

Title: *“Data Visualization using Matplotlib and Seaborn”*

Sivakami s
Bannariamman Institute Of Technology
Sathyamangalam

VISUALIZATION

Introduction

Creating a graphical or pictorial representation of data is called visualisation. It assists in turning objective data into insightful understanding. Comprehending complex data is made easier with the use of maps, graphs, and charts. Patterns, trends, and connections within data are highlighted through visualisation. It facilitates decision-making by making big datasets easier to understand. In general, data visualisation improves its clarity, interest, and usability.

1. MATPLOTLIB

Matplotlib is a widely used **Python library for data visualization**. It provides a flexible way to create **static, animated, and interactive plots**. With Matplotlib, users can present data in various formats such as **line, bar, histogram, scatter, pie, and area plots**. It is often used in **data analysis, machine learning, and scientific research** because it integrates well with **NumPy and Pandas**. Matplotlib also allows **customization of colors, labels, styles, and layouts**, making visualizations clear and professional.

Common Plots in Matplotlib

1. **Line Plot** – Shows trends or relationships over time/sequence.
2. **Bar Plot** – Compares categories or groups using rectangular bars.
3. **Histogram** – Displays frequency distribution of a dataset.
4. **Scatter Plot** – Shows relationships and correlations between two variables.
5. **Pie Chart** – Represents proportions or percentages of a whole.
6. **Area Plot** – Similar to a line plot but emphasizes magnitude with filled areas.

Overview of Matplotlib

Matplotlib is a widely used **open-source data visualization library in Python**. It provides powerful tools for creating a variety of **static, interactive, and animated plots**. Matplotlib is considered the foundation of Python visualization, as many other libraries such as **Seaborn, Pandas visualization, and Plotly** build upon or integrate with it.

Key Features of Matplotlib

- **Customization:** Full control over figure size, colors, labels, grids, and styles.
- **Wide Range of Plots:** Supports line plots, bar charts, histograms, scatter plots, pie charts, area plots, and more.
- **Integration with NumPy & Pandas:** Works seamlessly with numerical and tabular data, making it ideal for scientific computing.

- **Flexibility:** Can produce publication-quality graphics with detailed customization.
- **Cross-platform Support:** Works across different operating systems and supports multiple output formats (PNG, PDF, SVG, etc.).

Why Matplotlib is Popular among Data Scientists

Matplotlib is popular because it combines **simplicity and flexibility**. Data scientists prefer it for:

- Quick creation of plots during exploratory data analysis.
- Precise control over every element of a figure for research/publication.
- Strong ecosystem support with Pandas, NumPy, and other data science tools.
- Large community, extensive documentation, and wide adoption in academia and industry.

1.Line Plot

A line plot displays data points connected by straight lines. It is mainly used to show **trends and changes over time**. The X-axis usually represents time or sequence, while the Y-axis shows values. Line plots are widely used in **time series analysis** like sales, stock prices, or temperature changes.

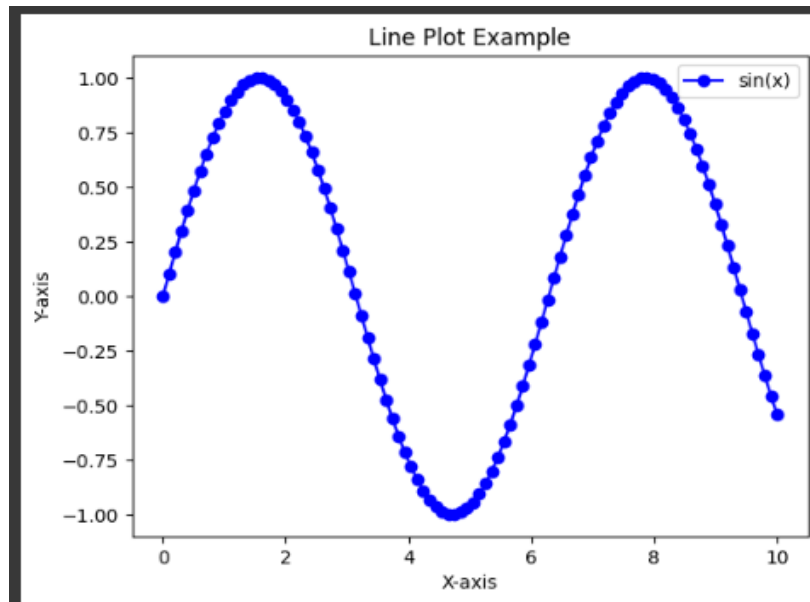
CODE SNIPPET :

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

x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.plot(x, y, label="sin(x)", color="blue", marker='o')
plt.title("Line Plot Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

OUTPUT :



Description:

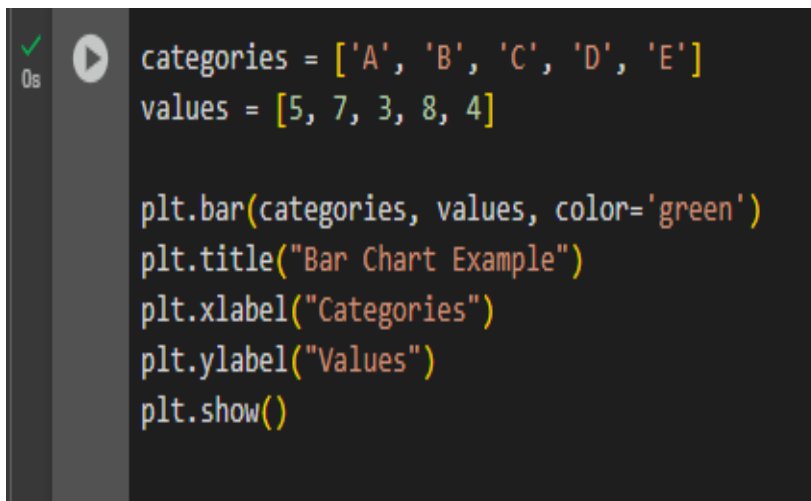
- `np.linspace(0, 10, 100)` generates 100 evenly spaced values between 0 and 10.
- `plt.plot()` is used to plot the line graph.
- `title()`, `xlabel()`, `ylabel()` add labels to the graph.
- `legend()` displays the label of the line.

- `show()` displays the final plot.

2.Bar Plot

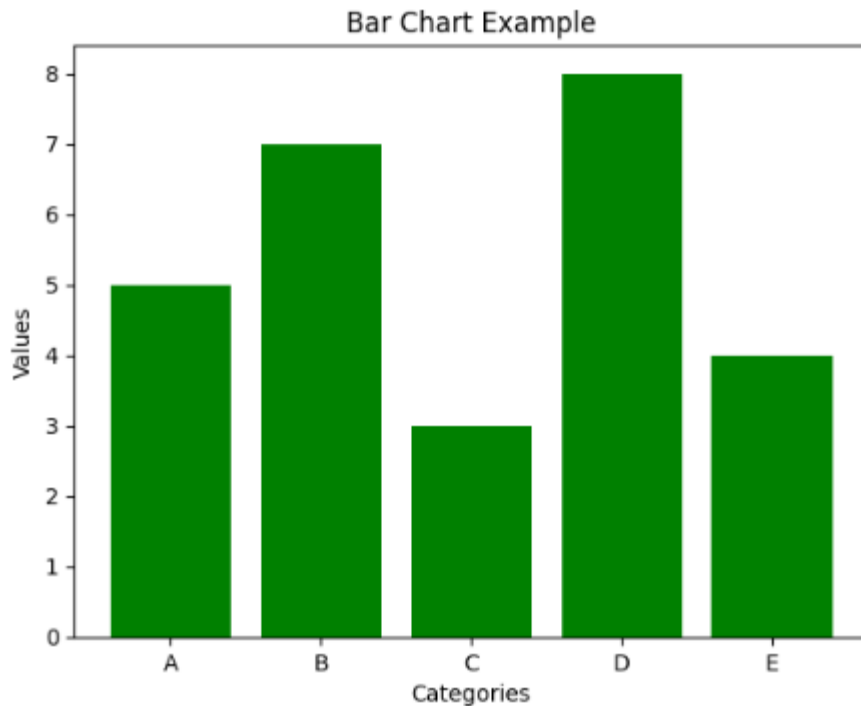
A bar plot uses rectangular bars to compare categories or groups. The **length/height** of each bar represents its value. It is useful for showing differences between discrete categories. Commonly used in **business, surveys, and comparisons**.

CODE SNIPPET :

A code editor window with a dark background. On the left, there is a vertical toolbar with a green checkmark icon and a play button icon. The code is written in a light-colored font. It defines two lists: 'categories' with values 'A', 'B', 'C', 'D', 'E' and 'values' with values 5, 7, 3, 8, 4. Then it uses 'plt.bar' to create a bar chart with 'color='green''. Finally, it sets a title 'Bar Chart Example', x-axis label 'Categories', y-axis label 'Values', and calls 'plt.show()' to display the plot.

```
categories = ['A', 'B', 'C', 'D', 'E']  
values = [5, 7, 3, 8, 4]  
  
plt.bar(categories, values, color='green')  
plt.title("Bar Chart Example")  
plt.xlabel("Categories")  
plt.ylabel("Values")  
plt.show()
```

OUTPUT :



Description:

- `categories` define the x-axis labels.
- `values` represent the bar heights.
- `plt.bar()` creates the bar chart.
- `xlabel()` and `ylabel()` label the axes.
- `show()` displays the chart.

3.Histogram

A histogram shows the **frequency distribution** of a dataset. It groups data into **intervals (bins)** and displays how many values fall into each range. Helps to identify patterns such as skewness, spread, or normal distribution. Commonly used in **statistics and probability analysis**.

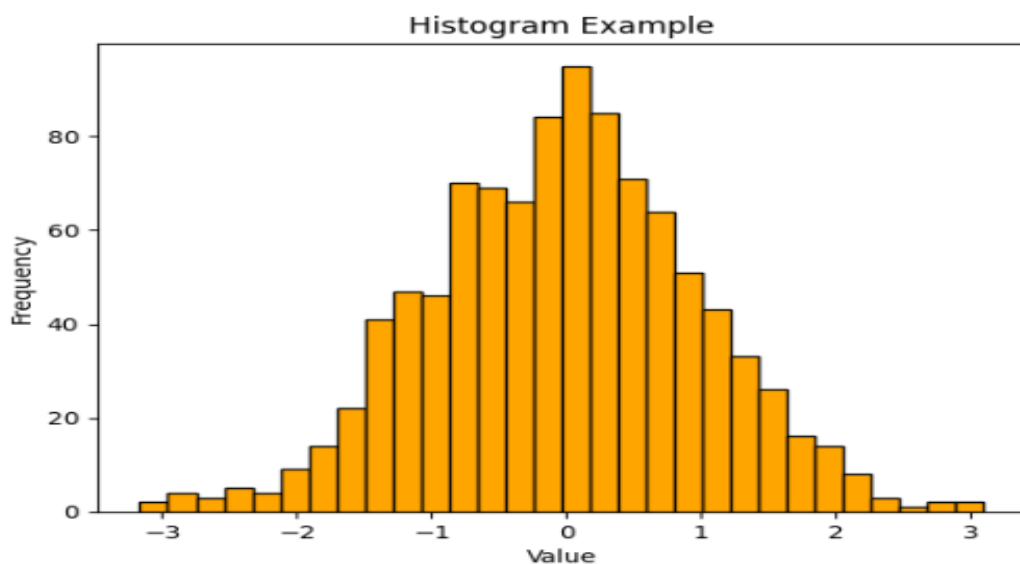
CODE SNIPPET:

```

data = np.random.randn(1000) # 1000 random numbers
plt.hist(data, bins=30, color='orange', edgecolor='black')
plt.title("Histogram Example")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()

```

OUTPUT :



Description:

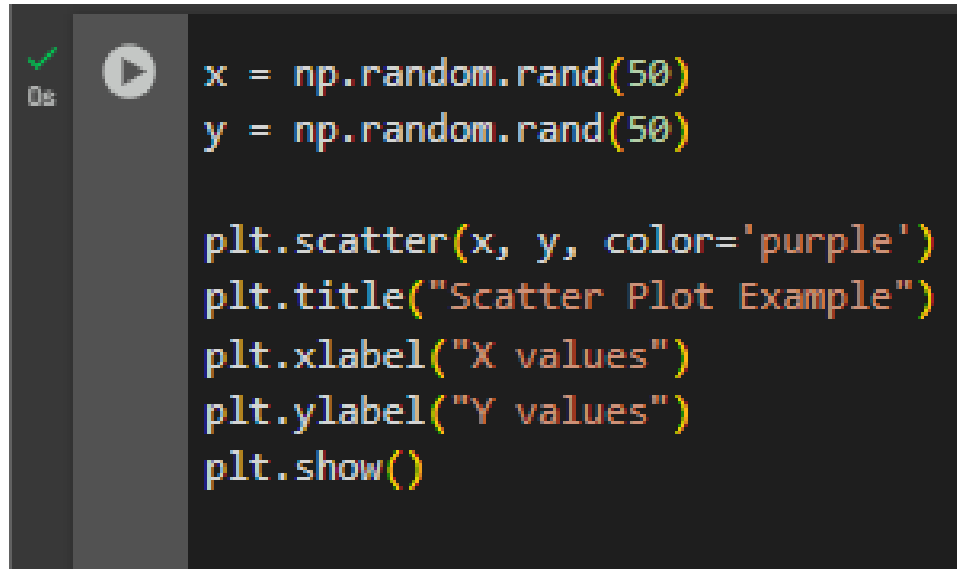
- `np.random.randn(1000)` generates 1000 random values from a normal distribution.
- `plt.hist()` creates the histogram.
- `bins=30` divides the data into 30 intervals.
- `color` sets the bar color and `edgecolor` outlines each bar.
- `show()` displays the histogram.

4.Scatter Plot

A scatter plot represents data as **points on a two-dimensional plane**. It shows the **relationship or correlation** between two variables.

Patterns like positive, negative, or no correlation can be observed.
Widely used in **data analysis and regression modeling**.

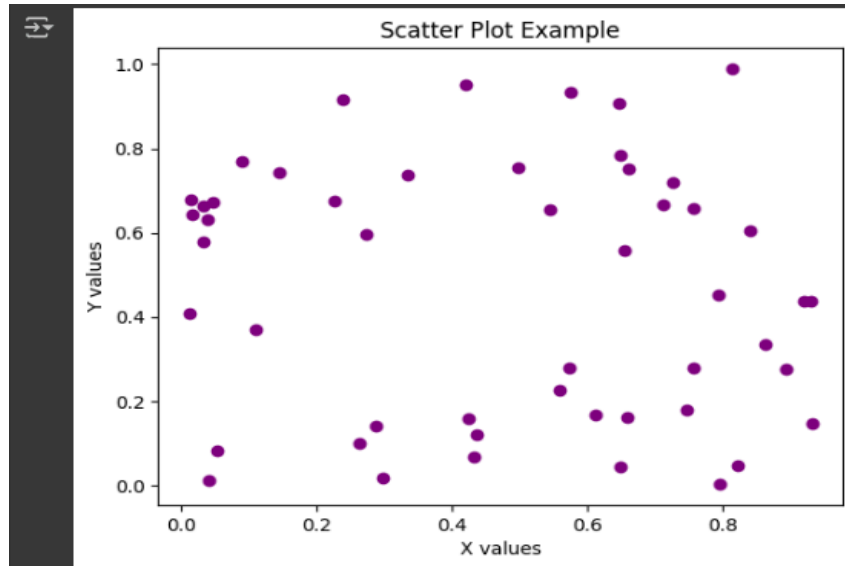
CODE SNIPPET :

A code editor snippet with a dark background. On the left, there is a vertical bar with a green checkmark icon and a play button icon. The code is written in a light-colored font. The code generates two random values, x and y, and plots them as a single purple point on a scatter plot. The plot is titled "Scatter Plot Example" and has axes labeled "X values" and "Y values".

```
x = np.random.rand(50)
y = np.random.rand(50)

plt.scatter(x, y, color='purple')
plt.title("Scatter Plot Example")
plt.xlabel("X values")
plt.ylabel("Y values")
plt.show()
```

OUTPUT :



Description:

- `np.random.rand(50)` generates 50 random values for x and y.
- `plt.scatter()` plots the scatter points.
- `title()`, `xlabel()`, `ylabel()` add graph labels.
- `show()` displays the scatter plot.

5.Pie Chart

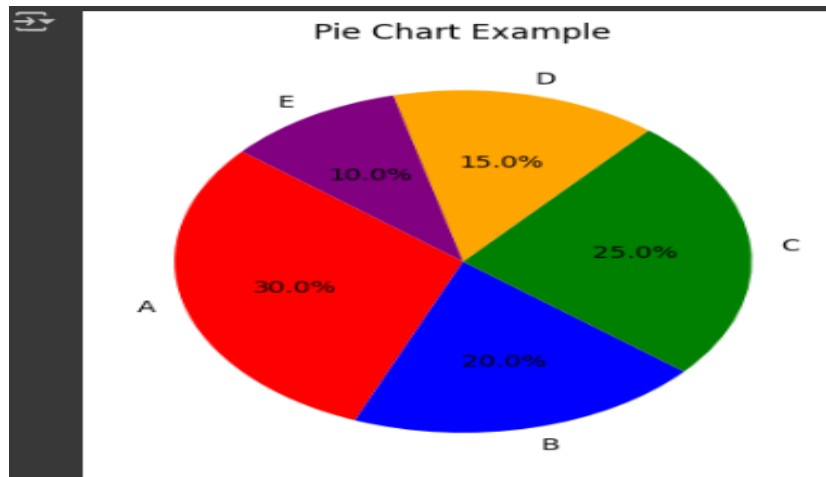
A pie chart shows data as **slices of a circle**, representing proportions of a whole. Each slice's size corresponds to the percentage of that category. It is best for showing **part-to-whole relationships**. Commonly used in **business reports, surveys, and percentage analysis**.

CODE SNIPPET :

```
✓ 0s sizes = [30, 20, 25, 15, 10]
labels = ['A', 'B', 'C', 'D', 'E']
colors = ['red', 'blue', 'green', 'orange', 'purple']

plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)
plt.title("Pie Chart Example")
plt.show()
```

OUTPUT :



Description:

- `sizes` represents the percentage values for each slice.
- `labels` names each slice of the pie.
- `colors` assigns colors to slices.
- `autopct` displays percentage values on the chart.
- `startangle=140` rotates the start position.
- `show()` displays the pie chart.

6.Area Plot

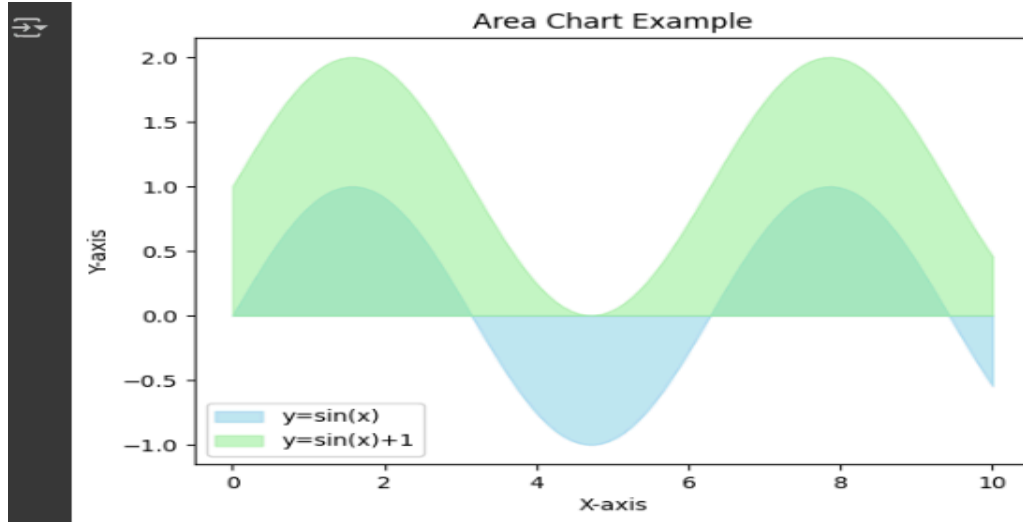
An area plot is similar to a line plot but the area below the line is **filled with color**. It emphasizes the **magnitude of values over time**. Helps in comparing multiple quantities with stacked areas. Commonly used in **cumulative data and trend visualization**.

CODE SNIPPET :

```
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.sin(x) + 1

plt.fill_between(x, y1, color="skyblue", alpha=0.5, label="y=sin(x)")
plt.fill_between(x, y2, color="lightgreen", alpha=0.5, label="y=sin(x)+1")
plt.title("Area Chart Example")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

OUTPUT :



Description:

- `np.linspace(0, 10, 100)` creates 100 values from 0 to 10.
- `y1` and `y2` define the functions to fill under.
- `plt.fill_between()` fills the area between curve and x-axis.
- `alpha=0.5` sets transparency.
- `legend()` shows labels of both filled areas.
- `show()` displays the chart.

2.SEABORN

Introduction to Seaborn

Seaborn is a **Python data visualization library** built on top of Matplotlib. It provides a **high-level interface** for creating attractive and informative statistical graphics with minimal code. Seaborn comes with **built-in themes, color palettes, and dataset support**, making plots more visually appealing compared to plain Matplotlib. It is especially powerful for **statistical data visualization**, such as showing distributions, relationships, and categorical comparisons.

Key Features of Seaborn

- **Beautiful Styles:** Comes with built-in themes and color palettes for clean and elegant plots.
- **Statistical Support:** Easily creates advanced plots like box plots, violin plots, swarm plots, and pair plots.
- **Integration:** Works seamlessly with Pandas DataFrames for handling tabular data.
- **Simplified Syntax:** Requires fewer lines of code than Matplotlib for complex visualizations.
- **Built-in Datasets:** Provides sample datasets (like *tips*, *iris*, *flights*) for practice.

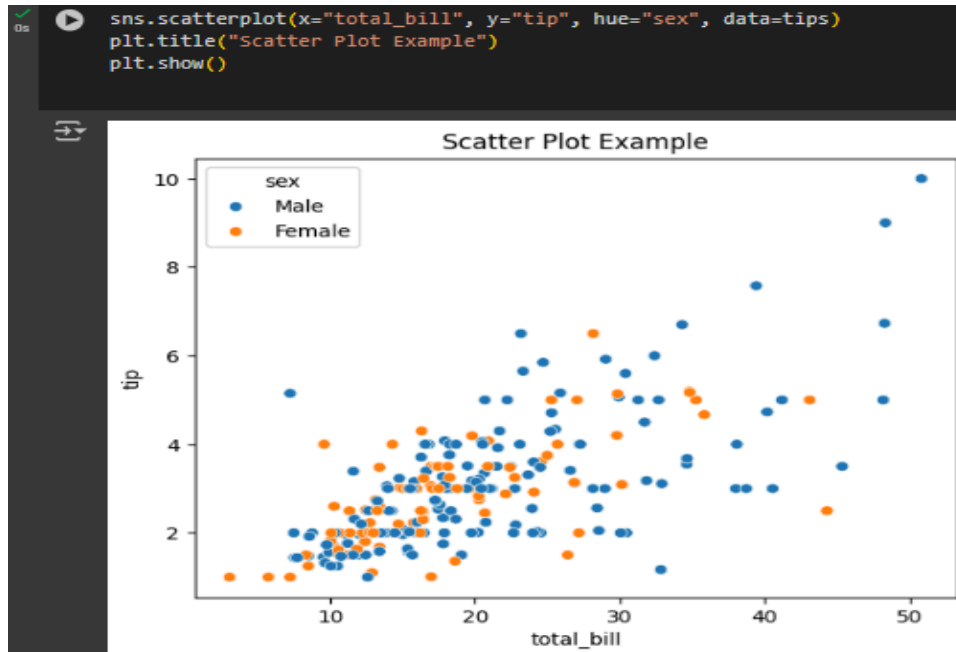
Why Seaborn is Popular among Data Scientists

Seaborn is widely used because it allows data scientists to quickly create **professional and publication-ready plots**. It reduces coding effort, handles complex statistical visualizations, and integrates well with Python's data analysis libraries like **Pandas and NumPy**. With its elegant design and simplicity, Seaborn is a go-to choice for **exploratory data analysis (EDA)** and reporting.

1. Scatter Plot

Scatter plots use points to show the relationship between two numeric variables. Each point represents a value in the dataset, and the parameter **hue** can be used to differentiate categories with colors. They are helpful in identifying patterns, clusters, or outliers in data.

CODE SNIPPET AND OUTPUT :



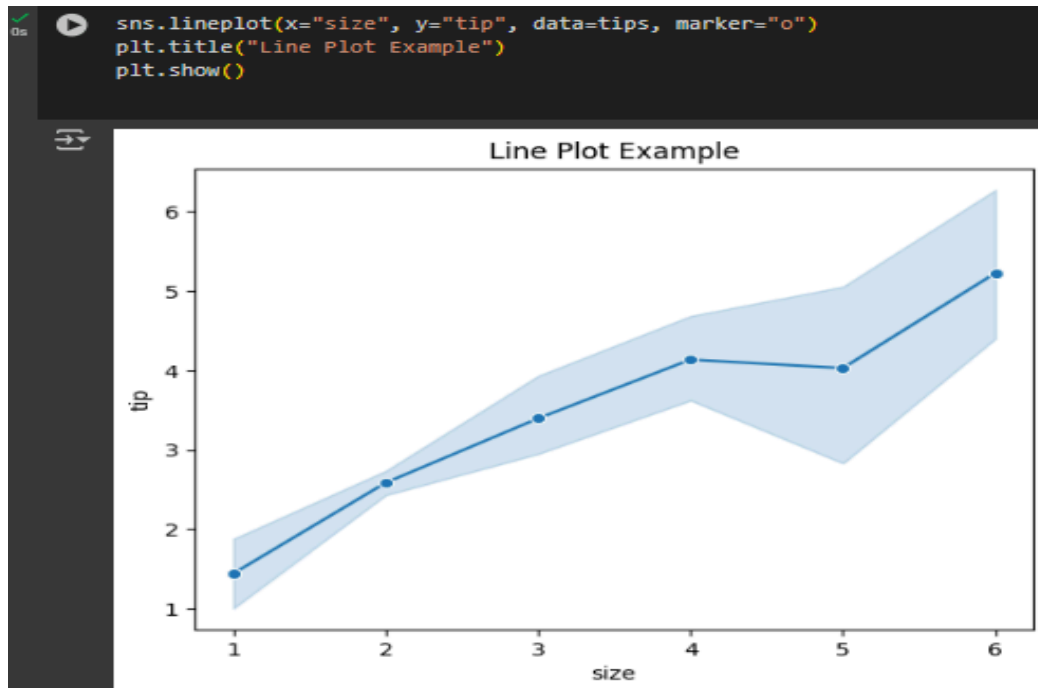
Description:

- `sns.scatterplot()` creates a scatter plot.
- `x="total_bill"` and `y="tip"` map dataset columns.
- `hue="sex"` colors the points by gender.
- `data=tips` specifies the dataset.
- `show()` displays the plot.

2.Line Plot

Line plots display data as a series of points connected by lines. They are used to observe trends and changes over a continuous variable, such as time or size. Markers can be added for better visualization of each point.

CODE SNIPPET AND OUTPUT :



Description:

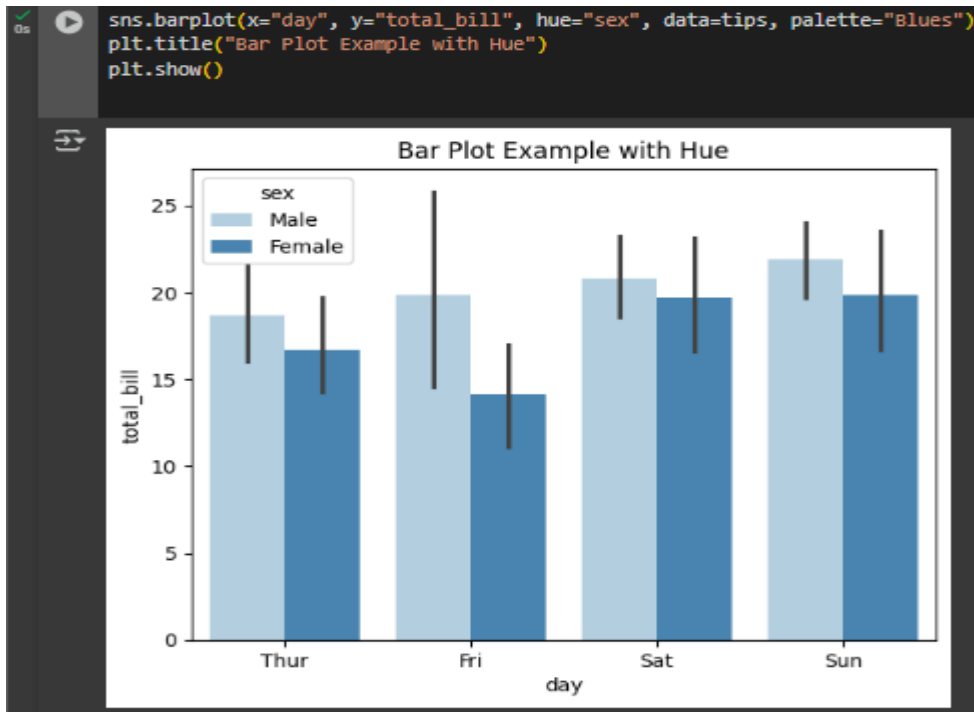
- `sns.lineplot()` draws a line graph.
- `x="size"` and `y="tip"` plot tip amounts against group size.
- `marker="o"`

3. Bar Plot

Bar plots represent the average values of a numerical variable for each category. They use rectangular bars to show comparisons between groups. Seaborn automatically calculates and plots the mean with confidence intervals

CODE SNIPPET AND OUTPUT :

.



Description:

- `sns.barplot()` creates bars showing average values.
- `x="day"` places days on x-axis.
- `y="total_bill"` calculates average total bill for each day.
- `palette="Blues"` applies a blue color scheme.
- `show()` displays the bar plot.

4. Count Plot

Count plots show the frequency of categories in a dataset. Each bar represents how many times a category appears, and the `hue` parameter splits the bars into subcategories for comparison.

CODE SNIPPET AND OUTPUT :



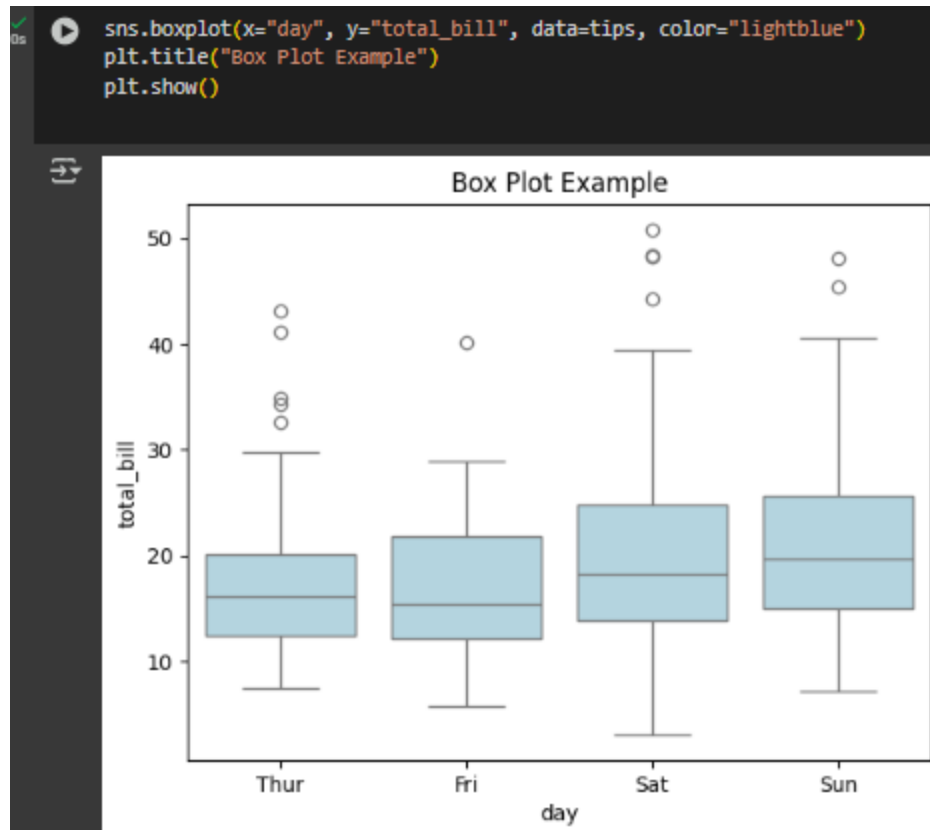
Description:

- `sns.countplot()` counts the number of occurrences of each category.
- `x="day"` shows days of the week.
- `hue="sex"` splits bars into male/female groups.
- `palette="Set1"` sets the color theme.
- `show()` displays the plot.

5. Box Plot

Box plots summarize the distribution of data through quartiles. The box shows the interquartile range (middle 50%), while whiskers represent variability. Outliers appear as individual points beyond the whiskers.

CODE SNIPPET AND OUTPUT :



Description:

- `sns.boxplot()` creates a box plot to show quartiles and outliers.
- `x="day"` groups data by days.
- `y="total_bill"` shows bill distribution.
- `palette="pastel"` gives soft colors.
- `show()` displays the chart.

6. KDE Plot

Kernel Density Estimation (KDE) plots show the probability distribution of a continuous variable as a smooth curve. Shading can be added to highlight the area under the curve. KDE plots are useful for visualizing data distributions more smoothly than histograms.

CODE SNIPPET AND OUTPUT :



Description:

- `sns.kdeplot()` generates the KDE curve.
- `x="total_bill"` plots the distribution of total bills.
- `fill=True` shades the area under the curve.
- `color="purple"` changes the curve color.
- `show()` displays the plot.

7. Heatmap

Heatmaps display numerical data as a matrix of colors. Darker or lighter shades indicate lower or higher values. Heatmaps are commonly used to represent correlation between variables or patterns across matrices.

CODE SNIPPET AND OUTPUT :



Description:

- `tips.corr()` creates a correlation matrix.
- `sns.heatmap()` visualizes the matrix.
- `annot=True` displays numbers inside cells.
- `cmap="coolwarm"` applies color gradient.
- `show()` displays the heatmap.

3.Comparison of Matplotlib and Seaborn

Matplotlib is the core Python visualization library that provides fine-grained control over plots. It allows you to customize every element of a chart — from axes, ticks, and labels to colors and styles. It is very flexible and supports a wide range of chart types,

including 2D and some 3D visualizations. However, its syntax can be verbose, and creating complex visualizations often requires multiple lines of code.

Seaborn, on the other hand, is built on top of Matplotlib and provides a high-level interface for statistical and categorical data visualization. It comes with beautiful default styles, color palettes, and simplified functions for creating common statistical plots like box plots, violin plots, pair plots, and regression plots. Seaborn is less flexible than Matplotlib in terms of full customization, but it is much easier to use for quick and attractive visualizations.

✓ Advantages of Matplotlib

- Highly flexible and customizable for any type of plot.
- Wide range of supported chart types, including 3D plots.
- Suitable for creating **publication-quality** figures.
- Strong community support and extensive documentation.

✓ Advantages of Seaborn

- High-level syntax makes plotting faster and easier.
- Attractive default themes and color palettes.
- Built-in support for **statistical plots** (e.g., regression, distribution, categorical plots).
- Seamless integration with **Pandas DataFrames** for direct plotting.
- Great for **exploratory data analysis (EDA)** and quick insights.

4. Conclusion

In conclusion, Matplotlib and Seaborn together provide a complete toolkit for data visualization in Python. Matplotlib offers flexibility and control for creating highly customized and publication-ready graphics, while Seaborn makes it simple to generate beautiful and informative

statistical plots with minimal effort. By learning both libraries, data scientists can handle quick exploratory analysis as well as detailed professional reporting, making data visualization more effective and impactful.

5.Resources / References

- 1.Matplotlib Documentation –** <https://matplotlib.org/stable/>
- 2.Seaborn Documentation –** <https://seaborn.pydata.org/>
- 3.Python Data Science Handbook (Jake VanderPlas, O'Reilly, 2016)**
- 4.W3Schools – Python Visualization –**
https://www.w3schools.com/python/matplotlib_intro.asp
- 5.GeeksforGeeks – Data Visualization in Python –**
<https://www.geeksforgeeks.org/python-data-visualization/>
- 6.Kaggle Datasets for Practice –** <https://www.kaggle.com/datasets>
- 7.Analytics Vidhya – Data Visualization Guide –**
<https://www.analyticsvidhya.com/blog/category/data-visualization/>