Visualization Libraries

1. Visualization Overview

Data visualization is a fundamental component of data science and analytics. It transforms raw numerical and categorical data into visual representations such as line plots, bar charts, histograms, scatter plots, heatmaps, and pie charts. Visualization helps analysts, researchers, and decision-makers to identify patterns, detect outliers, compare categories, and understand relationships between variables more effectively than raw tabular data.

Among Python's powerful visualization libraries, Matplotlib and Seaborn are widely used. While Matplotlib provides low-level control and customization, Seaborn simplifies statistical plotting and produces professional-quality visuals with less effort.

2. Matplotlib

2.1 Introduction

Matplotlib, introduced in 2002 by John D. Hunter, is the most fundamental visualization library in Python. It is designed to work seamlessly with NumPy arrays and Pandas DataFrames, and it forms the basis of many advanced visualization tools. Matplotlib visualizations are built around four main elements:

Figure: The entire plotting area, which can contain one or multiple subplots.

Axes: The actual plots or subplots within the figure, where the data is drawn.

Axis: The scales (X-axis and Y-axis) with ticks and labels.

Artists: All visual elements on the figure (lines, text, legends, shapes, etc.).

1. Figure

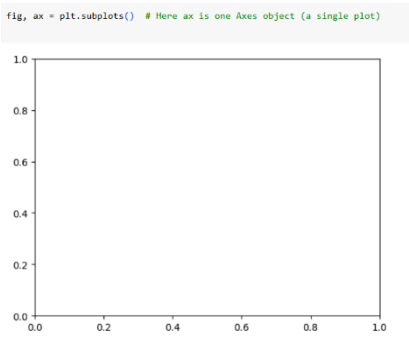
- Think of it as the **entire canvas or window** where everything is drawn.
- It can hold one or more **Axes (plots/subplots)**.
- Example: If you create a figure with 2x2 subplots, the figure is the big rectangle that holds all four plots.

```
import matplotlib.pyplot as plt

fig = plt.figure()  # This creates a blank figure (canvas)
```

🔹 2. Axes

- **Axes ≠ Axis.**
- Axes is basically **one plot (or subplot)** inside the figure.
- It contains:
  - X-axis
  - Y-axis
  - Data drawn (lines, bars, etc.)
- A figure can have multiple axes.

```
fig, ax = plt.subplots()  # Here ax is one Axes object (a single plot)

If you do fig, axs = plt.subplots(2,2) → you get 4 Axes objects inside
```



🔹 3. Axis

- Inside each **Axes**, you have **two Axis objects**:
  - **X-axis**
  - **Y-axis**
- They manage:
  - Ticks (numbers shown on the scale, e.g., 0, 1, 2…)

- Labels (like "Time (s)" or "Price ($)")

- Basically, they define the **scaling system** of the plot.

```
ax.set_xlabel("Time (s)")  # X-Axis
ax.set_ylabel("Value")     # Y-Axis
```

🔹 **4. Artists**

- These are the **actual things you see drawn** on the plot.
- Everything in Matplotlib is an Artist:
  - Lines (Line2D)
  - Text (titles, labels)
  - Shapes (rectangles, circles, patches)
  - Legends, grids, ticks
- Two types:
  - **Primitive artists** (basic objects like Line, Text)
  - **Composite artists** (groups like Axes, Figure, Legends that contain other artists)

Example:

```
ax.plot([1,2,3], [4,5,6])   # Line2D artist
ax.set_title("My Plot")     # Text artist
ax.legend(["Line"])         # Legend artist
```

2.2 Pyplot

- **pyplot** is the **main sub-library** of Matplotlib.
- It provides a **collection of functions** that make plotting **easy and MATLAB-like** (that's why it is called "py-**plot**").

- Most people import it like this:

import matplotlib.pyplot as plt

🔹 Key Features of pyplot

1. **Simplifies plotting**
    - You don’t need to create Figure/Axes manually every time.
    - Functions like plot(), bar(), hist(), scatter() handle everything behind the scenes.
2. **MATLAB-style interface**
    - Commands work sequentially (state-based), similar to MATLAB.
    - Example: You call plt.plot() → it knows which Figure and Axes to draw on.
3. **Quick visualization**
    - Useful for beginners and quick analysis.

🔹 Commonly Used pyplot Functions

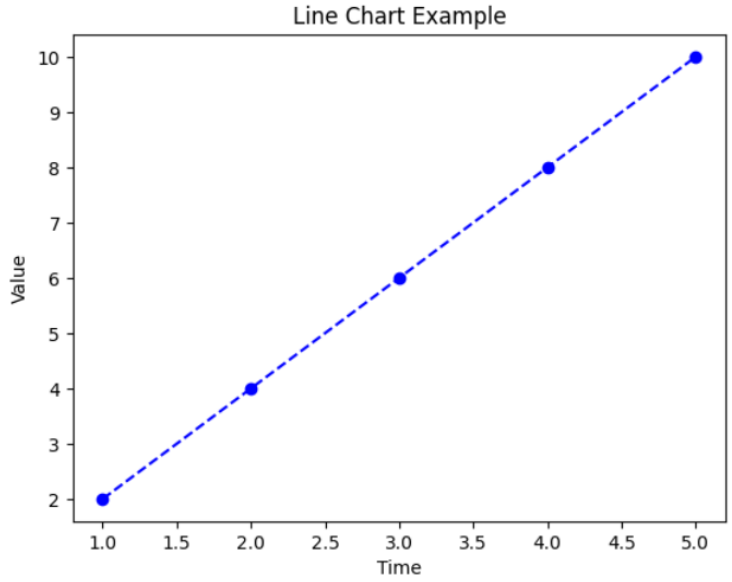
- plt.plot(x, y) → Line plot
- plt.bar(x, height) → Bar chart
- plt.hist(data) → Histogram
- plt.scatter(x, y) → Scatter plot
- plt.title("Title"), plt.xlabel("X-axis"), plt.ylabel("Y-axis") → Add labels
- plt.show() → Display the figure

2.3 Graph Types in Matplotlib

(a) Line Chart

- A line chart is one of the most commonly used graphs to **display data over a continuous interval or time period**.
- It uses points connected by straight lines to show **trends, progress, or changes**.
- Markers can be added at each point for clarity, and styles (dashed, dotted) can highlight patterns.
- It is especially useful for showing **upward or downward trends**.
- **Use Case:** Tracking stock prices, temperature changes, or population growth over years.

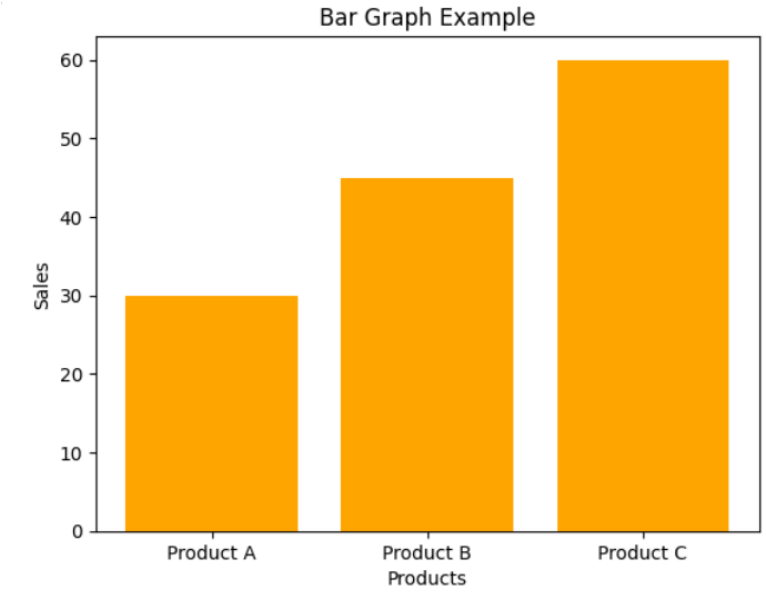
```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.plot(x, y, marker='o', linestyle='--', color='blue')
plt.title("Line Chart Example")
plt.xlabel("Time")
plt.ylabel("Value")
plt.show()
```



(b) Bar Graph

- A bar graph is used to **compare data across different categories**.
- The length or height of each bar represents the value of the category.
- It can be **vertical or horizontal** and can also be grouped or stacked for detailed comparisons.
- Bar graphs make it easy to compare **discrete items** like sales, votes, or quantities.
- **Use Case:** Comparing product sales, survey responses, or website traffic per month.

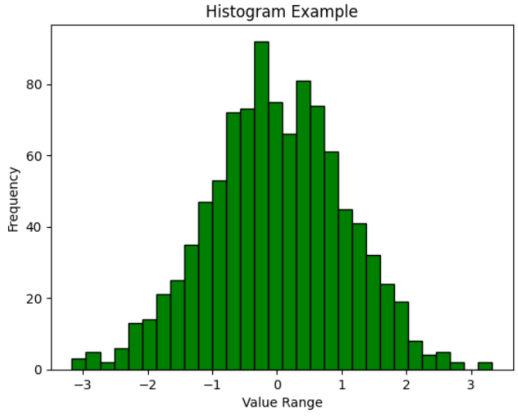
```
categories = ['Product A', 'Product B', 'Product C']
values = [30, 45, 60]
plt.bar(categories, values, color='orange')
plt.title("Bar Graph Example")
plt.xlabel("Products")
plt.ylabel("Sales")
plt.show()
```



(c) Histogram

- A histogram shows the **frequency distribution of continuous data**.
- Data is grouped into **intervals called bins**, and the height of each bar indicates how many values fall in that range.
- It helps visualize the **shape of the data distribution** (normal, skewed, etc.).
- Histograms are often used in **statistics and data analysis** to understand variability.
- **Use Case:** Distribution of student exam scores, ages, or incomes.

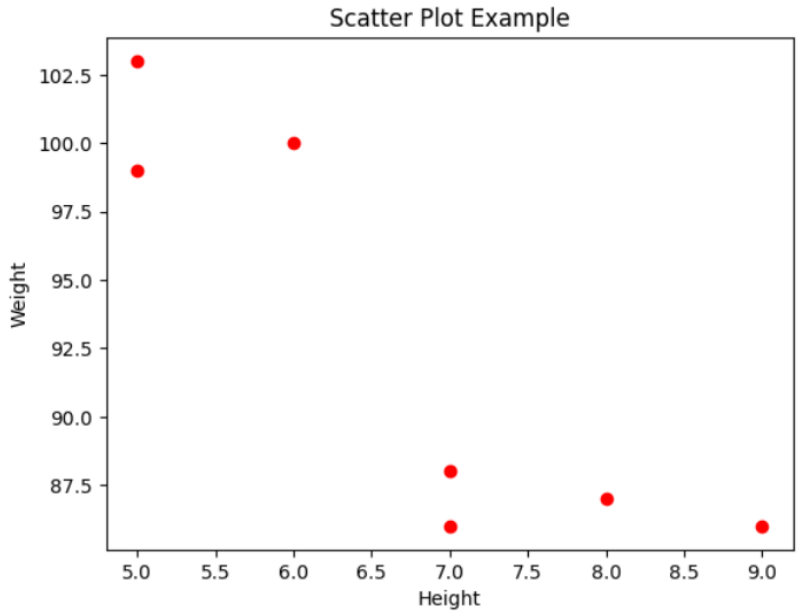
```
import numpy as np
data = np.random.randn(1000) # 1000 values from normal distribution
plt.hist(data, bins=30, color='green', edgecolor='black')
plt.title("Histogram Example")
plt.xlabel("Value Range")
plt.ylabel("Frequency")
plt.show()
```



(d) Scatter Plot

- A scatter plot displays the **relationship (correlation) between two variables** using dots.
- Each point represents one observation with values along the X-axis and Y-axis.
- It helps in identifying **patterns, clusters, or outliers** in the dataset.
- Scatter plots are often used in **regression analysis and machine learning** to check dependencies.
- **Use Case:** Height vs Weight, Advertising spend vs Sales revenue.

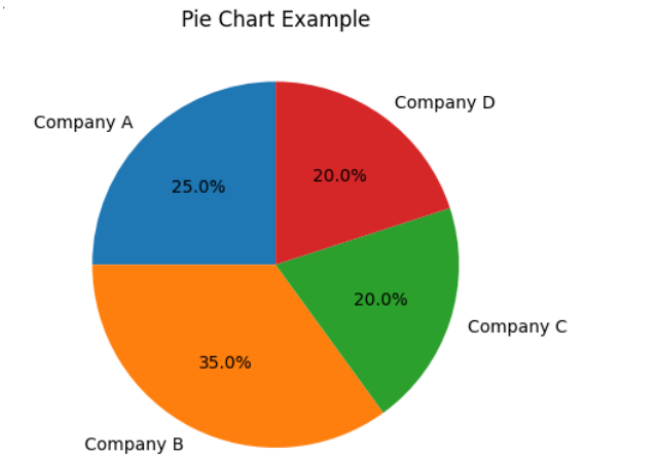
```
x = [5, 7, 8, 7, 6, 9, 5]
y = [99, 86, 87, 88, 100, 86, 103]
plt.scatter(x, y, color='red')
plt.title("Scatter Plot Example")
plt.xlabel("Height")
plt.ylabel("Weight")
plt.show()
```



(e) Pie Chart

- A pie chart represents data as **proportions of a whole**, divided into slices.
- Each slice corresponds to a category, and the angle of the slice shows its relative size.
- Percentages are often displayed inside or outside the slices for clarity.
- While effective for showing proportions, pie charts are **less useful when there are many categories**.
- **Use Case:** Market share of companies, budget allocation, or population by regions.

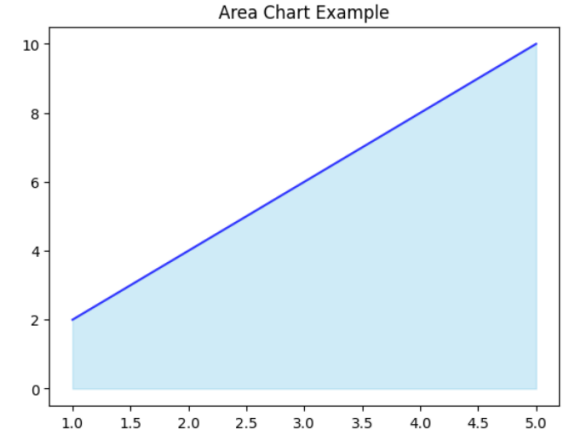
```
sizes = [25, 35, 20, 20]
labels = ["Company A", "Company B", "Company C", "Company D"]
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title("Pie Chart Example")
plt.show()
```



(f) Area Chart

- An area chart is like a **line chart with the area below the line filled with color**.
- It emphasizes the **magnitude of change** over time as well as the trend.
- It can also be used to show **cumulative data** when multiple lines are stacked.
- The shaded region helps in understanding the overall impact and scale of growth.
- **Use Case:** Visualizing population growth, rainfall over months, or revenue growth.

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 6, 8, 10]
plt.fill_between(x, y, color="skyblue", alpha=0.4)
plt.plot(x, y, color="blue", alpha=0.8)
plt.title("Area Chart Example")
plt.show()
```



3. Seaborn

3.1 Overview

- **Seaborn** is a **Python data visualization library** built on top of **Matplotlib**, created to make statistical graphics simpler and more attractive.
- Unlike Matplotlib, which requires more customization for professional-looking plots, Seaborn provides **beautiful default themes** and **easy-to-use functions** for common statistical visualizations.
- One of its biggest strengths is that it **works seamlessly with Pandas DataFrames**. You can directly pass column names instead of manually extracting data, which makes code shorter and cleaner.
- It is particularly good for **data exploration and statistical analysis**, where relationships, categories, and distributions need to be clearly represented.

Strengths of Seaborn:

1. **Built on Matplotlib** → Uses Matplotlib as its foundation but provides a higher-level interface.
2. **Simplifies statistical visualizations** → Handles complex plots (like violin plots, heatmaps, and regression plots) in just one function.
3. **Beautiful themes by default** → Comes with attractive styles and color palettes (like Set1, coolwarm, viridis).
4. **Integration with Pandas** → Can directly use Pandas DataFrames, making it great for data science tasks.

Categories of Plots in Seaborn

1. **Relational** → Focus on the relationship between two variables (e.g., scatterplot, lineplot).
2. **Categorical** → Compare categories of data (e.g., barplot, countplot, boxplot, violinplot).
3. **Distribution** → Show the distribution of data (e.g., histplot, kdeplot, rugplot).

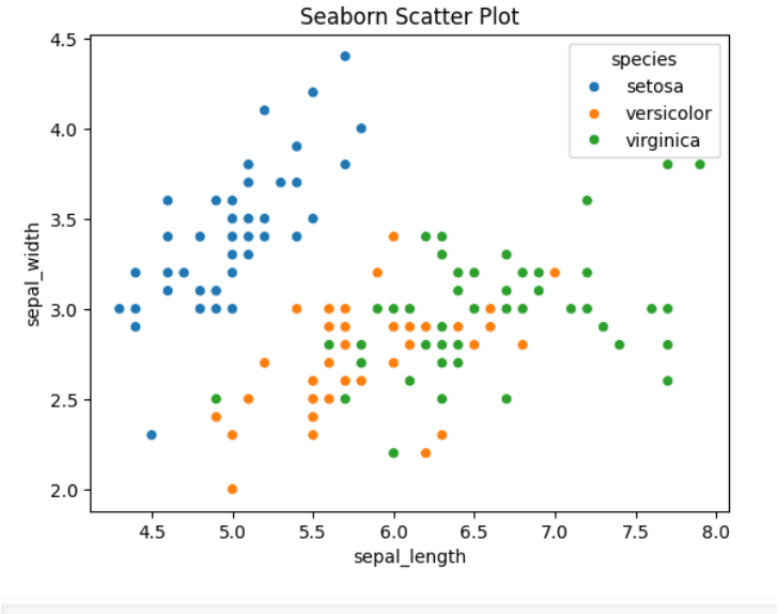
4. **Regression** → Display regression relationships (e.g., regplot, lmplot).
5. **Matrix** → Work with 2D matrix data (e.g., heatmap, clustermap).

3.2 Types of Graphs in Seaborn

(a) Scatter Plot

- A scatter plot is used to visualize the relationship between two numerical variables.
- In Seaborn, the hue parameter allows grouping data by categories and coloring them differently.
- This makes it easier to see patterns, clusters, or separations between groups in the dataset.
- Scatter plots are widely used in correlation analysis and classification problems.
- Example Use Case: Comparing petal length vs petal width in the Iris dataset, grouped by species.

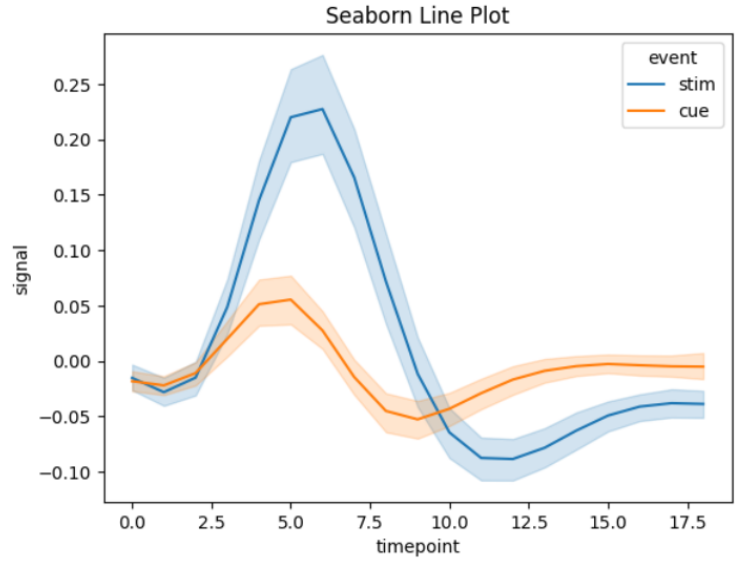
```
import seaborn as sns
import matplotlib.pyplot as plt
iris = sns.load_dataset("iris")
sns.scatterplot(x="sepal_length", y="sepal_width", hue="species", data=iris)
plt.title("Seaborn Scatter Plot")
plt.show()
```



(b) Line Plot

- A line plot is useful for **showing trends of a variable over time or another continuous axis**.
- Seaborn's lineplot() automatically adds confidence intervals, making it better than Matplotlib's basic line plot for statistical analysis.
- It can handle multiple groups using the hue parameter, which assigns different colors to categories.
- Line plots are ideal for **time-series data, experimental results, or signals**.
- **Example Use Case:** Visualizing how brain signal strength changes across different events in an fmri dataset.

```
fmri = sns.load_dataset("fmri")
sns.lineplot(x="timepoint", y="signal", data=fmri, hue="event")
plt.title("Seaborn Line Plot")
plt.show()
```



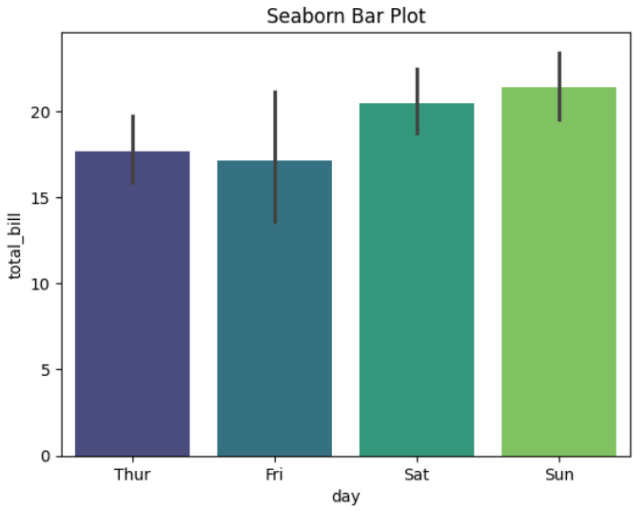
(c) Bar Plot

- A bar plot is used to **compare the average values of categories**.
- Unlike Matplotlib's bar(), Seaborn's barplot() automatically computes **mean and confidence intervals** for each category.
- You can also apply different color palettes to make it visually appealing.

- Bar plots are useful for **comparing groups, categories, or conditions**.
- **Example Use Case:** Showing the average restaurant bill on different days using the tips dataset.

```
tips = sns.load_dataset("tips")
sns.barplot(x="day", y="total_bill", data=tips, palette="viridis")
plt.title("Seaborn Bar Plot")
plt.show()
```

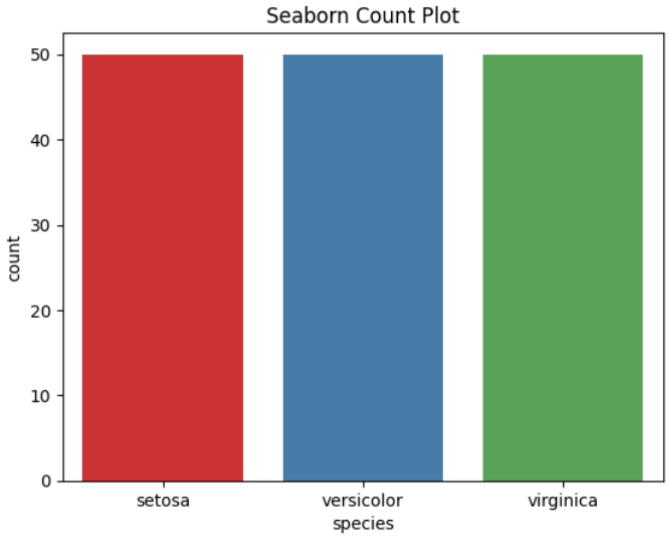
```
/tmp/ipython-input-3944205399.py:2: FutureWarning:

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

  sns.barplot(x="day", y="total_bill", data=tips, palette="viridis")
```



(d) Count Plot

```
sns.countplot(x="species", data=iris, palette="Set1")
plt.title("Seaborn Count Plot")
plt.show()
```

```
/tmp/ipython-input-597268860.py:1: FutureWarning:

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

  sns.countplot(x="species", data=iris, palette="Set1")
```

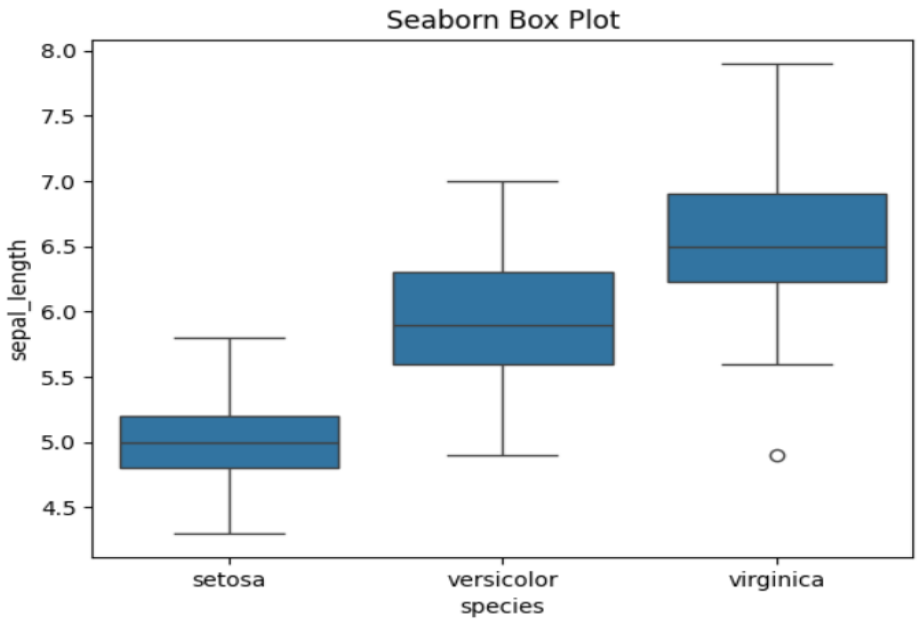


- A count plot shows the **frequency of categories** as bars.
- Unlike barplot(), it doesn't compute averages — it just **counts how many times each category appears**.
- It's very useful for categorical data exploration.
- Count plots can also use hue to show sub-categories inside each bar.
- **Example Use Case:** Counting the number of iris flowers of each species in the dataset.

(e) Box Plot

- A box plot (or whisker plot) is used to **show the distribution of data through quartiles**.
- It highlights the **median, upper/lower quartiles, and outliers**.
- Box plots are excellent for comparing **distributions between categories**.
- They are commonly used in **statistical analysis to detect variability and outliers**.
- **Example Use Case:** Comparing sepal length distribution across iris species.

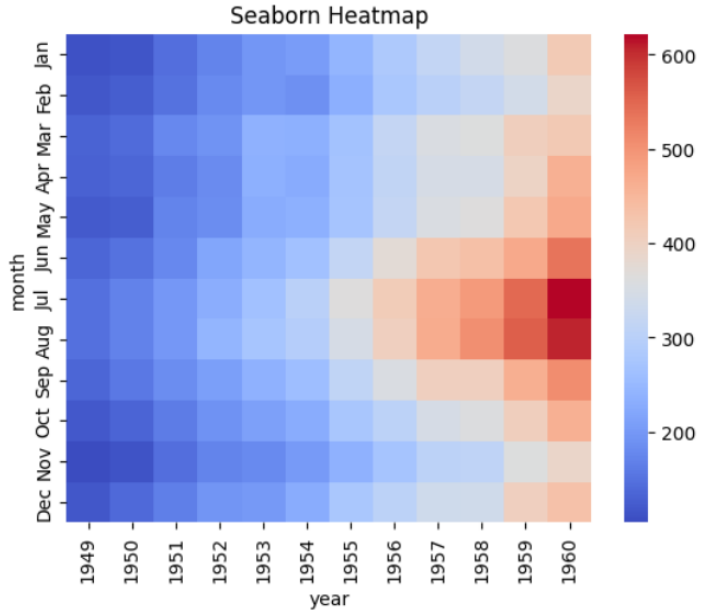
```
sns.boxplot(x="species", y="sepal_length", data=iris)
plt.title("Seaborn Box Plot")
plt.show()
```



(f) Heatmap

- A heatmap is used to **visualize matrix-like data in the form of colors**.
- Each cell is color-coded according to its value, making it easy to see **patterns and trends**.
- Heatmaps are widely used in **correlation analysis, financial data, and time-series comparisons**.
- With options like annot=True, values can be displayed inside the cells for clarity.
- **Example Use Case:** Visualizing the number of passengers over years and months in the flights dataset.

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

flights = sns.load_dataset("flights").pivot(index="month", columns="year", values="passengers")
sns.heatmap(flights, cmap="coolwarm")
plt.title("Seaborn Heatmap")
plt.show()
```

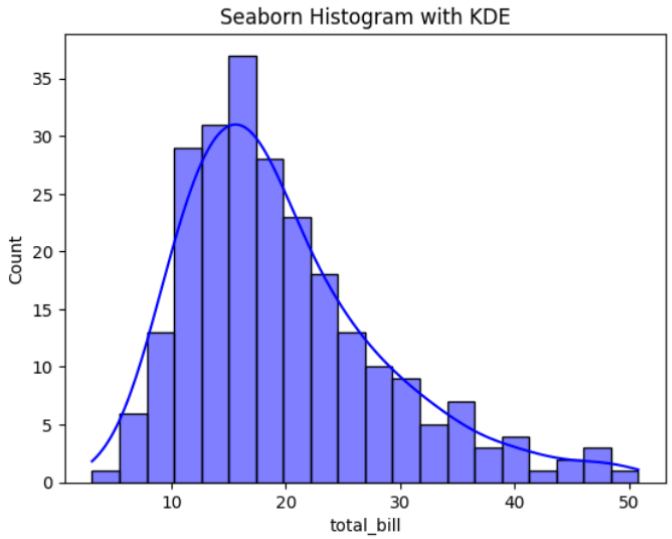


Additional Seaborn Graph Types

(g) Histogram Plot (histplot)

- A Histogram Plot is used to visualize the distribution of a single variable by dividing it into bins and showing how frequently values fall into each bin.
- Unlike Matplotlib's hist(), Seaborn's histplot() comes with better styling, flexibility, and the option to add Kernel Density Estimation (KDE) overlay.
- It is useful for identifying data patterns such as skewness, normal distribution, or concentration of values.
- Histograms are widely used in statistics, probability analysis, and data exploration.
- Use Case: Analyzing the distribution of restaurant bills or student exam scores.

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

tips = sns.load_dataset("tips")
sns.histplot(tips["total_bill"], bins=20, kde=True, color="blue")
plt.title("Seaborn Histogram with KDE")
plt.show()
```



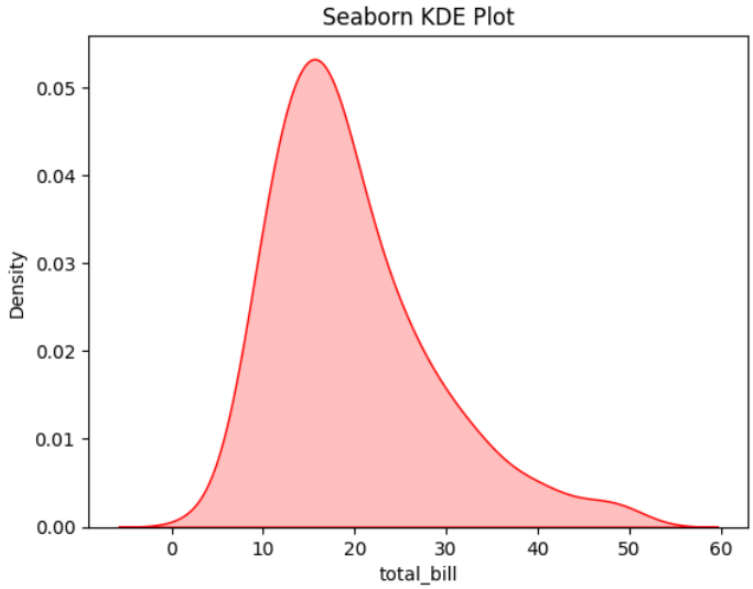
(h) KDE Plot (kdeplot)

- A Kernel Density Estimation (KDE) Plot is a smoothed version of the histogram.
- Instead of showing counts in bins, it estimates the probability density function of a continuous variable.
- It helps in understanding the shape of the distribution without the blocky appearance of histograms.

- KDE plots are great for comparing two or more distributions on the same axis.
- Use Case: Understanding the distribution of tips given in restaurants compared across weekdays.

```
sns.kdeplot(data=tips["total_bill"], shade=True, color="red")
plt.title("Seaborn KDE Plot")
plt.show()
```

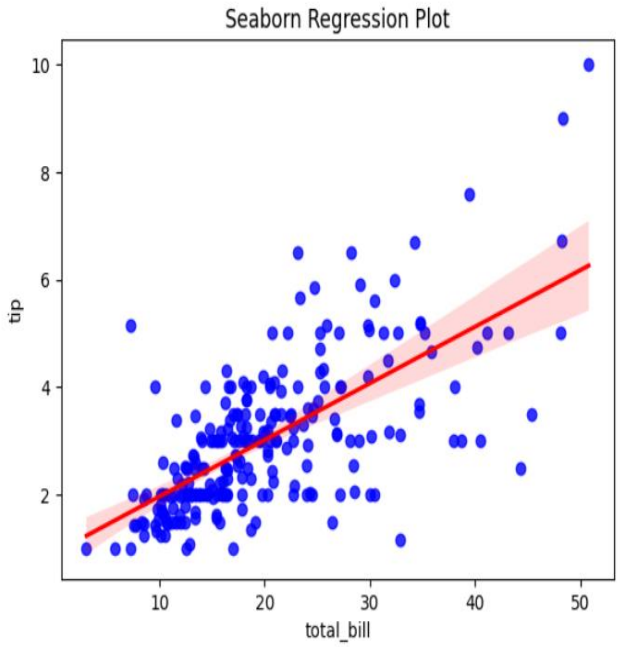
```
/tmp/ipython-input-2483287523.py:1: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  sns.kdeplot(data=tips["total_bill"], shade=True, color="red")
```



(i) Regression Plot (regplot)

- A Regression Plot shows the relationship between two variables along with a regression line.
- It not only plots the scatter points but also fits a linear regression model by default.
- Confidence intervals are automatically shown, making it useful for statistical analysis.
- It is often used in predictive modeling and correlation studies.
- Use Case: Exploring the relationship between restaurant bills (total_bill) and tips (tip).

```
sns.regplot(x="total_bill", y="tip", data=tips, scatter_kws={"color":"blue"}, line_kws={"color":"red"})
plt.title("Seaborn Regression Plot")
plt.show()
```



4.Matplotlib vs Seaborn

4.1. Introduction

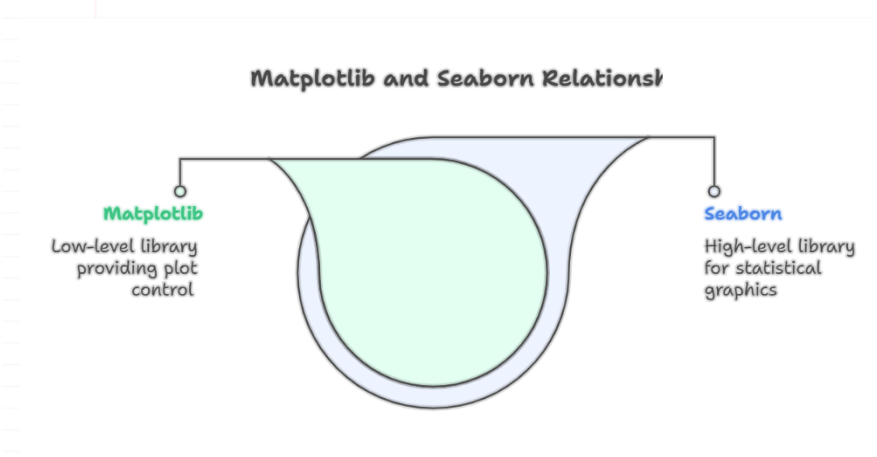
- **Matplotlib** (introduced in 2002 by John Hunter) is the **fundamental plotting library** in Python.
  It provides low-level control over every aspect of a figure (size, color, axes, labels, etc.).
  Almost every other Python visualization library (including Seaborn) is built on top of it.

- **Seaborn** (introduced in 2014 by Michael Waskom) is a **high-level interface built on Matplotlib**.
  It is designed to simplify **statistical visualizations** and comes with **beautiful default themes** and **easy integration with Pandas DataFrames**.

4.2. Detailed Comparison

Feature	Matplotlib	Seaborn
Level	Low-level library (gives complete control over plots).	High-level library (built on Matplotlib).
Purpose	General-purpose plotting (line, bar, scatter, etc.).	Specializes in **statistical visualizations** (boxplot, violin, heatmap, regplot).
Ease of Use	Requires more code for customization (titles, labels, grids).	Shorter, cleaner syntax with built-in defaults.
Style/Theme	Default style is very basic (needs manual styling).	Comes with **attractive themes** (darkgrid, whitegrid, ticks, etc.).
Integration with Pandas	Can plot Pandas data, but requires manual passing of arrays.	Works directly with Pandas DataFrames (hue, col, row parameters).
Customization	Extremely customizable (every detail can be modified).	Limited customization (falls back to Matplotlib if deep customization is needed).
Statistical Support	Does not directly support statistical analysis.	Designed for statistics: confidence intervals, regression lines, distribution plots.
Performance	Faster for simple plots (line, scatter, bar).	Slightly slower (adds extra processing for themes & stats).
Learning Curve	Steeper (requires understanding of Figures, Axes, Artists).	Easier for beginners (higher-level abstraction).
Community Support	Large, widely used (basis for all Python viz).	Growing but relies on Matplotlib community too.

Feature	Matplotlib	Seaborn
Examples	Line chart of stock prices, bar chart of sales.	Heatmap of correlations, regression analysis, violin plot of distributions.



4.3. Advantages & Disadvantages of Matplotlib vs Seaborn

Feature	Matplotlib	Seaborn
Advantages	- Very powerful and flexible (can create any kind of 2D/3D plot). - Gives low-level control over every plot element (ticks, axes, colors, labels). - Works as the foundation for other libraries (Seaborn, Pandas plotting). - Supports complex customizations and animations. - Large community and extensive documentation.	- Built on top of Matplotlib, easier to use for statistical plots. - Provides high-level functions (barplot, violinplot, pairplot, heatmap). - Better default styles and color palettes (aesthetic visuals without much code). - Great for **statistical data visualization** (KDE, regression, categorical plots). - Integrates easily with Pandas DataFrames.
Disadvantages	- Verbose (requires more lines of code for styling). - Default plots are not very attractive (require customization). - Can be overwhelming for beginners due to too many options. - Statistical plots (like regression, KDE) are not built-in.	- Limited flexibility compared to raw Matplotlib (less control over fine details). - Slower for large datasets. - Depends on Matplotlib (cannot work without it). - Fewer options for **3D plotting** and advanced customizations.