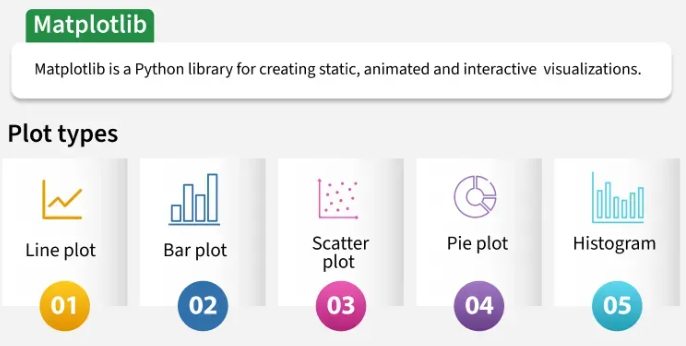
**DATA VISUALIZATION WITH MATPLOTLIB AND SEABORN**

**MATPLOTLIB:**

Matplotlib is an open-source**visualization library** for the Python programming language, widely used for creating **static**, **animated**and **interactive plots.** It provides an **object-oriented API** for embedding plots into applications using general-purpose GUI toolkits like Tkinter, Qt, GTK and wxPython. It offers a variety of plotting functionalities, including line plots, bar charts, histograms, scatter plots and 3D visualizations. Created by John D. Hunter in 2003, Matplotlib has become a fundamental tool for data visualization in Python, extensively used by data scientists, researchers and engineers worldwide.



**What is Matplotlib in Python used for?**

With Matplotlib, we can perform a wide range of visualization tasks, including:

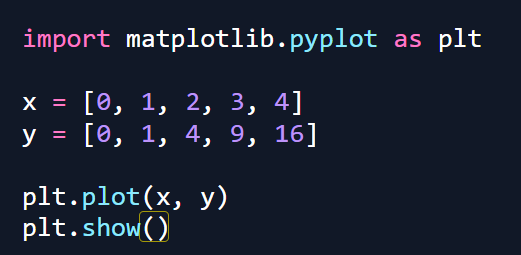
* Creating basic plots such as line, bar and scatter plots.
* Customizing plots with labels, titles, legends and color schemes.
* Adjusting figure size, layout and aspect ratios.
* Saving plots in various formats like PNG, PDF and SVG.
* Combining multiple plots into subplots for better data representation.
* Creating interactive plots using the widget module.

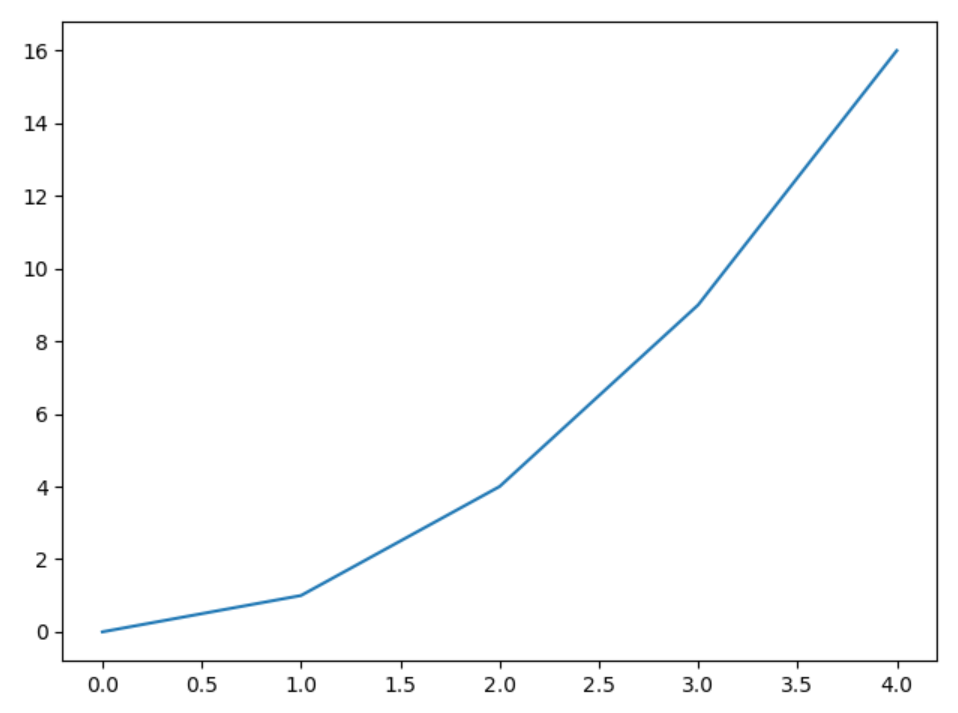
Matplotlib is a powerful and versatile open-source plotting library for Python, designed to help users visualize data in a variety of formats. Developed by John D. Hunter in 2003, it enables users to graphically represent data, facilitating easier analysis and understanding.**If you want to convert your boring data into interactive plots and graphs, Matplotlib is the tool for you**.

**Important Facts to know:**

* **Matplotlib Pyplot:** The pyplot module is a collection of functions that make Matplotlib work like MATLAB, providing a simple interface for creating plots.
* **Figure and Axes:** In Matplotlib, figures represent the overall container, while axes refer to the individual plots within a figure.
* **Integration with Pandas:** Matplotlib works seamlessly with Pandas DataFrames, enabling efficient data visualization.

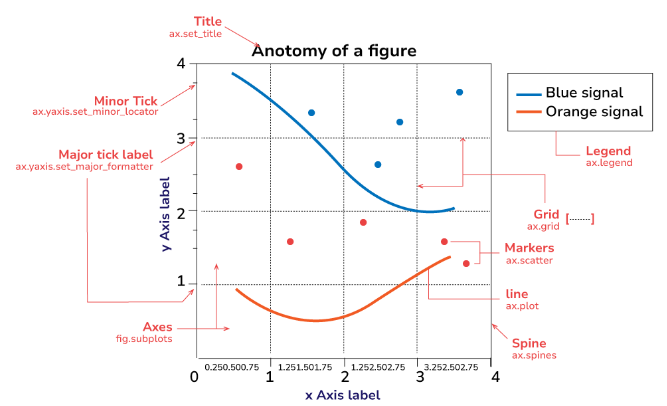
**EXAMPLE:**

****

****

**Components or Parts of Matplotlib Figure:**

**Anatomy of a Matplotlib Plot:** This section dives into the key components of a Matplotlib plot, including figures, axes, titles, and legends, essential for effective data visualization.



The parts of a Matplotlib figure include (as shown in the figure above):

* **Figure**: The overarching container that holds all plot elements, acting as the canvas for visualizations.
* **Axes**: The areas within the figure where data is plotted; each figure can contain multiple axes.
* **Axis**: Represents the x-axis and y-axis, defining limits, tick locations, and labels for data interpretation.
* **Lines and Markers**: Lines connect data points to show trends, while markers denote individual data points in plots like scatter plots.
* **Title and Labels**: The title provides context for the plot, while axis labels describe what data is being represented on each axis.

**Matplotlib Pyplot:**

Pyplot is a module within Matplotlib that provides a MATLAB-like interface for making plots. It simplifies the process of adding plot elements such as lines, images, and text to the axes of the current figure. **Steps to Use Pyplot:**

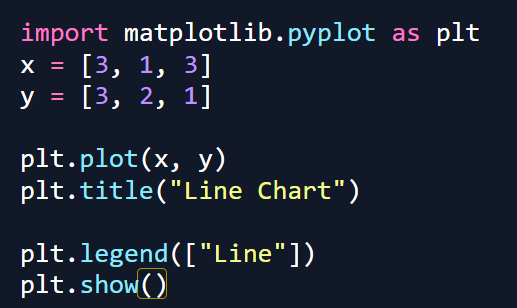
* **Import Matplotlib:** Start by importing matplotlib.pyplot as plt.
* **Create Data:** Prepare your data in the form of lists or arrays.
* **Plot Data**: Use plt.plot() to create the plot.
* **Customize Plot**: Add titles, labels, and other elements using methods like plt.title(), plt.xlabel(), and plt.ylabel().
* **Display Plot**: Use plt.show() to display the plot.

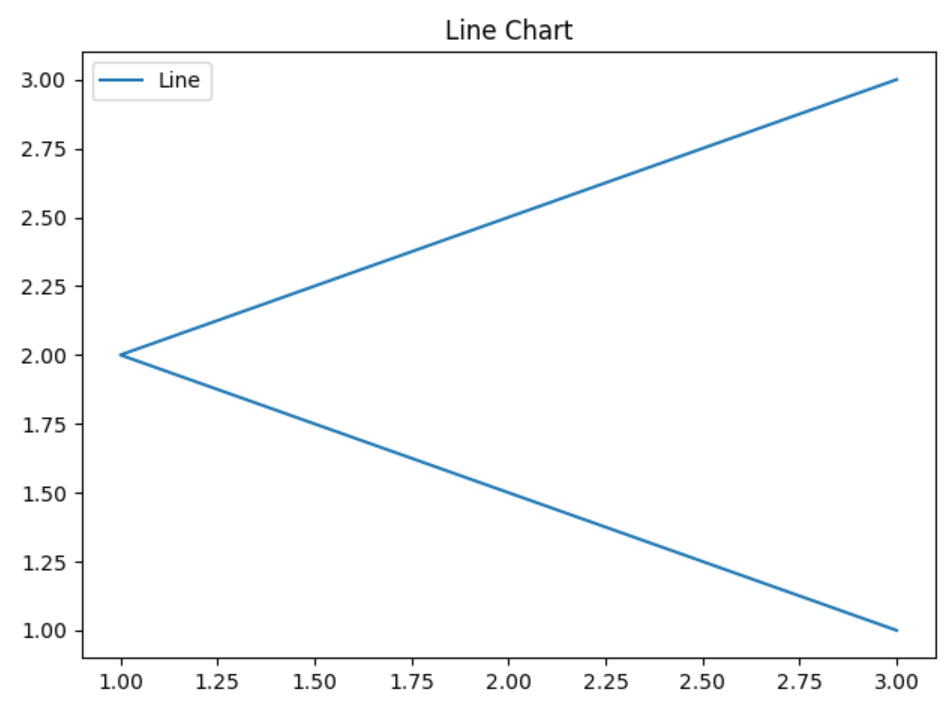
**Different Types of Plots in Matplotlib:**

Matplotlib offers a wide range of plot types to suit various data visualization needs. Here are some of the most commonly used types of plots in Matplotlib:

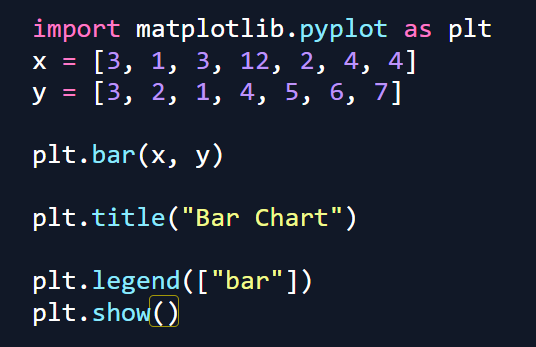
* 1. Line Graph
* 2. Bar Chart
* 3. Histogram
* 4. Scatter Plot
* 5. Pie Chart
* 6. 3D Plot

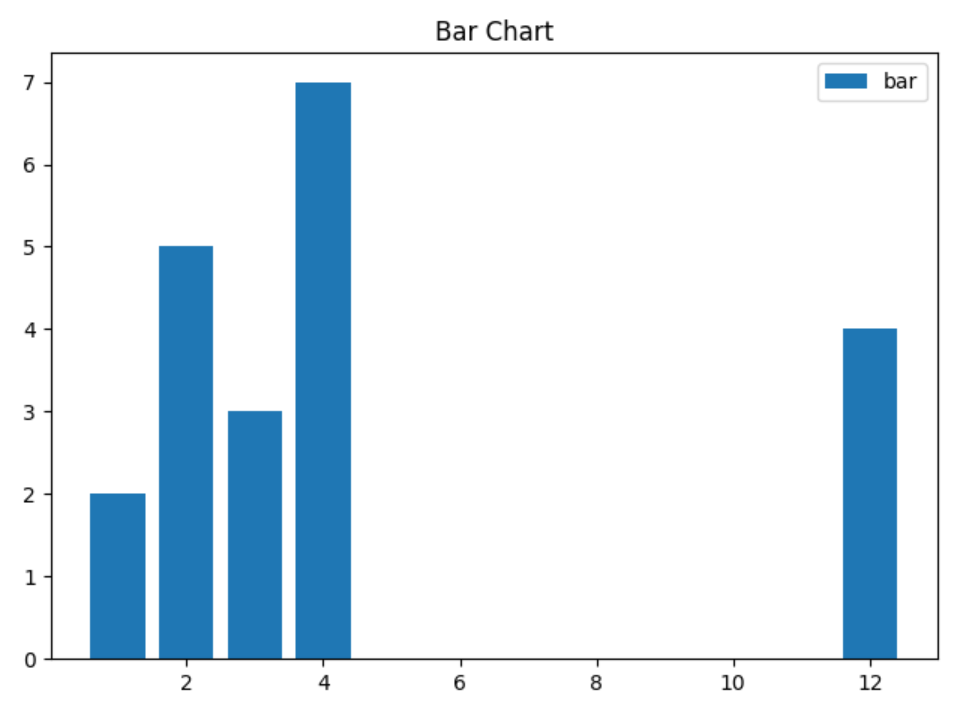
1. **LINE GRAPH**

****

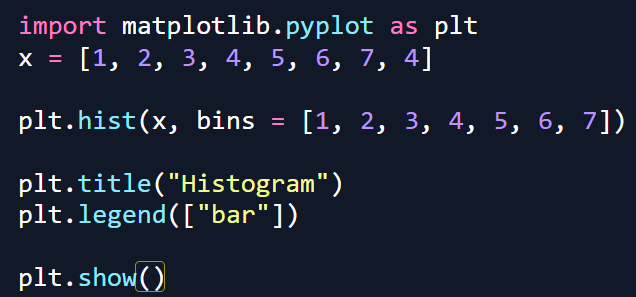


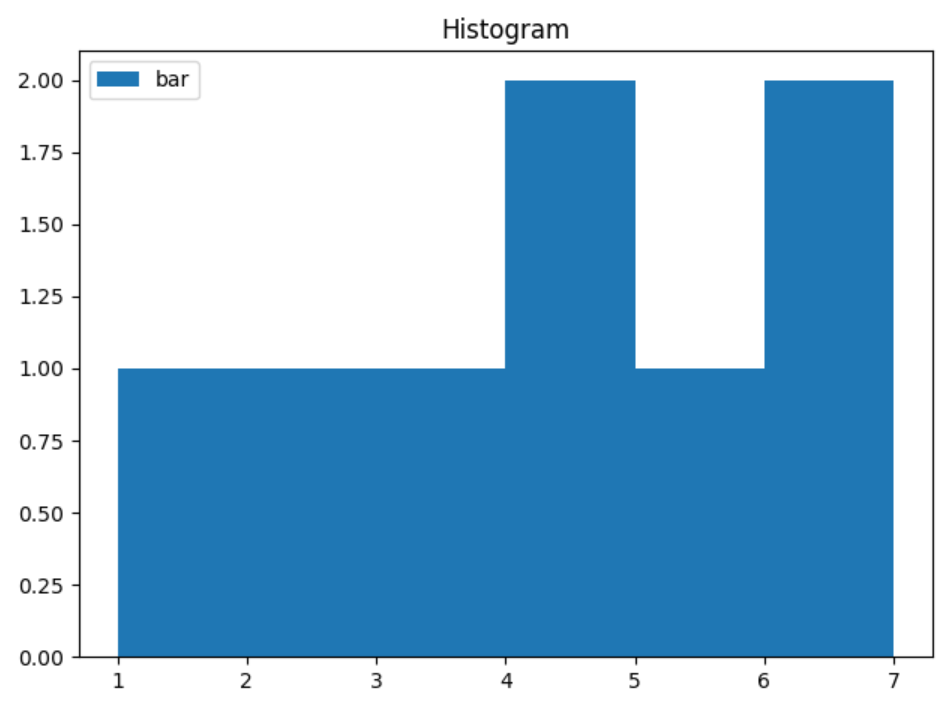
1. **BAR GRAPH**

****

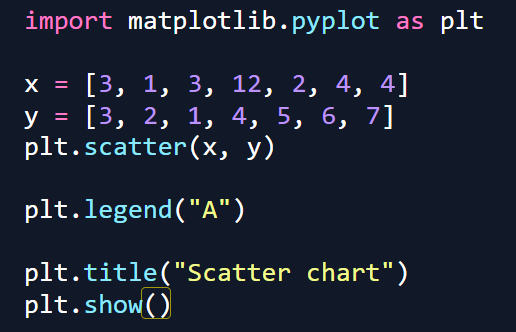
****

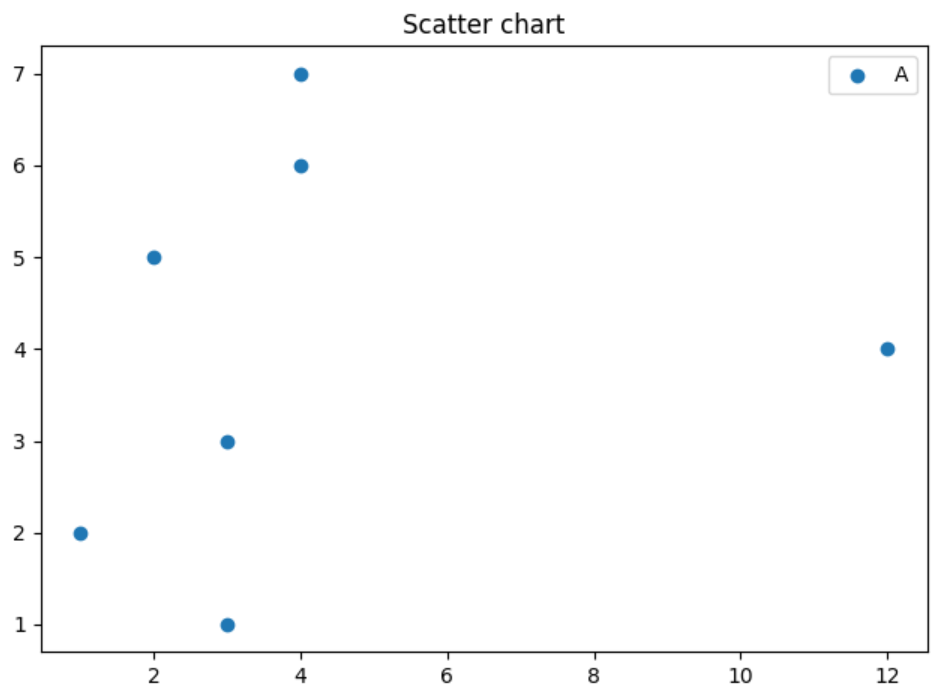
1. **HISTOGRAM**

****

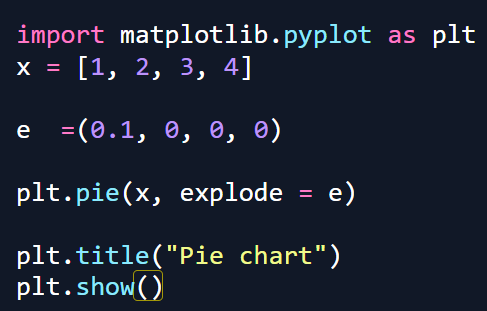
****

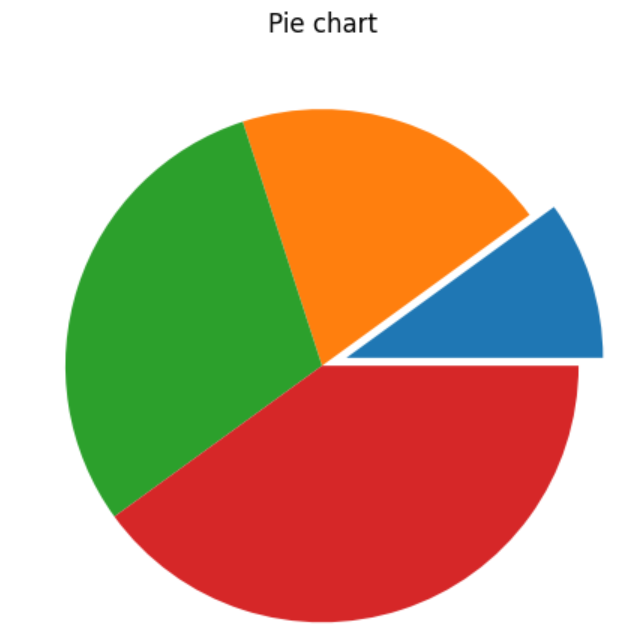
1. **SCATTER PLOT**

****

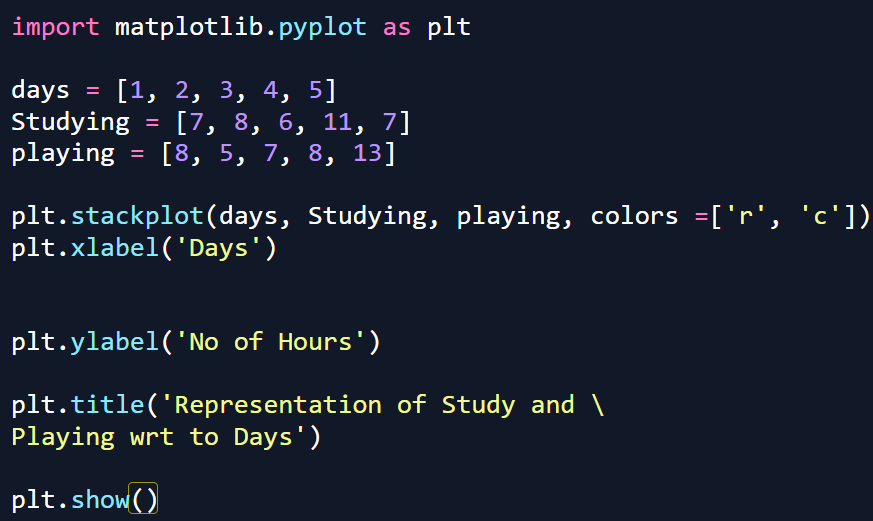
****

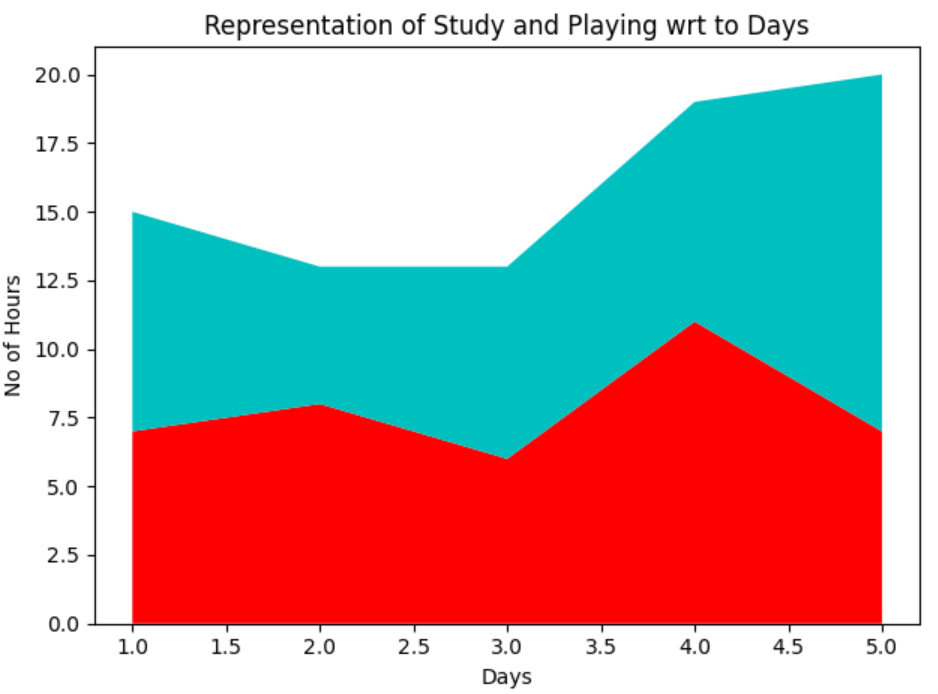
1. **PIE CHART**

****

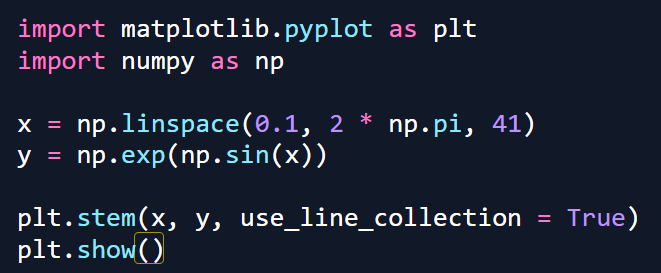
****

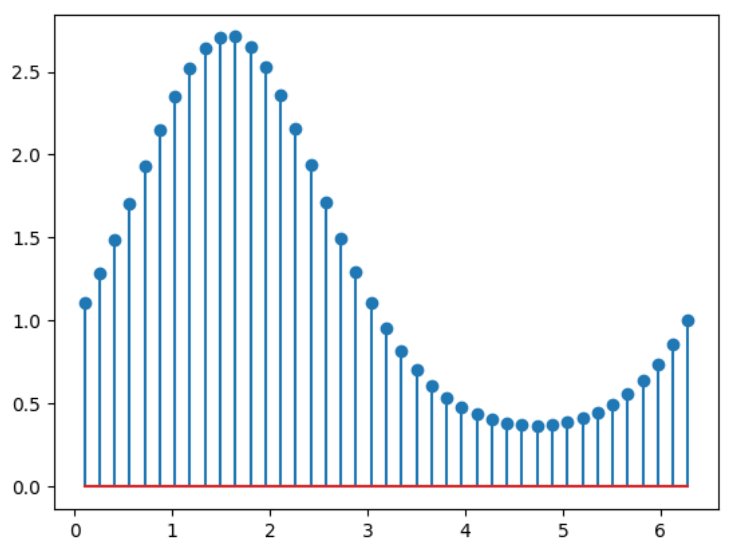
1. **STACK PLOT**

****

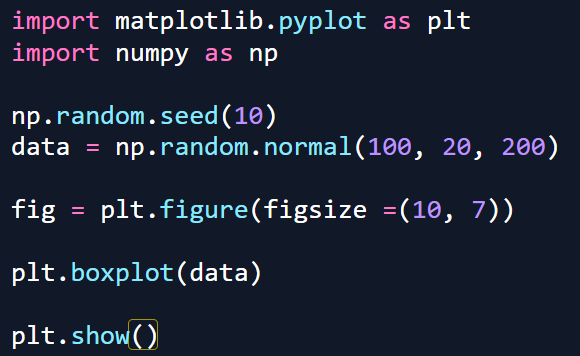
****

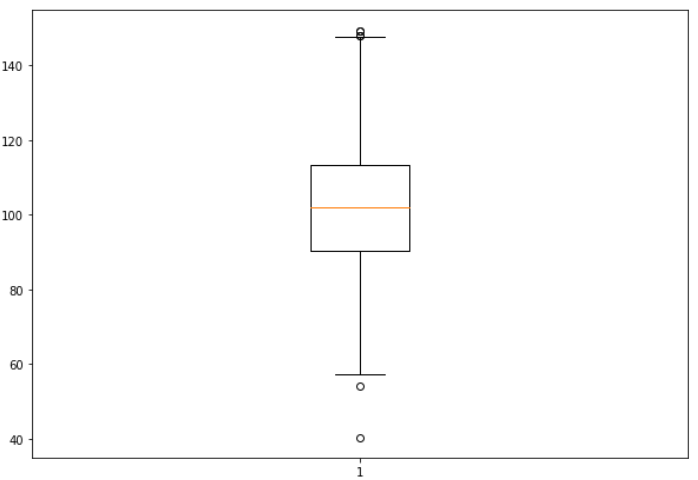
1. **STEM PLOT**

****

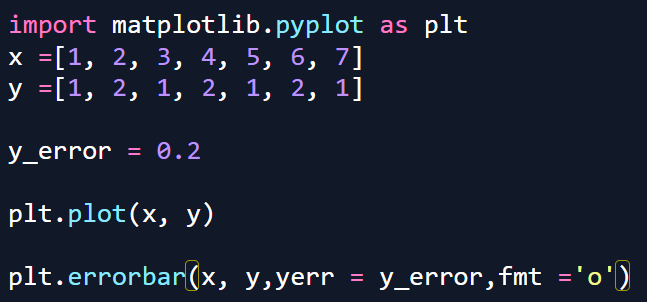
****

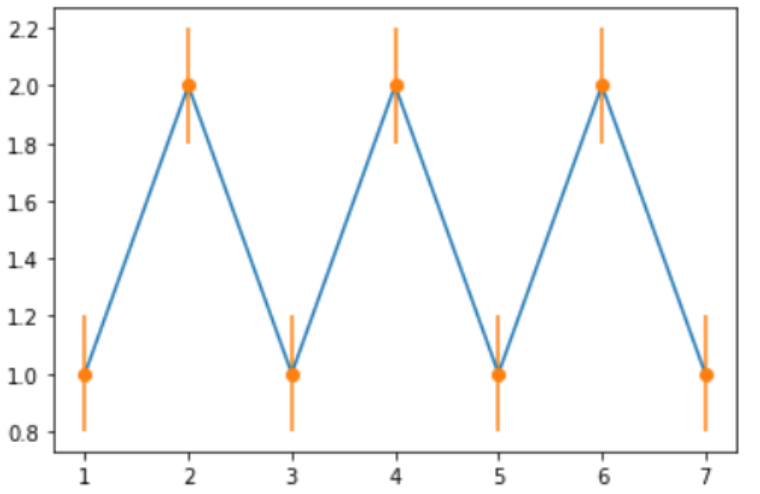
1. **BOX PLOT**

****

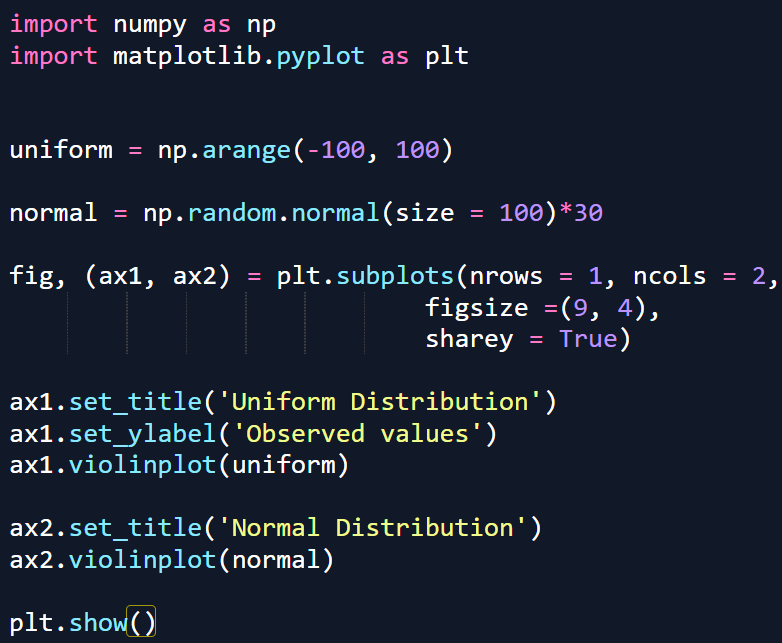
****

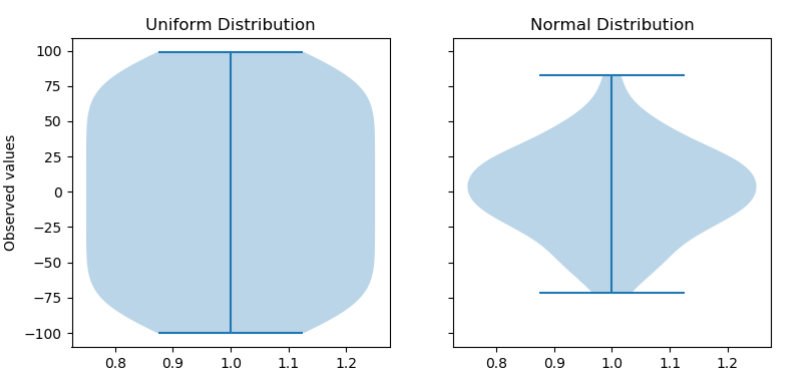
1. **ERROR PLOT**

****

****

1. **VIOLIN PLOT**

****

****

**Key Features of Matplotlib:**

* **Versatile Plotting**: Create a wide variety of visualizations, including line plots, scatter plots, bar charts, and histograms.
* **Extensive Customization**: Control every aspect of your plots, from colors and markers to labels and annotations.
* **Seamless Integration with NumPy**: Effortlessly plot data arrays directly, enhancing data manipulation capabilities.
* **High-Quality Graphics**: Generate publication-ready plots with precise control over aesthetics.
* **Cross-Platform Compatibility**: Use Matplotlib on Windows, macOS, and Linux without issues.
* **Interactive Visualizations**: Engage with your data dynamically through interactive plotting features.

**What is Matplotlib Used For?**

Matplotlib is a Python library for data visualization, primarily used to create static, animated, and interactive plots. It provides a wide range of plotting functions to visualize data effectively.

**Key Uses of Matplotlib:**

* **Basic Plots:**Line plots, bar charts, histograms, scatter plots, etc.
* **Statistical Visualization:** Box plots, error bars, and density plots.
* **Customization:**Control over colors, labels, gridlines, and styles.
* **Subplots & Layouts:**Create multiple plots in a single figure.
* **3D Plotting:**Surface plots and 3D scatter plots using mpl\_toolkits.mplot3d.
* **Animations & Interactive Plots:** Dynamic visualizations with FuncAnimation.
* **Integration:** Works well with Pandas, NumPy and Jupyter Notebooks.

**SEABORN:**

**Seaborn** is a library mostly used for statistical plotting in Python. It is built on top of Matplotlib and provides beautiful default styles and color palettes to make statistical plots more attractive.

In this tutorial, we will learn about Python Seaborn from basics to advance using a huge dataset of seaborn basics, concepts, and different graphs that can be plotted.

Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on top matplotlib library and is also closely integrated with the data structures from pandas.  
Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs so that we can switch between different visual representations for the same variables for a better understanding of the dataset.

**Different categories of plot in Seaborn**

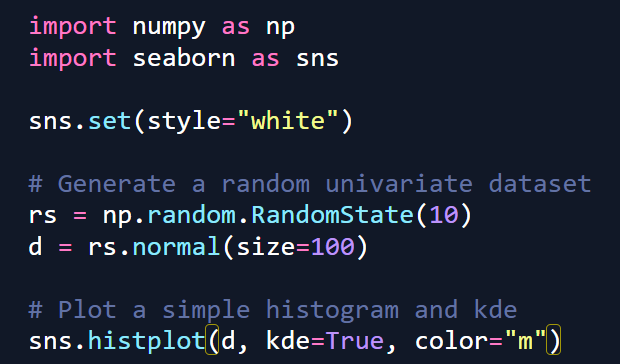
Plots are basically used for visualizing the relationship between variables. Those variables can be either completely numerical or a category like a group, class, or division.

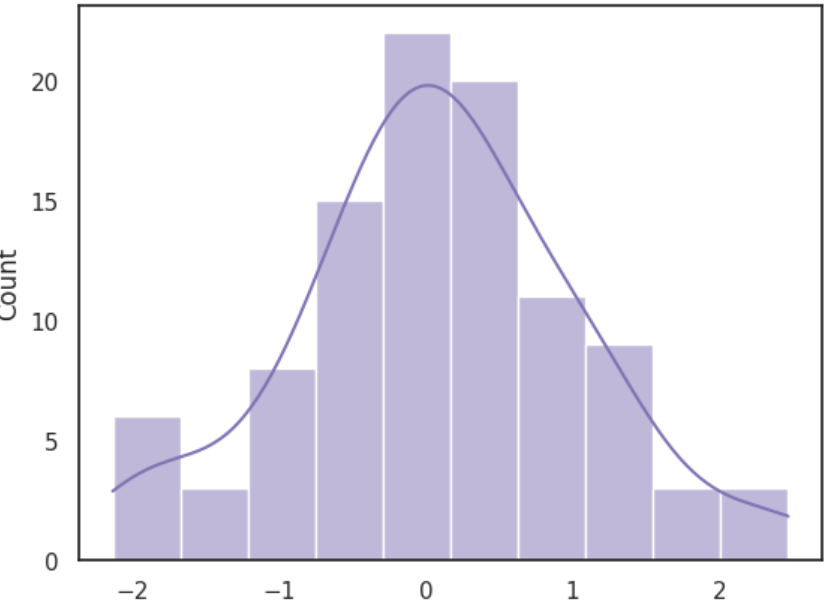
* **Relational plots:** This plot is used to understand the relation between two variables.
* **Categorical plots:**This plot deals with categorical variables and how they can be visualized.
* **Distribution plots:**This plot is used for examining univariate and bivariate distributions
* **Regression plots:**The regression plots in Seaborn are primarily intended to add a visual guide that helps to emphasize patterns in a dataset during exploratory data analyses.
* **Matrix plots:** A matrix plot is an array of scatterplots.
* **Multi-plot grids:**It is a useful approach to draw multiple instances of the same plot on different subsets of the dataset.

**Some basic plots using seaborn:**

**Histplot:**  Seaborn Histplot is used to visualize the univariate set of distributions(single variable). It plots a histogram, with some other variations like kdeplot and rugplot. The Histplot function takes several arguments but the important ones are

* **data**: This is the array, series, or dataframe that you want to visualize. It is a required parameter.
* **x**: This specifies the column in the data to use for the histogram. If your data is a dataframe, you can specify the column by name.
* **y**: This specifies the column in the data to use for the histogram when you want to create a bivariate histogram. By default, it is set to **None**, meaning that a univariate histogram will be plotted.
* **bins**: This specifies the number of bins to use when dividing the data into intervals for plotting. By default, it is set to "auto", which uses an algorithm to determine the optimal number of bins.
* **kde**: This parameter controls whether to display a kernel density estimate (KDE) of the data in addition to the histogram. By default, it is set to **False**, meaning that a KDE will not be plotted.

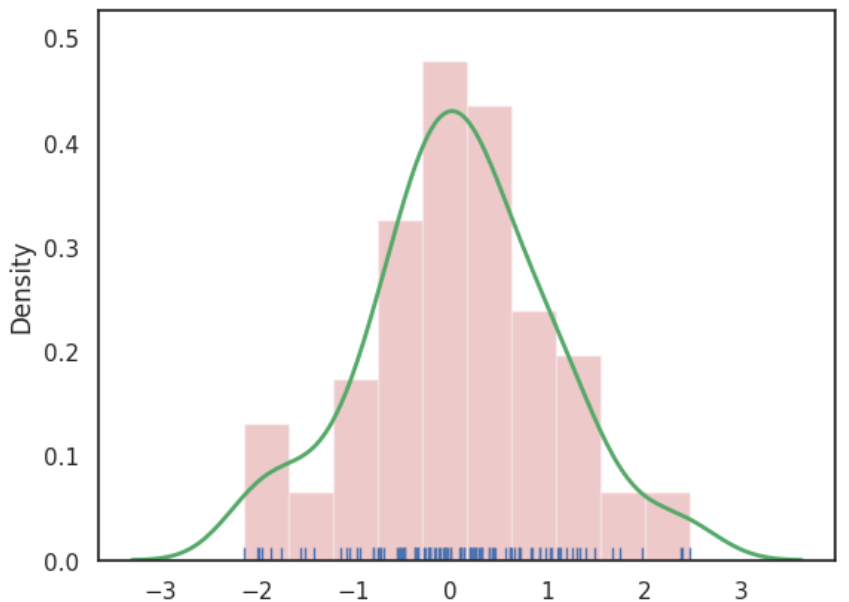




**Distplot**: Seaborn distplot is used to visualize the univariate set of distributions(Single features) and plot the histogram with some other variations like kdeplot and rugplot.

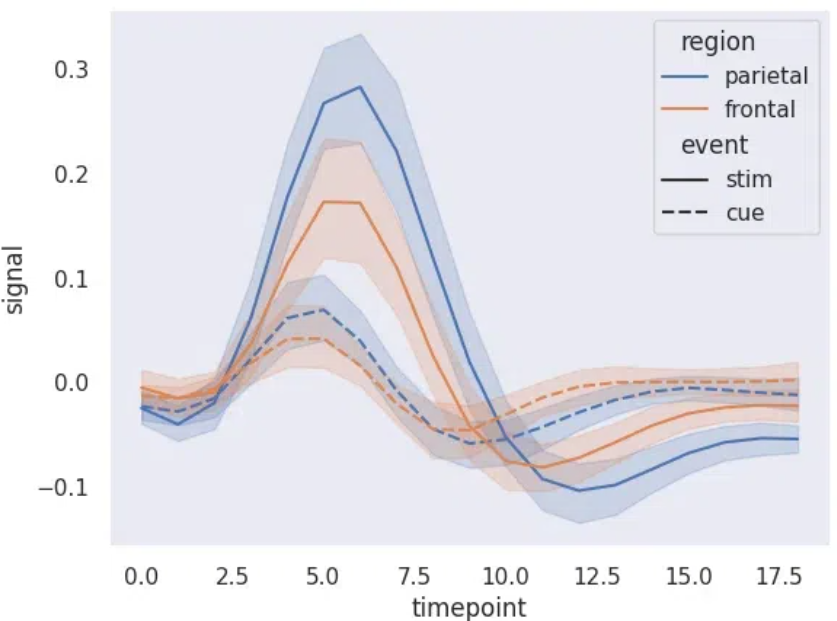
The function takes several parameters, but the most important ones are:

* **a**: This is the array, series, or list of data that you want to visualize. It is a required parameter.
* **bins**: This specifies the number of bins to use when dividing the data into intervals for plotting. By default, it is set to "auto", which uses an algorithm to determine the optimal number of bins.
* **kde**: This parameter controls whether to display a kernel density estimate (KDE) of the data in addition to the histogram. By default, it is set to **True**, meaning that a KDE will be plotted.
* **hist**: This parameter controls whether to display the histogram of the data. By default, it is set to **True**, meaning that a histogram will be plotted.

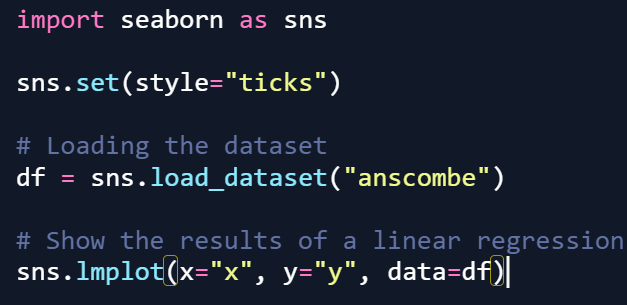


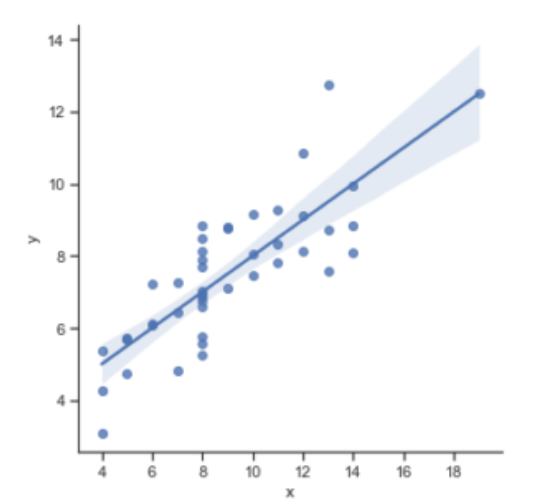
**Lineplot:** The line plot is one of the most basic plots in the seaborn library. This plot is mainly used to visualize the data in the form of some time series, i.e. in a continuous manner.





**Lmplot:**  The lmplot is another most basic plot. It shows a line representing a linear regression model along with data points on the 2D space and x and y can be set as the horizontal and vertical labels respectively.



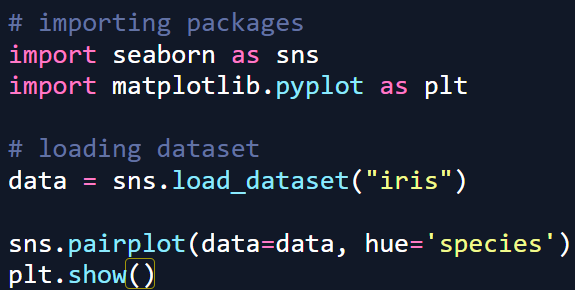


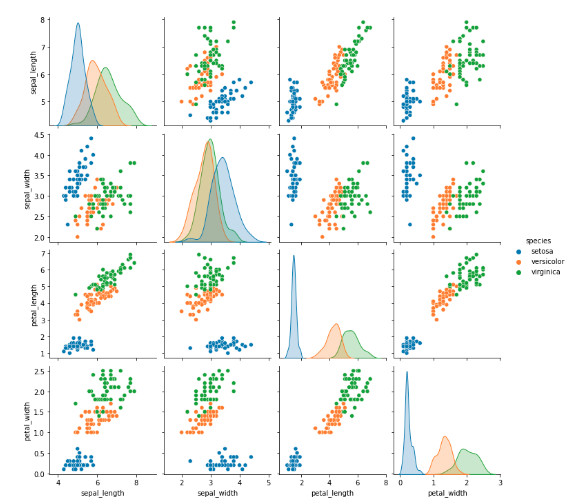
**Pairplot**:

represents pairwise relation across the entire dataframe and supports an additional argument called hue for categorical separation. What it does basically is create a jointplot between every possible numerical column and takes a while if the dataframe is really huge. It is plotted using the **pairplot()** method.

**SYNTAX:**

pairplot(data[, hue, hue\_order, palette, …])

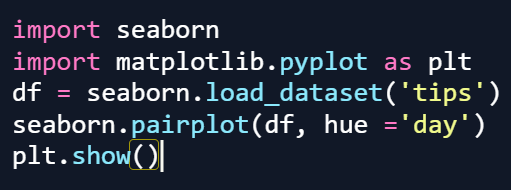


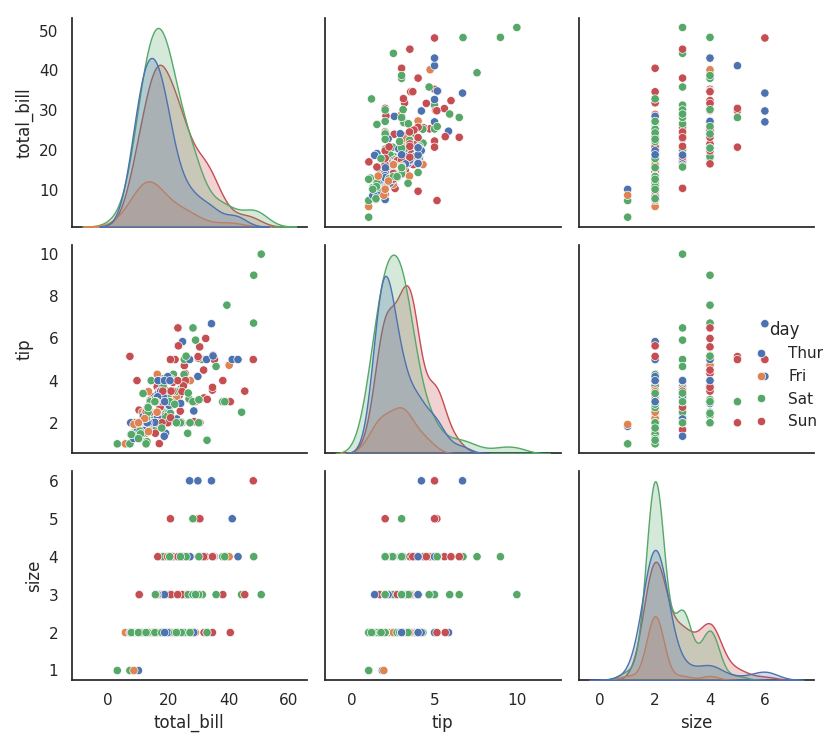


**seaborn.pairplot()**method is used for visualizing relationships between multiple variables in a dataset. By creating a grid of scatter plots it helps to identify how different features interact with each other to identify patterns, correlations and trends in data. In this article, we will see how to implement seaborn.pairplot() in python.

**EX 1 Pairplot with Hue by Day:**

We will use **hue**parameter to color-code points based on the **day**column. This helps to distinguish between different days of the week.

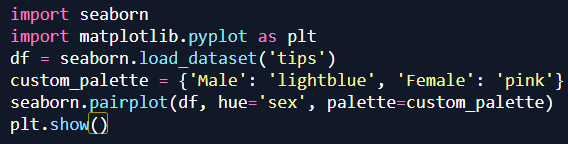


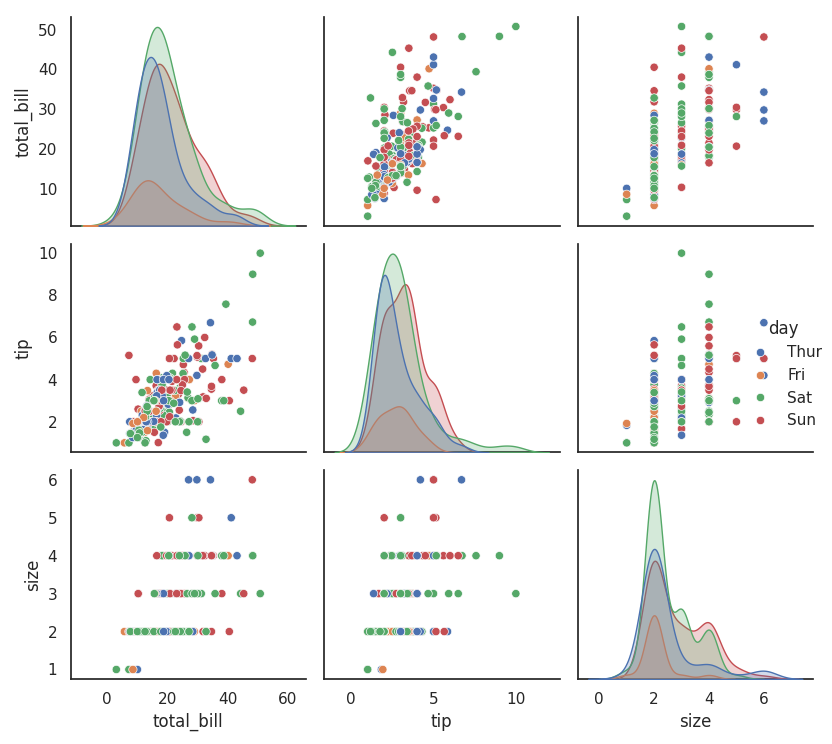


A grid of scatter plots showing the relationships between the numerical features in the **tips** dataset with color coding based on the **day**column is formed.

**Example 2: Pairplot with Custom Palette by Sex**

We will use the **hue and palette**parameter to color-code points based on the **sex**column helps in distinguishing between male and female customers. Here we defined colour palette as blue and pink using **custom\_palette**.

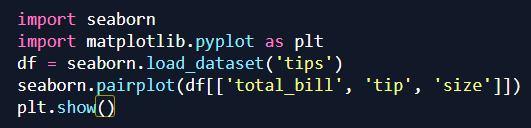


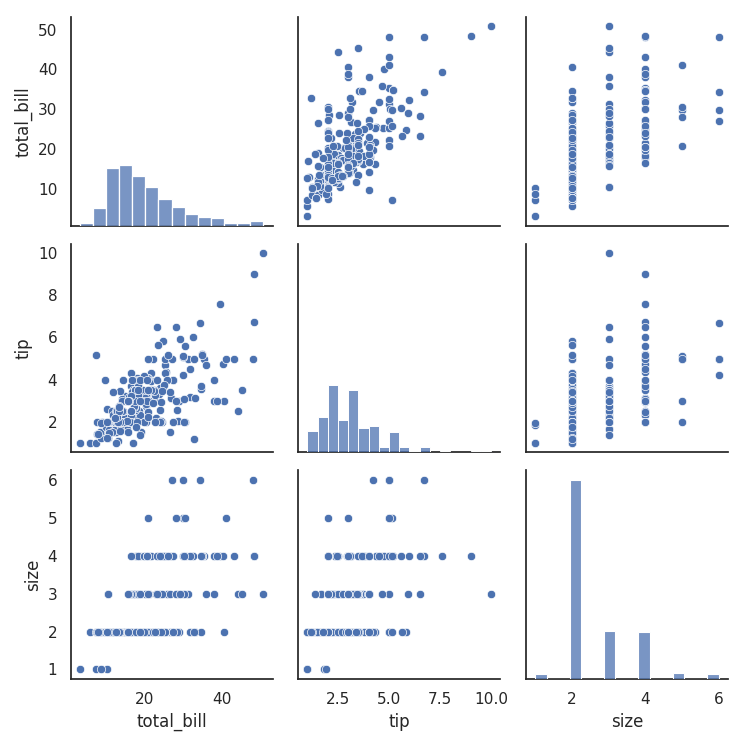


Scatter plots will be color-coded based on the **sex**column which allows us to distinguish between male and female customers.

**Example 3: Pairplot with Specific Variables**

We can focus on specific variables in the **tips** dataset. Here we visualize only the total\_bill, tip and size features using **{x, y}\_vars parameter.**



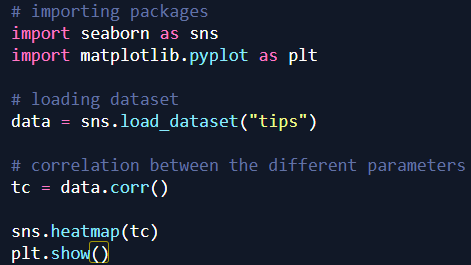


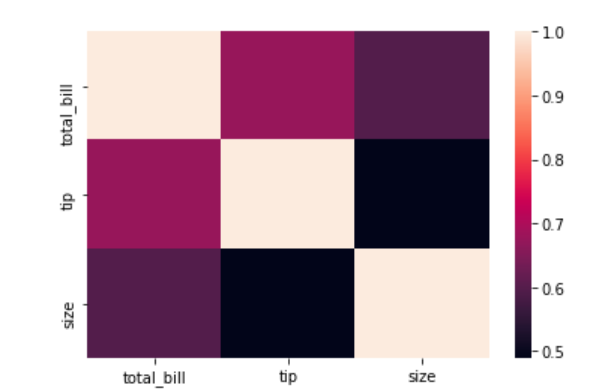
**Heatmap**

**Heatmap** is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. it can be plotted using the **heatmap()** function.

**Syntax:**

seaborn.heatmap(data, \*, vmin=None, vmax=None, cmap=None, center=None, annot\_kws=None, linewidths=0, linecolor=’white’, cbar=True, \*\*kwargs)

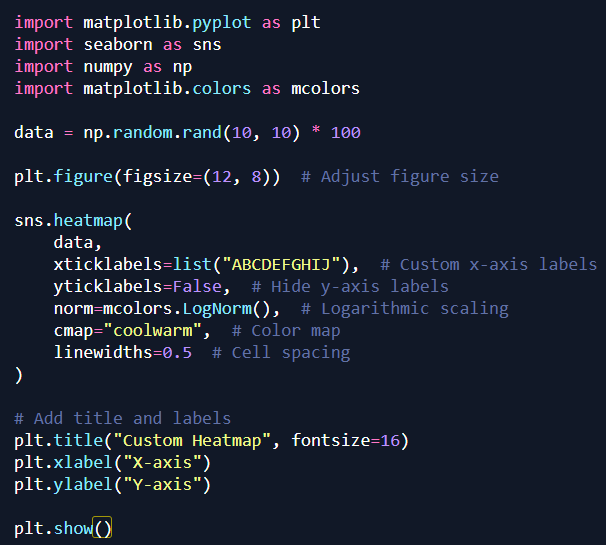


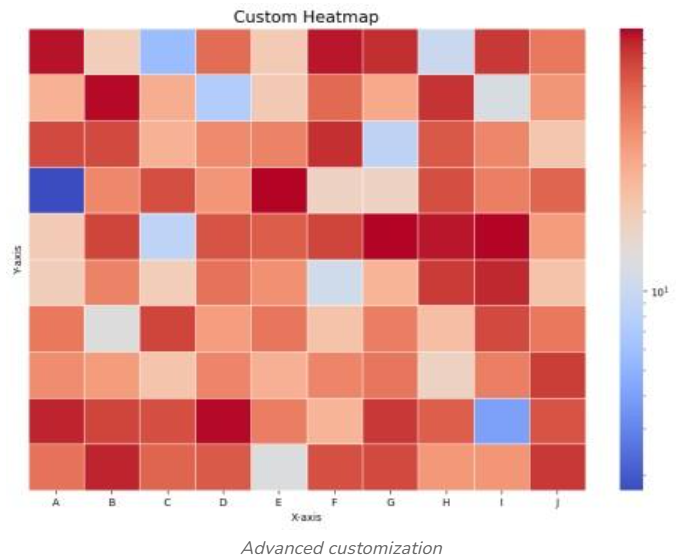


A **heatmap** is a graphical representation of data where individual values are represented by color intensity. It is widely used in data analysis and visualization to identify patterns, correlations and trends within a dataset. **Heatmaps in**Seaborn can be plotted using the**seaborn.heatmap() function**, which offers extensive customization options. Let's explore different methods to create and enhance heatmaps using Seaborn.

**Advanced customizations in seaborn heatmap:**

**Seaborn's heatmap()** functionprovides various options for customization, allowing users to enhance visualization by adjusting color schemes, labels, scaling and spacing.





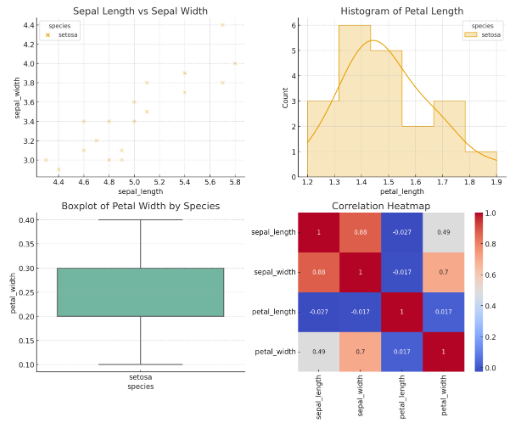
**Explanation:**

* **plt.figure(figsize=(12, 8))**enlarges the figure for better visibility.
* **xticklabels=list("ABCDEFGHIJ")**replaces default labels with A-J.
* **yticklabels=False** removes y-axis labels for clarity.
* **norm=mcolors.LogNorm()** enhances contrast for varied data.
* **cmap="coolwarm" u**ses a blue-to-red gradient.
* **linewidths=0.5** adds thin separation lines.
* **plt.title("Custom Heatmap", fontsize=16)** adds a title with a readable font size.
* **plt.xlabel("X-axis") & plt.ylabel("Y-axis")**labels the axes for clear understanding.

**PROJECT:**

**Creating a dashboard for visualizing relationships between features in a dataset**

****

****

**1. Scatter Plot: Sepal Length vs Sepal Width**

* **Observation:** The scatter plot shows clusters corresponding to the three species: *Setosa*, *Versicolor*, and *Virginica*.
* *Setosa* is clearly separated with smaller sepal length and higher sepal width.
* *Versicolor* and *Virginica* overlap somewhat but can be distinguished by larger sepal length in *Virginica*.
* **Conclusion:** Sepal measurements are helpful but not always sufficient for perfect classification.

**2. Histogram: Petal Length Distribution**

* **Observation:** The histogram shows three distinct peaks corresponding to the three species.
* *Setosa* has short petal lengths (around 1–2 cm).
* *Versicolor* lies in the middle range (3–5 cm).
* *Virginica* has the largest petal lengths (5–7 cm).
* **Conclusion:** Petal length is a strong distinguishing feature between species.

**3. Boxplot: Petal Width across Species**

* **Observation:** The boxplot shows clear separation:
  + *Setosa* has the smallest petal width (0.1–0.6 cm).
  + *Versicolor* has medium widths (1–1.8 cm).
  + *Virginica* has the largest (1.8–2.5 cm).
* There is little overlap between species, making petal width an excellent classification feature.
* **Conclusion:** Petal width is one of the most reliable predictors of species.

**4. Heatmap: Feature Correlations**

* **Observation:**
  + Petal length and petal width are **highly correlated** (~0.96).
  + Sepal length is moderately correlated with petal size.
  + Sepal width shows weak or negative correlation with other features.
* **Conclusion:** Petal measurements (length & width) are better predictors of species than sepal measurements.