Data Visualization using Matplotlib and Seaborn

1. INTRODUCTION TO VISUALIZATION

Visualization is the process of representing data, information, or concepts in a graphical or pictorial format. Instead of analyzing raw numbers or text, visualization helps to present information in a way that is easy to understand, interpret, and analyze.

It transforms complex datasets into charts, graphs, maps, and interactive dashboards, enabling people to identify patterns, trends, and insights quickly.

Definition

Visualization can be defined as:
"The graphical representation of information and data that enables easier understanding, communication, and decision-making."

➢ Importance of Visualization

1. **Simplifies complex data** – Large datasets become easier to interpret.
2. **Highlights trends and patterns** – Makes hidden relationships visible.
3. **Supports decision-making** – Provides insights for better business, scientific, and engineering decisions.
4. **Enhances communication** – Easier to present results to both technical and non-technical users.
5. **Interactive exploration** – Tools allow users to drill down and explore data.

➢ Types of Visualization

- **Statistical Charts** – Bar chart, Line chart, Pie chart, Histogram, Scatter plot.
- **Advanced Visuals** – Heatmaps, Donut charts, Tree maps, Violin plots.
- **Geographical Visualization** – Maps showing location-based data.
- **Interactive Dashboards** – Combination of visuals with filters for analysis

➢ Applications

- **Business Analytics** (sales trends, customer insights)
- **Scientific Research** (medical imaging, simulations)
- **Education** (illustrations of concepts)
- **Engineering** (system designs, performance analysis)
- **Machine Learning & AI** (model performance visualization)

Python provides powerful libraries for visualization such as Matplotlib and Seaborn.

2. MATPLOTLIB

Matplotlib is a popular Python library used for creating static, animated, and interactive visualizations.
It was developed by John D. Hunter in 2003 and is widely used in data science, machine learning, statistics, and scientific computing.

It provides a simple way to convert data into graphs and plots such as line charts, bar graphs, scatter plots, pie charts, histograms, and more.

➢ Key Features of Matplotlib

1. **Versatile plotting** – Supports 2D and basic 3D plotting.
2. **Variety of plots** – Line, bar, histogram, scatter, pie, stack, donut, etc.
3. **Highly customizable** – Control colors, labels, styles, axes, grid, legends, etc.
4. **Integration** – Works well with NumPy, Pandas, and Jupyter Notebook.
5. **Output formats** – Can export figures to PNG, JPG, PDF, SVG and others.
6. **Interactive plots** – Supports zooming, panning, and interactive exploration.

➢ Core Components

1. **Figure** → The outer container where everything is drawn.
2. **Axes** → The actual graph or subplot area inside the figure.
3. **Axis** → The x and y scale, ticks, and limits.
4. **Artists** → Anything visible on the plot (lines, text, labels, shapes).

➢ Pyplot

pyplot is a submodule of Matplotlib, usually imported as:

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

➢ Advantages of Matplotlib

- Easy to use and beginner-friendly.
- Rich set of plotting options.
- Works seamlessly with other libraries (NumPy, Pandas, Seaborn).
- Produces high-quality professional plots.

➢ Limitations

- Syntax can be lengthy for advanced plots.
- Visualization is **static by default** (limited interactivity compared to Plotly).
- 3D plotting support is basic.

➢ Applications of Matplotlib

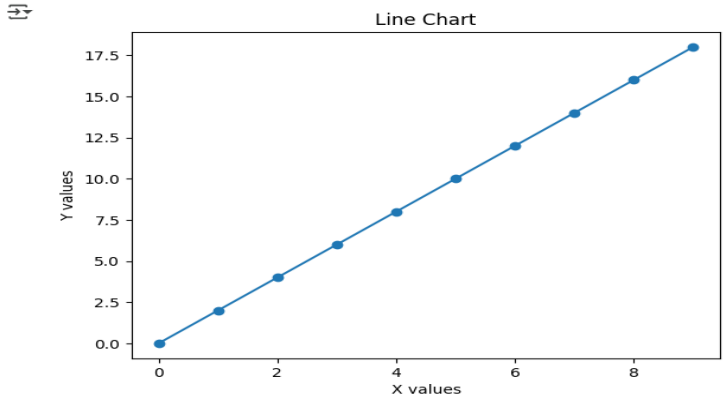
- **Data Analysis** – Visualizing sales, finance, and scientific data.
- **Machine Learning** – Plotting accuracy, loss, and model results.
- **Education** – Explaining mathematical and statistical concepts.
- **Engineering & Research** – Simulation results, experimental data visualization.

It provides functions for line, bar, scatter, pie, histogram, and more.

1. Line Chart

— Represents data points connected by straight lines.
— Useful for showing **trends over time** or continuous data.

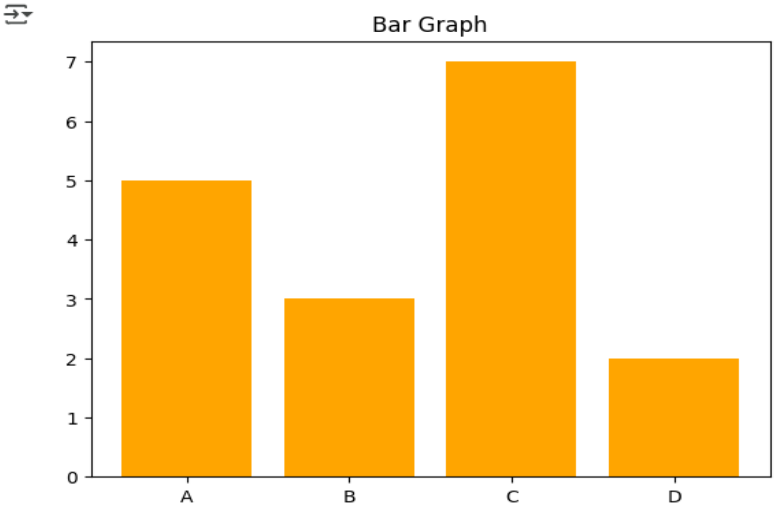
```
[4]  import numpy as np
     import matplotlib.pyplot as plt

     x = np.arange(0,10,1)
     y = x*2
     plt.plot(x,y,marker="o")
     plt.title("Line Chart")
     plt.xlabel("X values")
     plt.ylabel("Y values")
     plt.show()
```



2. Bar Graph

— Uses rectangular bars to show the **comparison between categories**.
— Height/length of the bar represents the value.

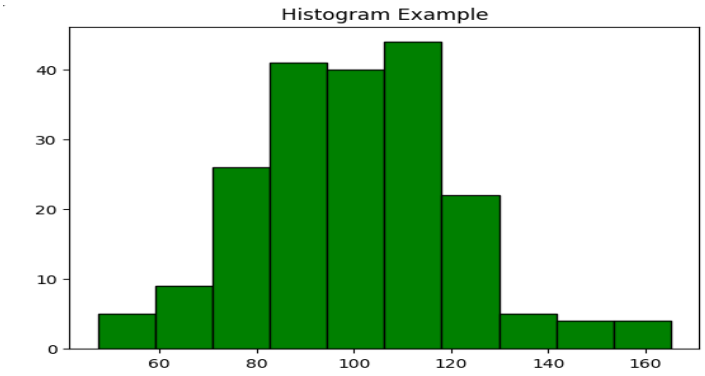
```
[5]  categories = ["A","B","C","D"]
     values = [5,3,7,2]
     plt.bar(categories, values, color="orange")
     plt.title("Bar Graph")
     plt.show()
```



3. Histogram

— Similar to bar chart but shows the **frequency distribution of data**.
— Divides data into intervals (bins) and shows how many values fall in each bin.

```
data = np.random.normal(100,20,200)
plt.hist(data, bins=10, color="green", edgecolor="black")
plt.title("Histogram Example")
plt.show()
```

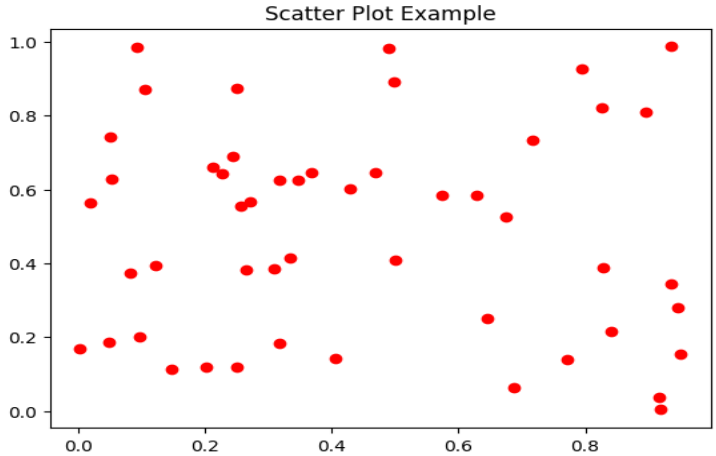


---

4. Scatter plot

— Displays data points on an **X-Y axis**.
— Shows **relationships or correlations** between two variables.

```
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x,y,color="red")
plt.title("Scatter Plot Example")
plt.show()
```

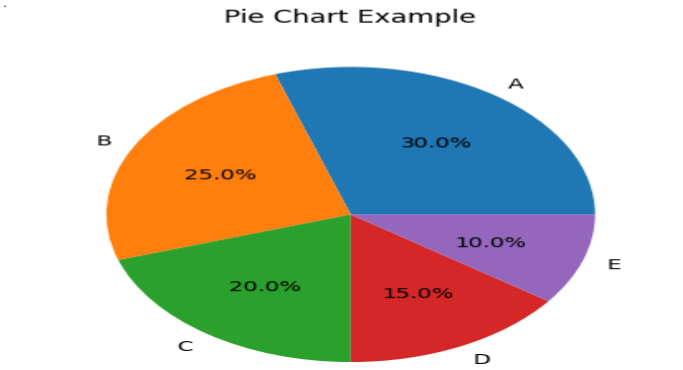


---

5. Pie Chart

— Circular chart divided into slices.
— Each slice represents a **proportion** of the whole.

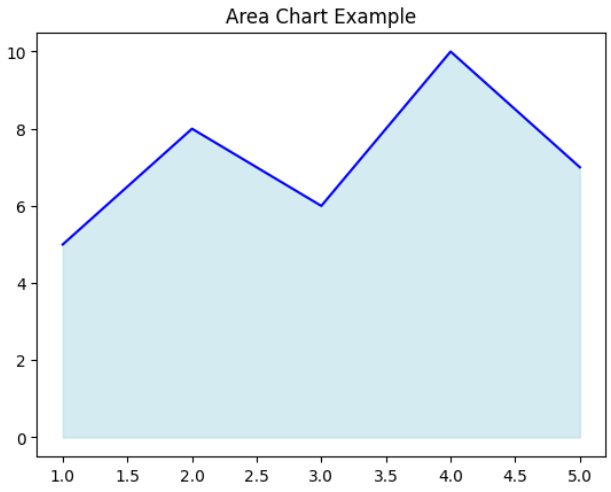
```
sizes = [30,25,20,15,10]
labels = ["A","B","C","D","E"]
plt.pie(sizes, labels=labels, autopct="%1.1f%%")
plt.title("Pie Chart Example")
plt.show()
```



6. Area Chart

— Similar to a line chart but the area under the line is **filled with color**.
— Useful for showing **cumulative trends** over time.

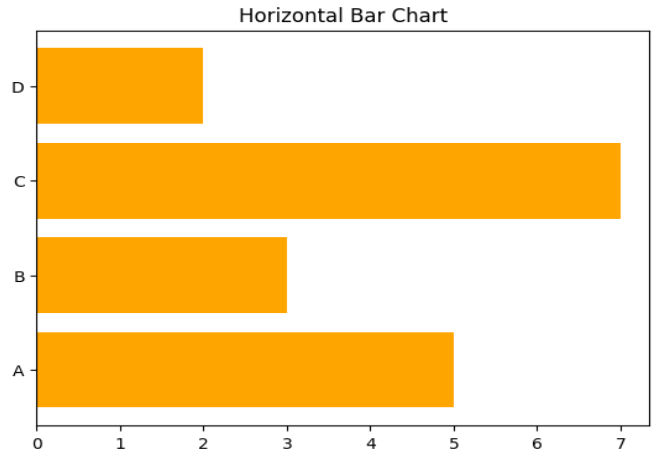
```
days = [1,2,3,4,5]
sales = [5,8,6,10,7]
plt.fill_between(days, sales, color="lightblue", alpha=0.5)
plt.plot(days, sales, color="blue")
plt.title("Area Chart Example")
plt.show()
```



7. Horizontal Bar Chart

— Same as bar chart but bars are **drawn horizontally**.
— Useful when category names are long or comparison is easier horizontally.

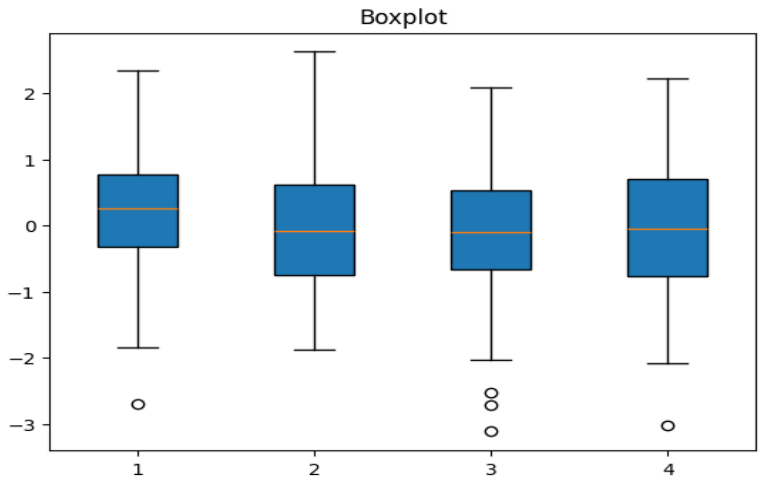
```
plt.barh(categories, values, color='orange')
plt.title("Horizontal Bar Chart")
plt.show()
```



8. Boxplot

— Summarizes data using **minimum, first quartile, median, third quartile, and maximum**.
— Useful for detecting **outliers and data spread**.

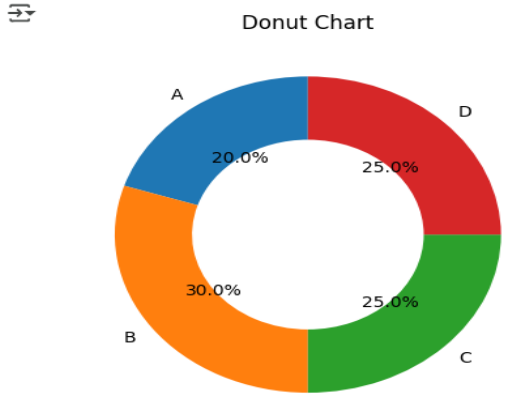
```
data = np.random.randn(100, 4)
plt.boxplot(data, patch_artist=True)
plt.title("Boxplot")
plt.show()
```



9. Donut chart

— Variation of pie chart with a **hole in the center**.
— Used to show proportions while leaving space in the middle for extra info.

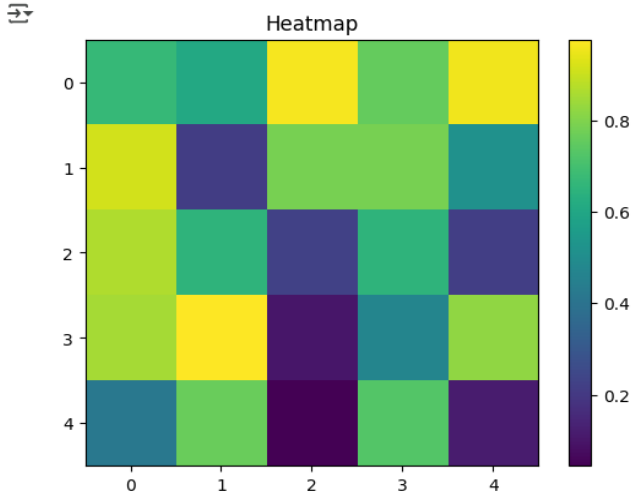
```
sizes = [20, 30, 25, 25]
labels = ["A", "B", "C", "D"]
plt.pie(sizes, labels=labels, autopct="%1.1f%%", startangle=90,
        wedgeprops=dict(width=0.4))   # width < 1 creates hole
plt.title("Donut Chart")
plt.show()
```



10. Heatmap

— Uses **color variations** to represent values in a 2D matrix or table.
— Commonly used for **correlation matrices** or **geographical intensity maps**.

```
[12] matrix = np.random.rand(5,5)
     plt.imshow(matrix, cmap="viridis", interpolation="nearest")
     plt.colorbar()
     plt.title("Heatmap")
     plt.show()
```



3. 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 much simpler code than Matplotlib.

Developed by **Michael Waskom**, Seaborn is widely used in **data analysis, machine learning, and statistics** because it integrates easily with **Pandas DataFrames** and has built-in functions for **statistical plots**.

➢ Key Features

1. **High-level interface** – Simple commands to create complex plots.
2. **Beautiful default styles** – Better aesthetics compared to Matplotlib.
3. **Integration with Pandas** – Works directly with DataFrames.
4. **Built-in statistical support** – Regression plots, distributions, categorical plots.
5. **Variety of plots** – Bar plots, boxplots, violin plots, heatmaps, swarm plots, etc.
6. **Themes and palettes** – Built-in themes (darkgrid, whitegrid, dark, etc.) and color palettes for professional visuals.

➢ Categories of Seaborn Plots

1. **Relational Plots**
   - sns.scatterplot() → relationship between two variables.
   - sns.lineplot() → trend or time series.
2. **Categorical Plots**
   - sns.barplot() → average values across categories.
   - sns.countplot() → frequency of categories.
   - sns.boxplot() → distribution with quartiles & outliers.
   - sns.violinplot() → combination of boxplot & KDE distribution.
   - sns.swarmplot() → categorical scatter plot.
3. **Distribution Plots**
   - sns.histplot() → histogram of data distribution.
   - sns.kdeplot() → probability density estimate.
   - sns.distplot() (older, replaced by histplot + kdeplot).
4. **Matrix Plots**
   - sns.heatmap() → visualize correlation or tabular data with color intensity.

➢ Advantages of Seaborn

- Simple syntax compared to Matplotlib.
- Attractive and professional-looking plots by default.
- Directly supports Pandas DataFrames.
- Provides statistical visualization (like regression).
- Easy integration with Jupyter notebooks.

➢ Limitations

- Less flexible than Matplotlib for very custom designs.
- Slightly slower for very large datasets.
- Relies on Matplotlib for backend (so not fully independent).

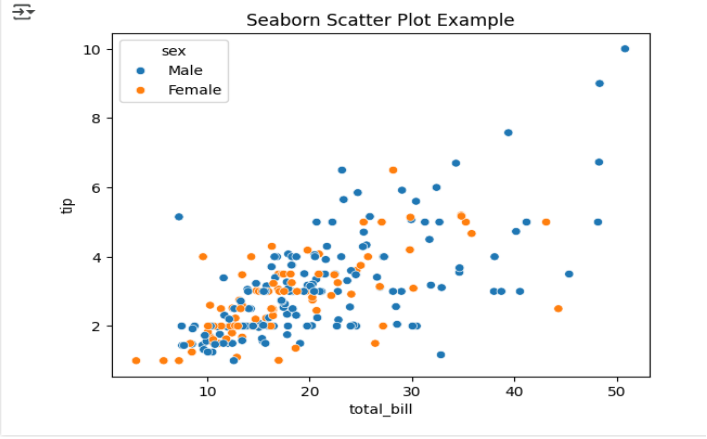
➢ Applications of Seaborn

- Data Analysis – Exploring datasets visually.
- Machine Learning – Feature correlation heatmaps, distribution checks.
- Statistics – Regression analysis, hypothesis testing visualization.
- Business Analytics – Customer segmentation, sales insights.

1. Scatter Plot

— Represents individual data points on an **X-Y axis**.
— Used to study **relationships or correlations** between two variables.
— Example: Height vs Weight.

```
import seaborn as sns
tips = sns.load_dataset("tips")
sns.scatterplot(x="total_bill", y="tip", hue="sex", data=tips)
plt.title("Seaborn Scatter Plot Example")
plt.show()
```

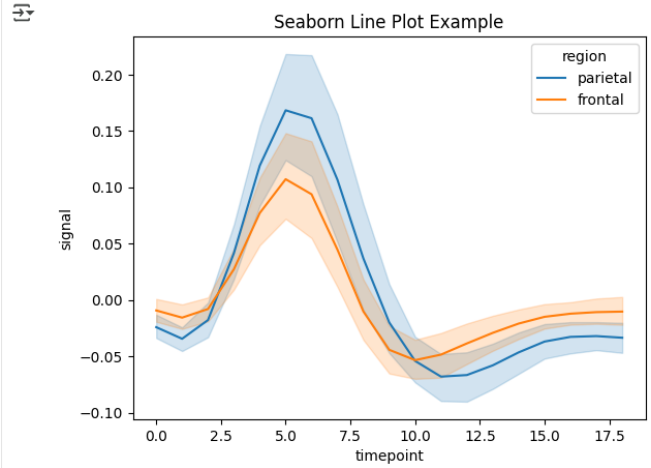


---

2. Line Plot

— Displays data points connected by straight lines.
— Best for showing **trends, patterns, or changes over time**.
— Example: Stock price over months.

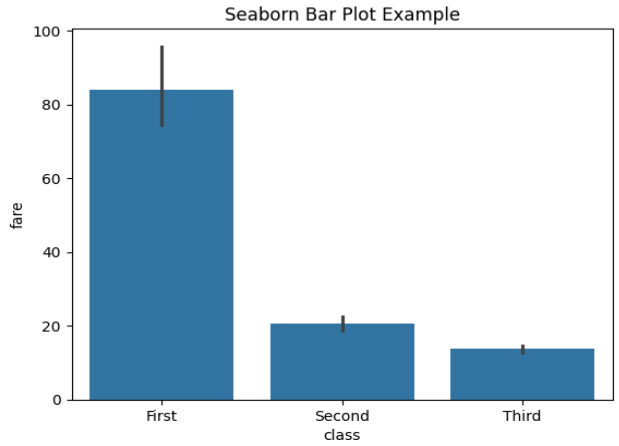
```
fmri = sns.load_dataset("fmri")
sns.lineplot(x="timepoint", y="signal", hue="region", data=fmri)
plt.title("Seaborn Line Plot Example")
plt.show()
```



3. Bar Plot

— Uses rectangular bars to represent data values.
— Compares **different categories or groups**.
— Example: Sales comparison across products.

```
titanic = sns.load_dataset("titanic")
sns.barplot(x="class", y="fare", data=titanic)
plt.title("Seaborn Bar Plot Example")
plt.show()
```

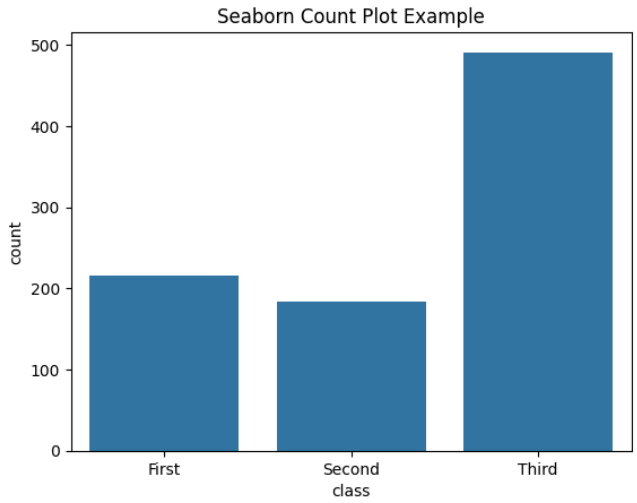


---

4. Count Plot

— A special type of bar plot that shows the **frequency (count) of categories**.
— Does not require numerical values; it automatically counts occurrences.
— Example: Number of students in each department.

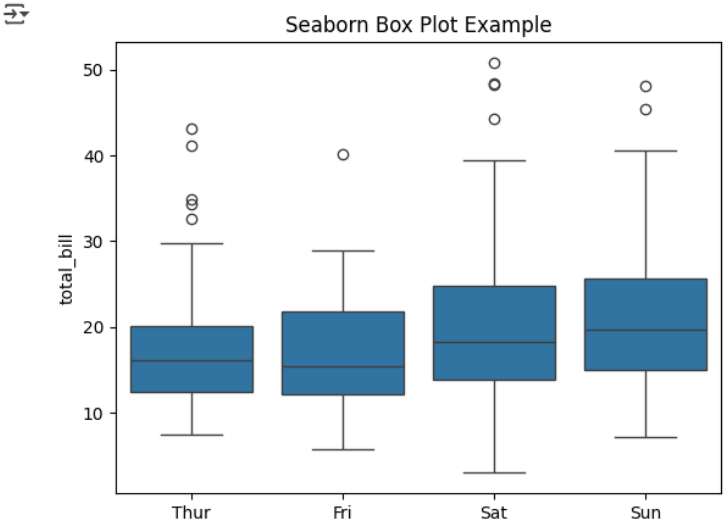
```
sns.countplot(x="class", data=titanic)
plt.title("Seaborn Count Plot Example")
plt.show()
```



5. Box Plot

— Summarizes data using **minimum, first quartile, median, third quartile, and maximum**.
— Useful for identifying **outliers and spread** of data.
— Example: Distribution of exam marks in a class.

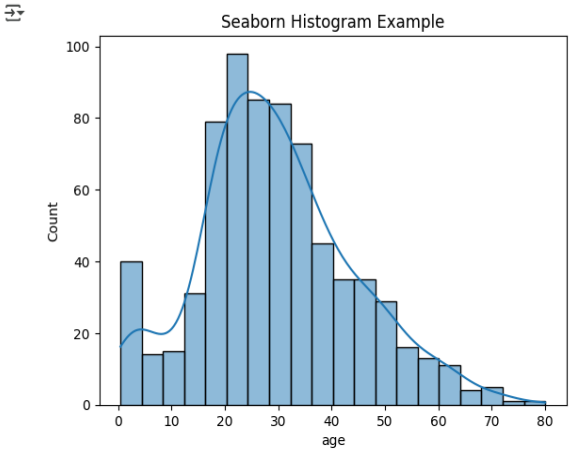
```
sns.boxplot(x="day", y="total_bill", data=tips)
plt.title("Seaborn Box Plot Example")
plt.show()
```



6. Histogram

— Shows the **frequency distribution** of continuous data.
— Divides data into **intervals (bins)** and counts how many values fall in each bin.
— Example: Distribution of ages in a survey.

```
sns.histplot(titanic["age"].dropna(), bins=20, kde=True)
plt.title("Seaborn Histogram Example")
plt.show()
```

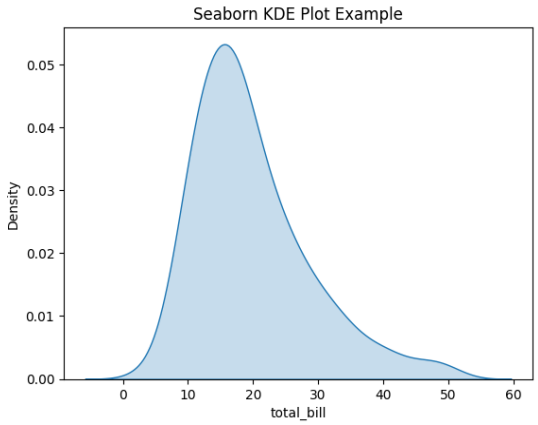


7. KDE Plot

— A smooth curve that estimates the **probability density function** of a dataset.
— Similar to histogram but continuous and smoother.
— Example: Income distribution in a population.

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

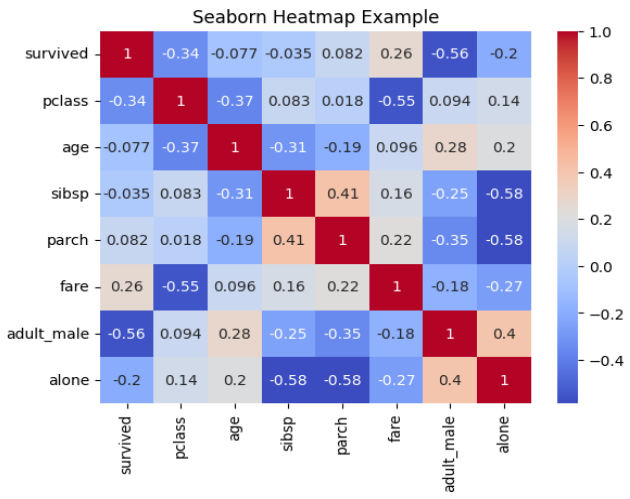
```
/tmp/ipython-input-1551424417.py:1: FutureWarning:

`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)
```



8. Heatmap

— Uses **color intensity** to represent values in a matrix or 2D dataset.
— Often used for **correlation analysis** or to visualize large tables.
— Example: Correlation between features in a dataset.

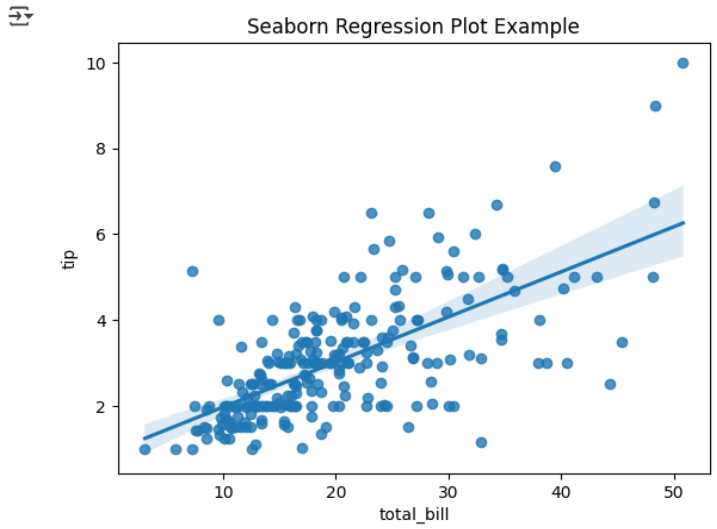
```
corr = titanic.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap="coolwarm")
plt.title("Seaborn Heatmap Example")
plt.show()
```



9. Regression Plot

— Shows the relationship between two variables with a **fitted regression line**.
— Useful for **predictive analysis** and identifying linear trends.
— Example: Advertising spend vs Sales revenue.

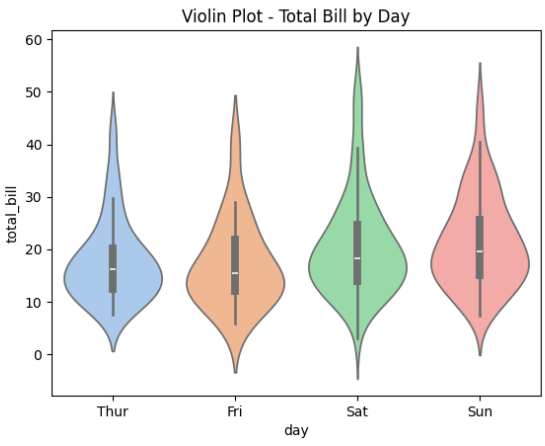
```
sns.regplot(x="total_bill", y="tip", data=tips)
plt.title("Seaborn Regression Plot Example")
plt.show()
```



10. Violin plot

```
tips = sns.load_dataset("tips")
sns.violinplot(x="day", y="total_bill", data=tips, palette="pastel")
plt.title("Violin Plot - Total Bill by Day")
plt.show()
```

```
/tmp/ipython-input-452674314.py:4: FutureWarning:

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

  sns.violinplot(x="day", y="total_bill", data=tips, palette="pastel")
```



4. Comparison: Matplotlib vs Seaborn

Feature	Matplotlib	Seaborn
Type	Low-level plotting library	High-level plotting library built on Matplotlib
Syntax	More detailed & manual	Simpler & more concise
Customization	Very flexible (you can control every detail)	Limited but elegant (auto styles, themes)
Default Style	Basic, not very attractive	Beautiful, publication-ready by default
Best For	Creating custom plots, fine-grained control	Quick statistical plots & data analysis
Chart Types	Line, bar, scatter, histogram, pie, 3D plots, etc.	Distribution plots (hist, kde), categorical plots (bar, violin, swarm), regression plots, heatmaps
Integration	Works with NumPy arrays & pure Python lists	Works best with Pandas DataFrames
Learning Curve	Steeper, more coding needed	Easier for beginners in data science
Speed of Plotting	Slower to write but more powerful	Faster to write, fewer lines of code
When to Use	When you need full control over chart appearance (custom dashboards, research plots)	When doing **EDA (Exploratory Data Analysis)**, quick visualization, or statistical graphics