VISUALISATION

Visualization is a method of presenting information visually. Data visualization involves displaying data in graphical or pictorial formats to make it easier to understand and interpret. It helps in gaining insights and identifying patterns or trends within the data.

In Python, there are several libraries available for data visualization, such as matplotlib, seaborn and plotly.

Let's explore matplotlib and seaborn in more details

MATPLOTLIB

Introduction to Matplotlib

Matplotlib is a powerful 2D plotting library for Python introduced in **2002 by John Hunter**. It is primarily built on NumPy and is used to create high-quality, publication-ready plots.

Core Components of a Matplotlib Plot

A typical plot in Matplotlib consists of the following main components:

1. Figure

- The outermost container for a plot.
- It can contain one or multiple Axes (subplots).

2. Axes

- These are the actual plots within the Figure.
- An Axes contains Axis, title, labels, and plot elements.
- Example functions:
    - set_ xlabel() – Sets the label for the X-axis.
    - set_ ylabel() – Sets the label for the Y-axis.

3. Axis

- Responsible for handling the numerical ranges and tick marks on the plot.
- It determines what values are shown and how they're formatted.

4. Artists

- Every visible element in the plot (lines, texts, shapes) is an artist.
- Includes everything from the plot lines to legends and title

Pyplot Sub-Library

pyplot is a sub-library of Matplotlib that provides a simple interface for plotting. Most plotting functions like plot(), bar(), hist(), and scatter() reside in this module.

Importing Matplotlib and NumPy:

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

NumPy is often used alongside Matplotlib for creating data arrays and performing calculations.

Types of Plots in Matplotlib

LINE CHART

Line plots are the most basic form of plotting, connecting consecutive data points with a line.

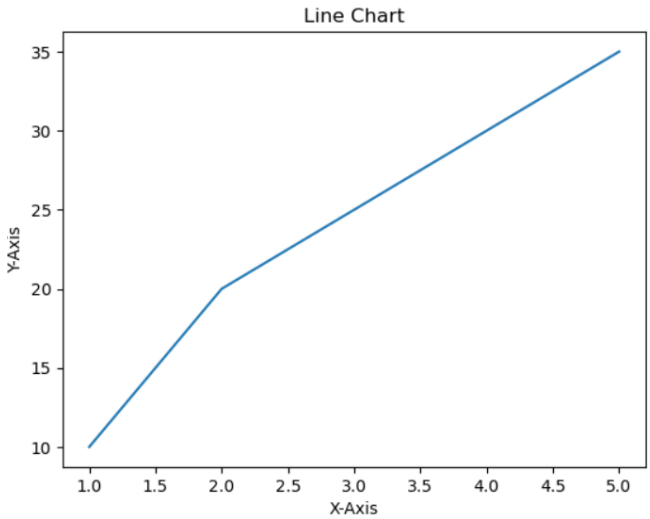
Code sample:

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

x = np.array([1, 2, 3, 4, 5])
y = np.array([10, 20, 25, 30, 35])

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

Output:



Description:

This code plots a simple line graph using points (x, y).

title() and xlabel()/ylabel() - label the chart or add metadata

plot() - draws the line.

show() - displays it.

Use:

To show trends or changes over time (e.g., growth, temperature, stock prices).

BAR GRAPH

Bar charts display categorical data with rectangular bars.

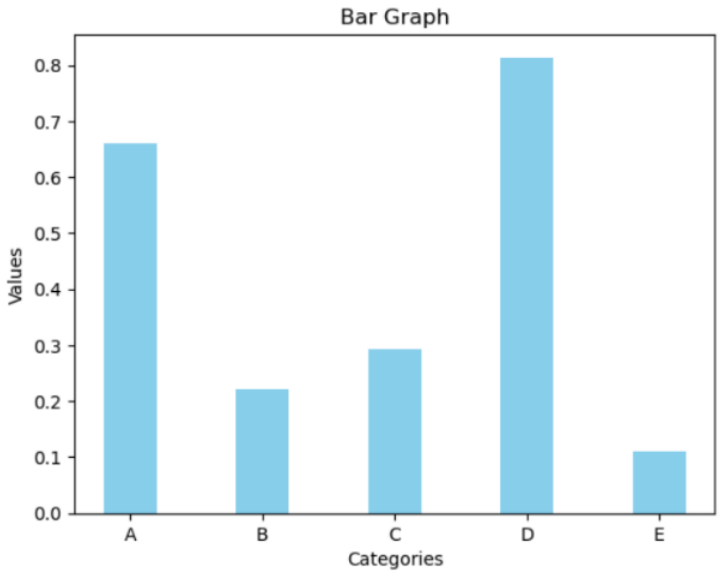
Code sample:

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

x = np.array(["A", "B", "C", "D", "E"])
y = np.random.rand(5)

plt.bar(x, y, color='skyblue', width=0.4)
plt.title("Bar Graph")
plt.xlabel("Categories")
plt.ylabel("Values")
plt.show()
```

Output:



Description:

bar() - draws vertical bars.

color - sets the bar color, and width adjusts the bar size.

show()- displays the chart.

Use:

To compare values across categories, like sales, marks, or counts.

HISTOGRAM

Histograms show the frequency distribution of data values grouped into bin

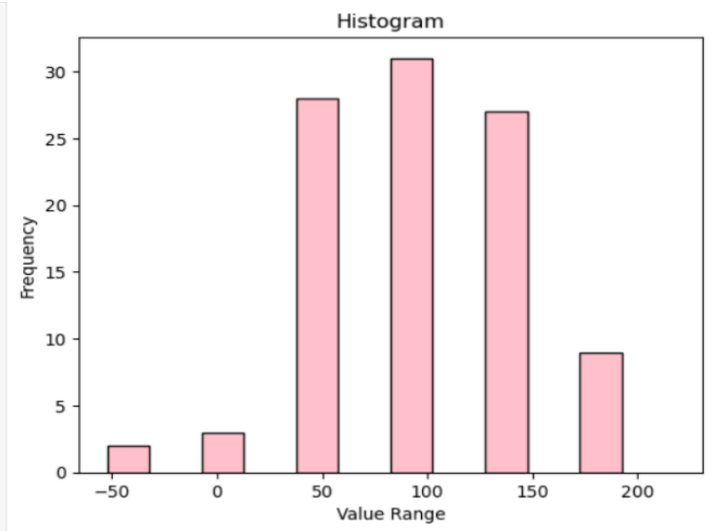
Code sample:

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

data = np.random.normal(100, 50, 100)

plt.hist(data, bins=6, color='pink', edgecolor='black', width=20 )
plt.title("Histogram")
plt.xlabel("Value Range")
plt.ylabel("Frequency")
plt.show()
```

Output:



Description:

This code creates a histogram from 100 random values.

hist() divides the data into 10 bins (ranges).

color sets the bar color, and edgecolor outlines the bars.

xlabel() and ylabel() label the axes, and show() displays the plot.

Use:

To show the distribution of data and how frequently values occur within certain ranges.

SCATTER PLOT

Scatter plots show the relationship between two numeric variables using dots.

Code sample:

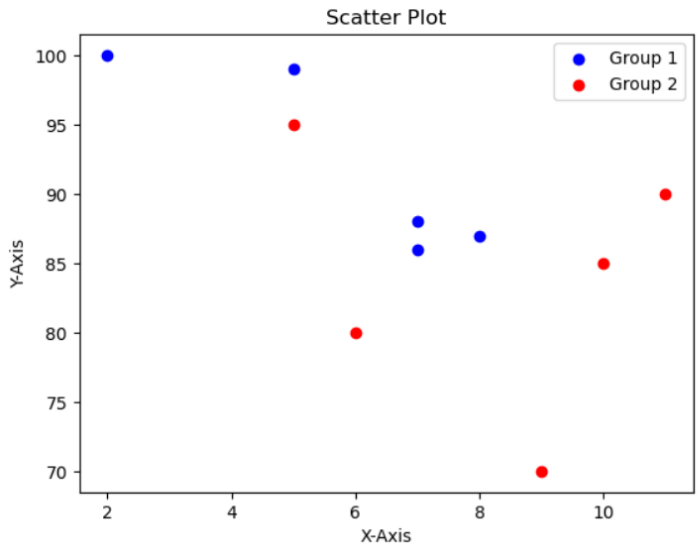
```
import matplotlib.pyplot as plt
import numpy as np

x1 = np.array([5, 7, 8, 7, 2])
y1 = np.array([99, 86, 87, 88, 100])

x2 = np.array([6, 9, 11, 10, 5])
y2 = np.array([80, 70, 90, 85, 95])

plt.scatter(x1, y1, color='blue', label='Group 1')
plt.scatter(x2, y2, color='red', label='Group 2')
plt.title("Scatter Plot")
plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
plt.legend()
plt.show()
```

output:



Description:
This scatter plot compares two groups of data using different colors.
Each point represents an (x, y) pair.
The legend() helps identify which points belong to which group.
Useful for spotting trends, clusters, or outliers.

Use:
To show relationships or comparisons between two sets of numerical data using dots.

PIE CHART

Pie charts display parts of a whole, divided by percentage.

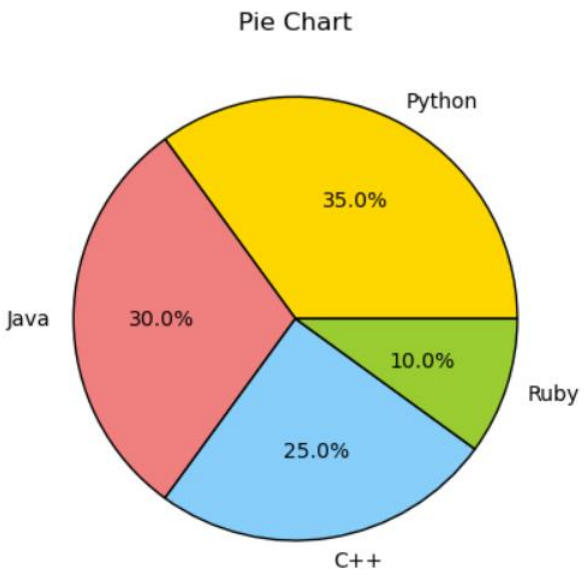
Code sample:

```
import matplotlib.pyplot as plt

labels = ['Python', 'Java', 'C++', 'Ruby']
sizes = [35, 30, 25, 10]
colors = ['gold', 'lightcoral', 'lightskyblue', 'yellowgreen']

plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%',
        wedgeprops={'linewidth': 1, 'edgecolor': 'black'})
plt.title("Pie Chart")
plt.show()
```

Output:



Description:
This pie chart divides a circle into slices based on given values.
Each slice represents a category's share of the total.
autopct displays the percentage, and wedgeprops adds borders for clear separation. It's ideal for visualizing parts of a whole.

Use:
To show proportions or percentage distribution of different categories in a dataset.

AREA CHART

Area charts fill the area below the line to the X-axis. It can be done by using fill_between() function or stackplot() function.

Use:

to show how a quantitative value changes over time or categories, emphasizing the magnitude of change.

using fill_between() :

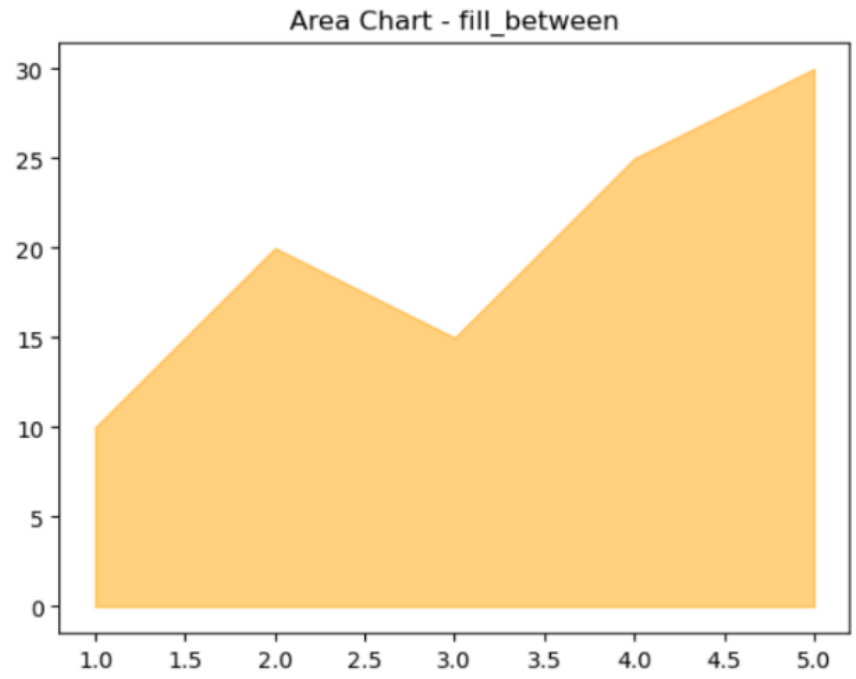
Code sample:

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

x = np.arange(1, 6)
y = np.array([10, 20, 15, 25, 30])

plt.fill_between(x, y, color='orange', alpha=0.5)
plt.title("Area Chart - fill_between")
plt.show()
```

Output:



using stackplot() :

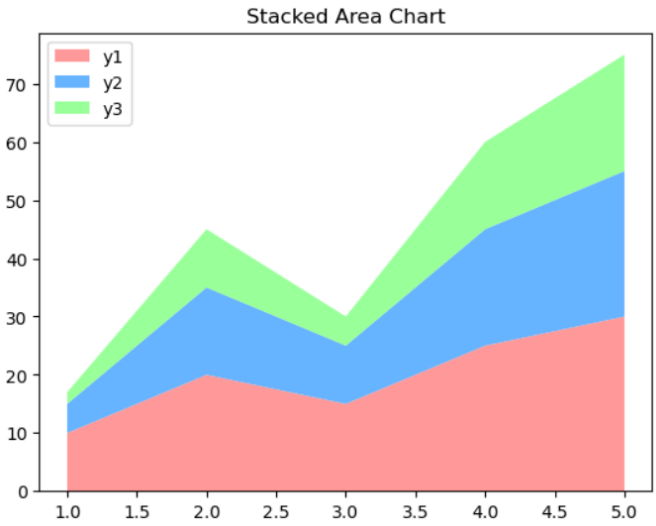
Code sample:

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

x = np.arange(1, 6)
y1 = [10, 20, 15, 25, 30]
y2 = [5, 15, 10, 20, 25]
y3 = [2, 10, 5, 15, 20]

plt.stackplot(x, y1, y2, y3, labels=['y1', 'y2', 'y3'], colors=['#ff9999','#66b3ff','#99ff99'])
plt.legend(loc='upper left')
plt.title("Stacked Area Chart")
plt.show()
```

Output:



SEABORN

Seaborn is a high-level data visualization library in Python, built on top of Matplotlib. It provides a user-friendly interface for creating visually appealing and informative statistical graphics. With built-in themes, color palettes, and abstraction of complex plot types, Seaborn enhances the visualization capabilities of Matplotlib, making it ideal for data exploration and presentation.

Introduction

In data science and statistical analysis, visual representation of data plays a vital role in understanding patterns, trends, and relationships among variables. Seaborn simplifies this process by providing a variety of plot types, categorized based on the nature of the data and the kind of insight required. The key categories of plots in Seaborn include:

- Relational plots
- Categorical plots
- Distribution plots
- Regression plots
- Matrix plots

Relational Plot

Relational plots are used to understand the relationship between two or more variables. Seaborn offers several plot types in this category:

- **scatterplot()** – Displays individual data points on a 2D plane, often used to identify correlations.
- **lineplot()** – Suitable for time series or continuous data; displays trends over intervals.
- **relplot()** – A figure-level function for creating multiple subplots with either scatter or line plots.

Categorical Plots

These plots are designed to visualize the distribution or relationship of categorical variables. Key plot types include:

- **barplot()** – Displays mean values with confidence intervals for categories.
- **countplot()** – Shows the count of observations in each category.
- **boxplot()** – Visualizes the distribution through quartiles and identifies outliers.
- **violinplot()** – Combines boxplot with a KDE plot for richer distribution insight.
- **swarmplot()** – Plots all data points, avoiding overlap.
- **pointplot()** – Shows the estimated values with confidence intervals using points.
- **catplot()** – A flexible, figure-level interface for creating various categorical plots.

Distribution Plots

Distribution plots are useful for understanding how data is spread across a variable. The main types include:

- **histplot()** – A histogram to visualize frequency distributions
- **kdeplot()** – Displays the Kernel Density Estimate of the distribution.
- **rugplot()** – Adds small vertical lines at each observation along the x-axis.
- **distplot()** *(Deprecated)* – A combination of histogram and KDE (now replaced by histplot() and kdeplot()).

Regression Plots

Regression plots are used to study relationships between variables along with a regression line, often used for prediction or trend analysis.

- **regplot()** – A simple scatter plot with a linear regression model fit.
- **lmplot()** – A figure-level function that allows for grouping and facetting**.**

Matrix Plots

Matrix plots are used to visualize relationships in tabular or matrix-like data, especially correlations between numerical variables.

- **heatmap()** – Displays data values as a matrix with varying color intensities.
- **clustermap()** – Adds hierarchical clustering to the heatmap for better grouping insight.

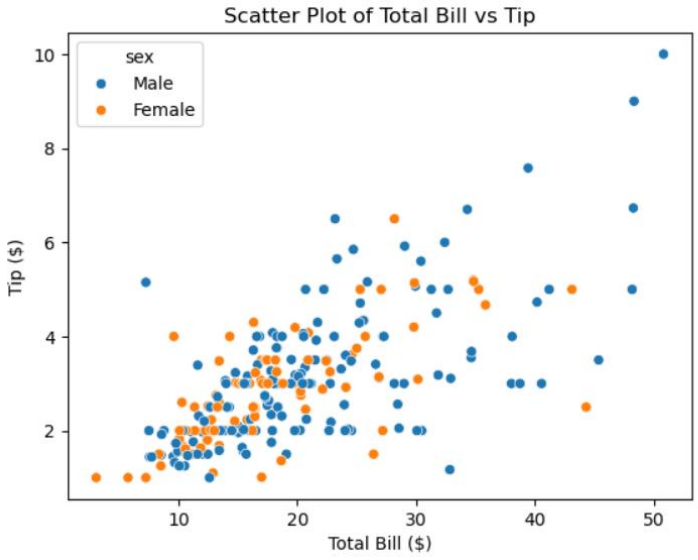
Here are some sample codes for some of the graphs.

SCATTER PLOT

Code sample:

```
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("tips")
sns.scatterplot(data=df, x="total_bill", y="tip", hue="sex")
plt.title("Scatter Plot of Total Bill vs Tip")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.show()
```

Output:



Description:

data=df: Uses the tips dataset.

x="total_bill", y="tip": Plots total bill on the x-axis and tip on the y-axis.

hue="sex": Colors the points by gender (Male/Female).

style="smoker": Uses different markers for smoker and non-smoker.

Use:

To show relationships or comparisons between two sets of numerical data using dots.

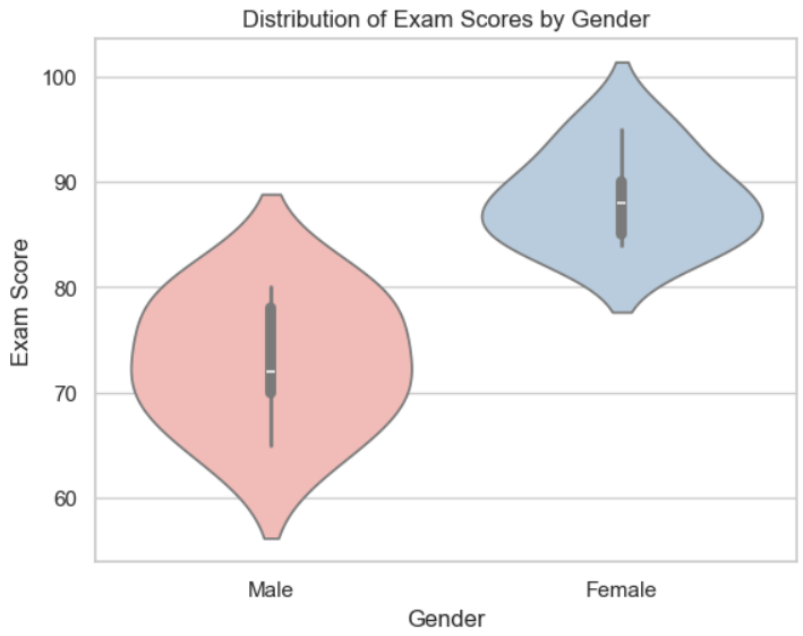
VIOLIN PLOT

Code sample:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
data = {
    "gender": ["Male", "Female", "Male", "Female", "Male", "Female", "Male", "Female", "Male", "Female"],
    "score": [78, 85, 72, 90, 65, 88, 80, 95, 70, 84]
}

df = pd.DataFrame(data)
sns.set(style="whitegrid")
sns.violinplot(data=df, x="gender", y="score", hue="gender", palette="Pastel1", legend=False)
plt.title("Distribution of Exam Scores by Gender")
plt.xlabel("Gender")
plt.ylabel("Exam Score")
plt.show()
```

Output:



Use:

visualize the distribution, central tendency, and variability of a numerical variable across different categories.

Description:

x="gender": Puts gender categories on the X-axis.

y="score": Puts exam scores on the Y-axis.

hue="gender": Uses different colors for each gender category.

palette="Pastel1": Applies a soft pastel color scheme.

legend=False: Hides the legend since gender is already shown on the X-axis.

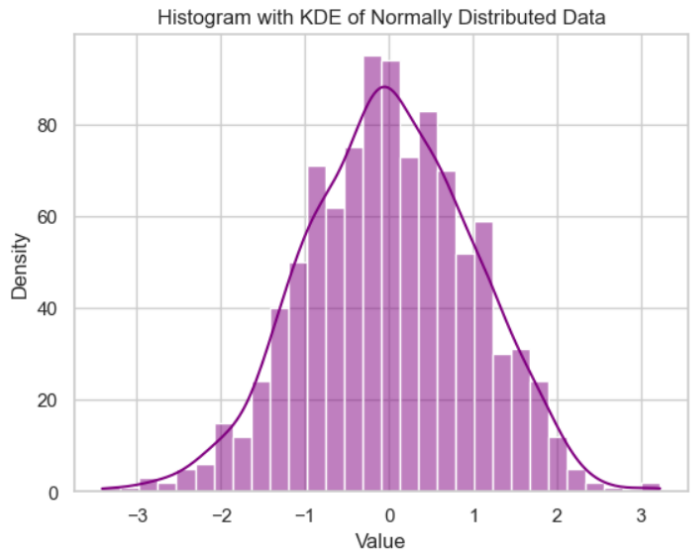
sns.set(style="whitegrid"): Sets a clean grid background for better readability.

DIST PLOT

Code sample:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
data = np.random.normal(loc=0, scale=1, size=1000)
sns.set(style="whitegrid")
sns.histplot(data, bins=30, kde=True, color="purple")
plt.title("Histogram with KDE of Normally Distributed Data")
plt.xlabel("Value")
plt.ylabel("Density")
plt.show()
```

Output:



Use:

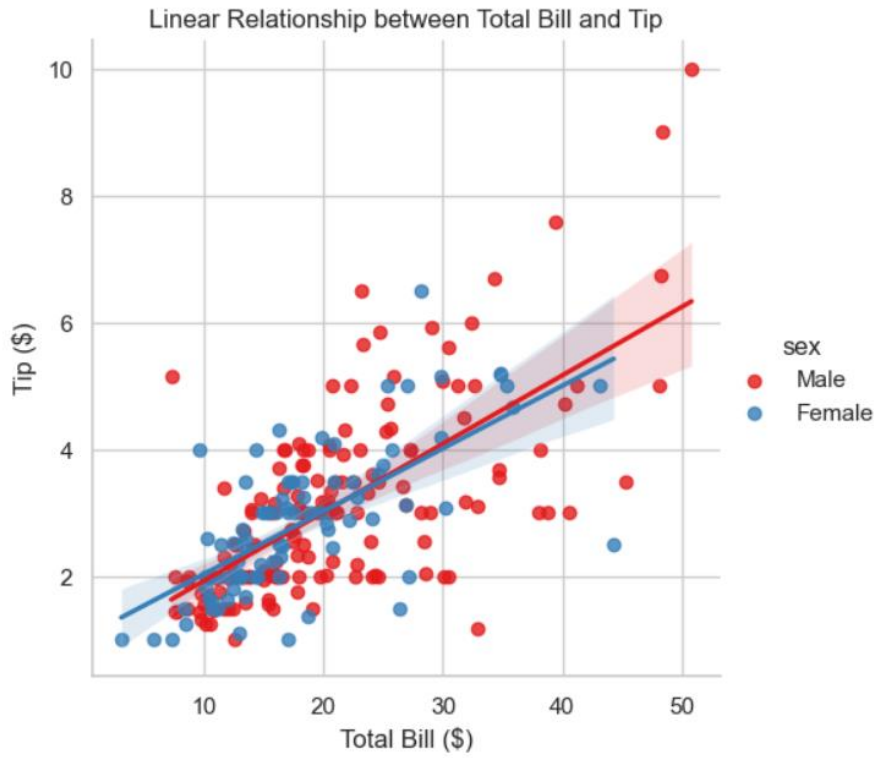
visualize the distribution of a dataset by combining a histogram and a kernel density estimate (KDE), helping to understand data shape and spread.

LM PLOT

Code sample:

```
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("tips")
sns.set(style="whitegrid")
sns.lmplot(data=df, x="total_bill", y="tip", hue="sex", palette="Set1")
plt.title("Linear Relationship between Total Bill and Tip")
plt.xlabel("Total Bill ($)")
plt.ylabel("Tip ($)")
plt.show()
```

Output:



Use:

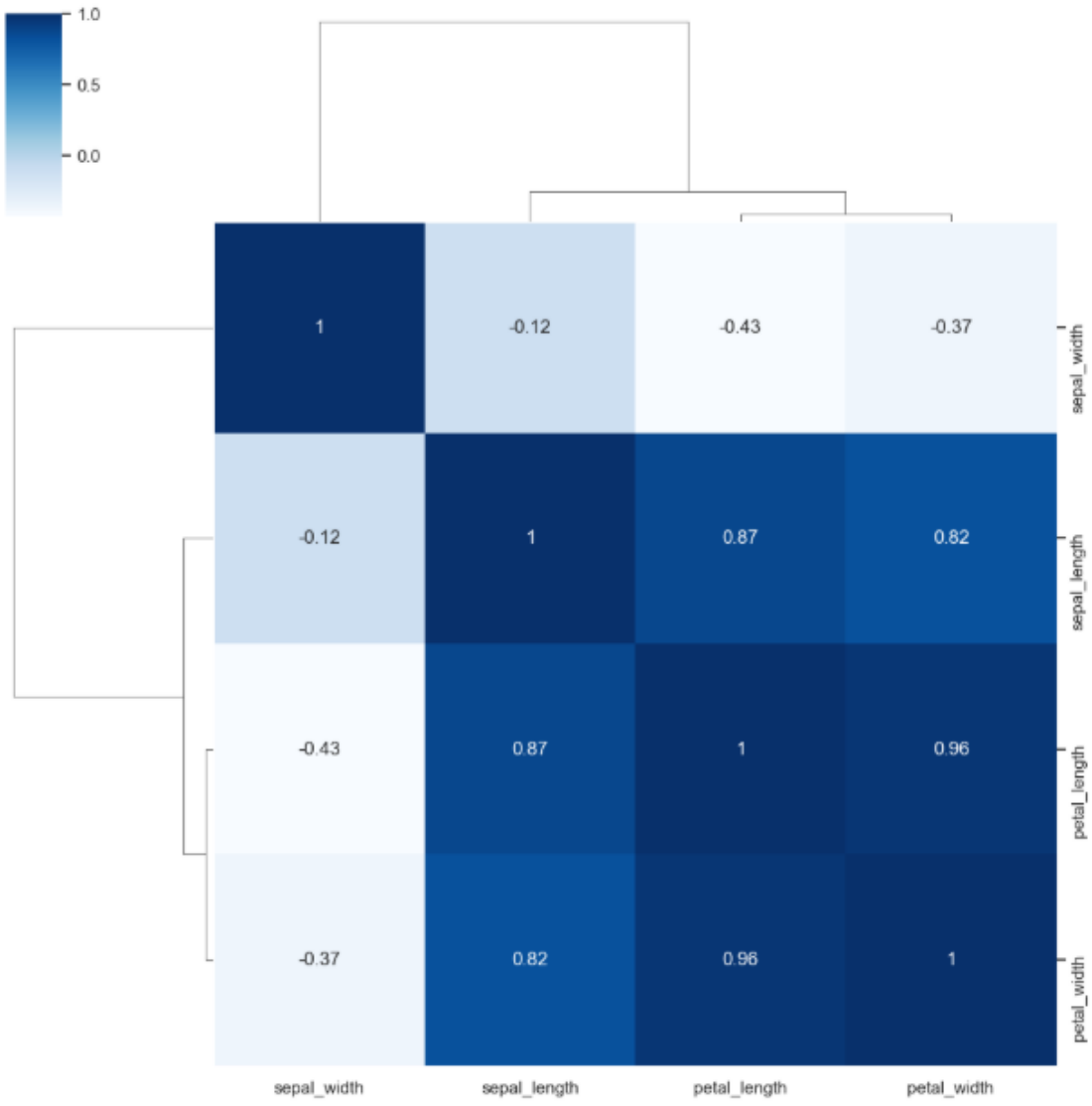
To create beautiful, statistical visualizations easily using pandas data structures.

CLUSTER MAP

Code sample:

```
import seaborn as sns
import matplotlib.pyplot as plt
df = sns.load_dataset("iris")
corr = df.drop(columns="species").corr()
sns.clustermap(corr, annot=True, cmap="Blues")
plt.show()
```

Output:



Use:

visually group similar rows and columns in a matrix using hierarchical clustering — commonly used in gene expression analysis, customer segmentation, or correlation heatmaps in data science.

Comparison of Matplotlib vs Seaborn :

Feature / Aspect	Matplotlib	Seaborn
Library Type	Low-level plotting library	High-level statistical data visualization library
Built On	Standalone library	Built on top of Matplotlib
Complexity	Requires more code and customization	Simpler, less code for attractive plots
Ease of Use	Steeper learning curve	Beginner-friendly with defaults
Default Styles	Basic and minimal	Prettier and more modern aesthetics by default
Statistical Plots	Needs manual implementation	Built-in support (e.g., boxplot, violinplot, lmplot)
Data Handling	Works well with arrays and lists	Works best with pandas DataFrames
Customization	Highly customizable	Customization available but limited to Seaborn's scope
Plot Types	Bar, line, scatter, histogram, pie, etc.	Includes all Matplotlib types plus KDE, heatmap, catplot
Themes/Palettes	Manual styling required	Built-in themes and color palettes
Subplots & Figures	Full control using plt.figure(), plt.subplot()	Easier with FacetGrid, but less control than Matplotlib
Performance	Slightly faster for basic plots	Slightly slower due to abstraction
Interactivity	Basic (requires extra libraries for advanced interaction)	Not built for interactivity (add-ons needed)