ShadowFox Data Science Internship

(Beginner Level Task – Visualization Library Documentation)

Understanding and Implementing Matplotlib for Data Insights:

History & Origin:

Matplotlib was created in 2003 by John D. Hunter, who wanted a Python-based visualization library that behaved like MATLAB's plotting system.

Since then, it has become the foundation for almost every other Python plotting library (including Seaborn, Pandas plot, and even some parts of Plotly).

Today, Matplotlib is maintained by an open-source community and is considered the standard plotting tool in Python for scientific computing.

What is Matplotlib?

Matplotlib is a **comprehensive data visualization library** in Python that allows you to create **static, animated, and interactive** plots with fine-grained control over every component of the figure.

It was created by **John D. Hunter** in 2003 to replicate MATLAB's plotting capabilities using Python's simplicity and flexibility. Since then, it has become the **backbone of most Python visualization tools**—including Seaborn, Pandas plots, and even Plotly in certain aspects.

Core Purpose:

Matplotlib enables data analysts, scientists, and engineers to **visualize data effectively** to:

- Explore patterns
- Spot trends
- Share insights
- Explain findings visually

Architecture – How Matplotlib Works:

Matplotlib has a **layered architecture**, which gives you complete control over your visuals.

Layer	**Description**
Figure	The entire canvas or window — one complete plot layout
Axes	A plot area (sub-figure) where data is drawn
Axis	Handles the ticks, scale, labels of X and Y axes
Artist	Anything drawn (line, text, shape, legend, etc.)

You can choose to use:

- **Pyplot interface** (plt.plot()) – quick and easy
- **Object-Oriented API** – for complex or custom plots

Matplotlib vs MATLAB:

Feature	MATLAB	Matplotlib (Python)
Language	Proprietary (MATLAB)	Open Source (Python)
Cost	Paid software	Free and open source
Plotting Syntax	Compact but less flexible	More flexible and powerful
Ecosystem	Closed	Open (NumPy, Pandas, Seaborn, etc.)
Community	Smaller, paid users	Massive open-source community

Why Is It So Popular?

- **Low-Level Control**: Customize everything—axes, lines, ticks, labels, fonts, grid, colors, spacing, etc.

- **Layered Structure**: You build plots like stacking bricks—first figure, then axes, then lines, then labels.

- **Integration-Ready**: Works smoothly with NumPy, Pandas, Jupyter Notebooks, Tkinter, and more.

- **Huge Plot Variety**: From basic bar and line charts to advanced subplots and polar plots.

- **Community & Stability**: 20+ years of community development and support with a rich ecosystem.

Key Features:

- Supports 2D plots: line, scatter, bar, pie, histogram, etc.
- Offers subplots and axes-level control
- Integrates with NumPy, Pandas, OpenCV, and Tkinter
- Can output to PNG, PDF, SVG, and other formats
- Used for both exploratory data analysis (EDA) and publication-quality visuals

When to Use:

- When you need fine control over every plot element
- For complex visual layouts
- For generating static visuals for reports and papers

Use Cases:

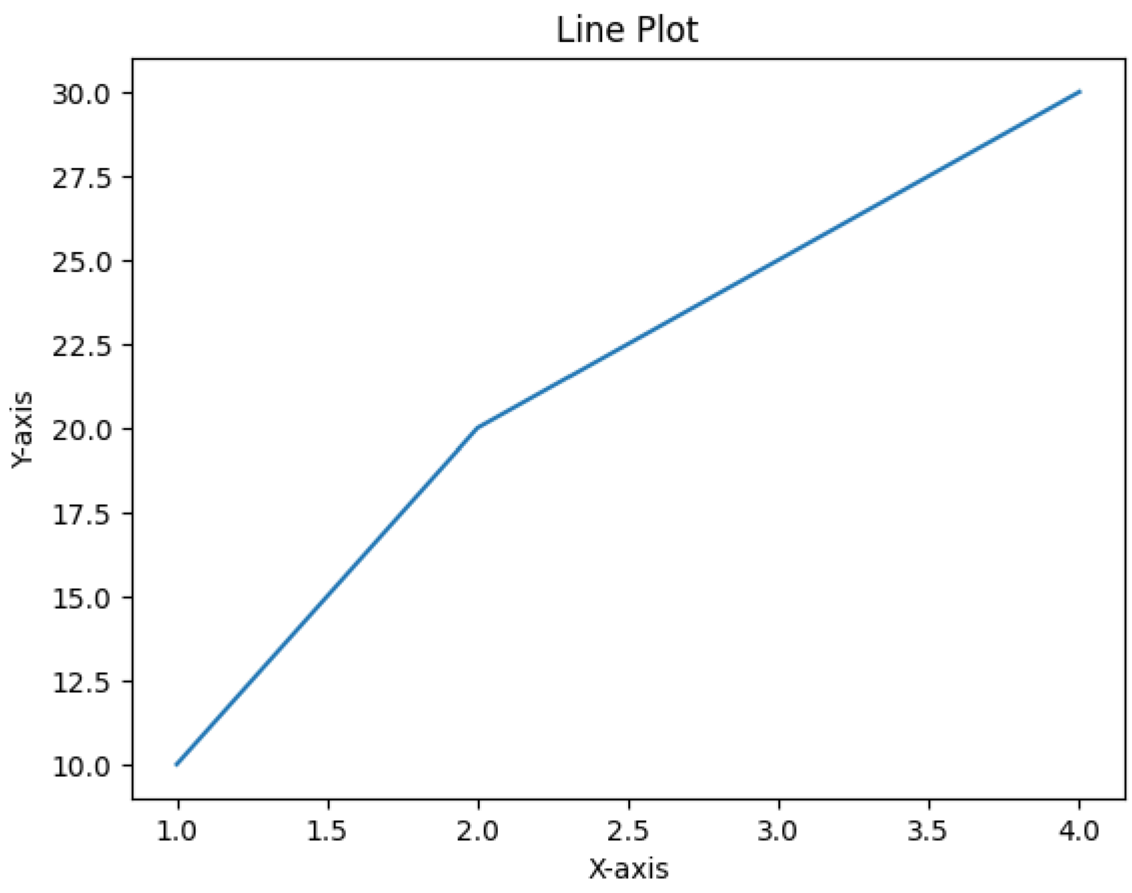
- Scientific and engineering plots
- Visualizing algorithms and processes
- Custom dashboards in apps
- Education

Practical Examples:

1) LinePlot:

```
#Matplotlib LinePlot
import matplotlib.pyplot as plt

x = [1, 2, 3, 4]
y = [10, 20, 25, 30]

plt.plot(x, y)
plt.title("Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

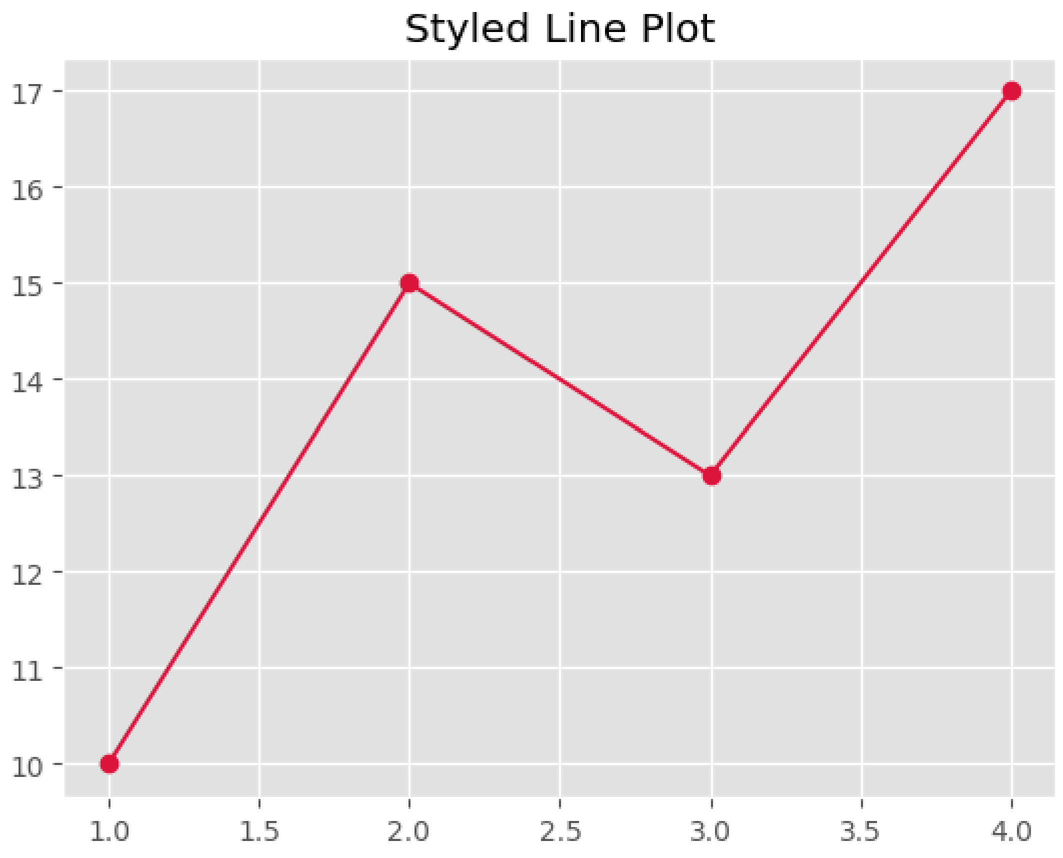
Output:



2) Styled Line Plot:

```
#Stylish Line Plot with Theme
plt.style.use('ggplot')  # Try 'seaborn-darkgrid', 'fivethirtyeight',
etc.

x = [1, 2, 3, 4]
y = [10, 15, 13, 17]

plt.plot(x, y, marker='o', color='crimson')
plt.title("Styled Line Plot")
plt.show()
```

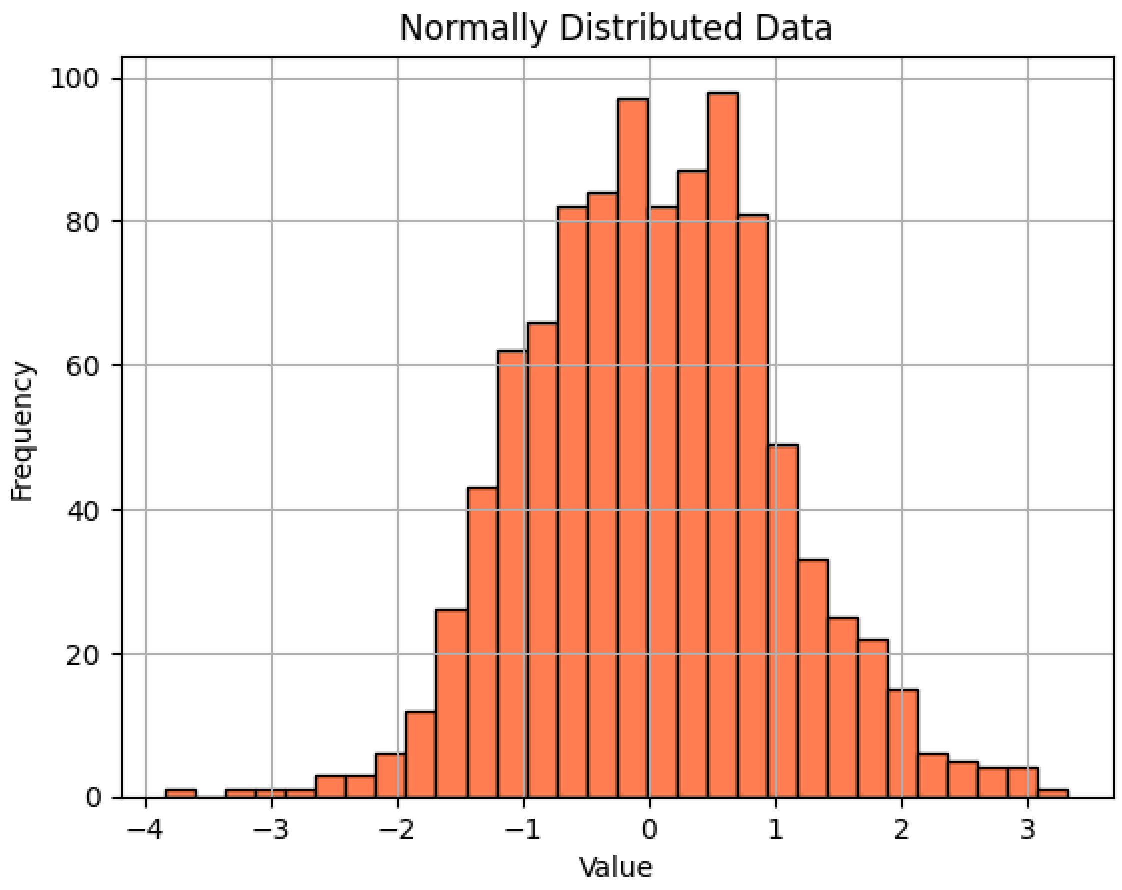
Output:



3) Custom Color Histogram:

```
#Custom Color Histogram
import numpy as np

data = np.random.randn(1000)

plt.hist(data, bins=30, color='coral', edgecolor='black')
plt.title("Normally Distributed Data")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.grid(True)
plt.show()
```

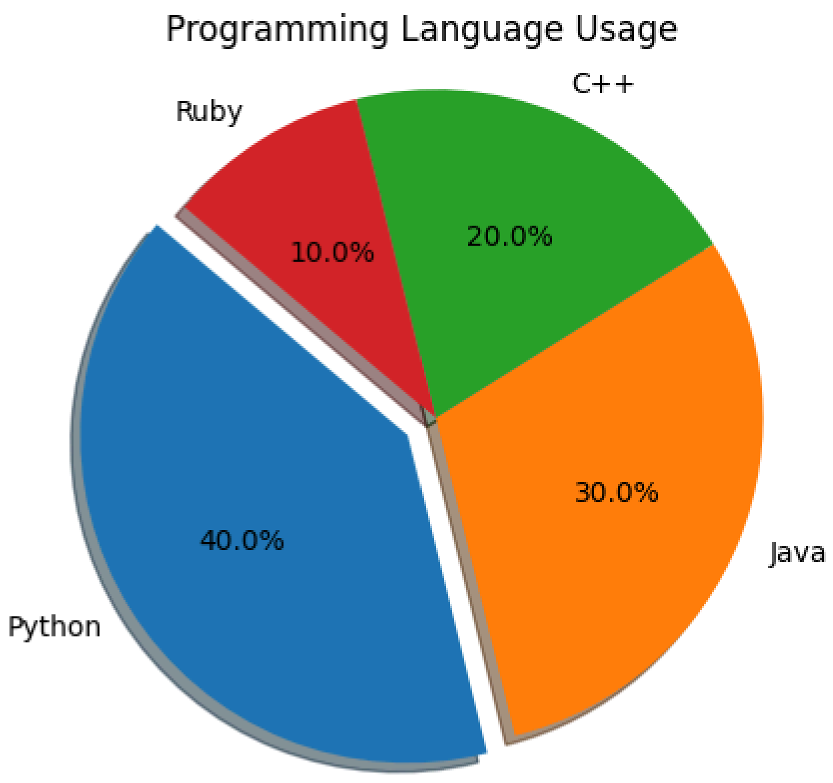
Output:



4) **Styled PieChart:**

```
#Styled Pie Chart with Explode
sizes = [40, 30, 20, 10]
labels = ['Python', 'Java', 'C++', 'Ruby']
explode = (0.1, 0, 0, 0)

plt.pie(sizes, labels=labels, autopct='%1.1f%%', explode=explode,
shadow=True, startangle=140)
plt.title("Programming Language Usage")
plt.axis('equal')
plt.show()
```

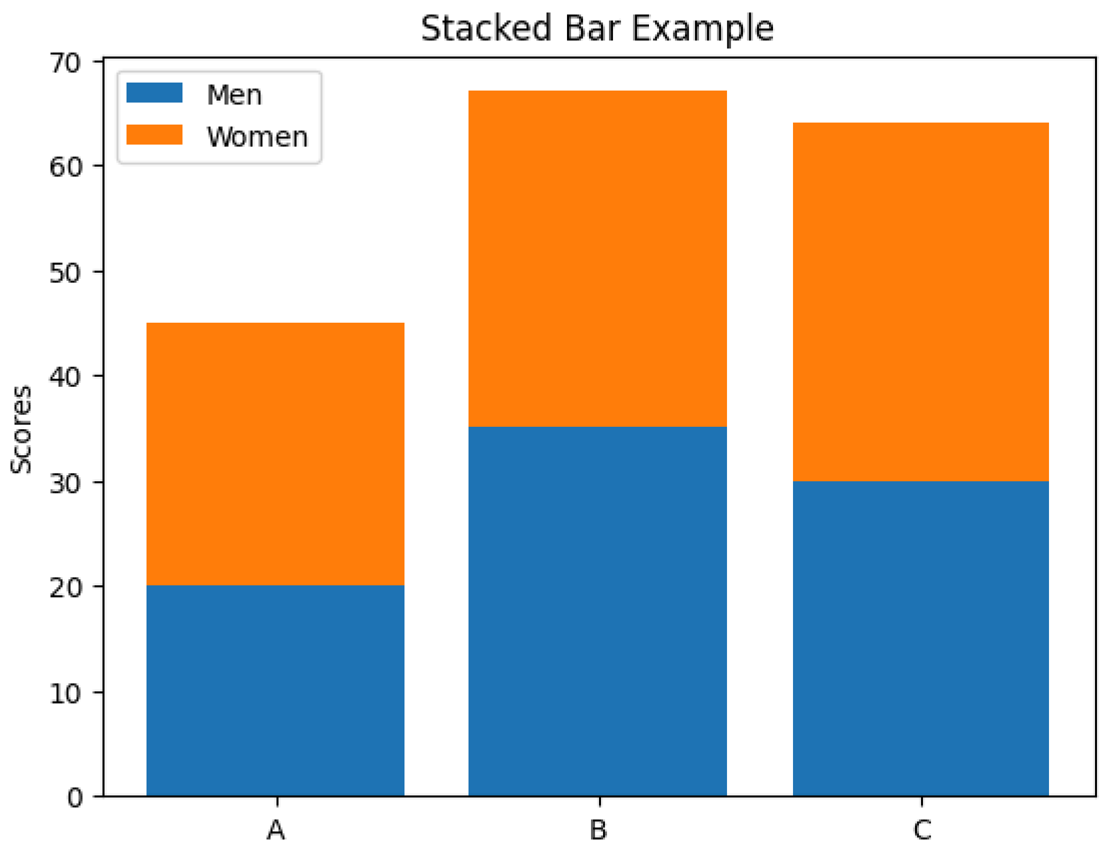
Output:



5) Stacked BarChart:

```python
#Stacked Bar Chart
import numpy as np

labels = ['A', 'B', 'C']
men = [20, 35, 30]
women = [25, 32, 34]

x = np.arange(len(labels))

plt.bar(x, men, label='Men')
plt.bar(x, women, bottom=men, label='Women')

plt.xticks(x, labels)
plt.ylabel('Scores')
plt.title('Stacked Bar Example')
plt.legend()
plt.show()
```

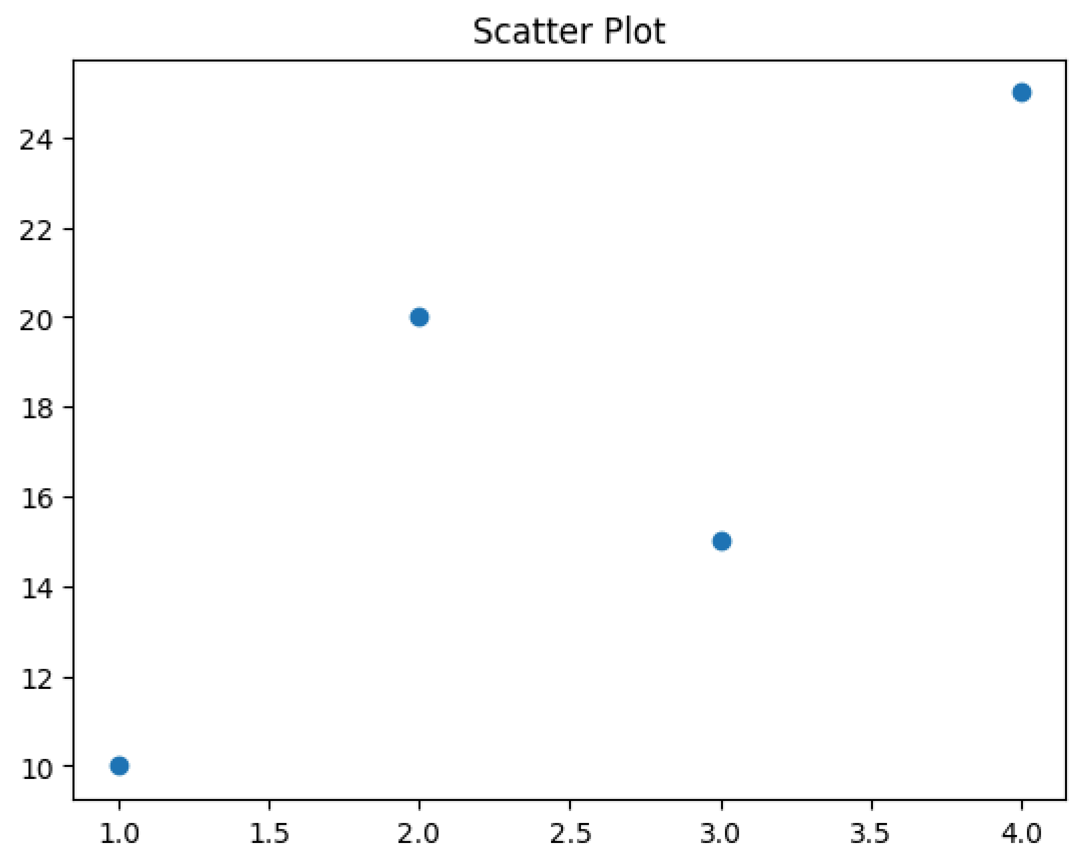
Output:



6) Scatter Plot:

```
#ScatterPlot
x = [1, 2, 3, 4]
y = [10, 20, 15, 25]

plt.scatter(x, y)
plt.title("Scatter Plot")
plt.show()
```

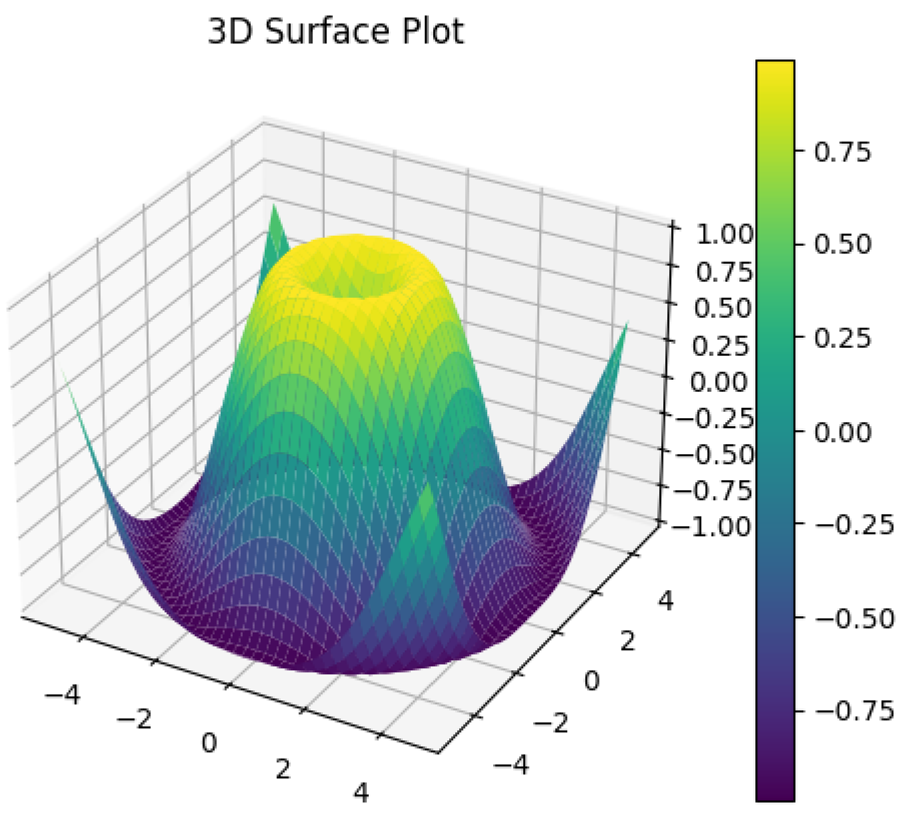
Output:



7) <u>3D Surface Plot</u>:

```
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
Z = np.sin(np.sqrt(X**2 + Y**2))

surf = ax.plot_surface(X, Y, Z, cmap=cm.viridis)
fig.colorbar(surf)

plt.title("3D Surface Plot")
plt.show()
```

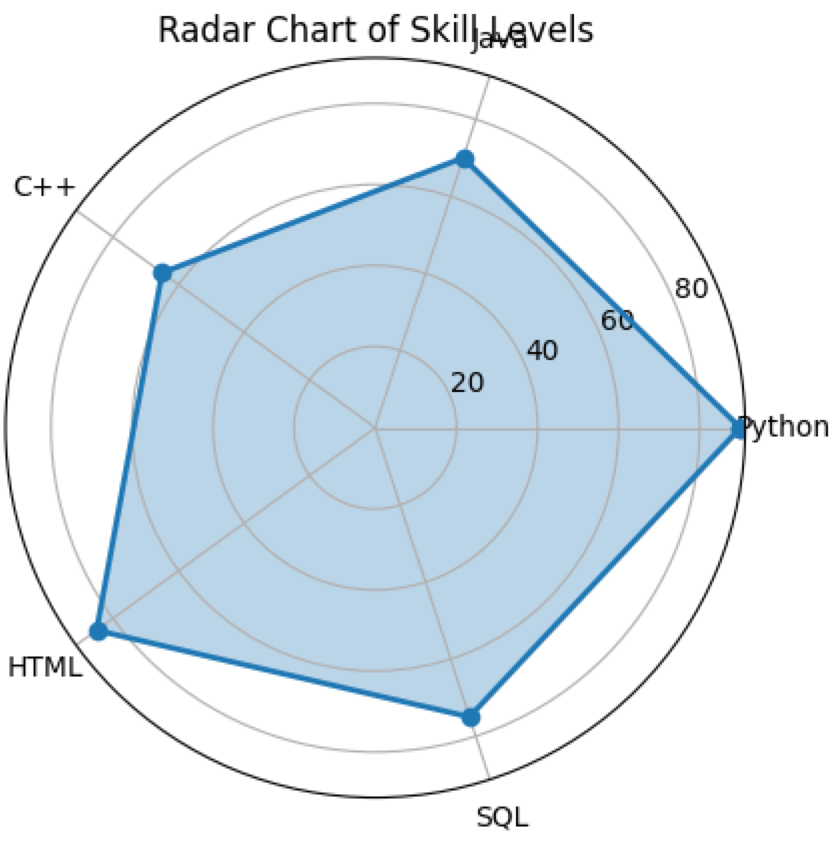
<u>Output</u>:



8) Radar Chart:

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

labels = ['Python', 'Java', 'C++', 'HTML', 'SQL']
values = [90, 70, 65, 85, 75]

angles = np.linspace(0, 2 * np.pi, len(labels),
endpoint=False).tolist()
values += values[:1]
angles += angles[:1]

fig, ax = plt.subplots(subplot_kw={'polar': True})
ax.plot(angles, values, 'o-', linewidth=2)
ax.fill(angles, values, alpha=0.3)
ax.set_thetagrids(np.degrees(angles[:-1]), labels)
plt.title('Radar Chart of Skill Levels')
plt.show()
```

Output:



Seaborn – Beautiful Statistical Visualization

History & Philosophy of Seaborn:

Seaborn was developed by **Michael Waskom** to simplify complex Matplotlib syntax and make beautiful plots with minimal code. It follows the principle of:

> “Simple things should be simple. Complex things should be possible.”

Its goal is to make **statistical visualization** more accessible, especially when working with **Pandas DataFrames**, which is the most common data structure in data science.

What is Seaborn?

Seaborn is a high-level Python data visualization library built on top of **Matplotlib**. It was developed to make it easier to create attractive and informative statistical plots with just a few lines of code.

Seaborn integrates tightly with **Pandas DataFrames**, allowing seamless plotting of complex data with minimal code. It is designed to work well for **Exploratory Data Analysis (EDA)** and is especially good at **group-wise comparisons**, **correlation analysis**, and **distribution visualization**.

Components of Seaborn:

Component Type	**Examples**	**Description**
Relational Plots	Scatterplot, lineplot	Compare two variables
Categorical Plots	Barplot, violinplot	Compare categories
Distribution Plots	histplot, kdeplot, boxenplot	Show how data is distributed
Matrix Plots	Heatmap, clustermap	Visualize grids or correlations
Multi-plot Grids	Facetgrid, pairplot, jointplot	Break data into multiple charts

Why Use Seaborn?

- Beautiful built-in themes and color palettes
- Easy statistical plots like violin plots, boxen plots, KDE, and heatmaps
- Works directly with Pandas datasets
- Ideal for visualizing distributions, categories, and relationships

When Should You Use Seaborn?

- When you want quick, aesthetically pleasing plots
- When you're doing EDA or comparing groups of data
- When you want to explore **patterns, outliers, and trends**

Seaborn Strengths:

- One-line statistical plots
- Automatic DataFrame integration
- Beautiful out-of-the-box visuals
- Built-in datasets for learning and testing
- Easy to use for beginners

Real-World Use Cases:

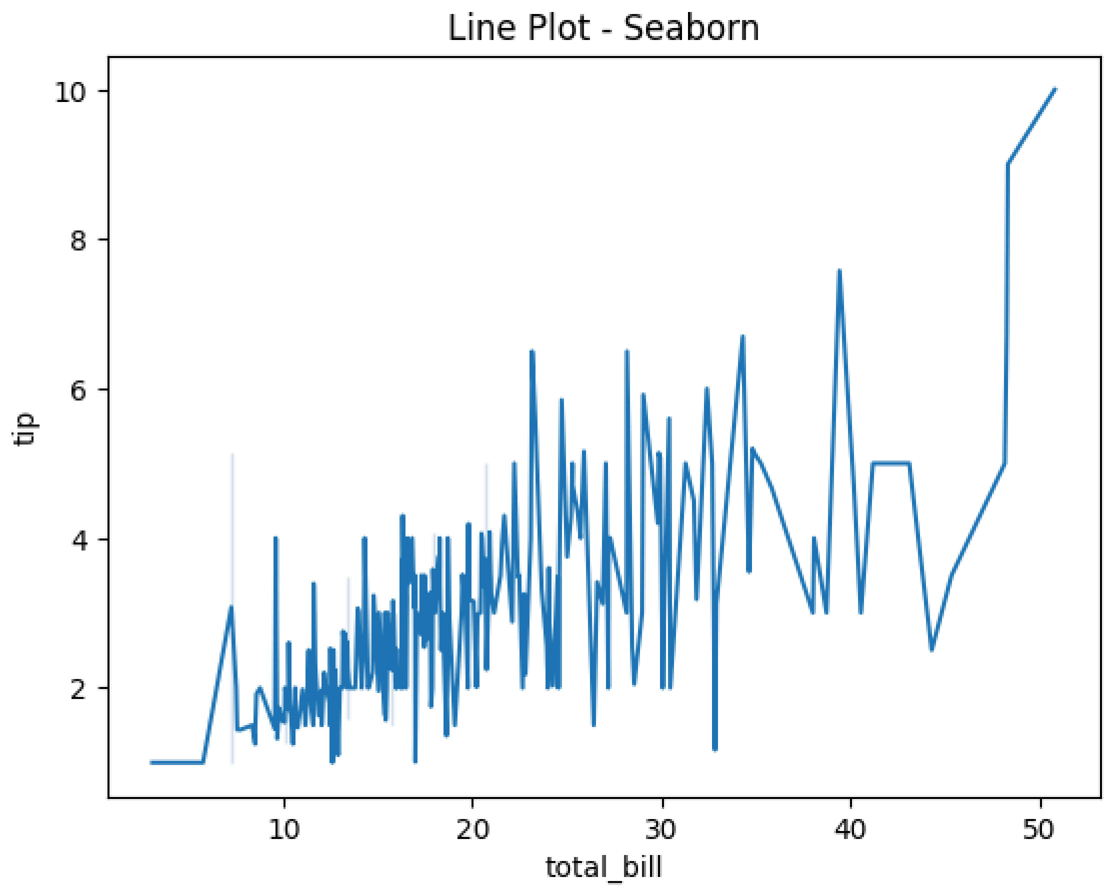
- Business Data
- Health/Medical Research
- Academia & Research
- Machine Learning

Practical Examples:

1) **LinePlot:**

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

tips = sns.load_dataset("tips")
sns.lineplot(data=tips, x="total_bill", y="tip")
plt.title("Line Plot - Seaborn")
plt.show()
```

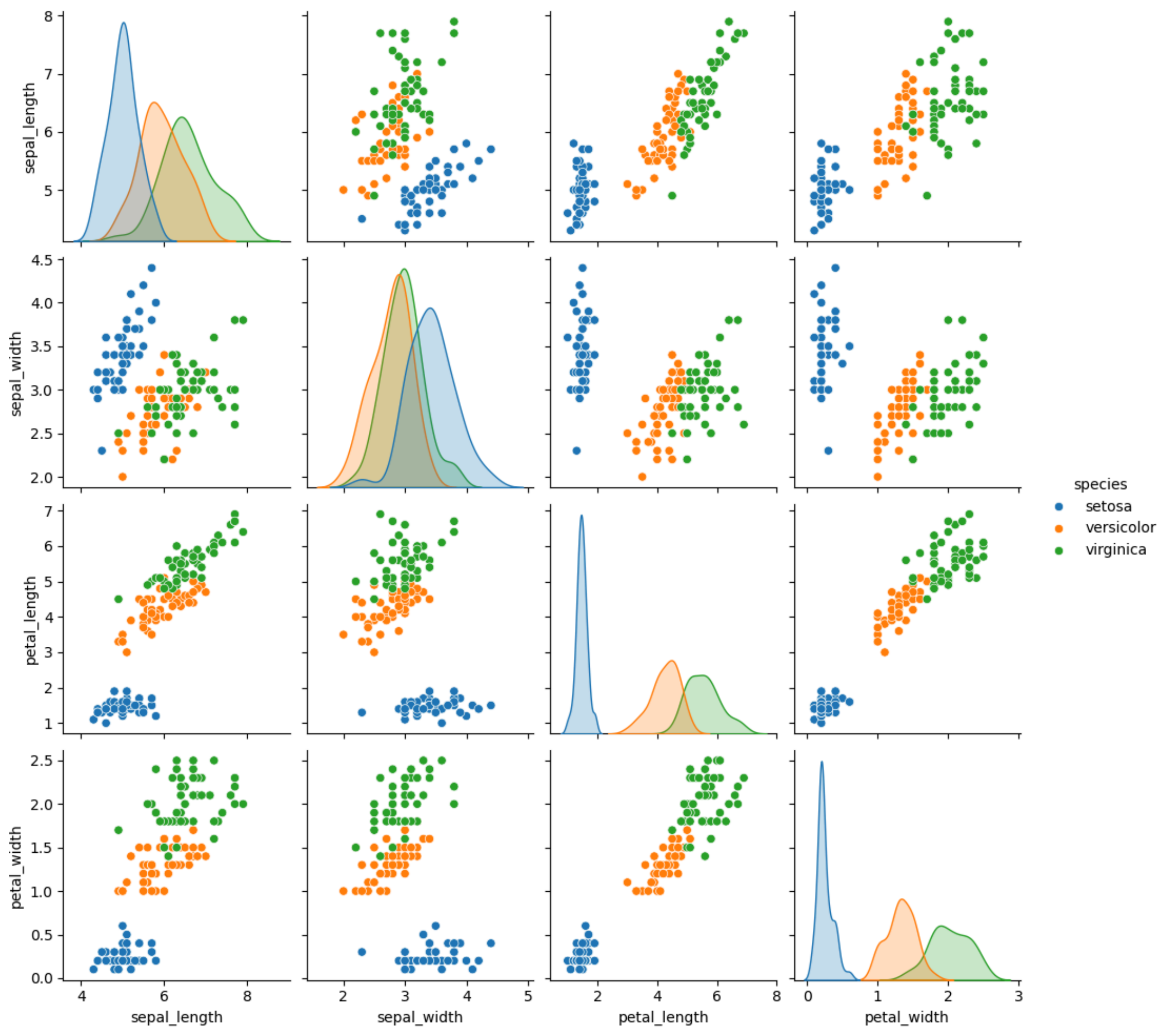
Output:



2) **PairPlot**:

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

df = sns.load_dataset("iris")
sns.pairplot(df, hue='species')
plt.show()
```

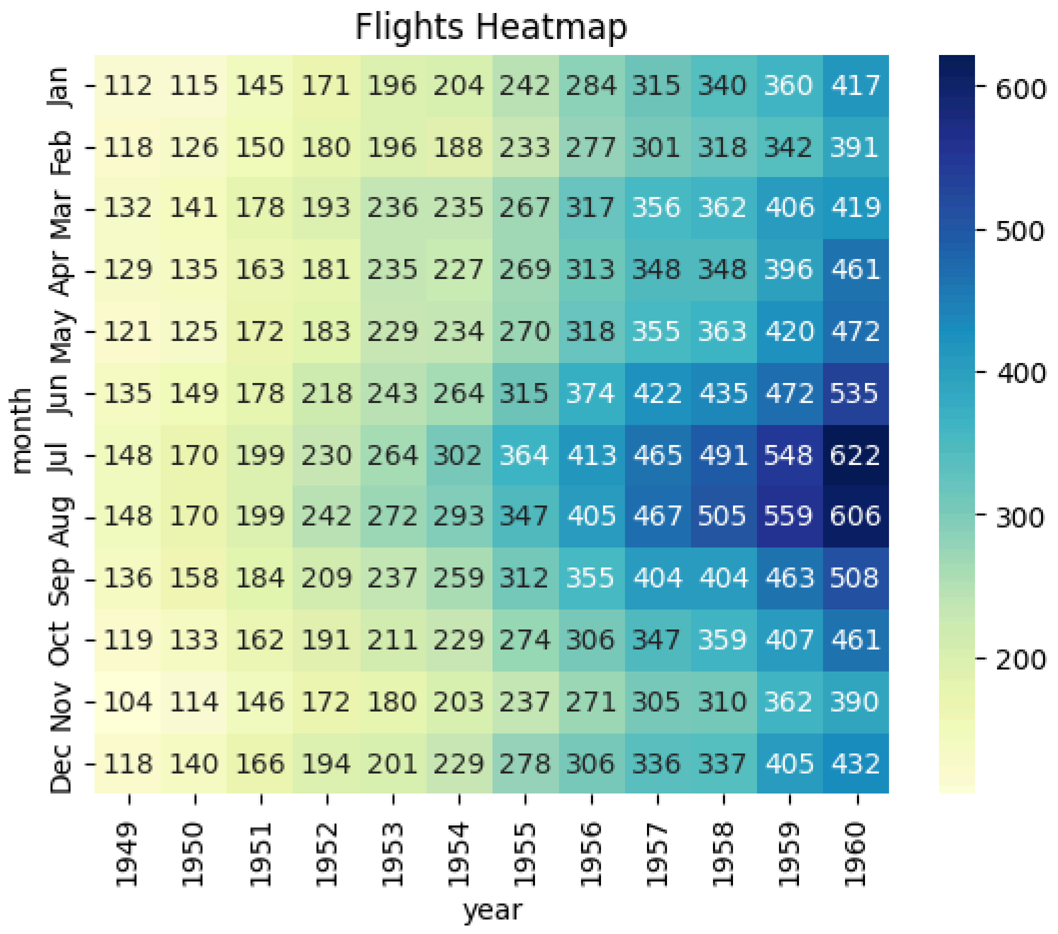
Output:



3) HeatMap:

```
#AdvancedHeatmap
data = sns.load_dataset("flights").pivot(index="month", columns="year",
values="passengers")
sns.heatmap(data, annot=True, fmt="d", cmap="YlGnBu")
plt.title("Flights Heatmap")
plt.show()
```

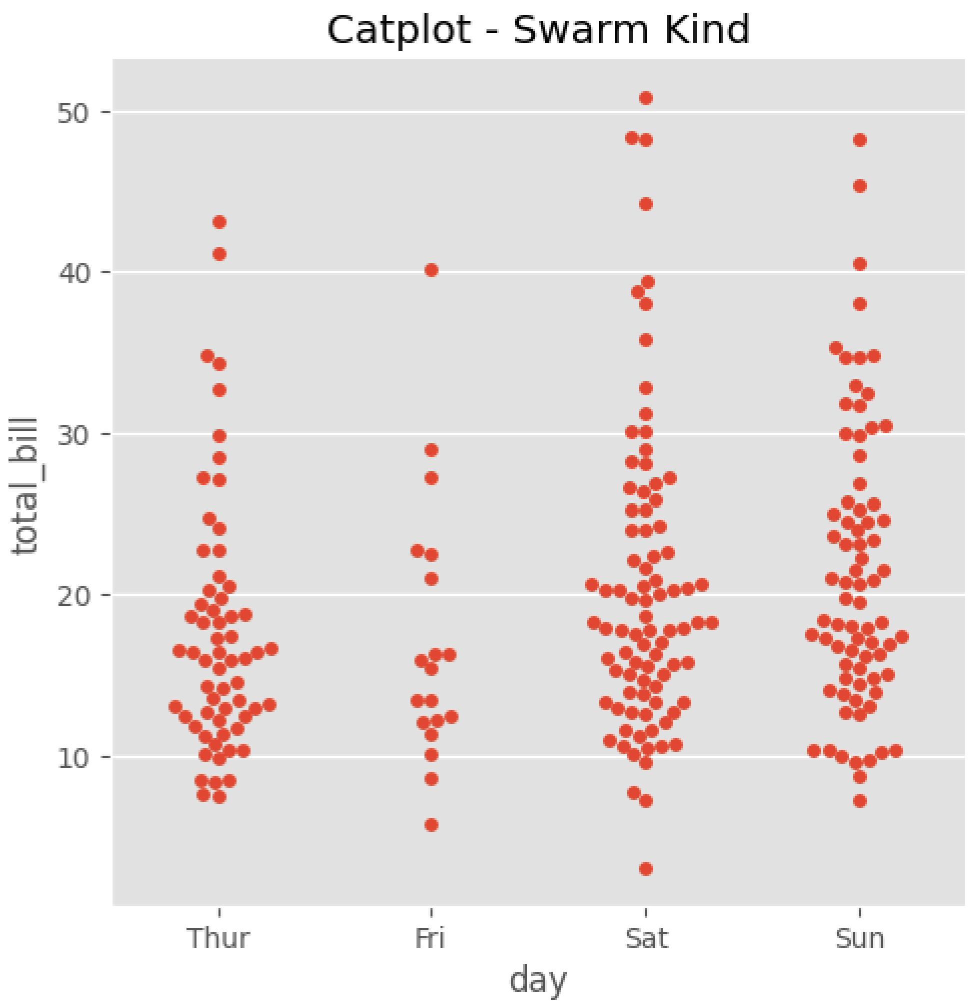
Output:



4) <u>CategoricalPlot</u>:

```
#Categorical Plot
sns.catplot(x="day", y="total_bill", kind="swarm", data=tips)
plt.title("Catplot - Swarm Kind")
plt.show()
```

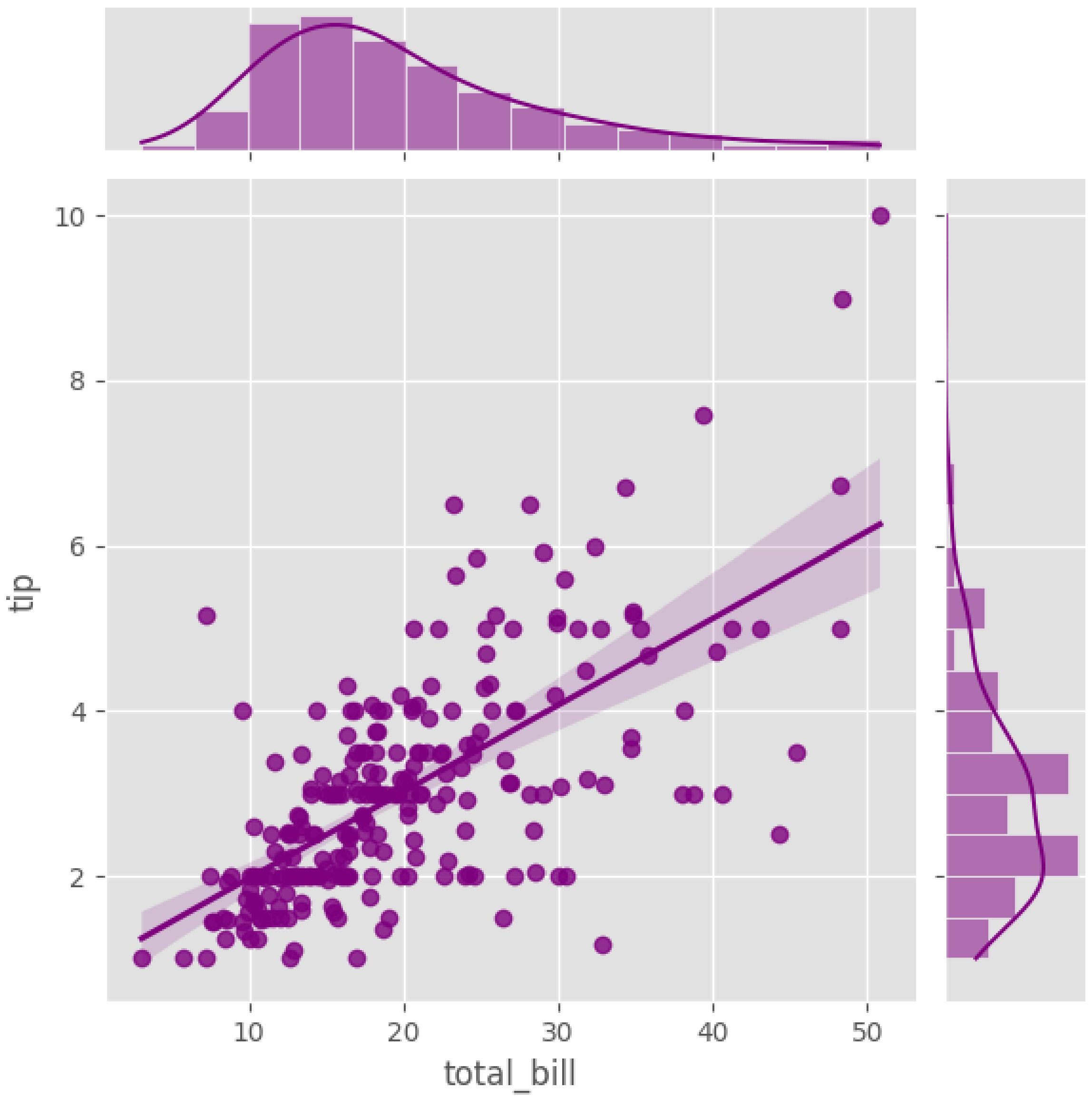
<u>Output</u>:



5) JointPlot:

```
#Joint Plot
sns.jointplot(data=tips, x="total_bill", y="tip", kind="reg",
color="purple")
plt.show()
```

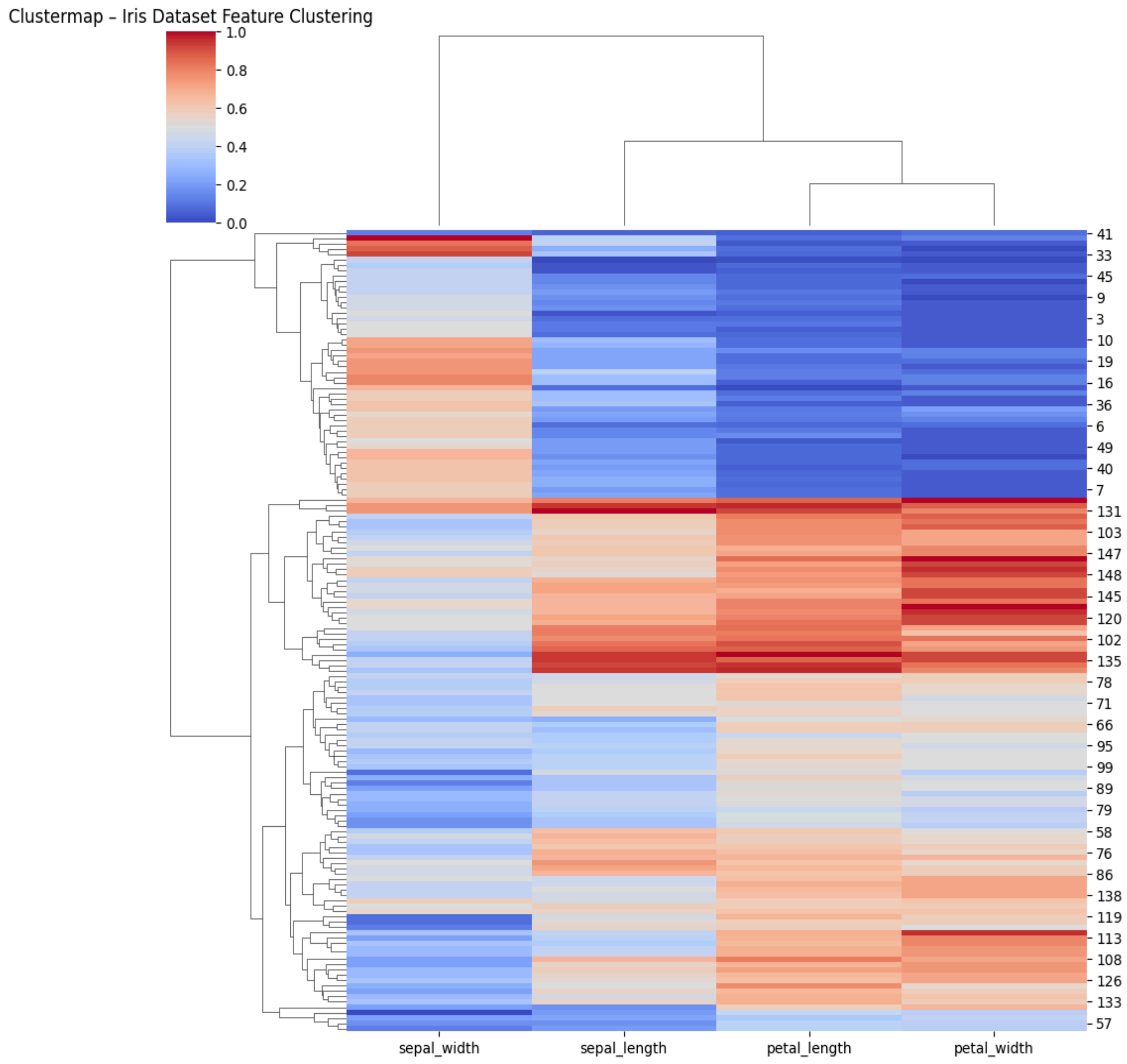
Output:



6) Heatmap with Clustering:

```
iris = sns.load_dataset("iris")
sns.clustermap(iris.drop("species", axis=1), cmap="coolwarm",
standard_scale=1)
plt.title("Clustermap - Iris Dataset Feature Clustering")
plt.show()
```

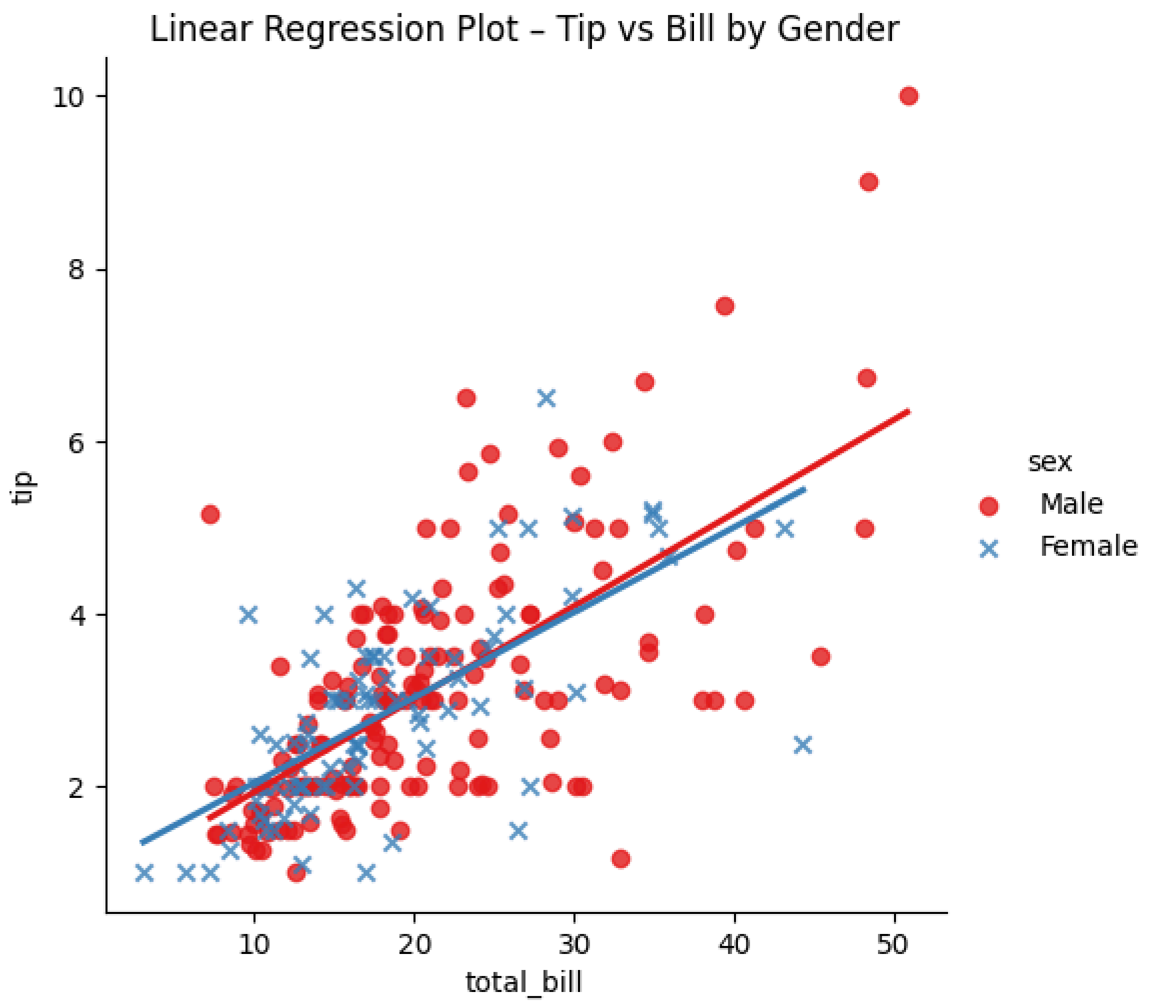
Output:



7) <u>Linear Regression Plot with Confidence Interval OFF</u>:

```
sns.lmplot(x="total_bill", y="tip", data=tips, ci=None, hue="sex",
markers=["o", "x"], palette="Set1")
plt.title("Linear Regression Plot - Tip vs Bill by Gender")
plt.show()
```

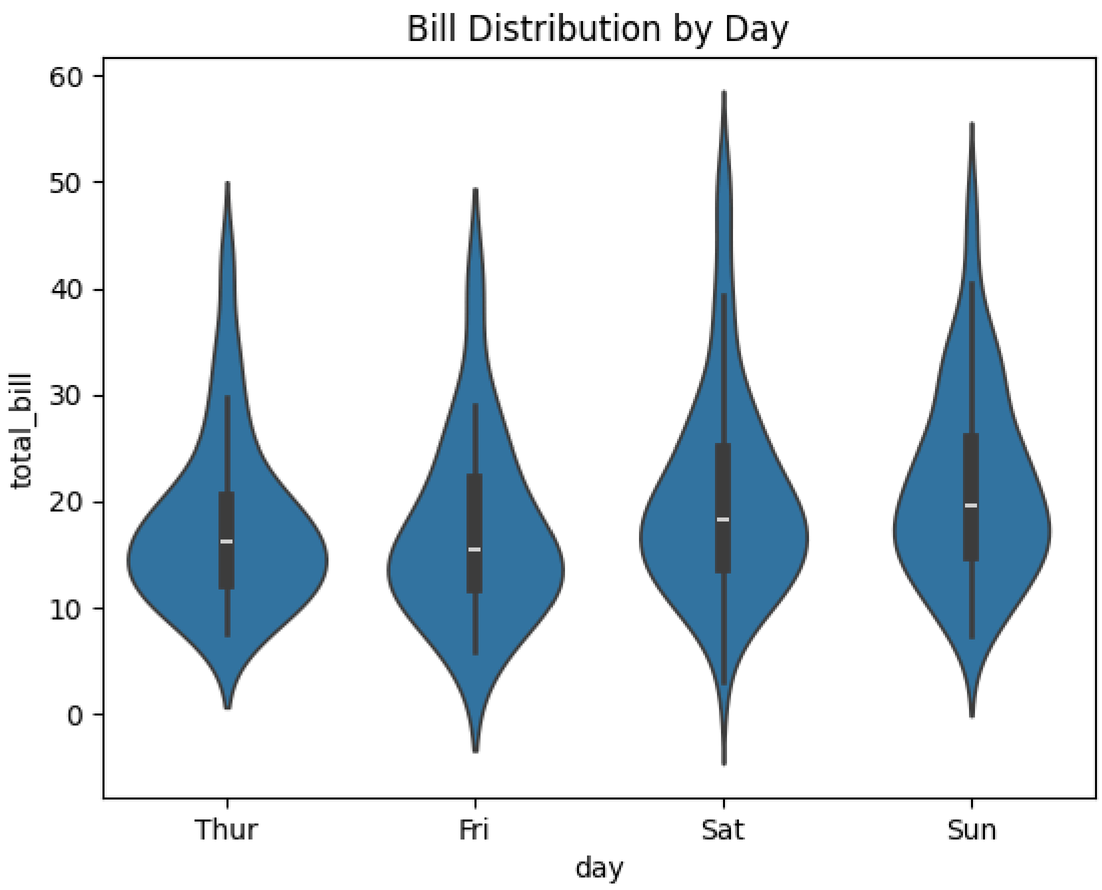
<u>Output</u>:



8) <u>Violin Distribution</u>:

```
#ViolinDistribution
sns.violinplot(x="day", y="total_bill", data=sns.load_dataset("tips"))
plt.title("Bill Distribution by Day")
plt.show()
```

<u>Output</u>:



Conclusion:

In this documentation, we explored two of the most powerful Python libraries for data visualization “Matplotlib and Seaborn”. Both libraries serve the same purpose: transforming raw data into meaningful and visually engaging stories, yet they do so in unique ways.

- Matplotlib offers low-level control, allowing us to customize every detail of our plots, making it ideal for highly specific or complex visualizations.

- Seaborn, on the other hand, simplifies the process of creating aesthetically pleasing and statistically insightful plots, making it a perfect companion for fast and informative analysis.

Through this beginner task, I not only learned how to create basic charts like line, bar, and scatter plots, but I also explored advanced plots such as radar charts, joint plots, heatmaps, violin plots, and layered visualizations. This helped me develop a deeper understanding of how data can be visualized from multiple angles and through different perspectives.