Matplotlib

created by :
The easylearn academy

What is Matplotlib

- **Matplotlib** is an open-source Python library for generating static, animated, and interactive visualizations.
- It was created by John D. Hunter in 2003 and is now maintained by a large community.
- Its flexibility and compatibility with other Python libraries like NumPy, Pandas, and SciPy make it a go-to tool for AI and ML practitioners.
- Open-source under a BSD-style license.

- **Purpose in AI/ML:**
  - Visualizations help understand data distributions, identify patterns, detect outliers, evaluate model performance, and communicate insights effectively.

- **Core Module:**
  - matplotlib.pyplot is the primary interface for creating plots, offering a MATLAB-like plotting experience.

Key Features

- **Plot Types:**
  - Line plots, scatter plots, bar charts, histograms, pie charts, box plots, heatmaps, and more.
  - Supports 2D and basic 3D plotting.
- **Customization:**
  - Control over titles, labels, colors, fonts, gridlines, legends, and more.
  - Create publication-quality figures.
- **Integration:**
  - Works seamlessly with Jupyter Notebooks, Pandas DataFrames, and NumPy arrays.
  - Compatible with AI/ML libraries like Scikit-learn, TensorFlow, and PyTorch.
- **Output Formats:**
  - Save plots as PNG, JPEG, SVG, PDF, etc.
  - Supports high-resolution outputs for reports or presentations.
- **Interactivity:**
  - Interactive plots in Jupyter (e.g., using %matplotlib inline).
  - Limited interactivity compared to tools like Plotly but supports basic zooming/panning.

Advantages of Matplotlib

- Simplifies complex Matplotlib tasks.
- Produces publication-quality visuals with minimal code.
- Excellent for exploratory data analysis (EDA).
- Customizable via Matplotlib for advanced tweaks.

Common Uses in AI/ML

- **Exploratory Data Analysis (EDA):**
  - Visualize data distributions, correlations, and outliers (e.g., histograms, scatter plots).
- **Model Evaluation:**
  - Plot metrics like loss curves, accuracy, or confusion matrices.
- **Feature Analysis:**
  - Display feature importance or relationships.
- **Result Communication:**
  - Create clear visuals for reports or presentations.

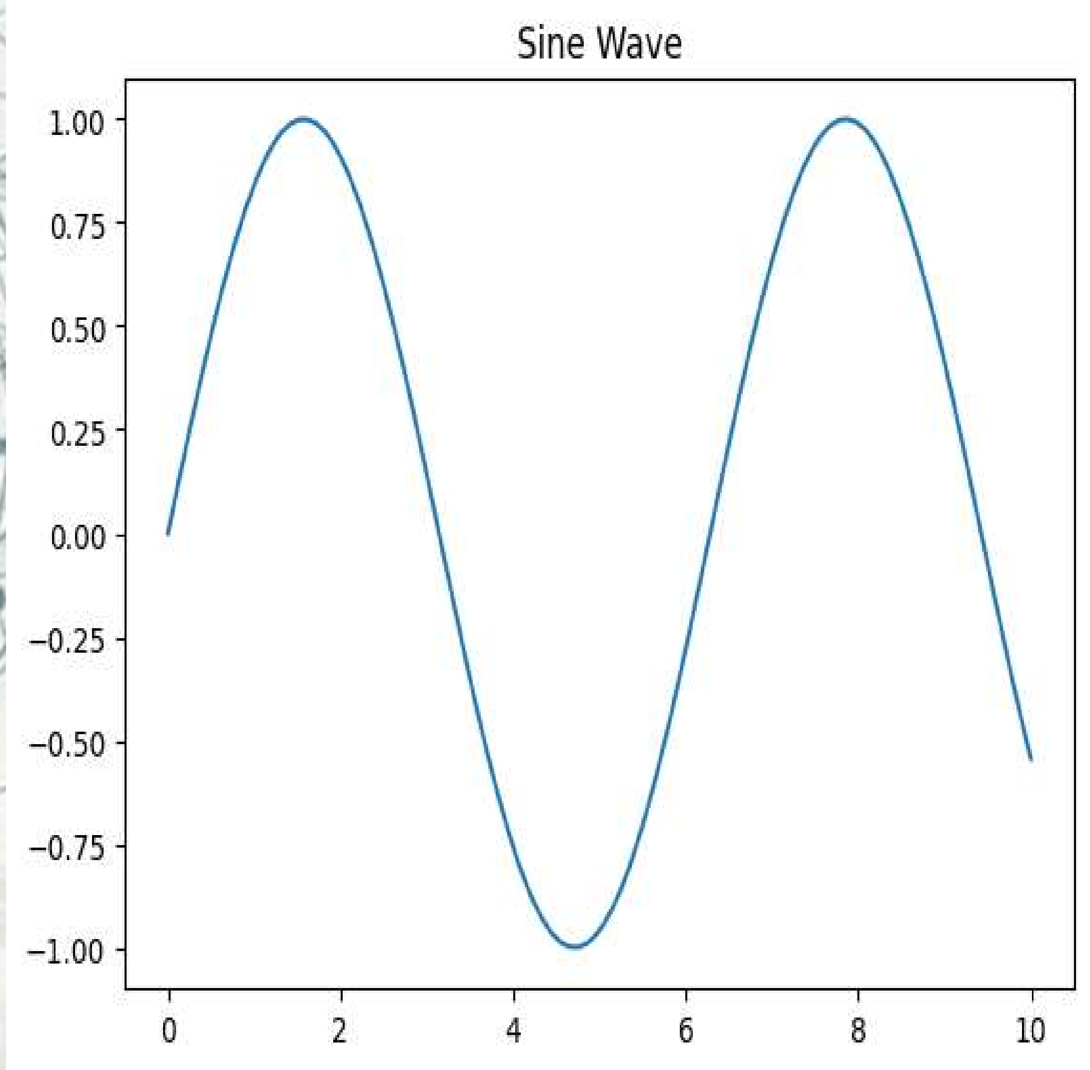
How to Get Started

- Install Matplotlib
  - pip install matplotlib

- **Basic Workflow:**
  - Import matplotlib.pyplot as plt.
  - Create data (e.g., using NumPy or Pandas).
  - Use plotting functions like plt.plot(), plt.scatter(), etc.
  - Customize with labels, titles, etc.
  - Display with plt.show() or save with plt.savefig().

Example

- import matplotlib.pyplot as plt
- import numpy as np
- x = np.linspace(0, 10, 100)
- y = np.sin(x)
- plt.plot(x, y)
- plt.title("Sine Wave")
- plt.show()



Type of Plot

- Line Plot
- Bar Graph
- Scatter Plot
- Step Plot
- Pie Chart
- Heat map
- Violin Plot
- Histogram
- Box Plot
- Candlestick Plot
- Polar Graph

Some common Method use in all graph

- **Sets the title of the graph. :-**
  - plt.title(label, fontsize=None, fontweight=None, loc='center', pad=None)
    - **label** - String for the title.
    - **fontsize** - Size of the title text (e.g., 12, 'large').
    - **fontweight** - Weight of the font (e.g., 'bold', 'normal').
    - **loc** - Title position ('center', 'left', 'right').
    - **pad** - Padding between title and plot (in points).
- **Sets the label for the x-axis. :-**
  - plt.xlabel(label, fontsize=None, fontweight=None, labelpad=None)
    - **label** - String for the x-axis label.
    - **labelpad** - Padding between label and axis (in points).
- **Sets the label for the y-axis. :-**
  - plt.ylabel(label, fontsize=None, fontweight=None, labelpad=None)
    - **label-** String for the y-axis label.

Some common Method use in all graph

- **Displays a legend for labeled elements (e.g., lines, bars). :-**
  - plt.legend(loc='best', fontsize=None, title=None)
    - **loc**: Position of legend (e.g., 'best', 'upper right', 'lower left').
    - **title**: Title for the legend.
- **Adds a grid to the plot for better readability. :-**
  - plt.grid(visible=True, which='major', axis='both', linestyle='--', linewidth=0.5)
    - **visible**: Show/hide grid (True/False).
    - **which**: Grid lines to show ('major', 'minor', 'both').
    - **axis**: Apply grid to 'x', 'y', or 'both'.
    - **linestyle**: Style of grid lines (e.g., '--', ':').
    - **linewidth**: Thickness of grid lines.

Some common Method use in all graph

- **Sets the range of the x-axis. :-**
  - plt.xlim(left=None, right=None)
    - **left**: Minimum x-axis value.
    - **right**: Maximum x-axis value.
- **Sets the range of the y-axis. :-**
  - plt.ylim(bottom=None, top=None)
    - **bottom**: Minimum y-axis value.
    - **top**: Maximum y-axis value.

Some common Method use in all graph

- **Customizes x-axis tick marks and labels. :-**
  - plt.xticks(ticks=None, labels=None, rotation=None, fontsize=None)
    - **ticks**: Positions of ticks (e.g., [1, 2, 3]).
    - **labels**: Labels for ticks (e.g., ['A', 'B', 'C']).
    - **rotation**: Angle of tick labels (e.g., 45 for diagonal).
    - **fontsize**: Size of tick labels.
- **Customizes y-axis tick marks and labels. :-**
  - plt.yticks(ticks=None, labels=None, rotation=None, fontsize=None)
    - Same as x-tick

How to Choose a Plot Type

Basic Plots
- Line Plot
- Scatter Plot
- Bar Chart
- Horizontal Bar Char
- Histogram
- Box Plot
- Pie Chart
- Area Plot
- Step Plot

Advanced 2D Plots
- Contour Plot
- Filled Contour Plot
- Heatmap
- Hexbin Plot
- Quiver Plot
- Stream Plot
- Error Bar Plot
- Stacked Bar Chart
- Stacked Area Plot
- Matrix Plot
- Polar Plot

Finance & Statistics
- Candle Stick Plot
- Violin Plot
- Lag Plot

Multivariate & Clustering
- Pair Plot
- Joint Plot
- Facet Grid
- Dendrogram
- Ternary Plot
- 3D & Surface Plots

3D Line Plot
- 3D Scatter Plot
- 3D Surface PlotTime Series

Time Series
- Time Series Plot

Line Plot

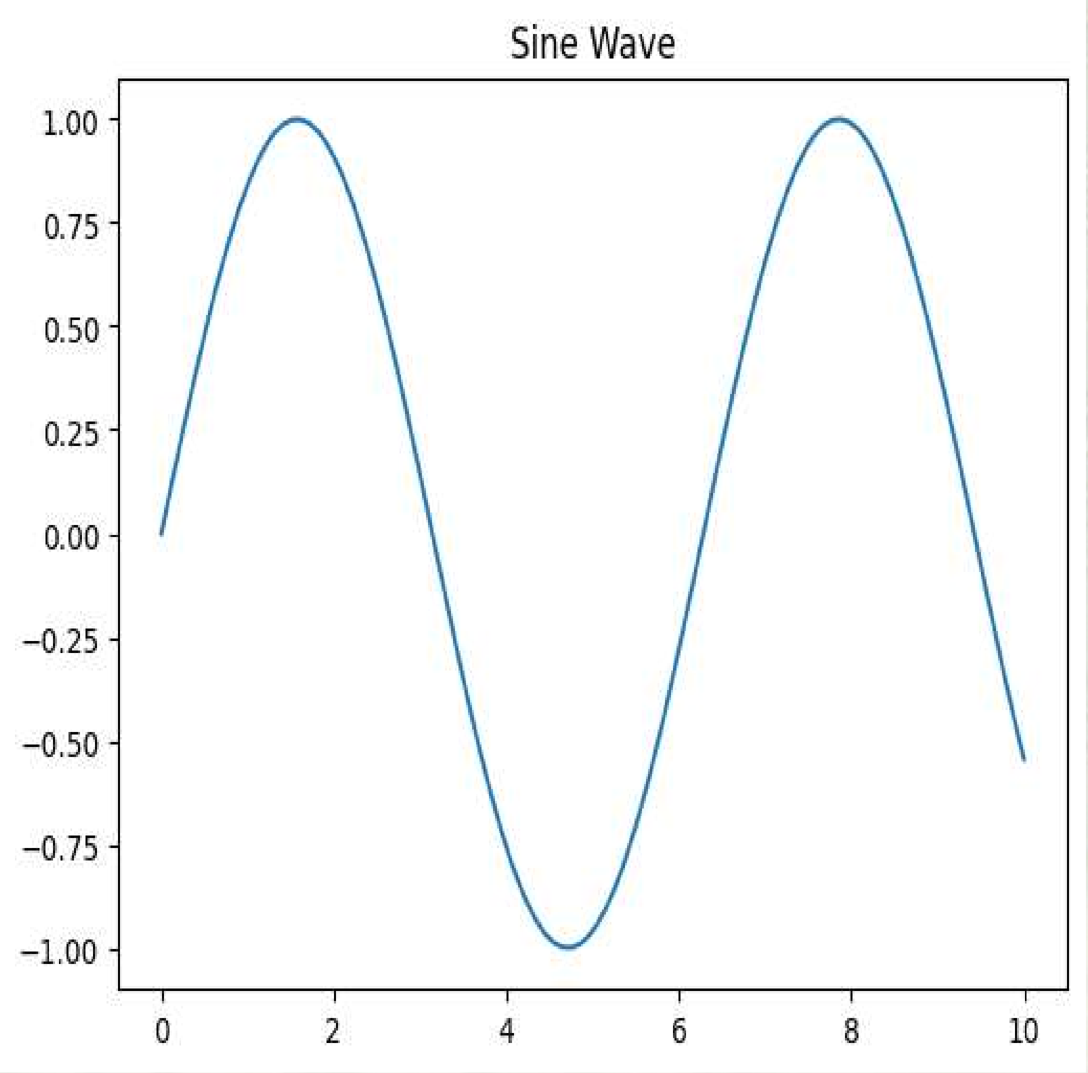
- **Purpose:**
  - Displays data points connected by straight lines to show trends or relationships over a continuous variable.
- **Use Case:**
  - Visualize training and validation loss/accuracy over epochs in neural networks.
  - Plot time series predictions (e.g., stock prices, sensor data).
  - Show model performance trends (e.g., learning curves).
- **Key Features:**
  - Smooth visualization of continuous data.
  - Supports multiple lines, custom styles (e.g., dashed, solid), and legends.
  - Easy to annotate and customize axes.

Line Plot

- **Syntax:**
  - plt.plot(x, y, color=None, linestyle=None, marker=None, linewidth=None, label=None)
- **Key Attribute:**
  - **x, y:** Data for x and y axes (arrays or lists of equal length).
  - **color (optional):** Color of the line (e.g., 'blue', 'r', or hex code).
  - **linestyle (optional):** Style of the line (e.g., '-' for solid, '--' for dashed, ':' for dotted).
  - **marker (optional):** Marker style for data points (e.g., 'o' for circles, '^' for triangles, None for no markers).
  - **linewidth (optional):** Thickness of the line (e.g., 1.5 for thicker lines).
  - **label(common):** Label for the line (used in legend).

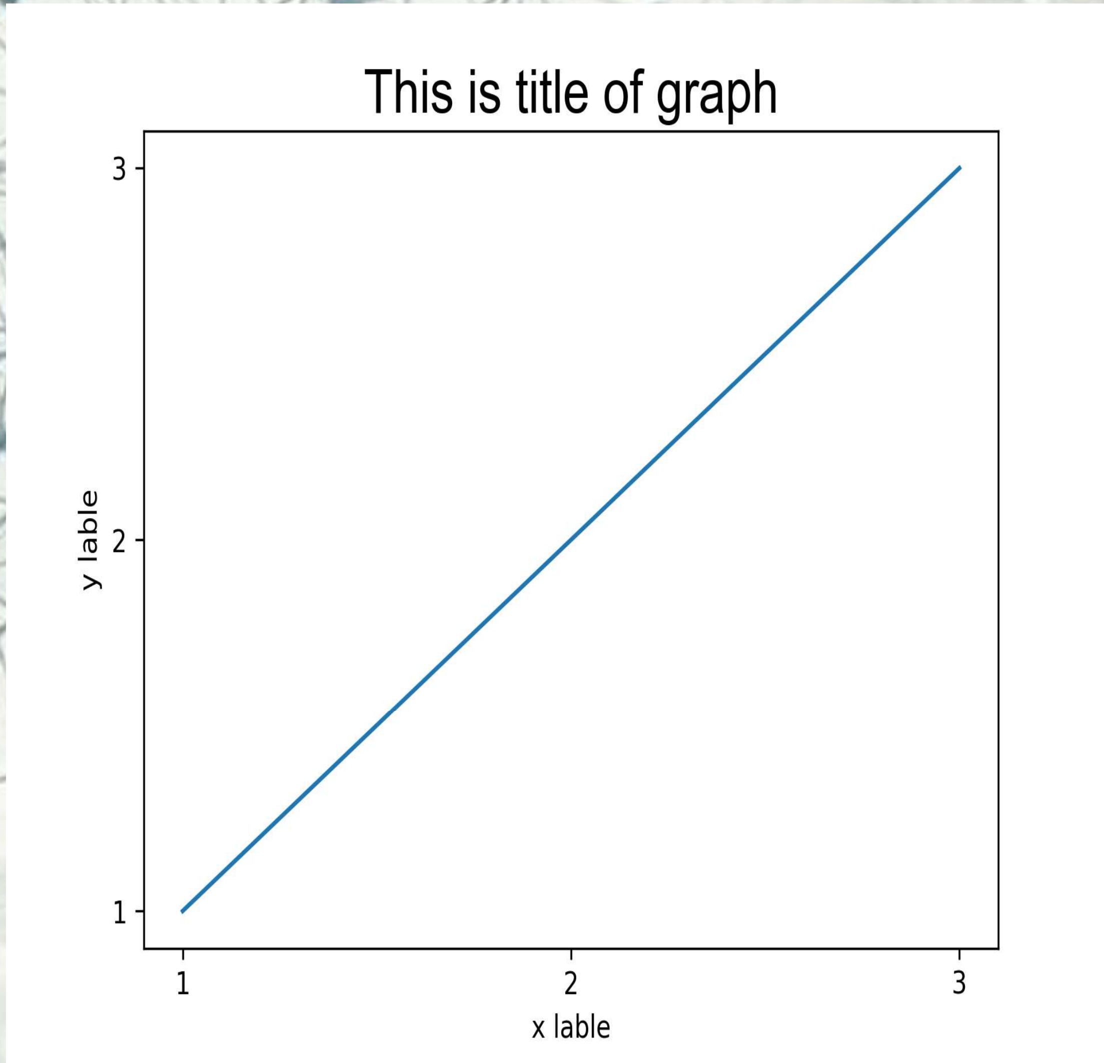
Example

- import matplotlib.pyplot as plt
- import numpy as np
- x = np.linspace(0, 10, 100)
- y = np.sin(x)
- plt.plot(x, y)
- plt.title("Sine Wave")
- plt.show()



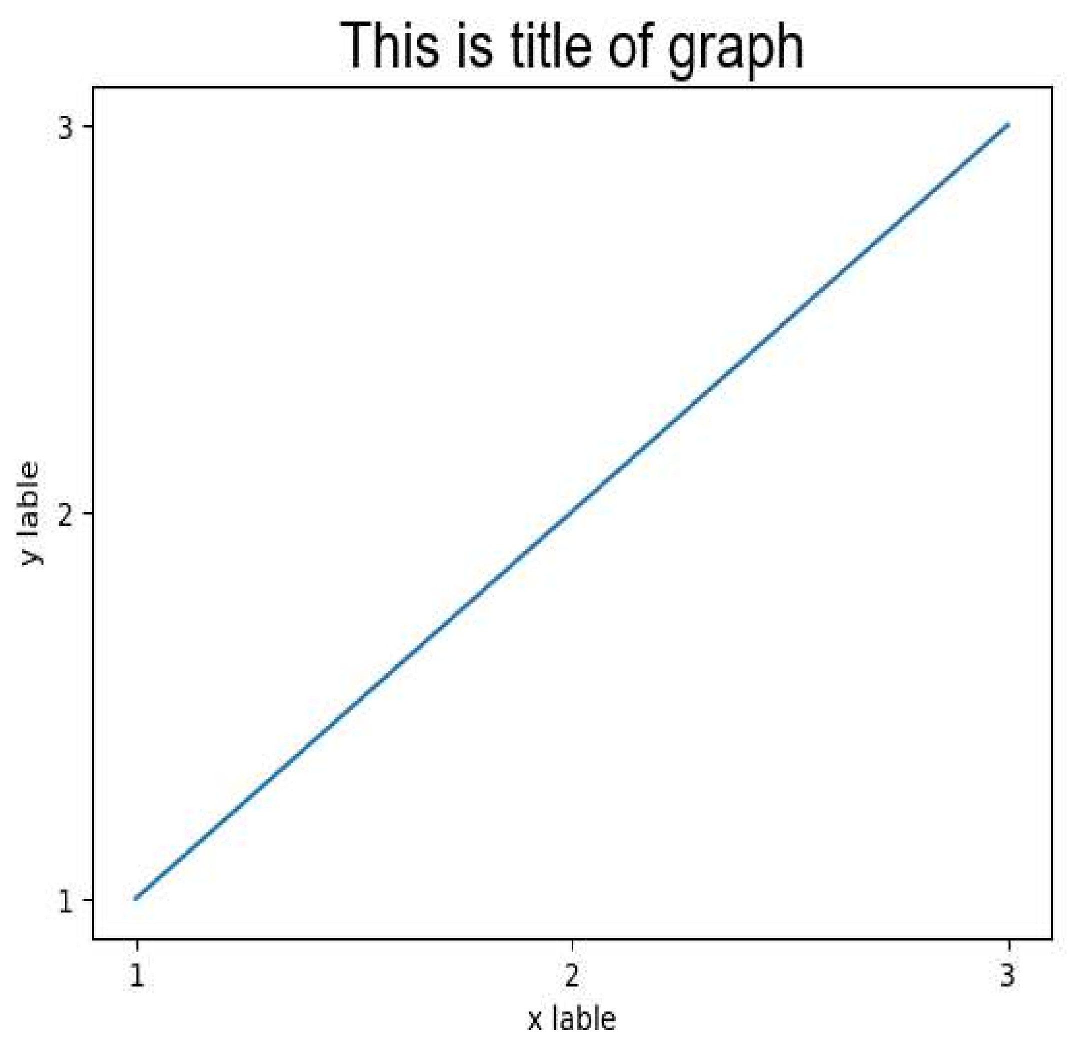
Example

- x1 = np.arange(0,10)
- y1 = np.arange(10,20)
- plt.figure(figsize=(3,2), dpi=100)
- plt.plot(x1,y1)
- plt.xlabel('X Label') **# for lable x**
- plt.ylabel('Y Label') **# for lable y**
- plt.title('Demo Graph') **# for title of visualization**
- plt.show()



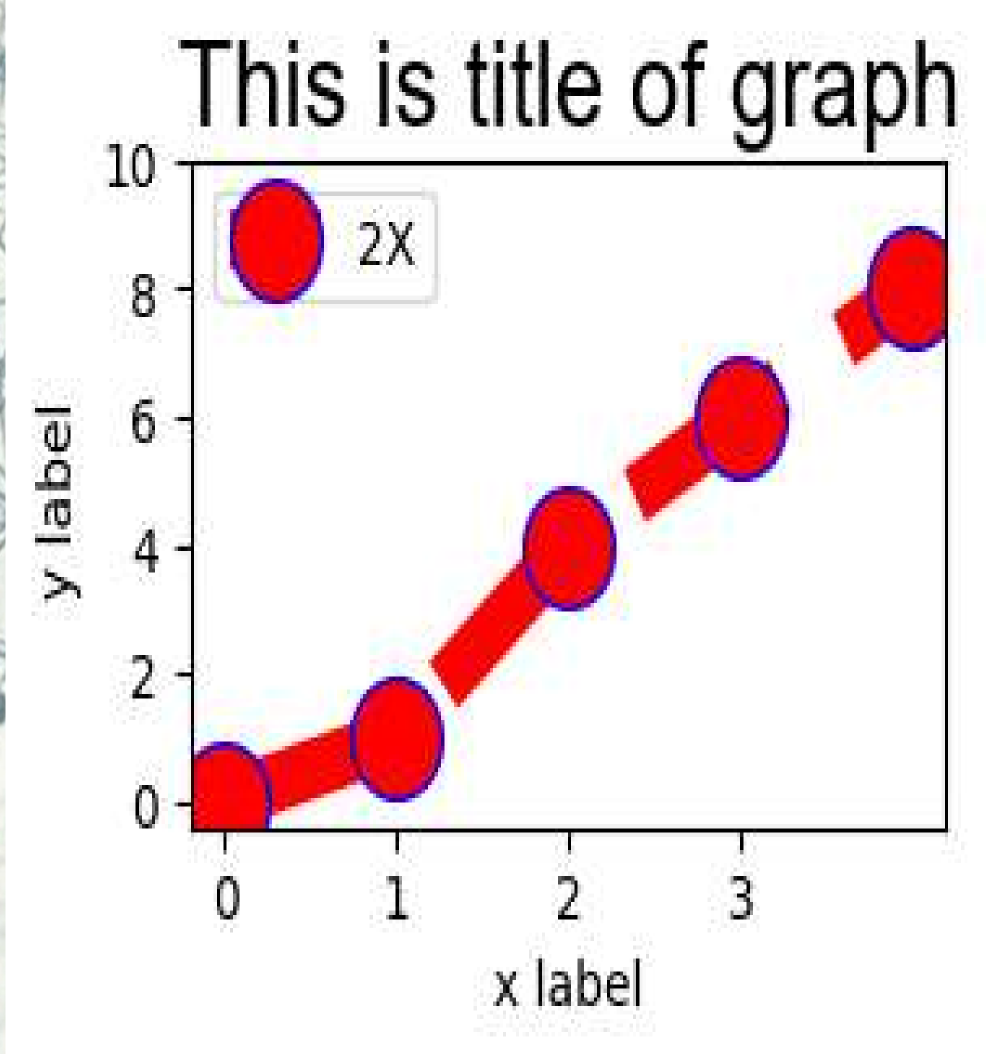
Example

- x =[1,2,3]
- y= [1,2,3]
- plt.plot(x,y) **# show graph**
- **# for x and y lable**
- plt.xlabel('x lable')
- plt.ylabel('y lable')
- **# for title**
- plt.title('This is title of graph' ,fontdict={'fontname':'Arial' , 'fontsize' : 20 })
- **# for x and y axis set**
- plt.xticks([1,2,3])
- plt.yticks([1,2,3])
- plt.savefig("linegraph1.png",dpi=300)
- plt.show()



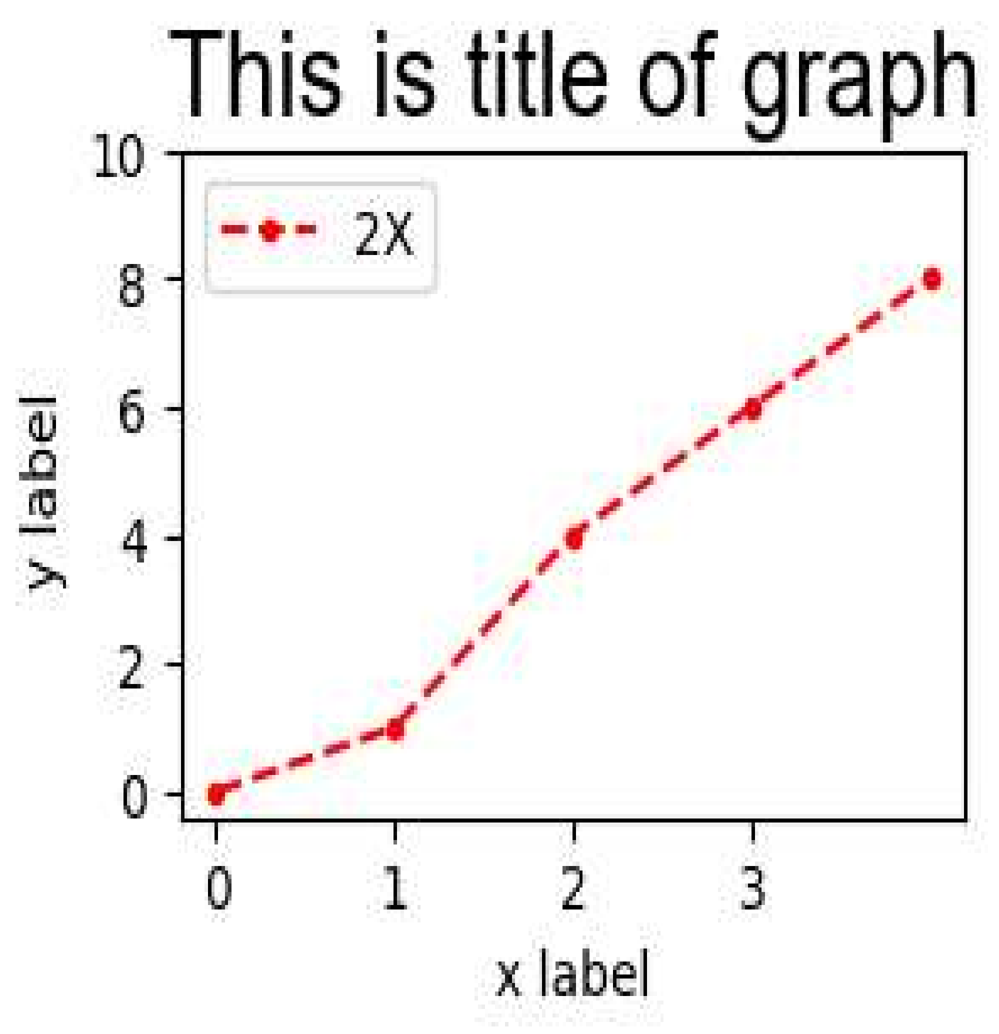
Example

- x =[0,1,2,3,4]
- y= [0,1,4,6,8]
- plt.figure(figsize=(3,2), dpi=100)**#set size of chart**
- **# set style of line and marker**
- plt.plot(x,y,label='2X',color='red',linewidth='10', markersize='20',marker='o',linestyle='--', markeredgecolor='b')
- plt.xlabel('x label')
- plt.ylabel('y label')
- plt.title('This is title of graph' ,fontdict={'fontname':'Arial' , 'fontsize' : 20 })
- plt.xticks([0,1,2,3])
- plt.yticks([0,2,4,6,8,10])
- **# set lable in graph**
- plt.legend()
- plt.show()



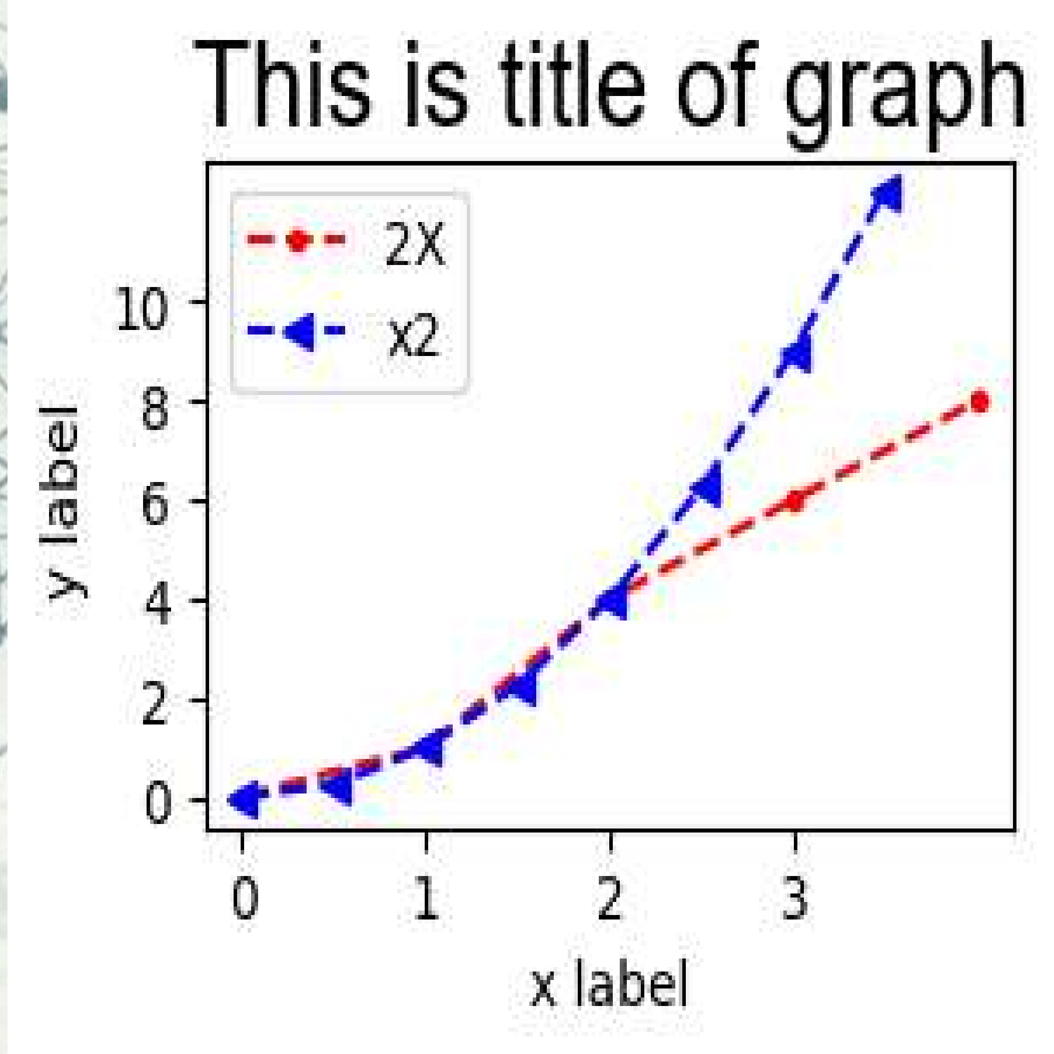
Example

- x =[0,1,2,3,4]
- y= [0,1,4,6,8]
- plt.figure(figsize=(3,2), dpi=100)
- **#use of shorthend notation 'color of line, marker, style of line'**
- plt.plot(x,y,'r.--', label='2X')
- plt.xlabel('x label')
- plt.ylabel('y label')
- plt.title('This is title of graph' ,fontdict= {'fontname':'Arial' , 'fontsize' : 20 })
- plt.xticks([0,1,2,3])
- plt.yticks([0,2,4,6,8,10])
- plt.legend()
- plt.show()



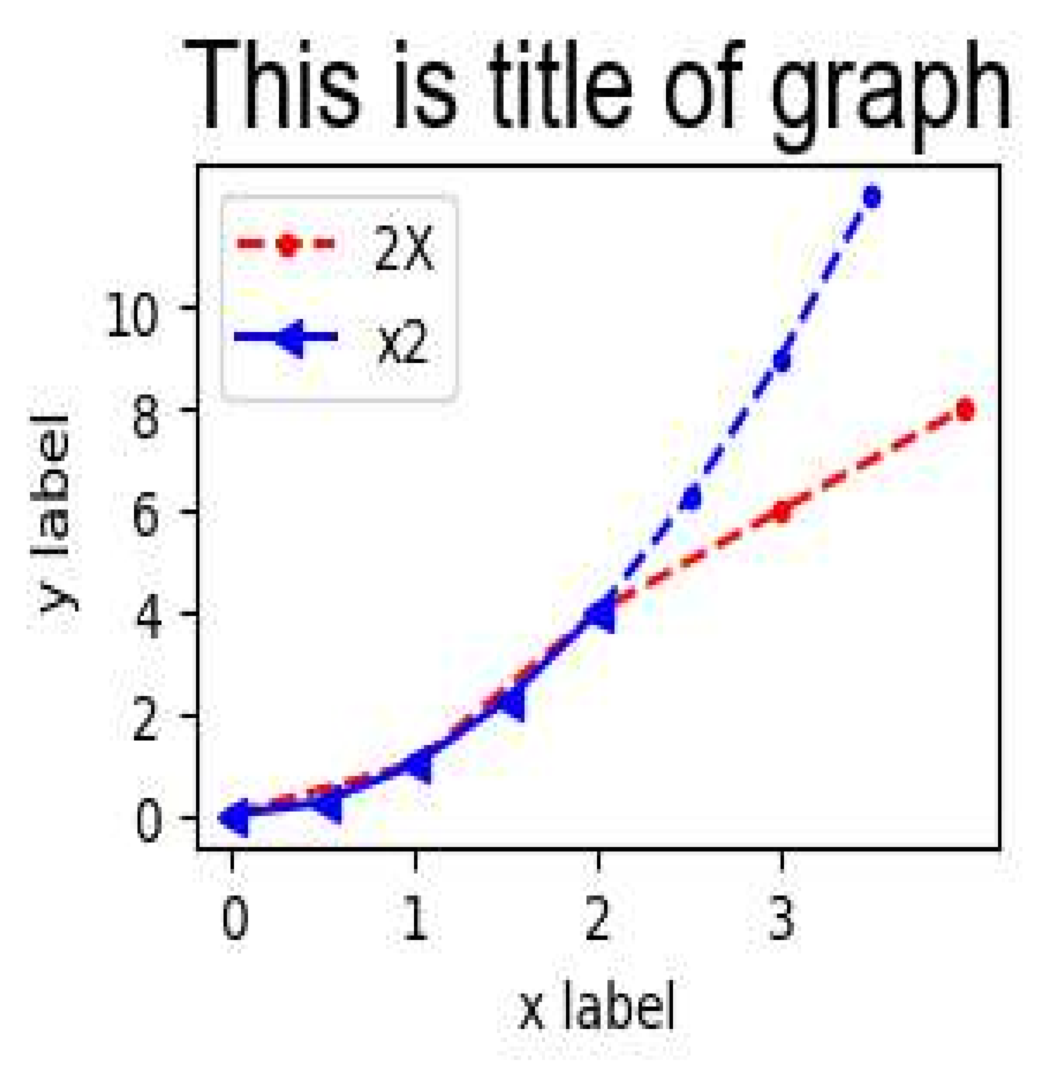
Example

- x =[0,1,2,3,4]
- y= [0,1,4,6,8]
- x2=np.arange(0,4,.5)
- plt.figure(figsize=(3,2), dpi=100)
- **# for line one**
- plt.plot(x,y,'r.--', label='2X')
- **# for line two**
- plt.plot(x2, x2**2,'b<--', label='x2')
- plt.xlabel('x label')
- plt.ylabel('y label')
- plt.title('This is title of graph' ,fontdict= {'fontname':'Arial' , 'fontsize' : 20 })
- plt.xticks([0,1,2,3])
- plt.yticks([0,2,4,6,8,10])
- plt.legend()
- plt.show()



Example

- x =[0,1,2,3,4]
- y= [0,1,4,6,8]
- x2=np.arange(0,4,.5)
- plt.figure(figsize=(3,2), dpi=100)
- plt.plot(x,y,'r.--', label='2X') **# for line one**
- **#for line two first part**
- plt.plot(x2[:5], x2[:5]**2,'b<-', label='x2')
- **# for line two second part**
- plt.plot(x2[4:],x2[4:]**2,'b.--')
- plt.xlabel('x label')
- plt.ylabel('y label')
- plt.title('This is title of graph' ,fontdict= {'fontname':'Arial' , 'fontsize' : 20 })
- plt.xticks([0,1,2,3])
- plt.yticks([0,2,4,6,8,10])
- **#for save file**
- plt.savefig("linegraph.png",dpi=300)
- plt.legend()



Scatter Plot

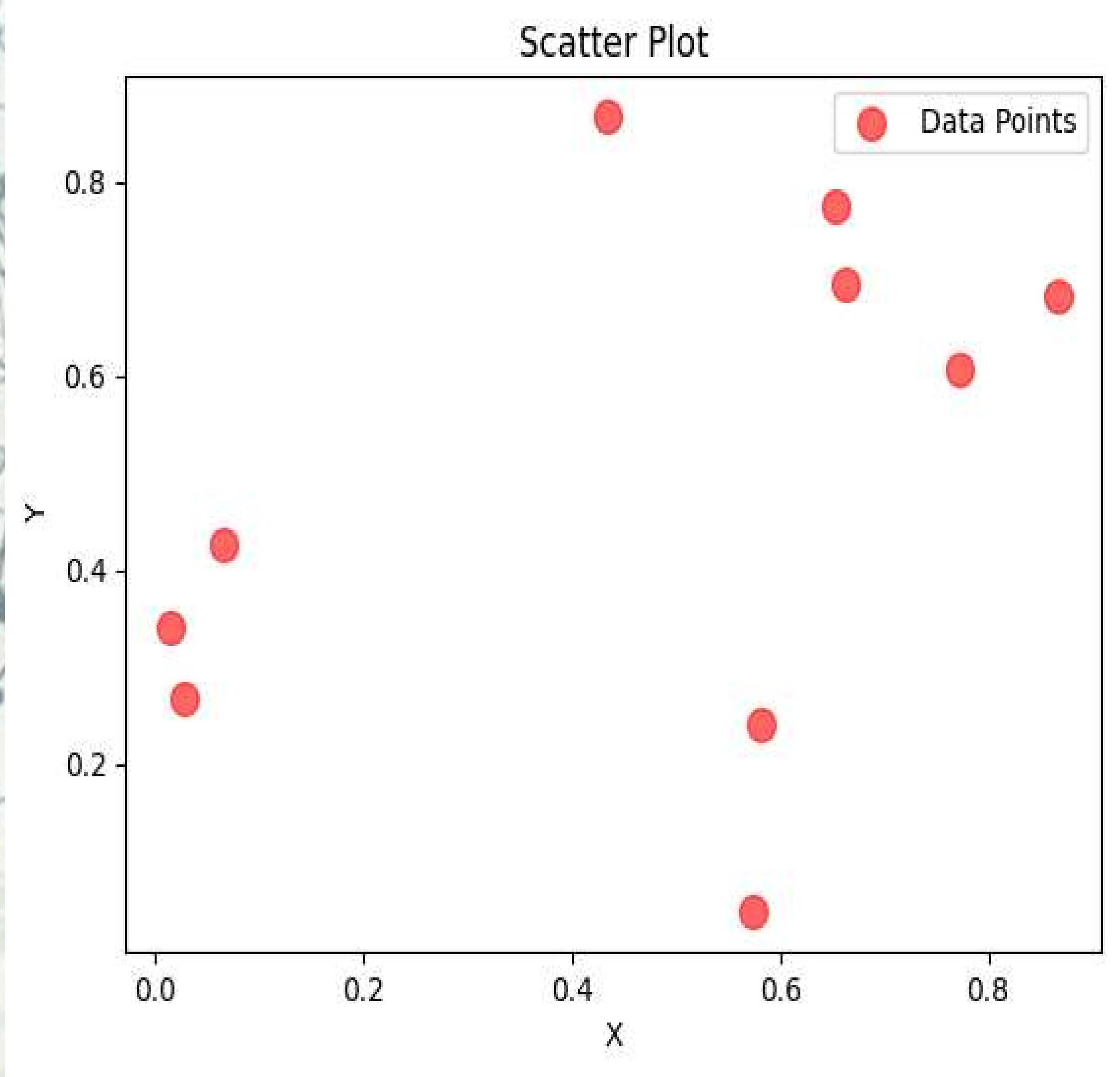
- **Purpose:**
  - Plots individual data points without connecting lines to show relationships between two variables.
- **Use Case:**
  - Visualize feature pairs in datasets for classification or clustering.
  - Display model predictions vs. actual values.
  - Plot clusters (e.g., K-Means) or dimensionality reduction results (e.g., PCA, t-SNE).
- **Key Features:**
  - Customizable marker size, color, and shape.
  - Supports color mapping for categorical data.
  - Ideal for 2D data exploration.

Scatter Plot

- **Syntax:**
  - plt.scatter(x, y, s=None, c=None, marker=None, alpha=None, label=None)
- **Key Attribute:**
  - **x, y:** Data for x and y axes (arrays of same length).
  - **s**: Size of markers (scalar or array for varying sizes).
  - **c**: Color of markers (scalar, array, or colormap).
  - **marker**: Marker style (e.g., 'o' for circles, '^' for triangles).
  - **alpha**: Transparency of markers (0 to 1).
  - **label**: Label for legend.

Example

- import matplotlib.pyplot as plt
- import numpy as np
- x = np.random.rand(10)
- y = np.random.rand(10)
- plt.scatter(x, y, s=100, c='red', marker='o', alpha=0.6, label='Data Points')
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Scatter Plot')
- plt.legend()
- plt.show()



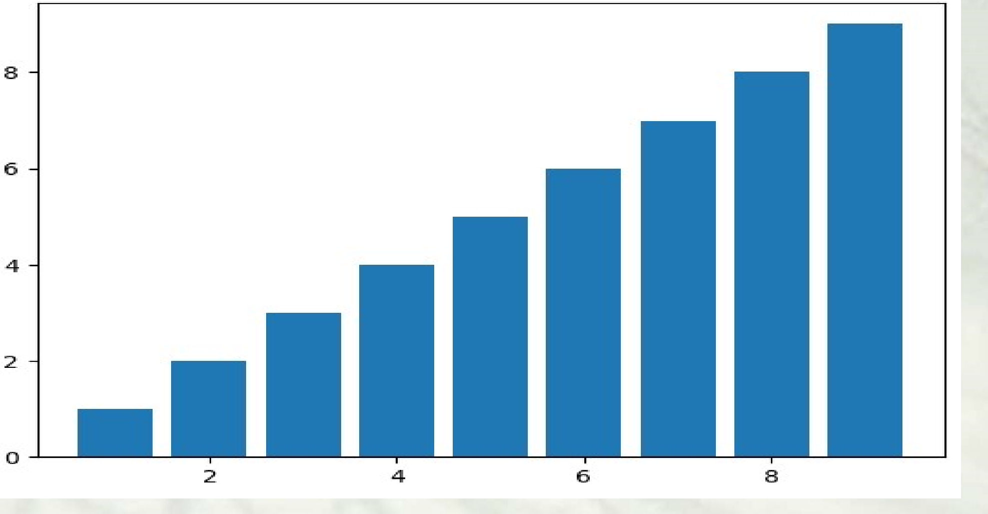
Bar Plot

- **Purpose:**
  - Represents categorical data with rectangular bars for comparison.
- **Use Case:**
  - Compare model performance metrics (e.g., accuracy, F1-score) across models.
  - Visualize feature importance scores (e.g., Random Forest, XGBoost).
  - Display class distribution in datasets.
- **Key Features:**
  - Supports vertical (`bar`) and horizontal (`barh`) bars.
  - Customizable bar width, color, and edge style.
  - Stacked or grouped bars for multiple categories.
- **Syntax:**
  - plt.bar(x, height, width=0.8, bottom=None, align='center', color=None, edgecolor=None, label=None)

Bar Plot

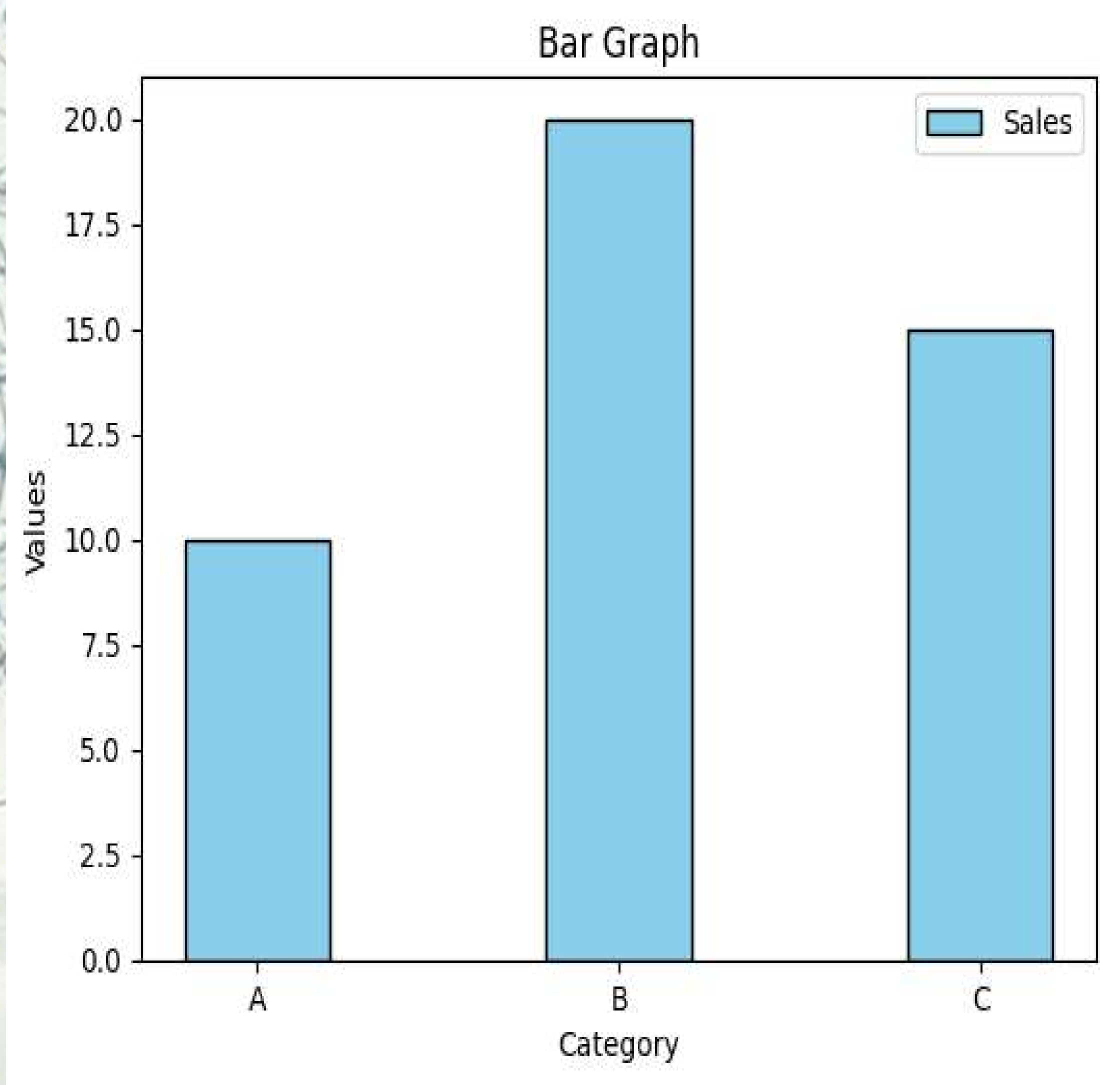
- **Key Attribute:**
  - **x**: Sequence of category positions (e.g., indices or labels).
  - **height**: Heights of the bars (values for each category).
  - **width**: Width of bars (default: 0.8, range 0 to 1).
  - **bottom**: Starting position for bars (default: 0, useful for stacked bars).
  - **align**: Alignment of bars relative to x ('center' or 'edge’).
  - **color**: Color of bars (e.g., 'blue', or list for multiple colors).
  - **edgecolor**: Color of bar edges.

- x1= (np.arange(1,10))
- y1= (np.arange(1,10))
- plt.bar(x1,y1)
- plt.show()



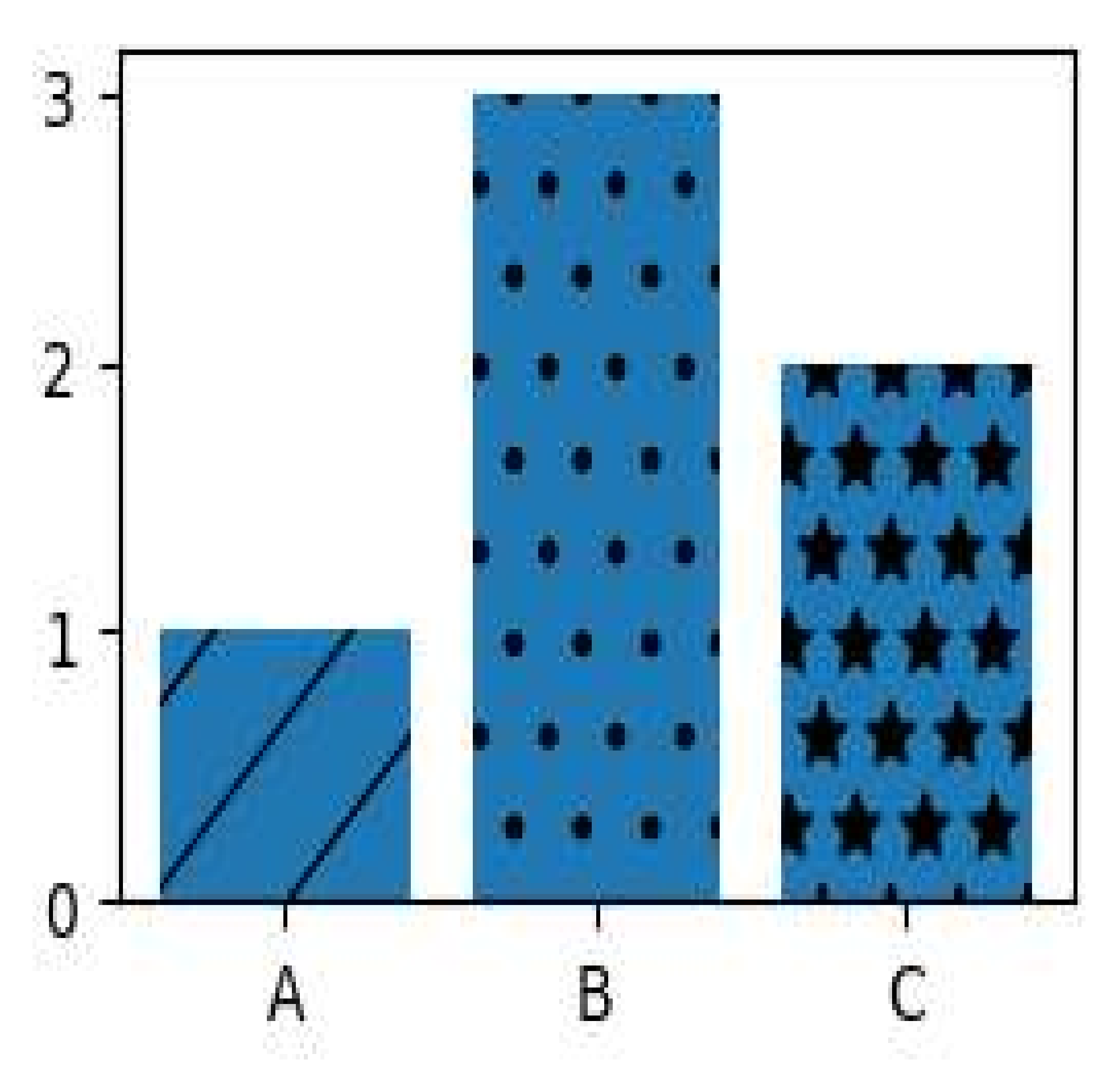
Example

- import matplotlib.pyplot as plt
- import numpy as np
- categories = ['A', 'B', 'C']
- values = [10, 20, 15]
- plt.bar(categories, values, width=0.4, color='skyblue', edgecolor='black', label='Sales')
- plt.xlabel('Category')
- plt.ylabel('Values')
- plt.title('Bar Graph')
- plt.legend()
- plt.show()



Example

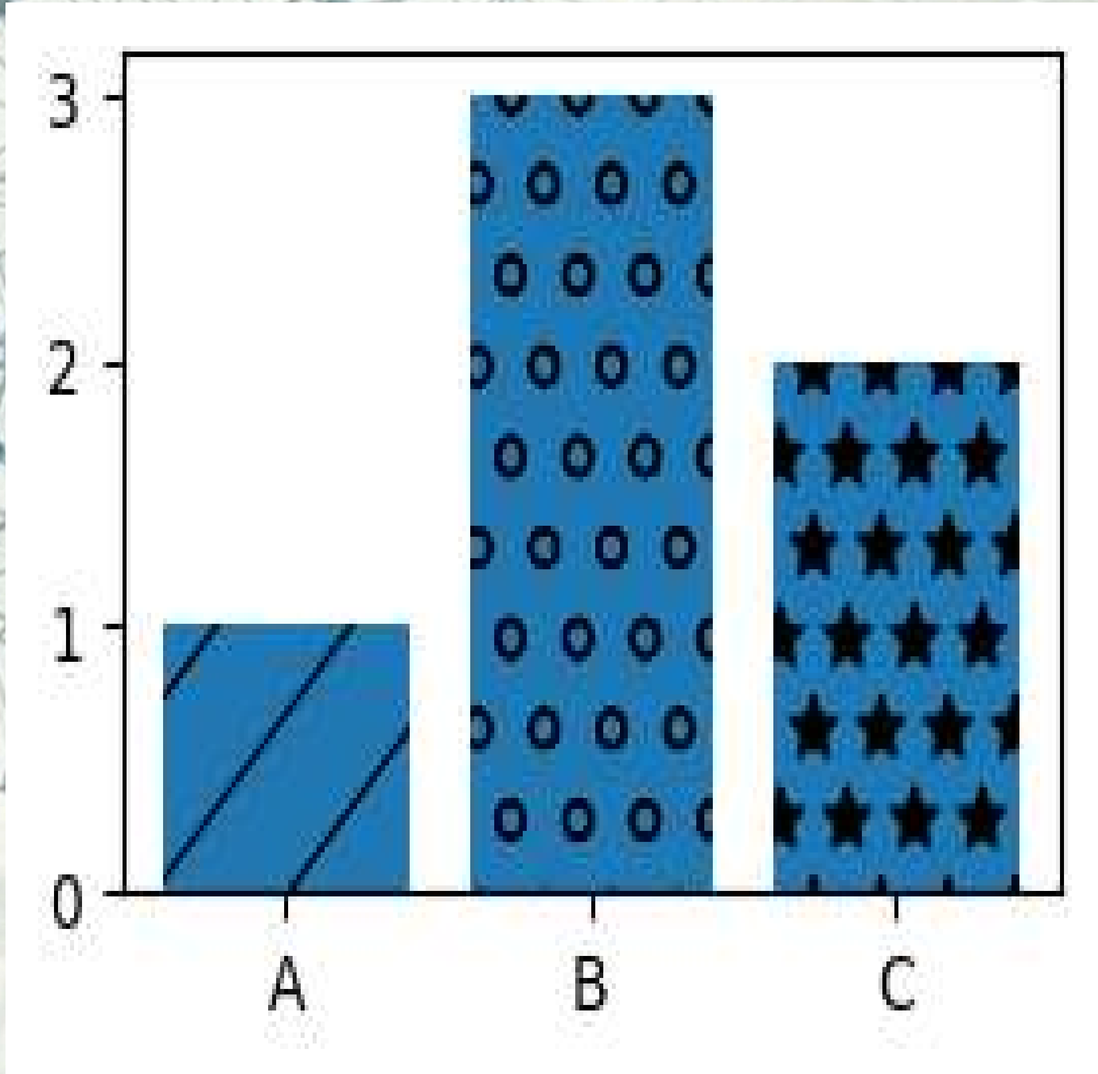
- import matplotlib.pyplot as plt
- import numpy as np
- label=['A','B','C']
- value=[1,3,2]
- plt.figure(figsize=(3,2),dpi=100)
- bars=plt.bar(label, value)
- bars[0].set_hatch('/')
- bars[1].set_hatch('.')
- bars[2].set_hatch('*')
- plt.show()



Example

- import matplotlib.pyplot as plt
- import numpy as np
- label=['A','B','C']
- value=[1,3,2]
- plt.figure(figsize=(3,2),dpi=100)
- patterns=['/','o','*']
- bars=plt.bar(label, value)
- # use of for loop
- for bar in bars:
-     bar.set_hatch(patterns.pop(0))

- plt.show()



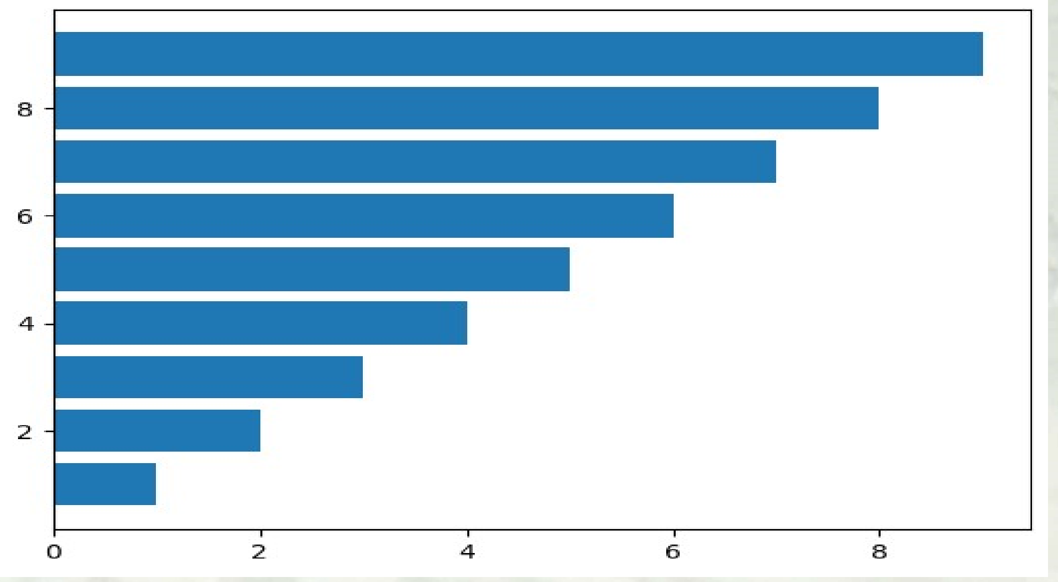
Horizontal Bar Chart

- **Purpose:**
  - Displays categorical data with horizontal bars for comparison, useful when category labels are long.
- **Use Case:**
  - Compare quantities across categories with long names.
  - Visualize rankings or survey results.
  - Display data where horizontal layout improves readability.
- **Key Features:**
  - Similar to bar chart but horizontal orientation.
  - Customizable bar height and color.
  - Ideal for text-heavy labels.
- **Syntax:**
  - plt.barh(y, width, height=0.8, color=None, edgecolor=None, align='center')

Horizontal Bar Chart

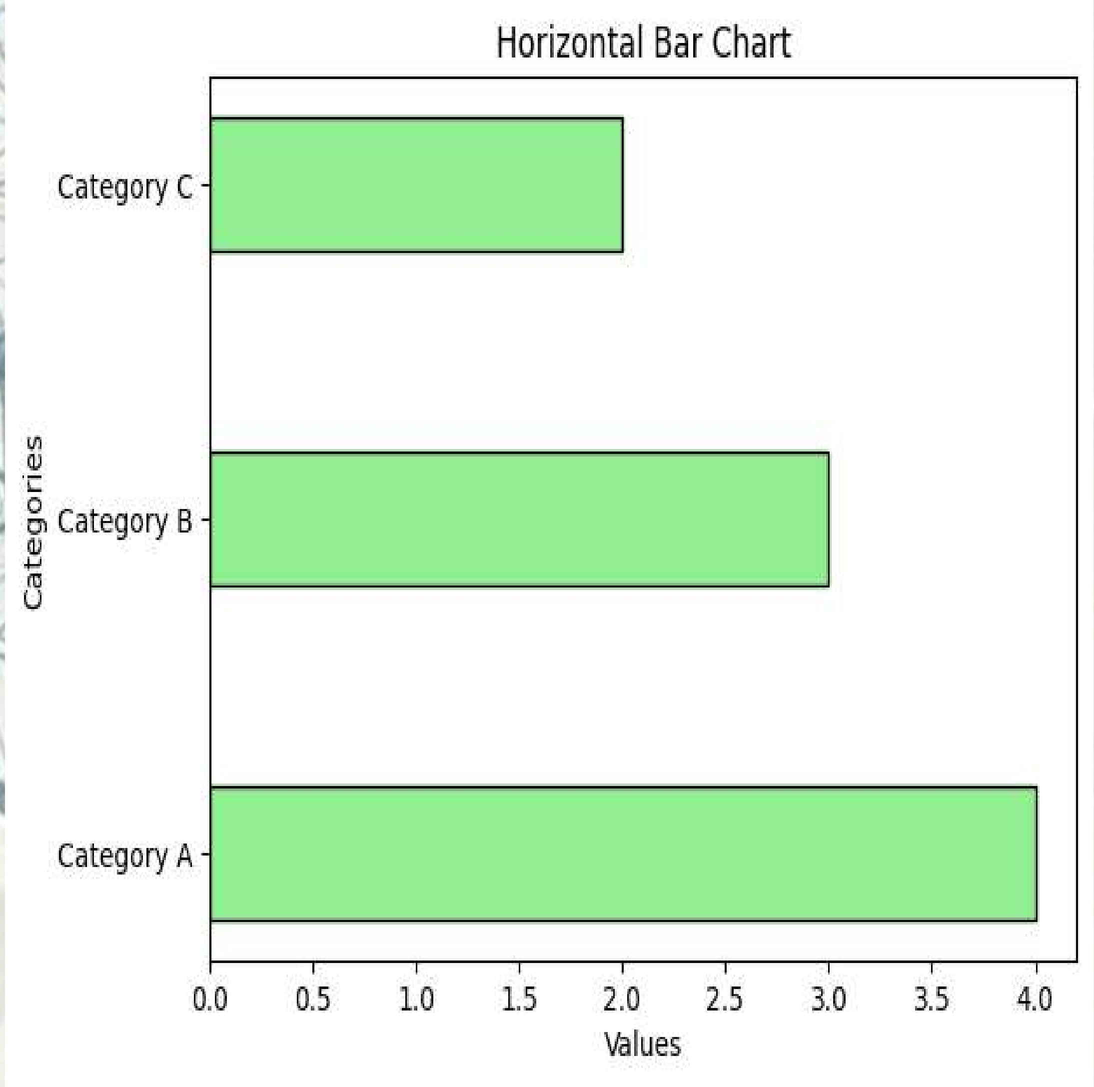
- **Key Attribute:**
  - **y:** Category labels or positions (array-like).
  - **width:** Lengths of bars (array-like).
  - **height:** Bar thickness (scalar or array).
  - **color:** Bar fill color.
  - **edgecolor:** Bar edge color.
  - **align:** Alignment of bars ('center' or 'edge').

- x1= (np.arange(1,10))
- y1= (np.arange(1,10))
- plt.barh(x1,y1)
- plt.show()



Example

- import matplotlib.pyplot as plt
- categories = ['Category A', 'Category B', 'Category C']
- values = [4, 3, 2]
- plt.barh(categories, values, height=0.4, color='lightgreen', edgecolor='black')
- plt.xlabel('Values')
- plt.ylabel('Categories')
- plt.title('Horizontal Bar Chart')
- plt.show()



Histogram plot

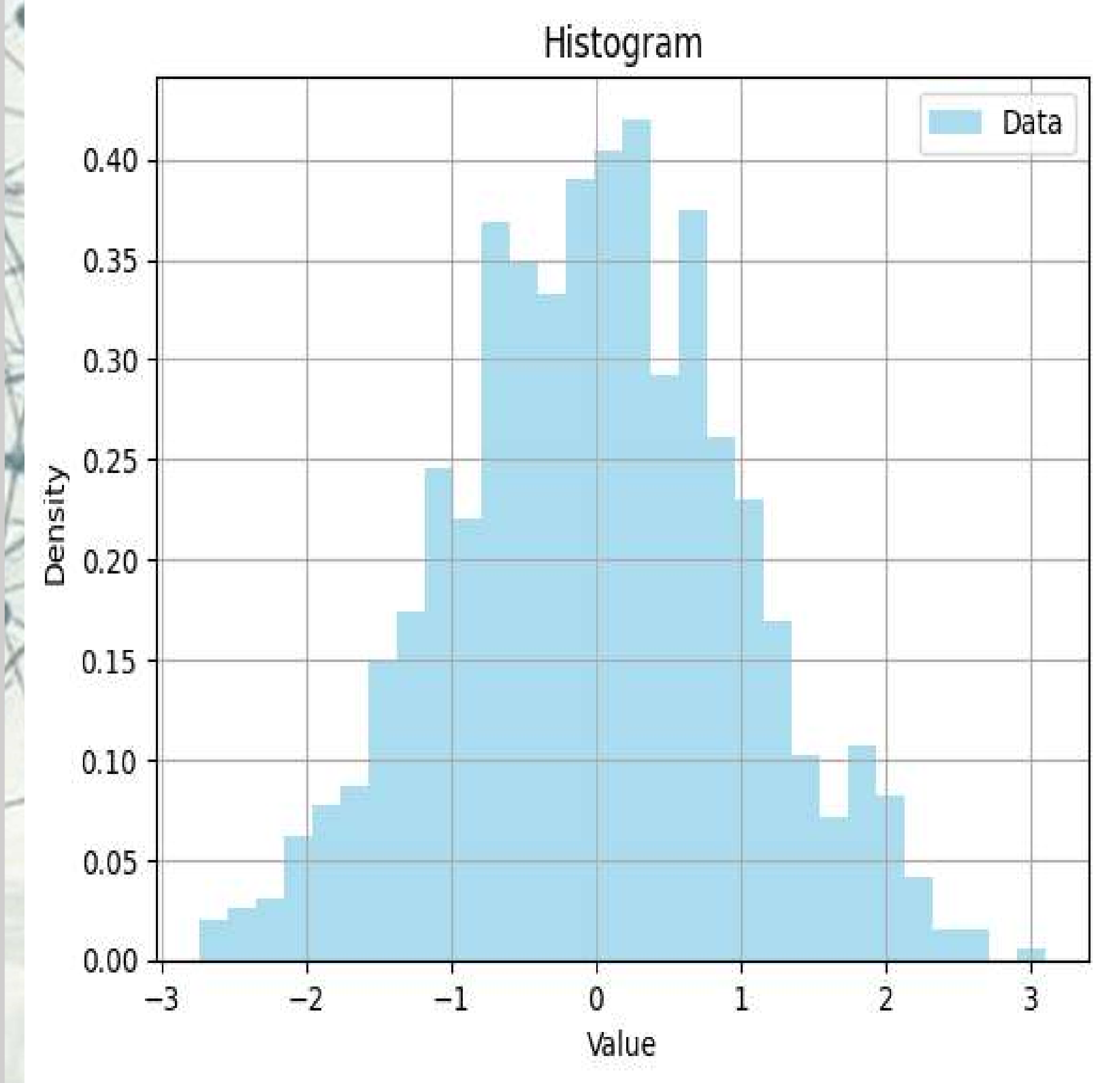
- **Purpose:**
  - Shows the distribution of a continuous variable by grouping data into bins.
- **Use Case:**
  - Analyze feature distributions to identify skewness or outliers.
  - Visualize prediction probability distributions.
  - Check data normalization effects.
- **Key Features:**
  - Adjustable bin size and number.
  - Supports density normalization and cumulative histograms.
  - Customizable colors and transparency.

Histogram Plot

- **Syntax:**
  - plt.hist(x, bins=None, density=False, color=None, alpha=None)
- **Key Attribute:**
  - **x:** Data to be binned (array-like).
  - **bins:** Number of bins or bin edges (int or sequence).
  - **density:** If True, plot probability density (area sums to 1).
  - **color:** Bar fill color.
  - **alpha:** Transparency of bars (0 to 1).
  - **label:** Label for legend.

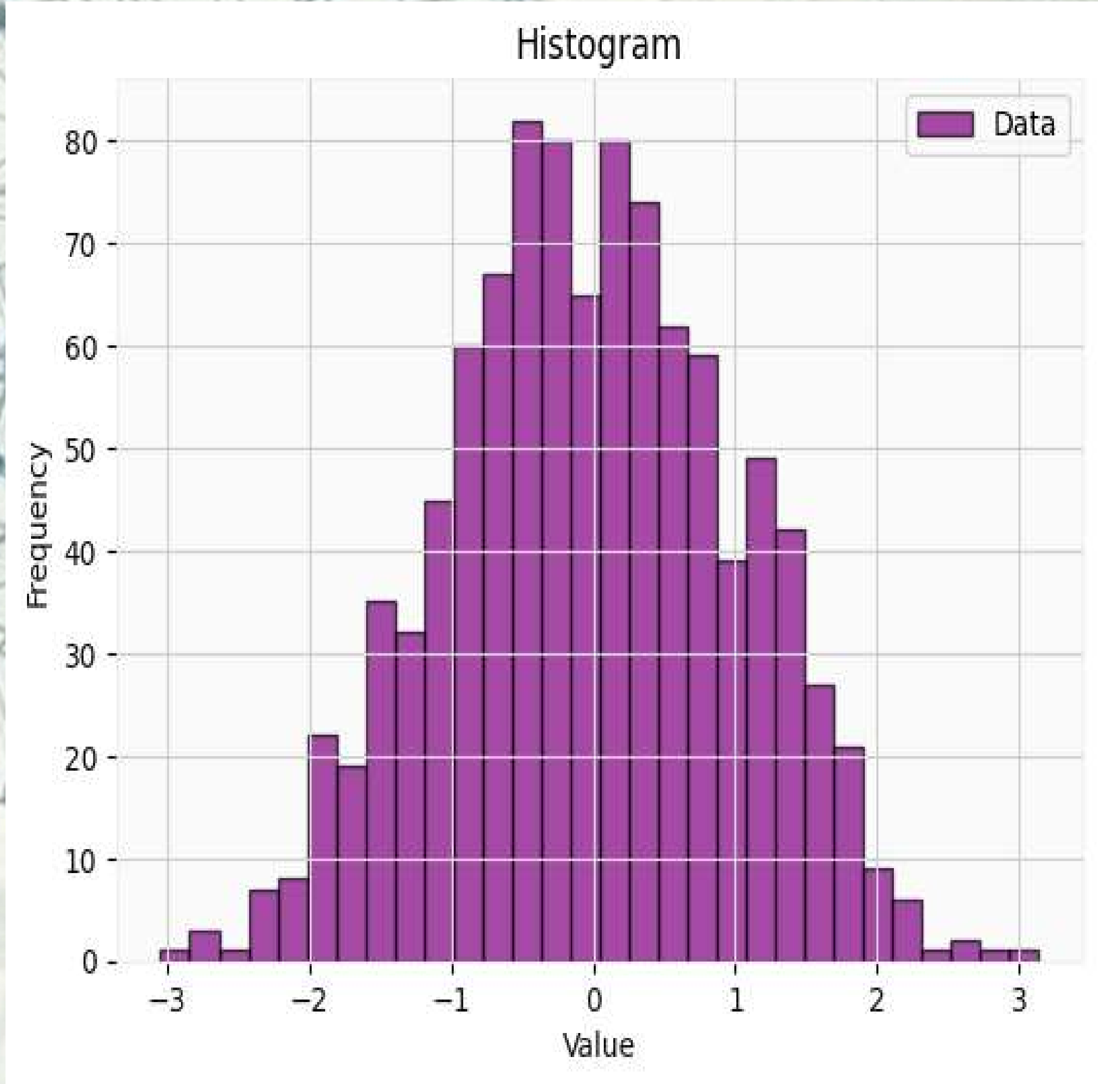
Example

- import matplotlib.pyplot as plt
- import numpy as np
- data = np.random.normal(0, 1, 1000)
- plt.hist(data, bins=30, density=True, color='skyblue',alpha=0.7, label='Data')
- plt.xlabel('Value')
- plt.ylabel('Density')
- plt.title('Histogram')
- plt.legend()
- plt.grid(True)
- plt.show()



Example

- import matplotlib.pyplot as plt
- import numpy as np
- data = np.random.randn(1000)
- plt.hist(data, bins=30, color='purple', edgecolor='black', alpha=0.7, label='Data')
- plt.xlabel('Value')
- plt.ylabel('Frequency')
- plt.title('Histogram')
- plt.legend()
- plt.show()



Box Plot

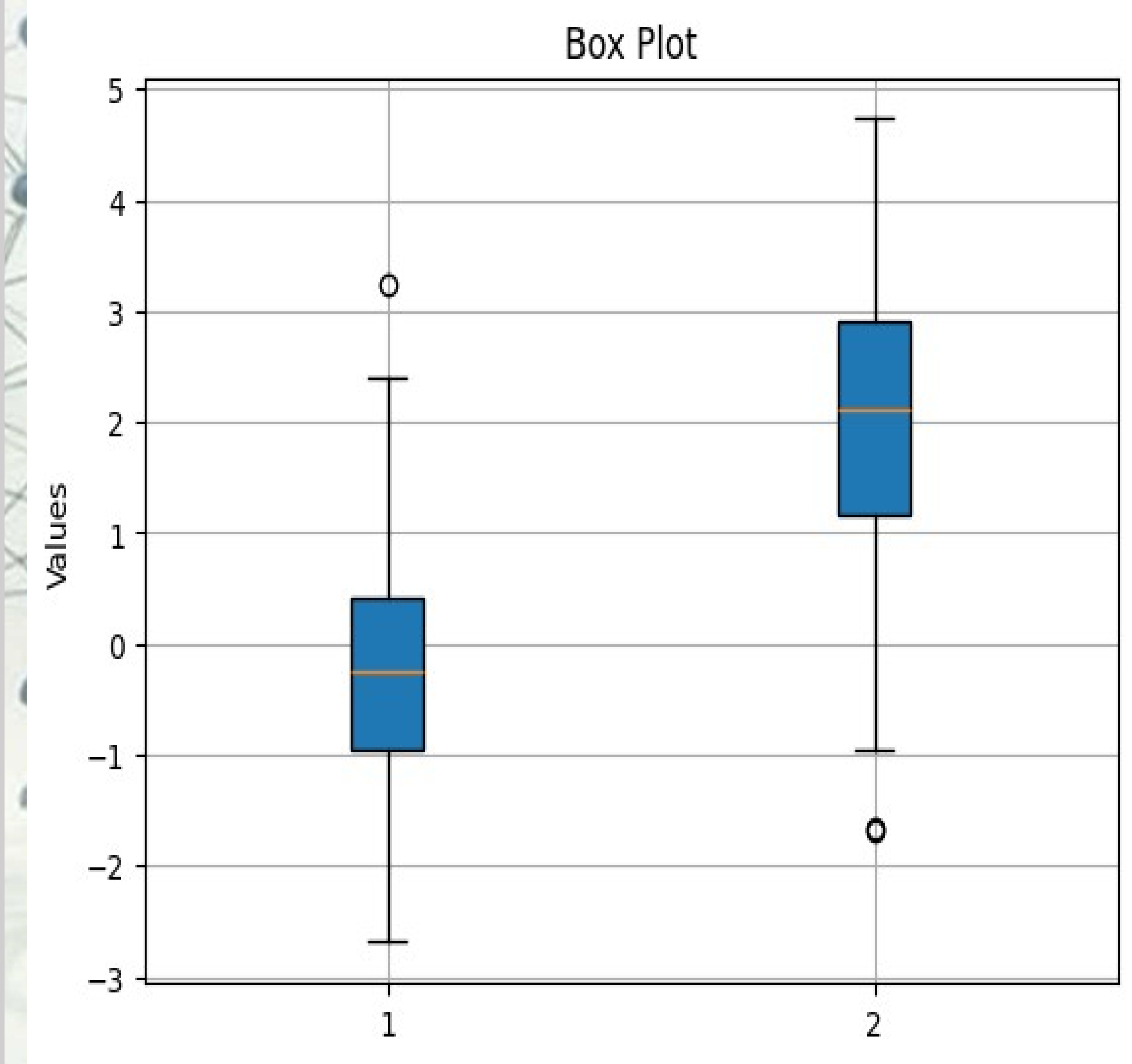
- **Purpose:**
  - Summarizes data distribution through quartiles, highlighting median, spread, and outliers.
- **Use Case:**
  - Compare distributions across groups (e.g., test scores by class).
  - Identify outliers in datasets.
  - Visualize data spread and skewness.
- **Key Features:**
  - Shows median, interquartile range (IQR), and outliers.
  - Customizable whisker length and box appearance.
  - Supports multiple box plots for comparison.

Box Plot

- **Syntax:**
  - plt.boxplot(x, vert=True, widths=None, patch_artist=False, labels=None)
- **Key Attribute:**
  - **x:** Data (array or list of arrays).
  - **vert:** If True, vertical boxes; if False, horizontal.
  - **widths:** Width of boxes (scalar or array).
  - **patch_artist:** If True, fill boxes with color.
  - **labs:** Labels for each box plot.

Example

- import matplotlib.pyplot as plt
- import numpy as np
- data = [np.random.normal(0, 1, 100), np.random.normal(2, 1.5, 100)]
- plt.boxplot(data, patch_artist=True, vert=True,)
- plt.ylabel('Values')
- plt.title('Box Plot')
- plt.grid(True)
- plt.show()



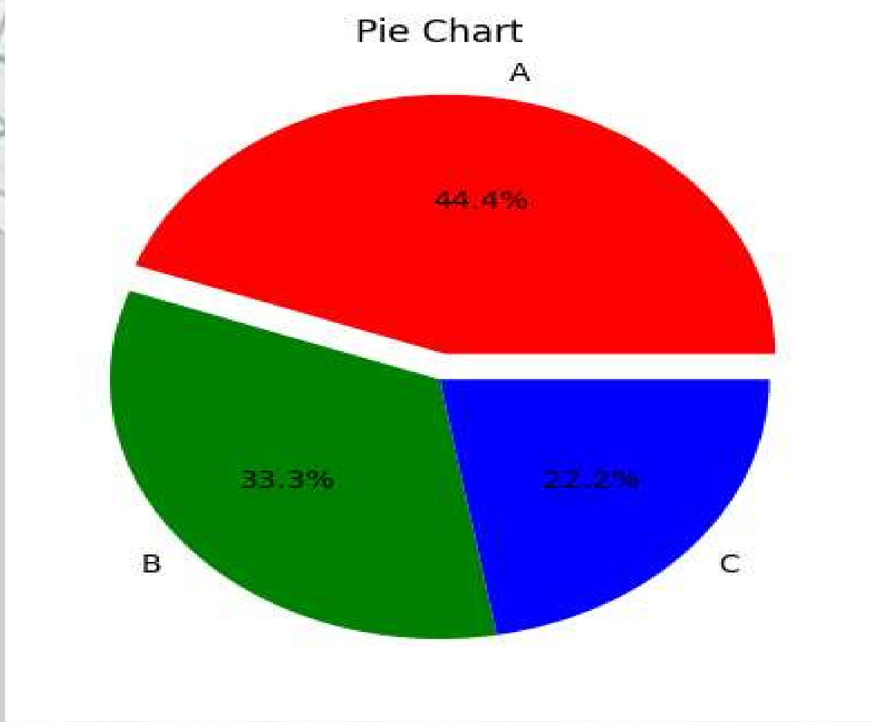
Pie Chart

- **Purpose:**
  - Displays proportions of categorical data as slices of a circle.
- **Use Case:**
  - Show percentage breakdowns (e.g., market share, budget allocation).
  - Visualize relative frequencies of categories.
  - Highlight composition of a whole.
- **Key Features:**
  - Customizable colors, labels, and explosion of slices.
  - Supports percentage display and shadows.
  - Best for small number of categories.
- **Syntax:**
  - plt.pie(x,labels=None,colors=None,autopct=None,explode=None,shadow=False)

Pie Chart

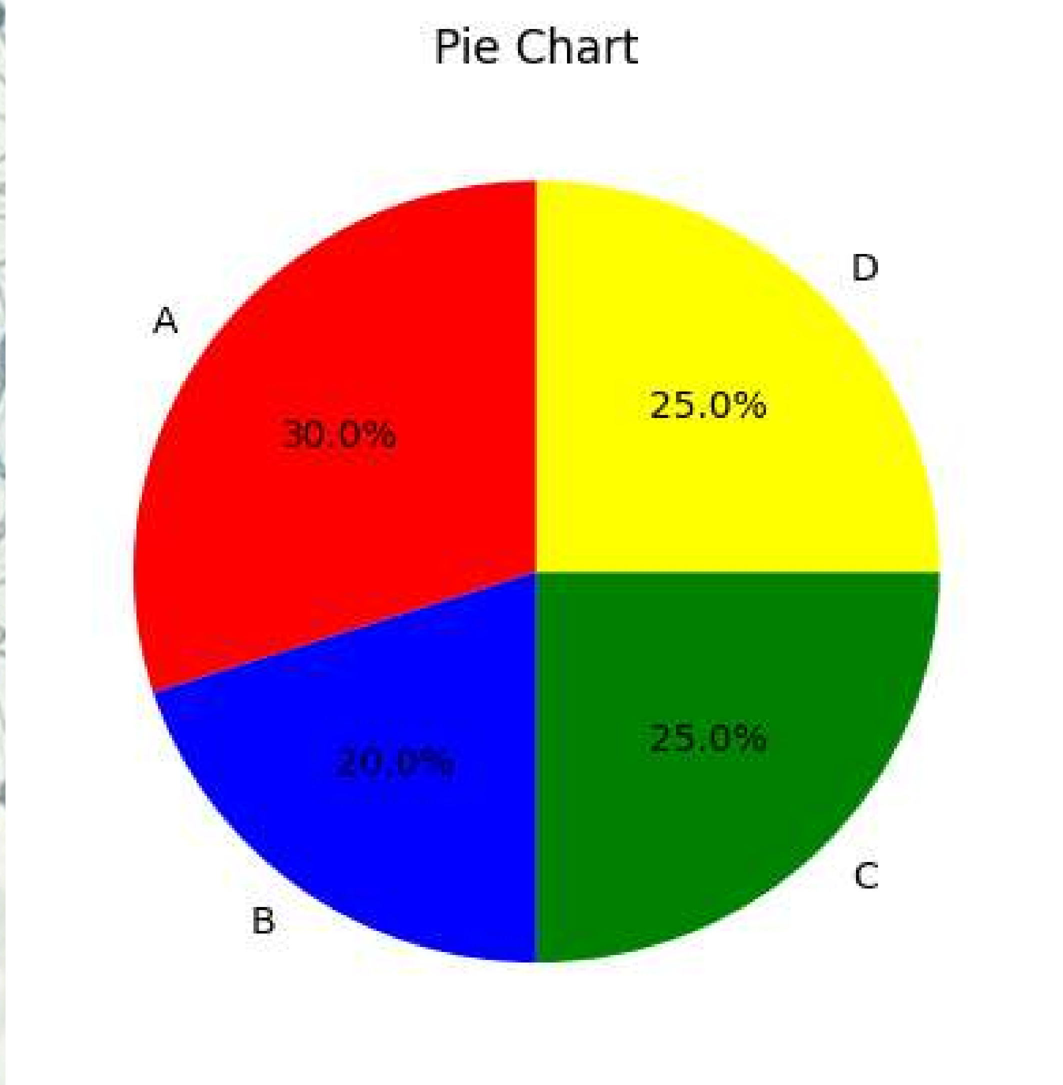
- **Key Attribute:**
  - **x:** Wedge sizes (array-like).
  - **labels:** Category names for each wedge.
  - **colors:** Colors for each wedge.
  - **autopct:** Format for percentage display (e.g., '%.1f%%').
  - **explode:** Offset for each wedge (array).
  - **shadow:** If True, adds shadow effect.

```
• import matplotlib.pyplot as plt
• labels = ['A', 'B', 'C']
• sizes = [40, 30, 20]
• plt.pie(sizes, labels=labels, colors=['red', 'green', 'blue'],
  autopct='%.1f%%', explode=[0.1, 0, 0])
• plt.title('Pie Chart')
• plt.show()
```



Example

- import matplotlib.pyplot as plt
- import numpy as np
- sizes = [30, 20, 25, 25]
- labels = ['A', 'B', 'C', 'D']
- plt.pie(sizes, labels=labels, colors=['red', 'blue', 'green', 'yellow'], autopct='%.1f%%', startangle=90)
- plt.title('Pie Chart')
- plt.show()



Area Plot

- **Purpose:**
  - Fills area under a curve or between curves to show cumulative or relative data.
- **Use Case:**
  - Visualize cumulative trends (e.g., stacked time series).
  - Show area under functions or between datasets.
  - Highlight differences between two curves.
- **Key Features:**
  - Customizable fill color and transparency.
  - Supports stacking multiple areas.
  - Useful for continuous data.
- **Syntax:**
  - plt.fill_between(x, y1, y2=0, color=None, alpha=None, label=None)

Area Plot

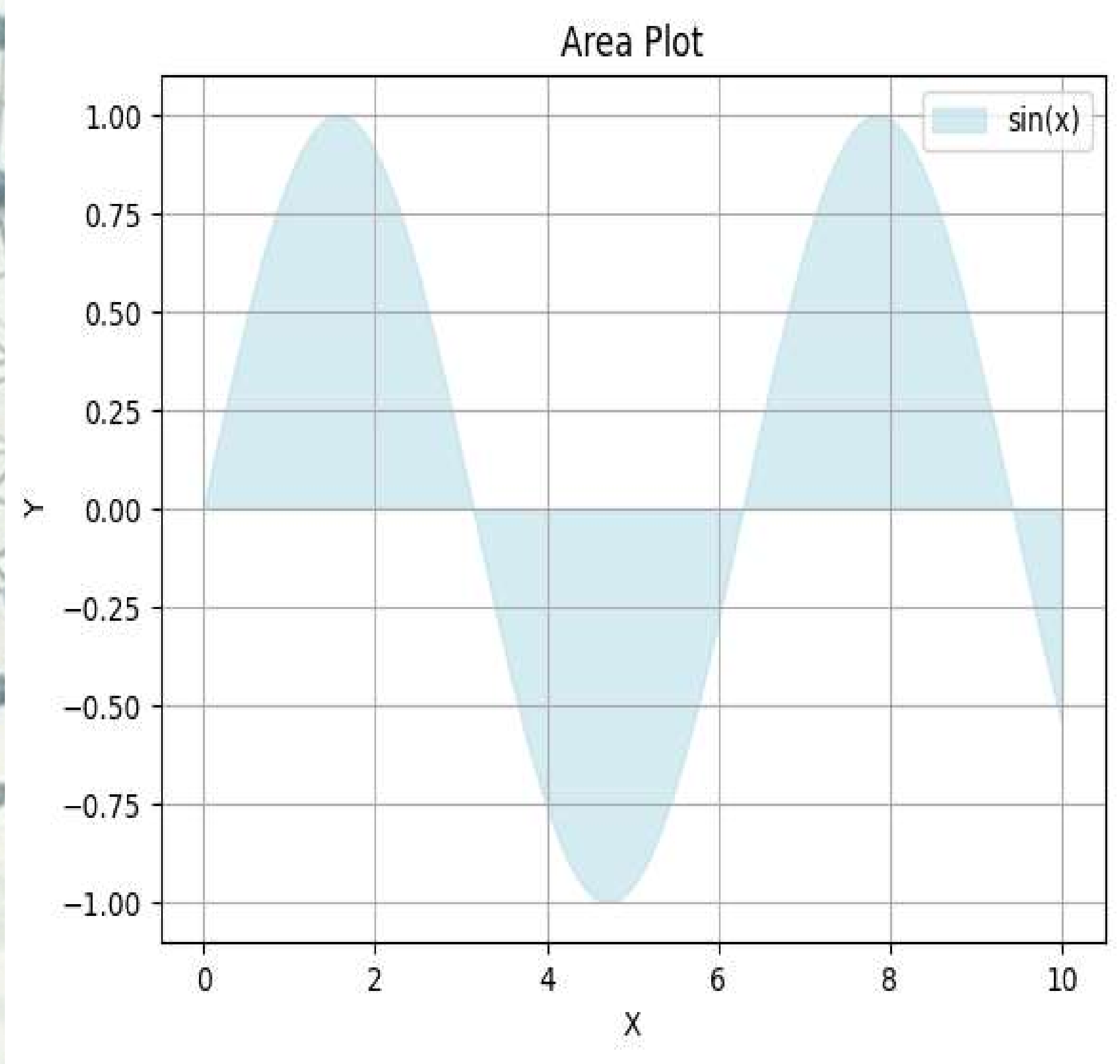
- **Syntax:**
  - plt.fill_between(x, y1, y2=0, color=None, alpha=None, label=None)

- **Key Attribute:-**
  - **x:** X-axis data (array-like).
  - **y1:** Y-axis data for upper boundary (array-like).
  - **y2:** Y-axis data for lower boundary (default 0).
  - **color:** Fill color.
  - **alpha:** Transparency of fill (0 to 1).
  - **label:** Label for legend.

Example

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

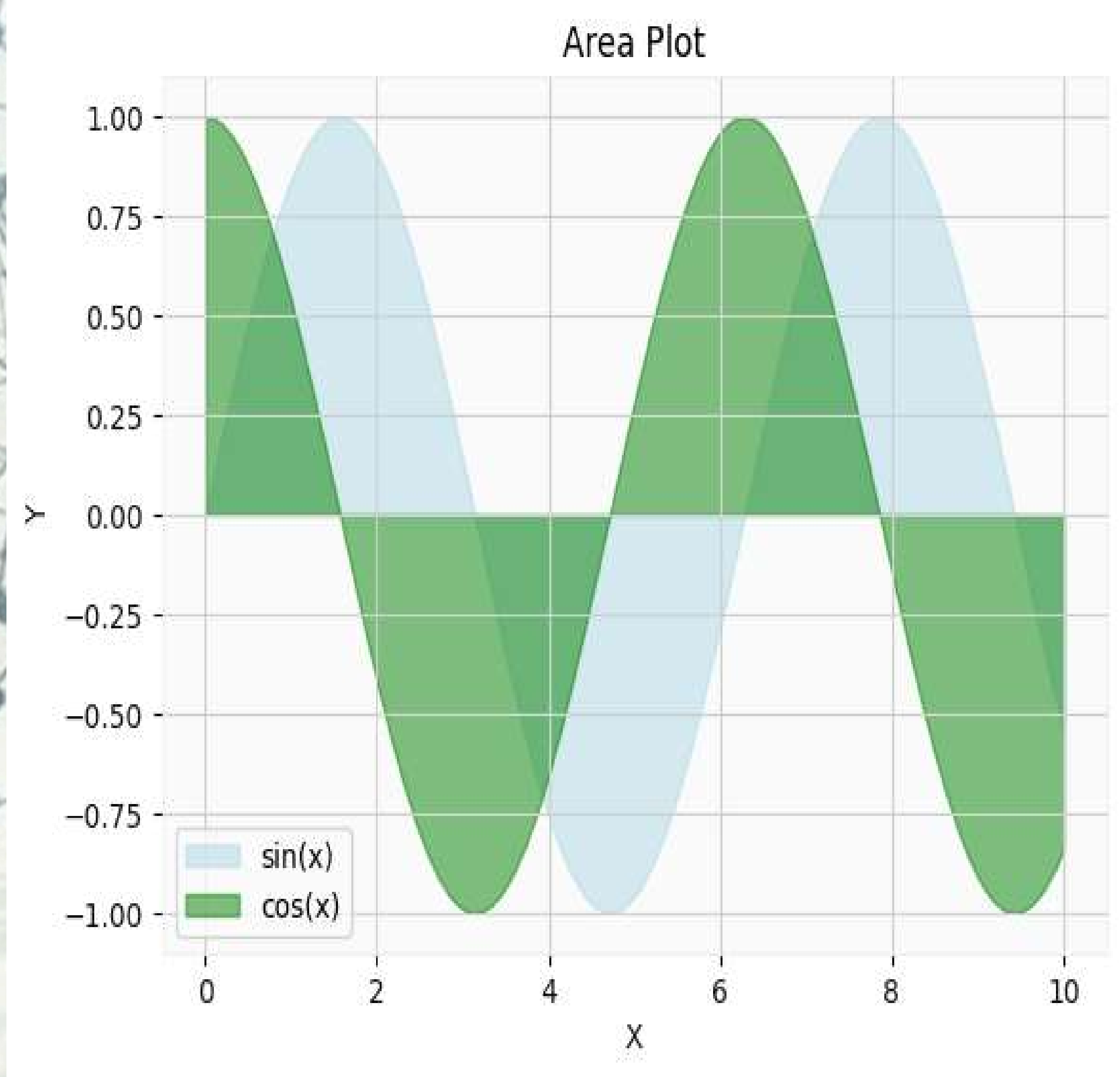
- x = np.linspace(0, 10, 100)
- y = np.sin(x)
- plt.fill_between(x, y, color='lightblue', alpha=0.5, label='sin(x)')
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Area Plot')
- plt.legend()
- plt.grid(True)
- plt.show()



Example

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

- x = np.linspace(0, 10, 100)
- y = np.sin(x)
- plt.fill_between(x, y, color='lightblue', alpha=0.5, label='sin(x)')
- plt.fill_between(x,np.cos(x), color='g', alpha=0.5, label='cos(x)')
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Area Plot')
- plt.legend()
- plt.grid(True)
- plt.show()



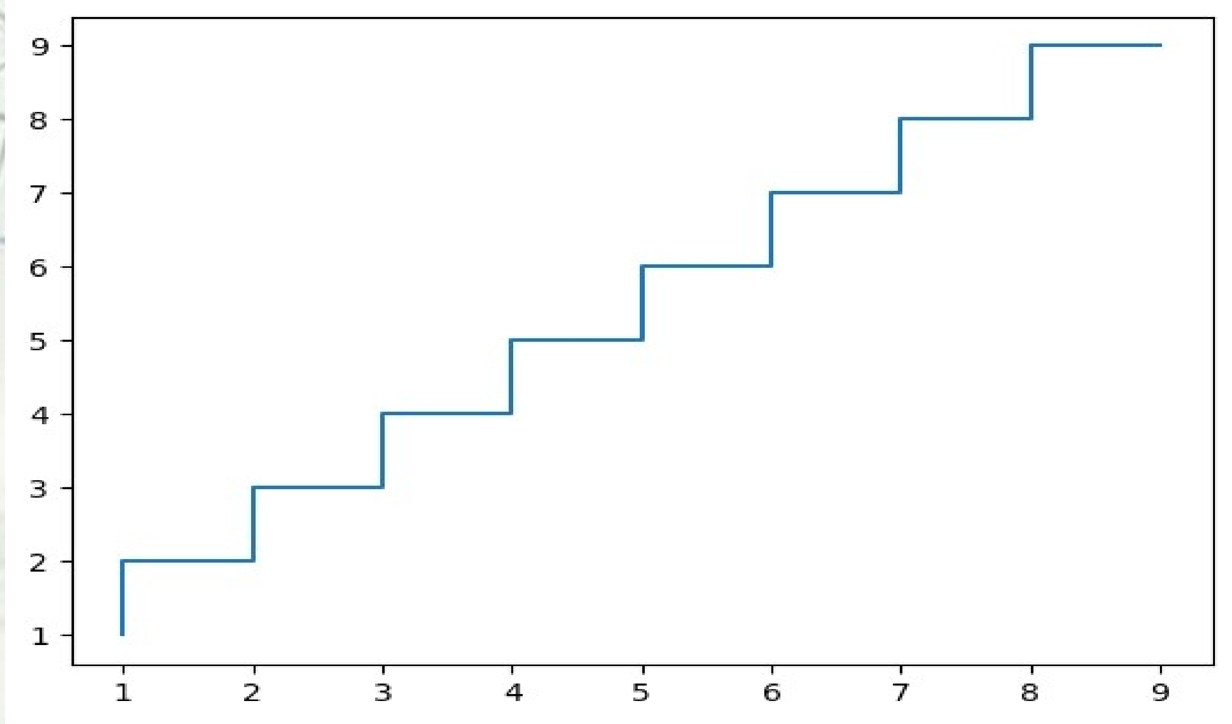
Step Plot

- **Purpose:**
  - Plots data as steps to represent discrete changes or piecewise constant data.
- **Use Case:**
  - Visualize discrete events (e.g., system state changes).
  - Show cumulative counts or step-wise trends.
  - Plot histograms with step-like appearance.
- **Key Features:**
  - Customizable step position (pre, post, or mid).
  - Supports multiple step lines.
  - Minimalist for discrete data.
- **Syntax:**
  - plt.step(x, y, where='mid', color=None, label=None)

Step Plot

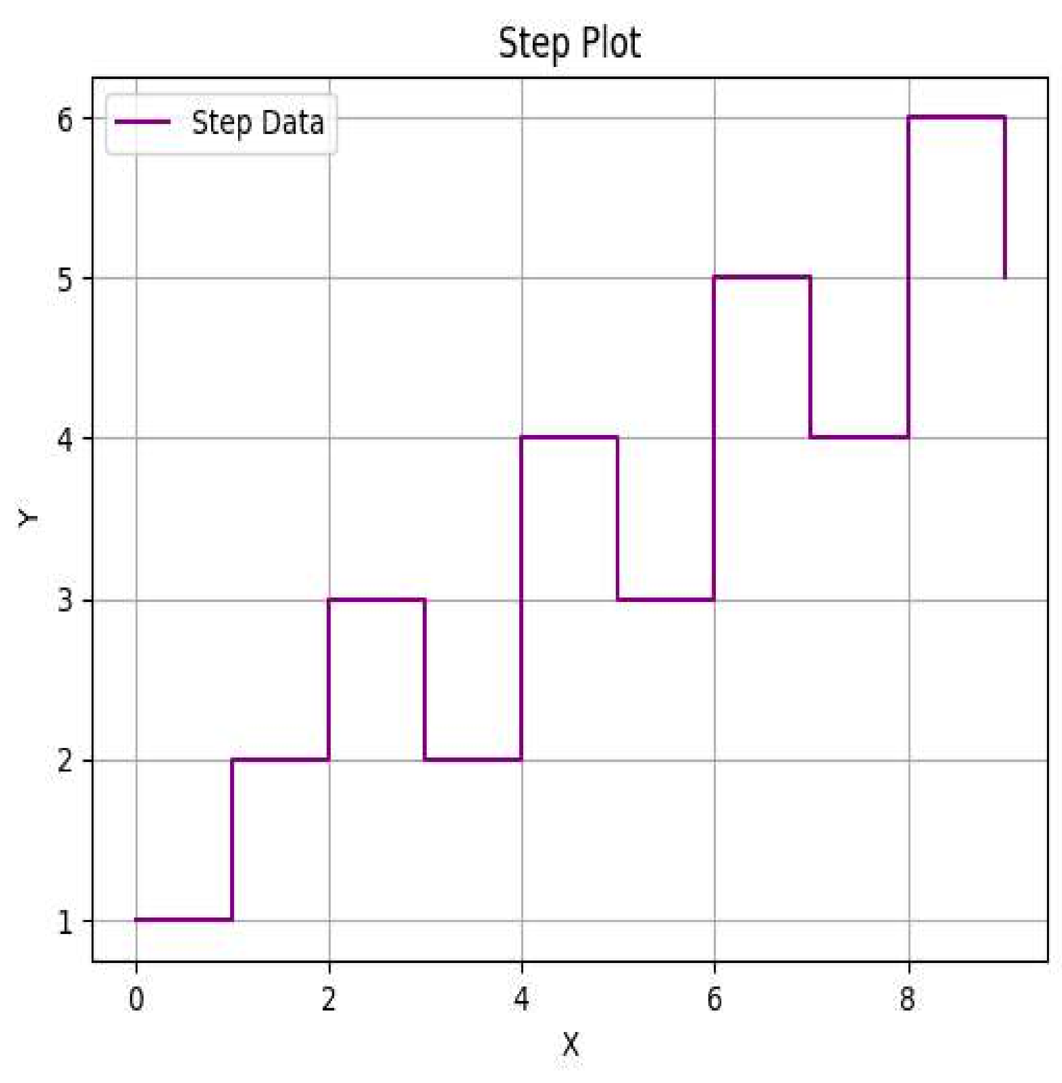
- **Key Attribute:**
  - **x:** X-axis data (array-like).
  - **y:** Y-axis data (array-like).
  - **where:** Step position ('pre', 'post', 'mid').
  - **color:** Line color.
  - **label:** Label for legend.

- import matplotlib.pyplot as plt
- import numpy as np
- x1= (np.arange(1,10))
- y1= (np.arange(1,10))
- plt.step(x1,y1)
- plt.show



Example

- import matplotlib.pyplot as plt
- import numpy as np
- x = np.arange(10)
- y = np.array([1, 2, 3, 2, 4, 3, 5, 4, 6, 5])
- plt.step(x,y,where='post', color='purple', label='Step Data')
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Step Plot')
- plt.legend()
- plt.grid(True)
- plt.show()



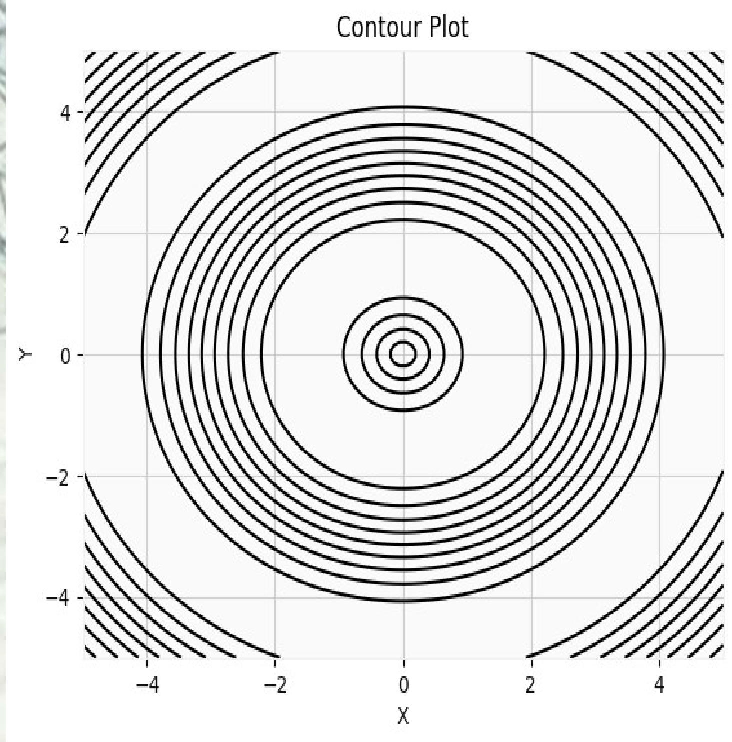
Contour Plot

- **Purpose:**
  - Displays 3D data in a 2D format by drawing contour lines where a function has constant values, useful for visualizing surfaces or fields.
- **Use Case:**
  - Visualize elevation or topographic maps.
  - Show temperature or pressure distributions in meteorology.
  - Analyze mathematical functions or optimization landscapes.
- **Key Features:**
  - Draws lines at specified levels of a 2D grid.
  - Customizable line styles, colors, and number of levels.
  - Can be combined with labels for contour values.
- **Syntax:**
  - plt.contour(X, Y, Z, levels=None, colors=None, linestyles=None)

Contour Plot

- **Key Attribute:**
  - **X, Y:** 2D arrays or 1D arrays defining the grid coordinates.
  - **Z:** 2D array of values to contour (height or intensity).
  - **levels:** Number of contour levels or specific values (e.g., [0, 1, 2]).
  - **colors:** Color of contour lines (single color or list).
  - **linestyles:** Style of contour lines (e.g., 'solid', 'dashed').

```
• import matplotlib.pyplot as plt
• import numpy as np
• x = np.linspace(-5, 5, 100)
• y = np.linspace(-5, 5, 100)
• X, Y = np.meshgrid(x, y)
• Z = np.sin(np.sqrt(X**2 + Y**2))
• plt.contour(X, Y, Z, levels=10, colors='black', linestyles='solid')
• plt.xlabel('X')
• plt.ylabel('Y')
• plt.title('Contour Plot')
• plt.grid(True)
• plt.show()
```



Filled Contour Plot

- **Purpose:**
  - Similar to contour plot but fills the areas between contour lines with colors to represent ranges of values.
- **Use Case:**
  - Visualize heatmaps or density maps (e.g., weather data).
  - Show regions of constant value in scientific data.
  - Highlight gradients in optimization problems.
- **Key Features:**
  - Fills areas between contours with colors.
  - Supports colormaps for gradient visualization.
  - Can be combined with plt.contour for outlined contours.

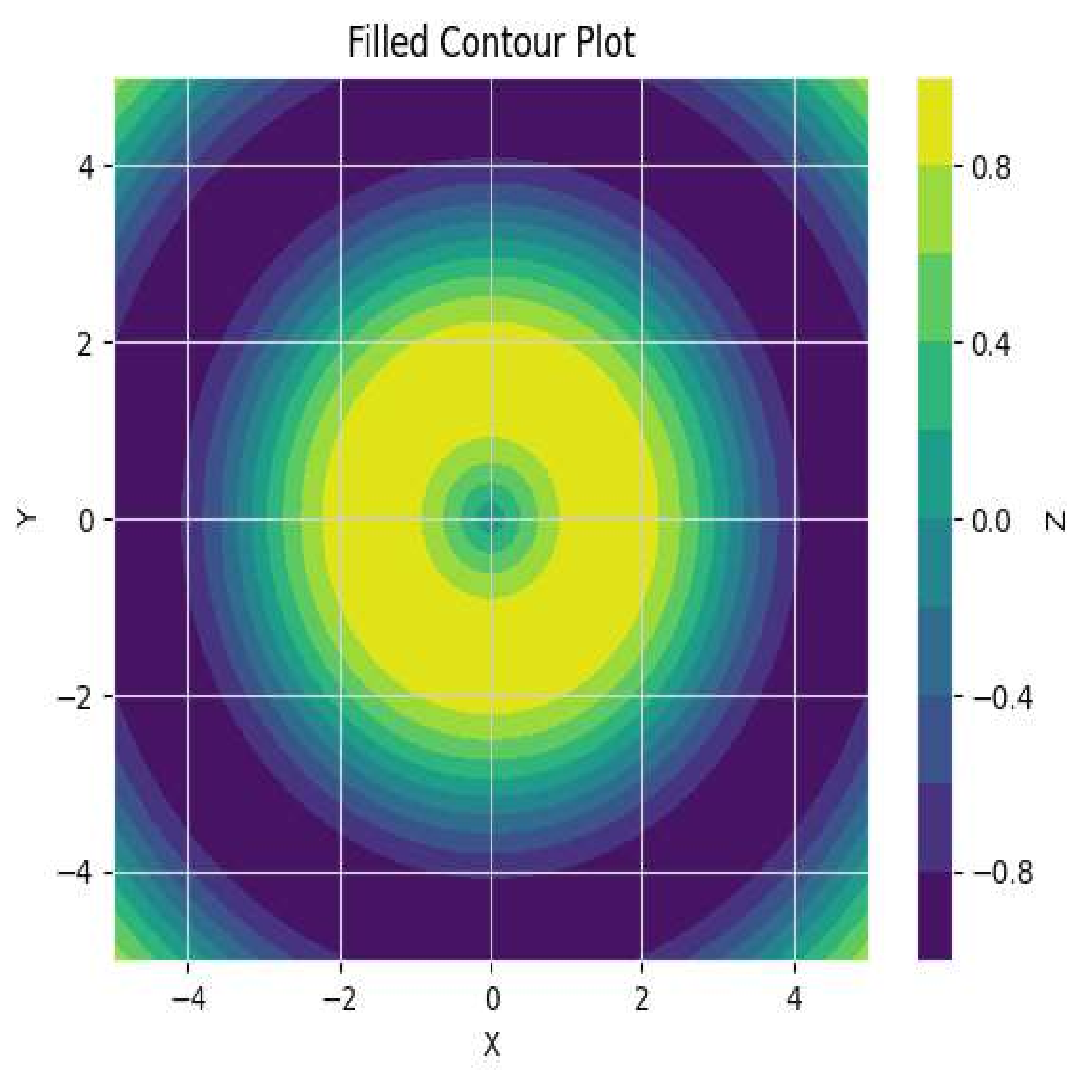
Filled Contour Plot

- **Syntax:**
  - plt.contourf(X, Y, Z, levels=None, cmap=None)
- **Key Attribute:-**
  - **X, Y:** 2D arrays or 1D arrays for grid coordinates.
  - **Z:** 2D array of values to fill.
  - **levels:** Number of levels or specific contour values.
  - **cmap:** Colormap for filled areas (e.g., 'viridis', 'hot').

Example

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

- x = np.linspace(-5, 5, 100)
- y = np.linspace(-5, 5, 100)
- X, Y = np.meshgrid(x, y)
- Z = np.sin(np.sqrt(X**2 + Y**2))
- plt.contourf(X, Y, Z, levels=10, cmap='viridis')
- plt.colorbar(label='Z')
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Filled Contour Plot')
- plt.show()



Heatmap

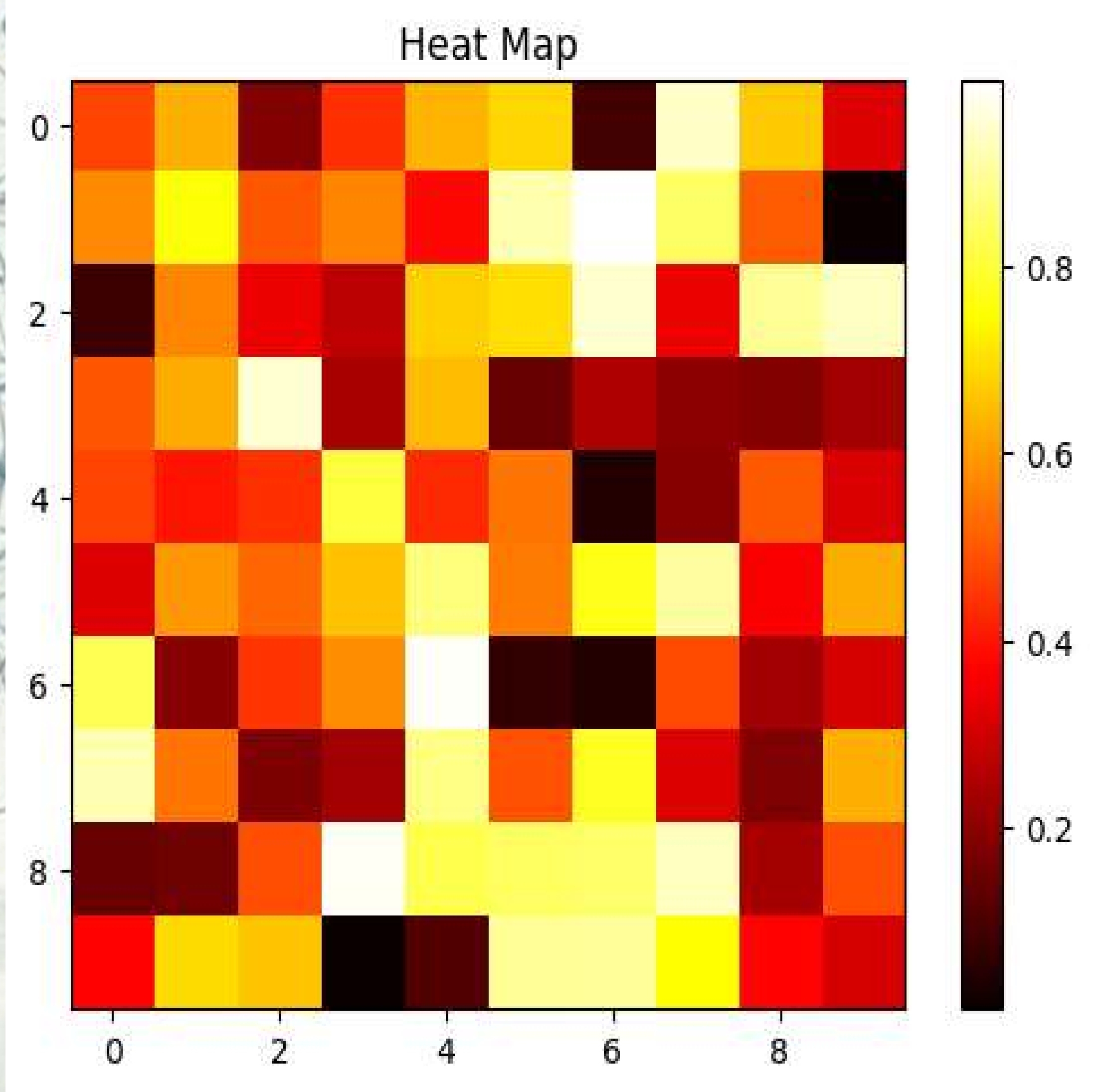
- **Purpose:**
  - Visualizes 2D data as a colored grid, where each cell's color represents the data value.
- **Use Case:**
  - Display correlation matrices or confusion matrices.
  - Visualize image data or pixel intensities.
  - Show spatial data distributions (e.g., population density).
- **Key Features:**
  - Uses colormaps to represent values.
  - Supports interpolation for smoother visuals.
  - Can include colorbars for value reference.

Heatmap

- **Syntax:**
  - plt.imshow(Z, cmap=None, interpolation=None, aspect=None)
- **Key Attribute:**
  - **Z:** 2D array of values to display.
  - **cmap**: Colormap (e.g., 'hot', 'cool', 'viridis').
  - **interpolation:** Interpolation method (e.g., 'nearest', 'bilinear').
  - **aspect:** Aspect ratio of the plot (e.g., 'equal', 'auto').

Example

- import matplotlib.pyplot as plt
- import numpy as np
- data = np.random.rand(10, 10)
- plt.imshow(data, cmap='hot', interpolation='nearest')
- plt.colorbar()
- plt.title('Heat Map')
- plt.show()



Hexbin Plot

- **Purpose:**
  - Aggregates dense 2D data into hexagonal bins, showing the density of points in each bin.
- **Use Case:**
  - Visualize large datasets with overlapping points (e.g., scatter data).
  - Analyze point density in scientific experiments.
  - Explore distributions in geospatial data.
- **Key Features:**
  - Hexagonal bins reduce overplotting in dense datasets.
  - Customizable bin size and colormap.
  - Can show counts or aggregated statistics per bin.

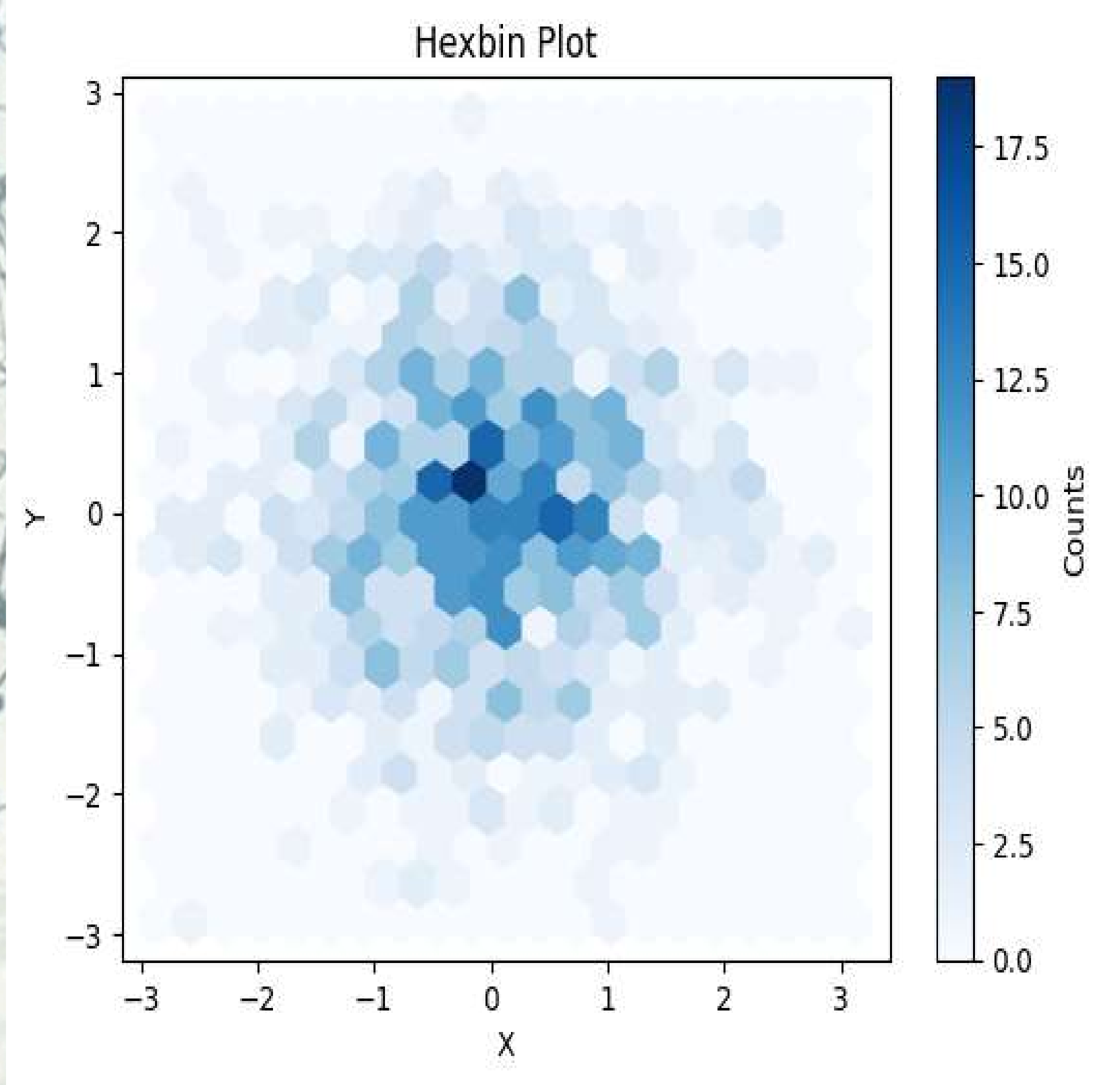
Hexbin Plot

- **Syntax:**
  - plt.hexbin(x, y, gridsize=30, cmap=None, reduce_C_function=None)
- **Key Attribute:**
  - **x, y:** Arrays of x and y coordinates.
  - **gridsize:** Number of hexagons in the x-direction.
  - **cmap:** Colormap for bin density (e.g., 'Blues').
  - **reduce_C_function:** Function to aggregate data in bins (e.g., np.mean).

Example

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

- x = np.random.randn(1000)
- y = np.random.randn(1000)
- plt.hexbin(x, y, gridsize=20, cmap='Blues')
- plt.colorbar(label='Counts')
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Hexbin Plot')
- plt.show()



Quiver Plot

- **Purpose:**
  - Displays vector fields by plotting arrows representing direction and magnitude at given points.
- **Use Case:-**
  - Visualize fluid flow or electromagnetic fields.
  - Show gradients or motion in scientific simulations.
  - Represent wind speed and direction in meteorology.
- **Key Features:**
  - Arrows indicate direction and magnitude of vectors.
  - Customizable arrow size, color, and scaling.
  - Works with 2D grids of vector components.

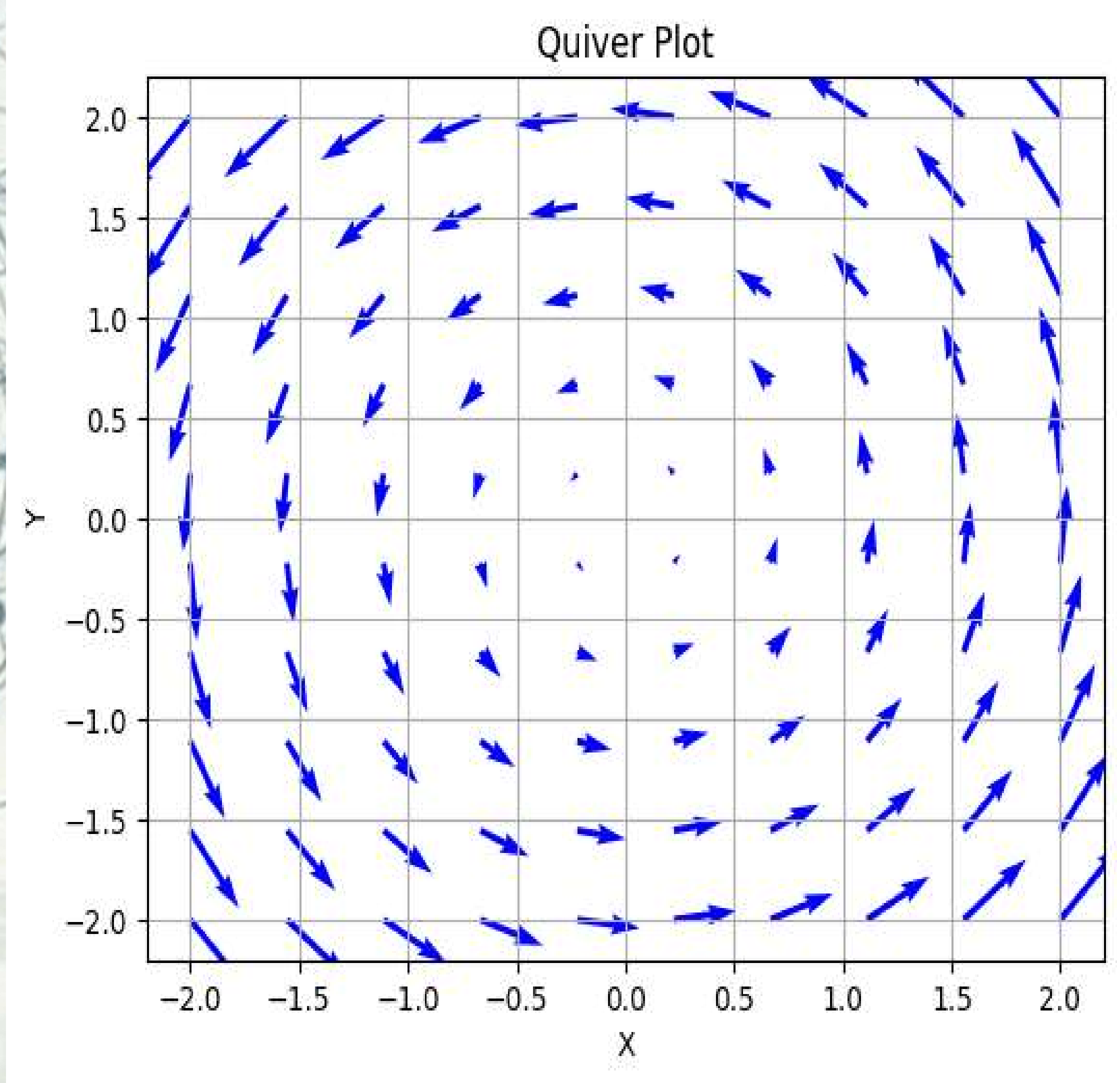
Quiver Plot

- **Syntax:**
  - plt.quiver(X, Y, U, V, scale=None, color=None)

- **Key Attribute:-**
  - X, Y: Coordinates of arrow starting points.
  - U, V: Vector components in x and y directions.
  - scale: Scaling factor for arrow lengths.
  - color: Color of arrows.

Example

- import matplotlib.pyplot as plt
- import numpy as np
- %matplotlib inline
- x = np.linspace(-2, 2, 10)
- y = np.linspace(-2, 2, 10)
- X, Y = np.meshgrid(x, y)
- U = -Y
- V = X
- plt.quiver(X, Y, U, V, color='blue')
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Quiver Plot')
- plt.grid(True)
- plt.show()



Stream Plot

- **Purpose:**
  - Visualizes vector fields with continuous streamlines, showing flow patterns more smoothly than quiver plots.
- **Use Case:**
  - Visualize fluid dynamics or airflow patterns.
  - Display electromagnetic fields or gradients.
  - Analyze complex flow in simulations.
- **Key Features:**
  - Draws smooth, continuous streamlines.
  - Customizable line width, color, and density.
  - Works with 2D grids of vector components.

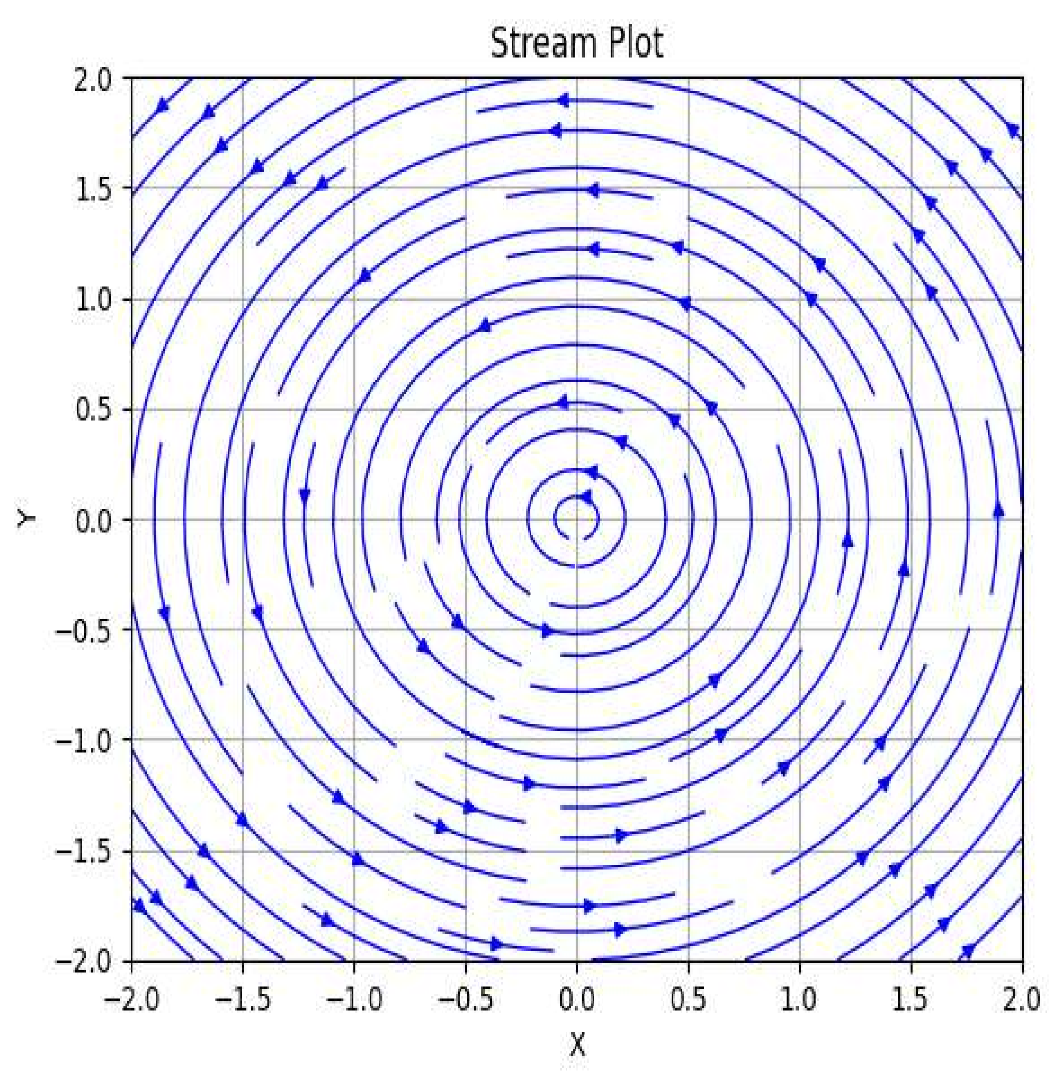
Stream Plot

- **Syntax:**
  - plt.streamplot(X, Y, U, V, density=1, linewidth=None, color=None)
- **Key Attribute:**
  - **X, Y:** 2D arrays defining grid coordinates.
  - **U, V:** 2D arrays of x and y vector components.
  - **density:** Controls streamline density.
  - **linewidth:** Width of streamlines (can vary with magnitude).
  - **color:** Color of streamlines or mapped to magnitude.

Example

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

- x = np.linspace(-2, 2, 20)
- y = np.linspace(-2, 2, 20)
- X, Y = np.meshgrid(x, y)
- U = -Y
- V = X
- plt.streamplot(X, Y, U, V, density=1, linewidth=1, color='blue')
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Stream Plot')
- plt.grid(True)
- plt.show()



Error Bar Plot

- **Purpose:**
  - Plots data points with error bars to indicate uncertainty or variability in measurements.
- **Use Case:**
  - Display experimental data with measurement errors.
  - Show statistical confidence intervals.
  - Visualize variability in scientific or financial data.
- **Key Features:**
  - Adds error bars to data points for x and/or y axes.
  - Customizable error bar styles and colors.
  - Supports asymmetric error ranges.

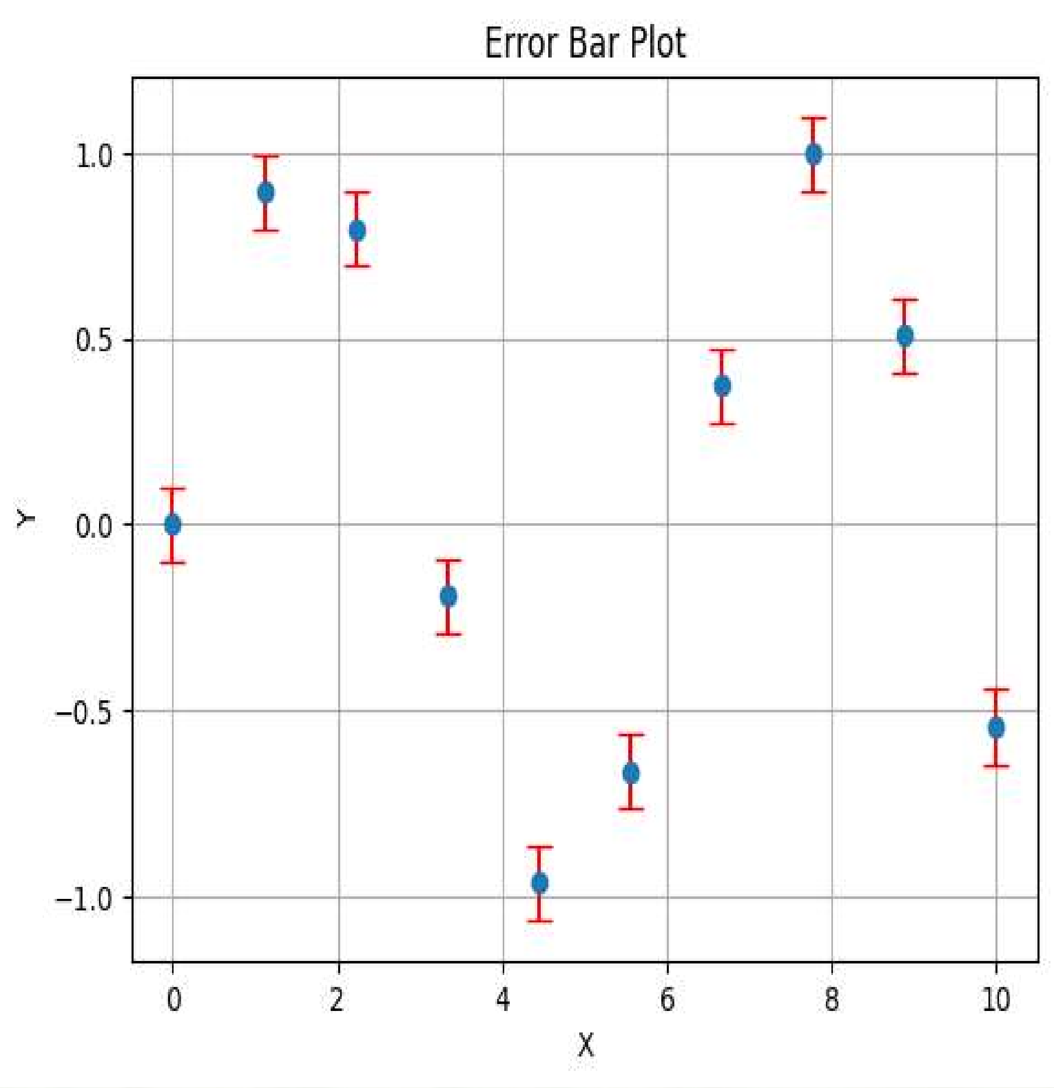
Error Bar Plot

- **Syntax:**
  - plt.errorbar(x, y, yerr=None, xerr=None, fmt=None, ecolor=None)
- **Key Attribute:**
  - x, y: Data for x and y axes.
  - yerr, xerr: Error values for y and x (scalar, array, or asymmetric).
  - fmt: Format of the data points (e.g., 'o' for circles).
  - ecolor: Color of error bars.

Example

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

- x = np.linspace(0, 10, 10)
- y = np.sin(x)
- yerr = 0.1
- np.random.rand(10)
- plt.errorbar(x, y, yerr=yerr, fmt='o', ecolor='red', capsize=5)
- plt.xlabel('X')
- plt.ylabel('Y')
- plt.title('Error Bar Plot')
- plt.grid(True)
- plt.show()



Stacked Bar

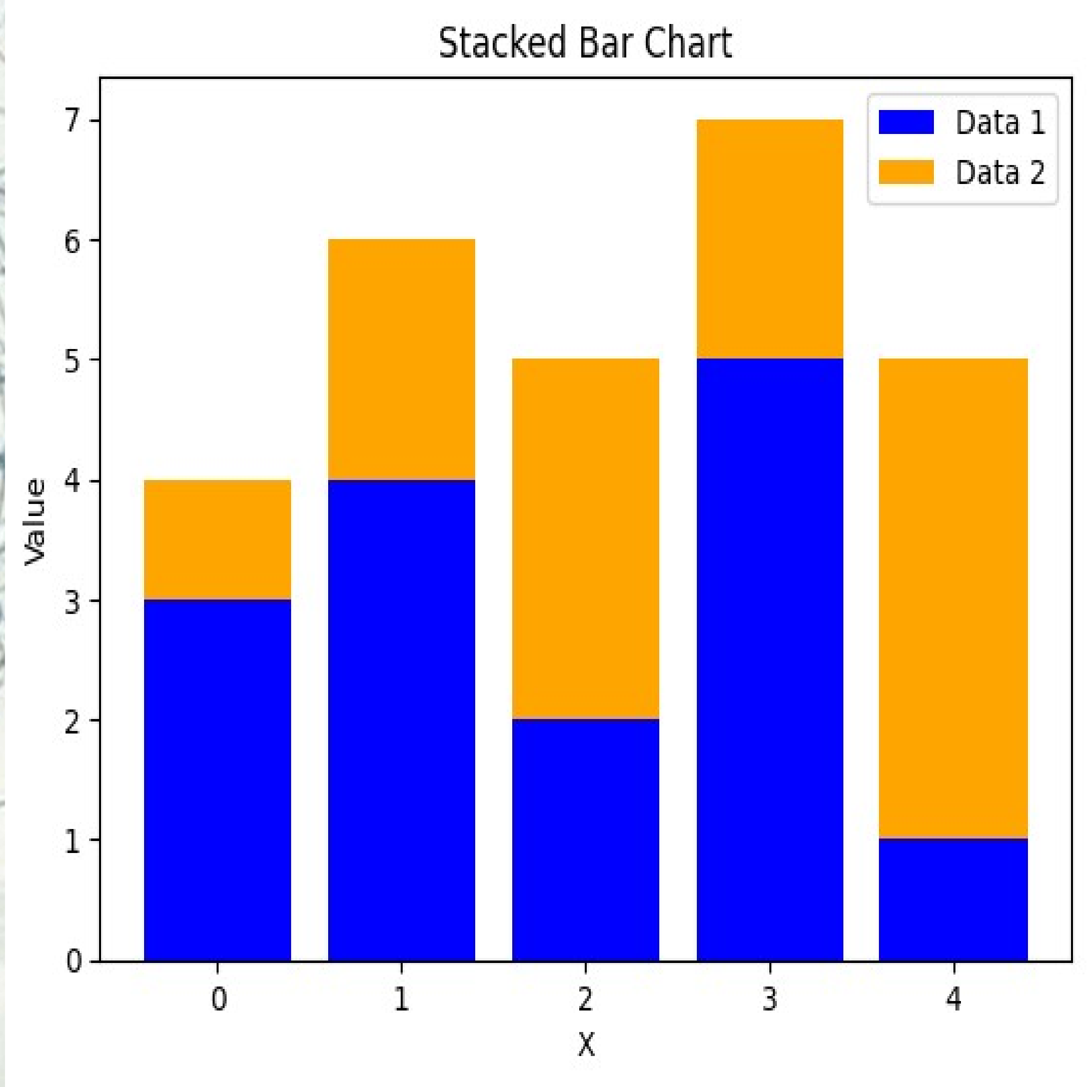
- **Purpose:**
  - Displays multiple datasets as bars stacked on top of each other to show cumulative contributions.
- **Use Case:**
  - Compare contributions of categories over time (e.g., sales by product).
  - Visualize parts-to-whole relationships.
  - Show cumulative metrics across groups.
- **Key Features:**
  - Stacks bars vertically to show total and individual contributions.
  - Customizable bar colors, widths, and labels.
  - Supports legends for multiple datasets.

Stacked Bar

- **Syntax:**
  - plt.bar(x, height, bottom=None, color=None, label=None)
- **Key Attribute:-**
  - **x**: Positions of bars on the x-axis.
  - **height**: Heights of bars or data values.
  - **bottom**: Starting height for each bar (for stacking).
  - **color**: Color of bars.
  - **label**: Label for legend.

Example

- import matplotlib.pyplot as plt
- import numpy as np
- x = np.arange(5)
- data1 = [3, 4, 2, 5, 1]
- data2 = [1, 2, 3, 2, 4]
- plt.bar(x,data1,color='blue', label='Data 1')
- plt.bar(x,data2,bottom=data1, color='orange', label='Data 2')
- plt.xlabel('X')
- plt.ylabel('Value')
- plt.title('Stacked Bar Chart')
- plt.legend()
- plt.show()



Stacked Area Plot

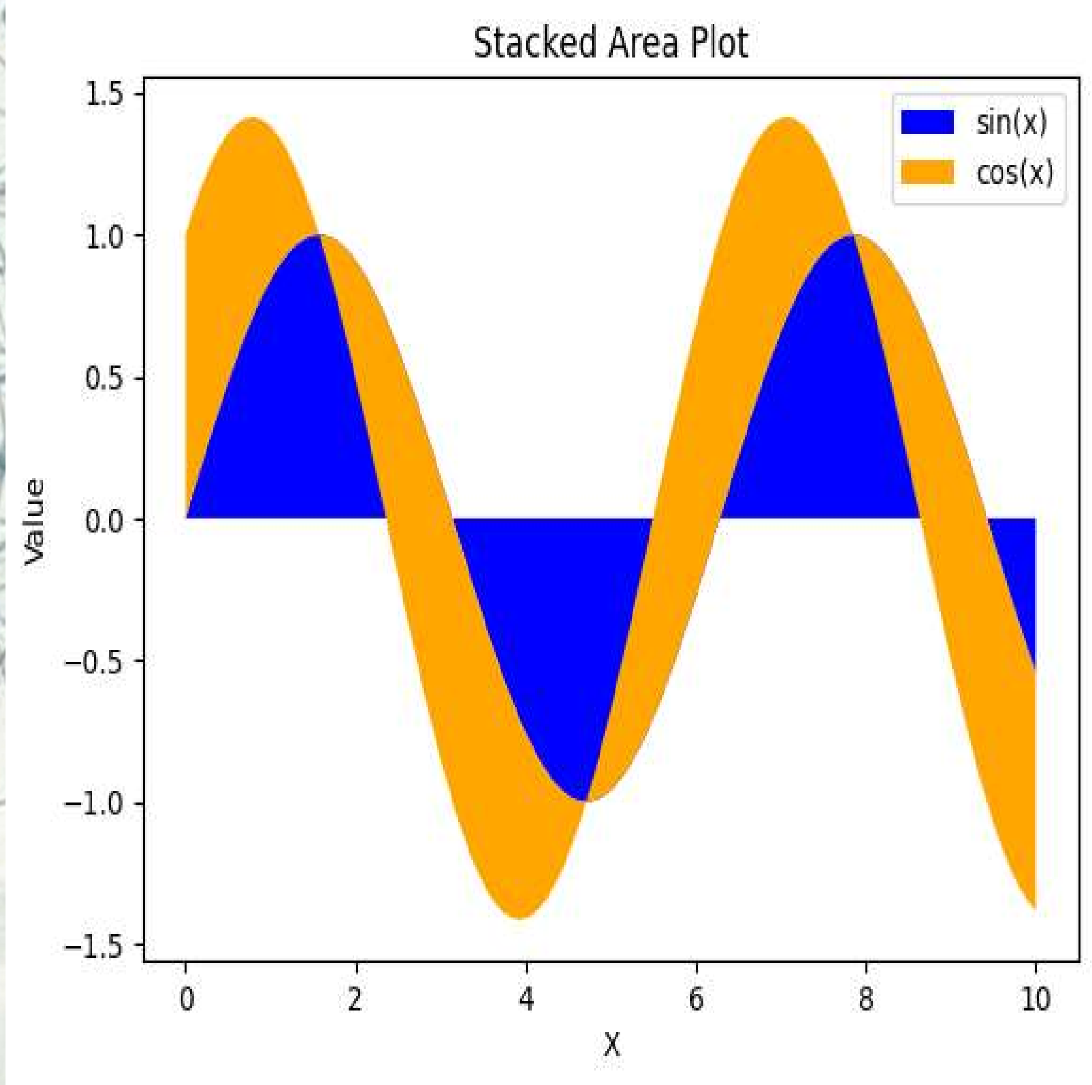
- **Purpose:**
  - Shows the cumulative contribution of multiple datasets over a continuous axis, filling areas between lines.
- **Use Case:**
  - Visualize time series contributions (e.g., energy sources over time).
  - Show parts-to-whole relationships in trends.
  - Analyze cumulative growth or decline.
- **Key Features:**
  - Fills areas under curves to show cumulative totals.
  - Supports multiple datasets with customizable colors.
  - Smooth visualization of trends.

Stacked Area Plot

- **Syntax:**
  - plt.stackplot(x, args, labels=None, colors=None)
- **Key Attribute:**
  - **x:** X-axis data (common for all datasets).
  - **args:** Sequence of y-data for each dataset to stack.
  - **labels:** Labels for legend.
  - **colors:** Colors for each dataset.

Example

- import matplotlib.pyplot as plt
- import numpy as np
- x = np.linspace(0, 10, 100)
- y1 = np.sin(x)
- y2 = np.cos(x)
- plt.stackplot(x, y1, y2, labels=['sin(x)', 'cos(x)'], colors=['blue', 'orange'])
- plt.xlabel('X')
- plt.ylabel('Value')
- plt.title('Stacked Area Plot')
- plt.legend()
- plt.show()



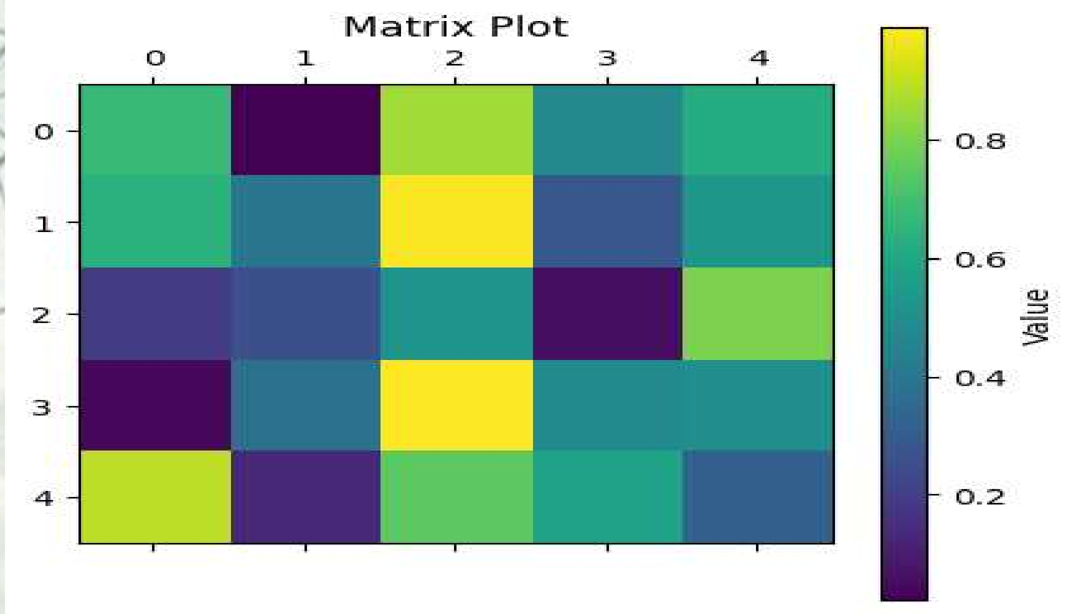
Matrix Plot

- **Purpose:**
  - Visualizes a 2D matrix as a colored grid, similar to a heatmap, with equal-sized cells.
- **Use Case:**
  - Display correlation matrices or adjacency matrices.
  - Visualize image data or pixel intensities.
  - Show structured data like confusion matrices.
- **Key Features:**
  - Automatically sets equal aspect ratio for cells.
  - Supports colormaps for value representation.
  - Can include colorbars for reference.

Matrix Plot

- **Syntax:**
  - plt.matshow(A, cmap=None, interpolation=None)
- **Key Attribute:**
  - **A:** 2D array to visualize.
  - **cmap:** Colormap (e.g., 'viridis', 'hot').
  - **interpolation:** Interpolation method (e.g., 'nearest').

- import matplotlib.pyplot as plt
- import numpy as np
- %matplotlib inline
- A = np.random.rand(5, 5)
- plt.matshow(A, cmap='viridis')
- plt.colorbar(label='Value')
- plt.title('Matrix Plot')
- plt.show()



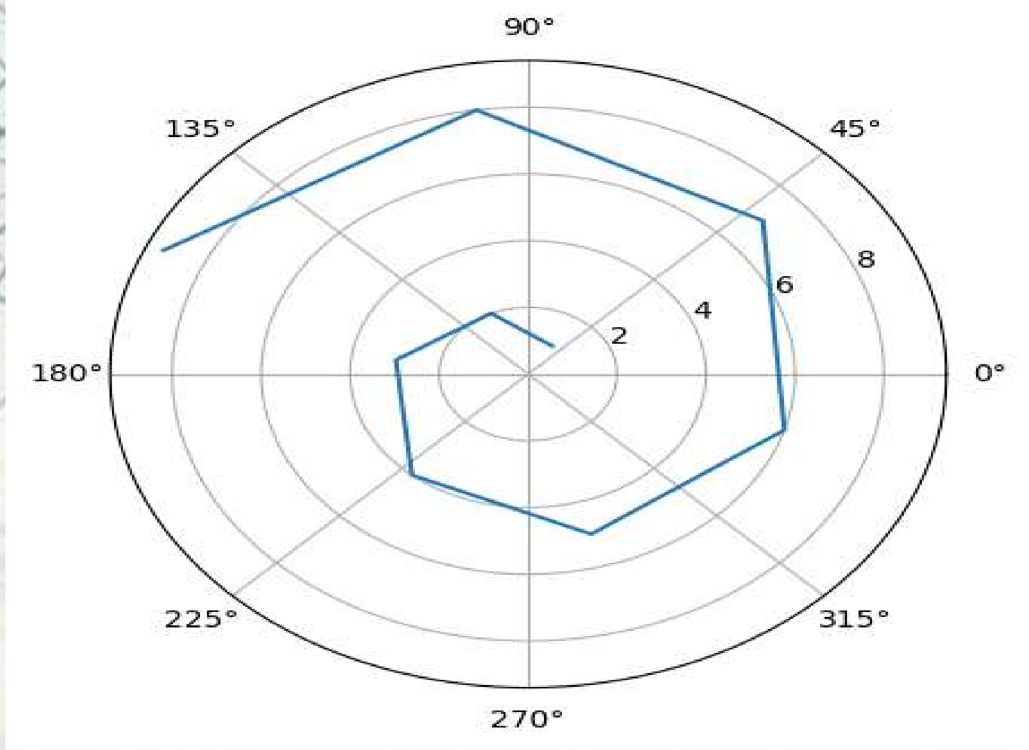
Polar Plot

- **Purpose:**
  - Plots data in polar coordinates, where points are defined by radius and angle, ideal for circular or periodic data.
- **Use Case:**
  - Visualize directional data (e.g., wind directions).
  - Plot periodic phenomena (e.g., antenna patterns).
  - Create radar or rose diagrams.
- **Key Features:**
  - Uses polar coordinates (theta, r) instead of Cartesian.
  - Customizable line styles, colors, and markers.
  - Supports radial and angular grid customization.
- **Syntax:**
  - plt.polar(theta, r, linestyle=None, color=None, marker=None)

Polar Plot

- **Key Attribute:**
- theta: Angles in radians for data points.
- r: Radial distances from the origin.
- linestyle: Style of the line (e.g., '-' for solid).
- color: Color of the plot.
- marker: Marker style for data points.

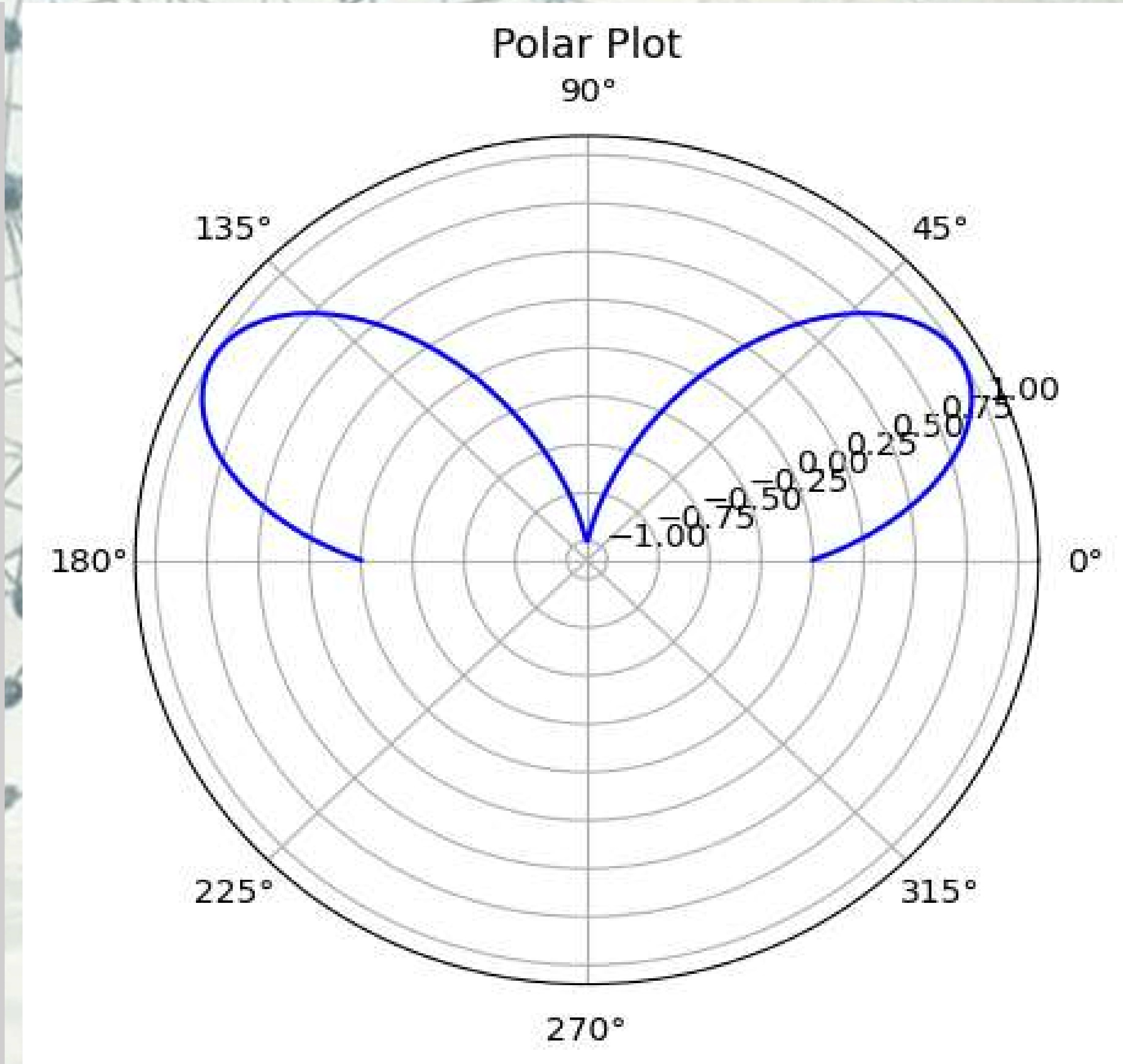
```
• import numpy as np
• import matplotlib.pyplot as plt
• %matplotlib inline
• x1= (np.arange(1,10))
• y1= (np.arange(1,10))
• plt.polar(x1,y1)
• plt.show()
```



Example

- import matplotlib.pyplot as plt
- %matplotlib inline
- import numpy as np

- theta = np.linspace(0, np.pi, 100)
- r = np.sin(3 * theta)
- plt.polar(theta, r, color='blue', linestyle='-')
- plt.title('Polar Plot')
- plt.show()



Candlestick Plot

- **Purpose:**
  - Visualizes financial time series data (e.g., stock prices) with open, high, low, and close values in a candlestick format.
- **Use Case:**
  - Display stock or cryptocurrency price movements.
  - Analyze financial market trends.
  - Visualize trading data over time.
- **Key Features:**
  - Shows price range and direction (up/down) for each period.
  - Customizable colors for bullish (up) and bearish (down) candles.
  - Requires data in OHLC (Open, High, Low, Close) format.

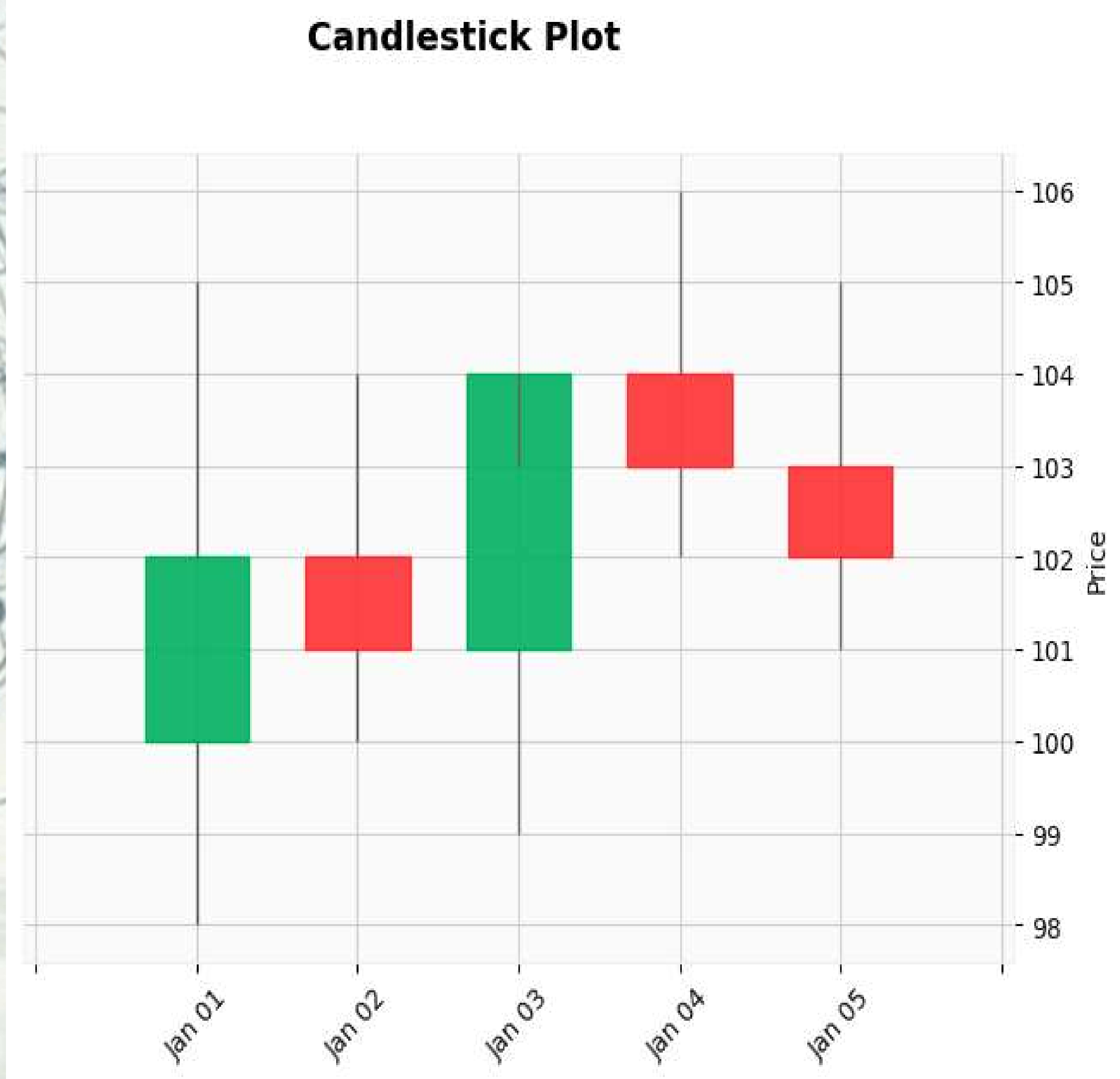
Candlestick Plot

- **Syntax:**
  - mpf.plot(data, type='candle')
- **Key Attribute:**
  - **data:** DataFrame with columns [Date, Open, High, Low, Close].
  - **type='candle':** Specifies candlestick plot.
  - **style:** Plot style (e.g., 'classic', 'yahoo').

Example

- import mplfinance as mpf
- import pandas as pd
- import numpy as np

- dates = pd.date_range('2023-01-01', periods=5)
- data = pd.DataFrame({
- 'Date': dates,
- 'Open': [100, 102, 101, 104, 103],
- 'High': [105, 104, 103, 106, 105],
- 'Low': [98, 100, 99, 102, 101],
- 'Close': [102, 101, 104, 103, 102]
- })
- data.set_index('Date', inplace=True)
- mpf.plot(data, type='candle', title='Candlestick Plot', style='yahoo')



Violin Plot

- **Purpose:**
  - Visualizes the distribution and density of data across multiple categories, combining box plot and kernel density estimation.
- **Use Case:**
  - Compare distributions across groups (e.g., test scores by class).
  - Analyze data spread and skewness in statistics.
  - Visualize data with multimodal distributions.
- **Key Features:**
  - Shows data density, median, and quartiles.
  - Customizable for multiple datasets.
  - Supports vertical or horizontal orientation.

Violin Plot

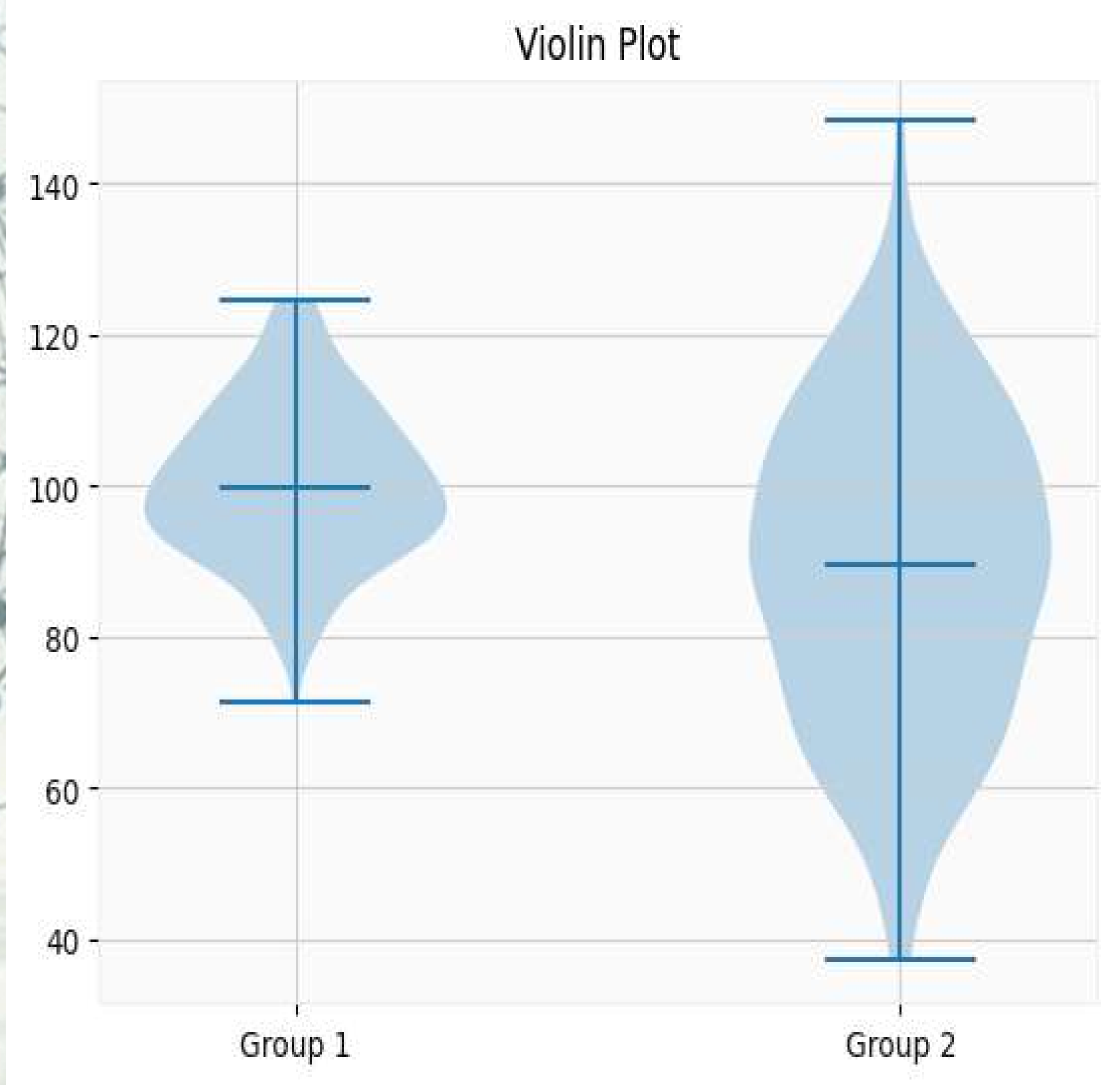
- **Syntax:**
  - plt.violinplot(dataset, positions=None, vert=True, showmeans=False)

- **Key Attribute:**
  - **dataset:** List of arrays or single array of data.
  - **positions:** X-positions for violins.
  - **vert:** Orientation (True for vertical, False for horizontal).
  - **showmeans:** Display mean values.

Example

- import numpy as np
- import matplotlib.pyplot as plt
- %matplotlib inline

- data = [np.random.normal(100, 10, 200), np.random.normal(90, 20, 200)]
- plt.violinplot(data, showmedians=True)
- plt.xticks([1, 2], ['Group 1', 'Group 2'])
- plt.title('Violin Plot')
- plt.show()



Example of subplot

- import numpy as np
- import matplotlib.pyplot as plt
- %matplotlib inline
- x = np.arange(-1,5,0.5)
- plt.subplot(2,3,1)
- plt.plot(x,x*2,'g.-')
- plt.subplot(2,3,2)
- plt.plot(x,np.cos(x),'b*--')
- plt.subplot(2,3,3)
- plt.plot(x,np.sin(x),c='r')
- plt.subplot(2,3,4)
- plt.plot(x,np.log1p(x),'g.--')
- plt.show()