt.title("Temperature Over 7 Days")
plt.show()
```

Output:



Description:

sns.lineplot() draws a continuous line showing temperature change across days.

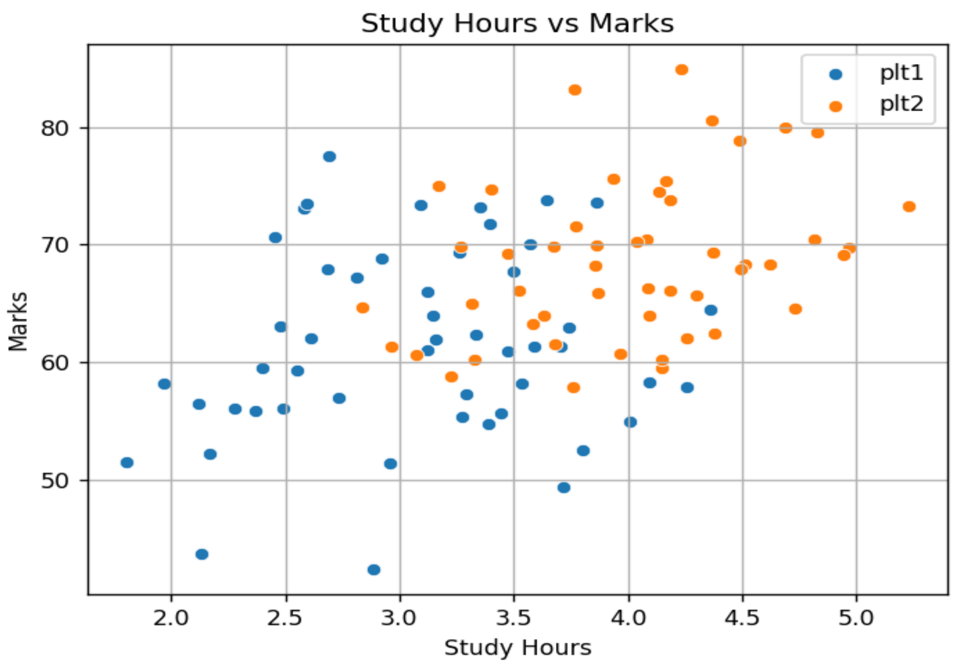
X-axis and Y-axis labels explain the data.

2. **Scatter Plot:** A scatter plot shows individual data points to reveal relationships, clusters, or correlations.
   Use Cases: Correlation between two variables.

Code Example:

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
hours1=np.random.normal(3,0.7,50)
marks1=np.random.normal(60,8,50)
hours2=np.random.normal(4,0.6,50)
marks2=np.random.normal(70,7,50)
sns.scatterplot(x=hours1,y=marks1)
sns.scatterplot(x=hours2,y=marks2)
plt.xlabel("Study Hours")
plt.ylabel("Marks")
plt.title("Study Hours vs Marks")
plt.legend(["plt1","plt2"])
plt.grid(True)
plt.show()
```

Output:



Description:

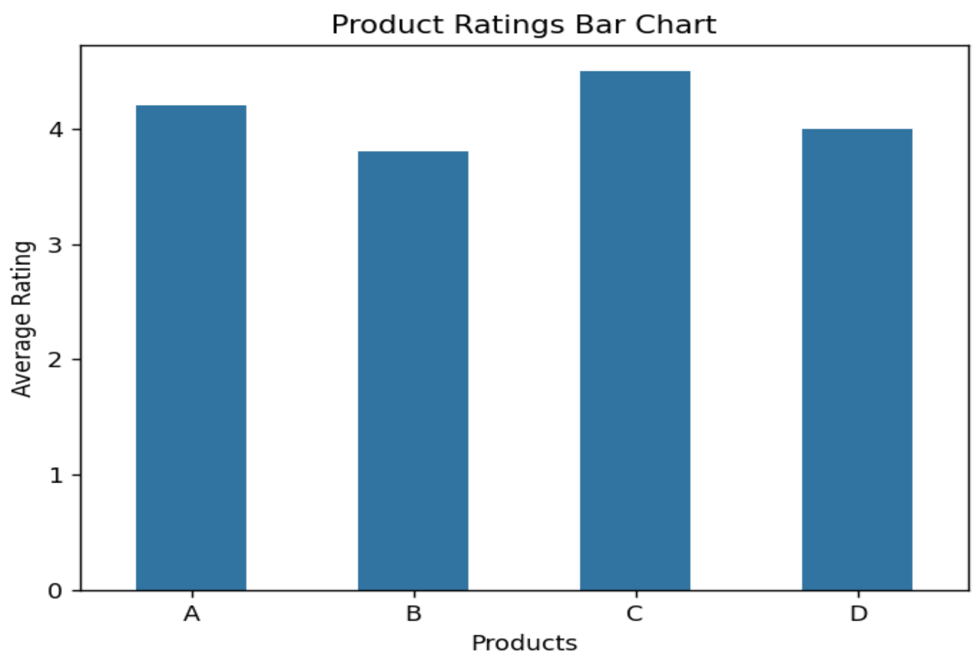
sns.scatterplot() plots points for each (hour, marks) pair. Helps visualize the relationship between two variables. Axis labels describe what the plot measures.

3. **Bar Chart:** A bar chart represents category-wise values using rectangular bars. Seaborn adds confidence intervals (CI) by default for aggregated data.
   Use Cases: Comparing average values across categories.

Code Example:

```
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
products=np.array(['A','B','C','D'])
ratings=np.array([4.2,3.8,4.5,4.0])
sns.barplot(x=products,y=ratings,errorbar=None,width=0.5)
plt.xlabel("Products")
plt.ylabel("Average Rating")
plt.title("Product Ratings Bar Chart")
plt.show()
```

Output:



Description:

sns.barplot() draws vertical bars for each product.

errorbar=None removes default confidence intervals.

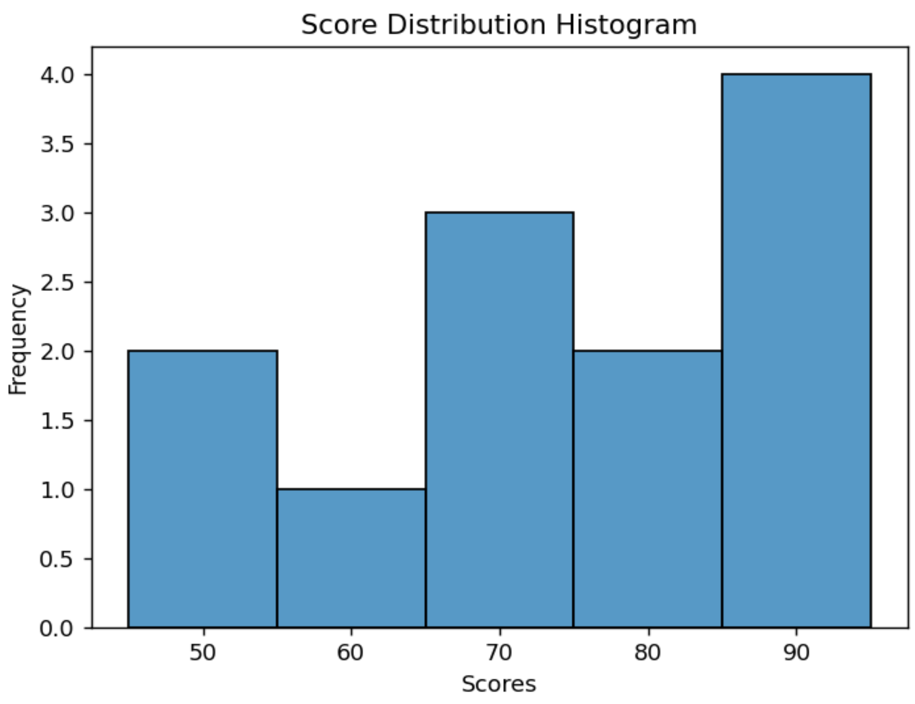
Labels and title explain what the bars represent.

4. **Histogram:** A histogram shows the distribution of numerical data divided into bins.
   Use Cases: Showing score distribution, understanding data spread.

Code Example:

```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
scores=np.array([45,50,60,65,70,72,75,80,85,90,92,95])
sns.histplot(scores,bins=5)
plt.xlabel("Scores")
plt.ylabel("Frequency")
plt.title("Score Distribution Histogram")
plt.show()
```

Output:



Description:

sns.histplot() divides the scores into 5 bins.

Frequency of scores in each range is shown as bars.

Labels and title explain the distribution.

Comparison between Matplotlib and Seaborn:

Comparison Factors	Matplotlib	Seaborn
Ease of Use	Requires more code and manual settings for colors, styles, and labels. Slightly harder for beginners.	Very easy to use. Automatically applies styles and reduces the amount of code needed. Good for beginners.
Customization	Highly customizable - you can change almost any part of the plot (axes, colors, fonts, lines, grid). Great for detailed work.	Limited customization. Gives clean and attractive plots but deep changes require Matplotlib commands.
Default Style	Basic and plain-looking by default. Needs manual styling to look attractive.	Modern, stylish, and clean default look without any extra code.
Interactivity	Supports basic interactivity when combined with extra tools (e.g., widgets, GUI apps).	No direct interactivity. Fully dependent on Matplotlib for any interactive features.
Performance with Large Datasets	Better performance for large datasets because it is lower-level and optimized.	Slower with very large datasets, especially statistical plots that compute additional data internally.
Plot Variety	Supports a wide range of basic and advanced plot types (scatter, line, bar, histogram, 3D plots, subplots).	Great for statistical plots like heatmaps, boxplots, violin plots, pair plots, etc. Designed for data analysis.
Coding Style	Low-level: you control everything manually, giving more flexibility but more work.	High-level: automatically handles many steps, reducing effort and making code shorter.
Best Use Case	Best for detailed, customized, publication-ready and scientific visualizations.	Best for quick, attractive, and statistical data visualizations during data analysis.