Overview of Matplotlib

What is Matplotlib ?

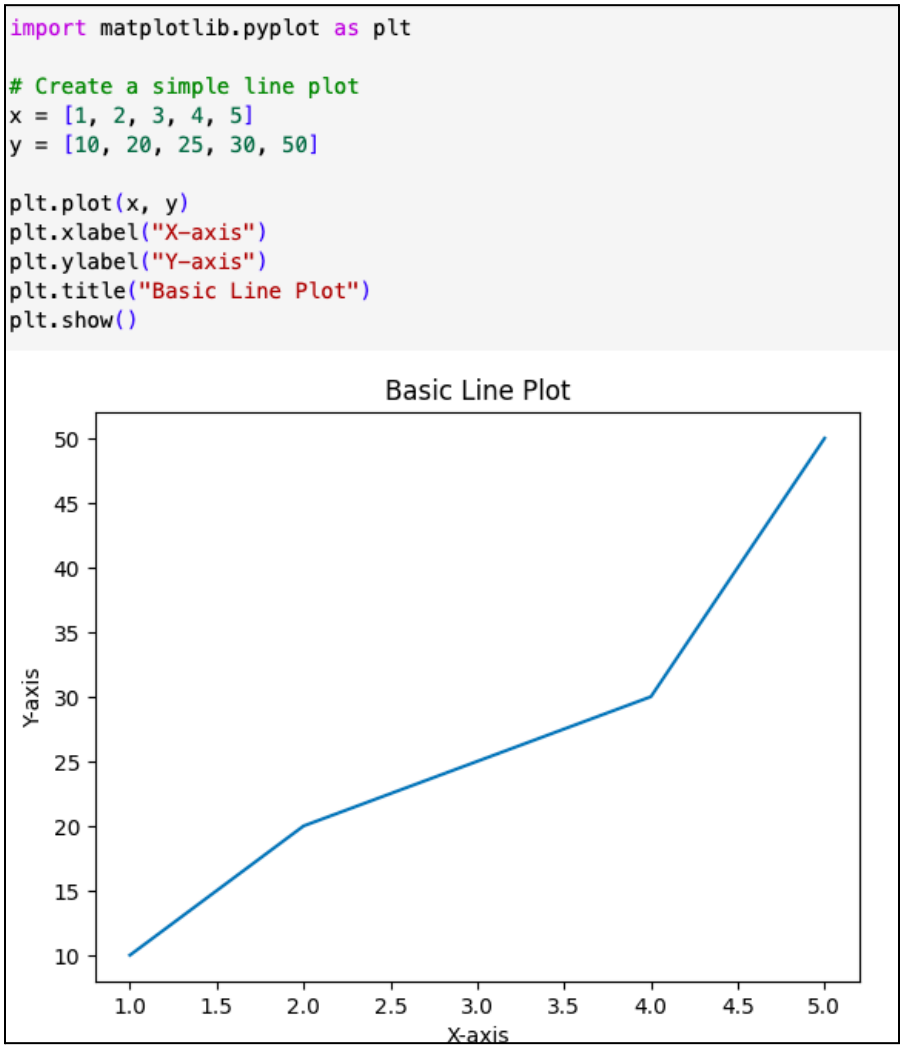
Matplotlib is a popular Python library used for creating static, animated, and interactive visualizations. It provides a comprehensive set of tools for generating plots, charts, and graphs, making it widely used in data science, engineering, and scientific computing.

How to Install:

```
pip install matplotlib
```

Code Example:

```python
import matplotlib.pyplot as plt

# Create a simple line plot
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 50]

plt.plot(x, y)
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Basic Line Plot")
plt.show()
```



Some commonly used methods:

Method	Description
`plot()`	Plot a line graph.
`scatter()`	Plot a scatter plot.
`hist()`	Create a histogram.
`bar()`	Create a bar chart.
`barh()`	Create a horizontal bar chart.
`xlabel()`	Label the x-axis.
`ylabel()`	Label the y-axis.
`title()`	Add a title to the plot.
`legend()`	Add a legend to the plot.
`show()`	Display the plot.
`savefig()`	Save the figure to an image file.
`subplots()`	Create multiple subplots in one figure.
`axis()`	Set the limits of the axes.
`xlim()`	Set the limits for the x-axis.
`ylim()`	Set the limits for the y-axis.
`grid()`	Add gridlines to the plot.
`annotate()`	Annotate a point on the plot.
`fill()`	Fill an area between curves.
`hist2d()`	Create a 2D histogram.
`imshow()`	Display an image or heatmap.

Pairwise Data Plots:

Pairwise plots display relationships between multiple variables in a dataset. They show how each pair of variables is related using scatter plots or other types.

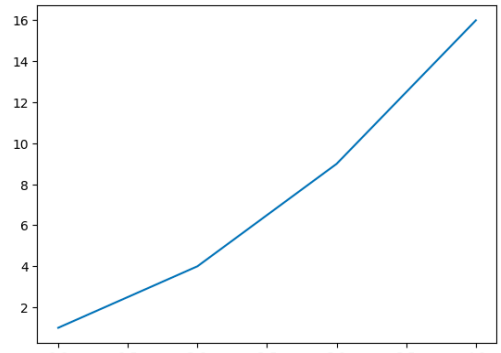
Some widely used Plot types:

1. Plot:

It is used to create line plots. It connects data points with a line.

Code example:

```python
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]

# Line plot
plt.plot(x, y)
plt.show()
```



Formatting style methods for Plot(x,y):

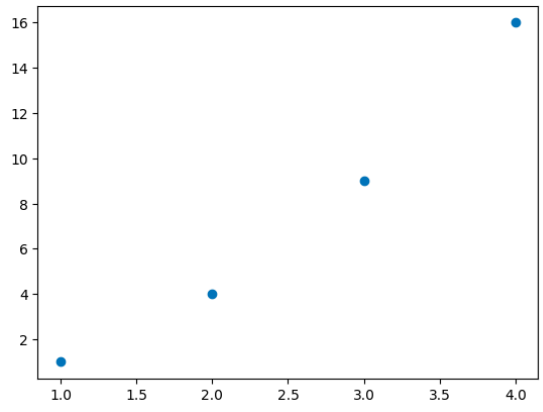
Method	Definition	Code Example
Color	Specifies the color of the plot line	`plt.plot(x, y, color='red')` `plt.plot(x, y, color='#FF5733')`
Line Style	Defines the style of the line (solid, dashed, etc.)	`plt.plot(x, y, linestyle='--')` `plt.plot(x, y, linestyle=':')`
Line Width	Sets the thickness of the line	`plt.plot(x, y, linewidth=2)`
Marker	Adds markers at data points	`plt.plot(x, y, marker='o')` `plt.plot(x, y, marker='x')`
Marker Size	Adjusts the size of the markers	`plt.plot(x, y, marker='o', markersize=8)`
Marker Edge Color	Specifies the edge color of the markers	`plt.plot(x, y, marker='o', markeredgecolor='blue')`
Marker Face Color	Specifies the color inside the markers	`plt.plot(x, y, marker='o', markerfacecolor='yellow')`
Alpha (Transparency)	Sets the transparency level of the line	`plt.plot(x, y, alpha=0.5)`

2. Scatter Plot:

Scatter plots show individual data points as dots on a two-dimensional graph. They are useful for visualizing relationships between two variables.

Code example:

```python
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]

# Scatter plot
plt.scatter(x, y)
plt.show()
```



Formatting style methods for Scatter Plot:

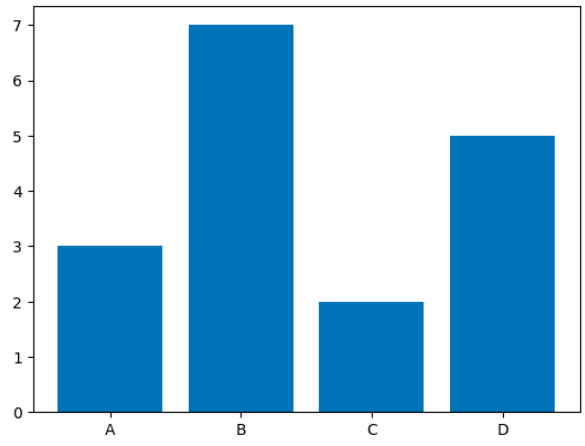
Method	Definition	Code Example
Size (`s`)	Sets the size of the points (in points squared).	`plt.scatter(x, y, s=100)`
Edge Color (`edgecolor`)	Specifies the color of the point's edge.	`plt.scatter(x, y, edgecolor='black')`
Face Color (`facecolor`)	Specifies the color of the point's interior.	`plt.scatter(x, y, facecolor='yellow')`
Line Width (`linewidths`)	Controls the width of the point's edge.	`plt.scatter(x, y, linewidths=2)`
Color Map (`c`)	Applies a color map based on the data values for coloring points.	`plt.scatter(x, y, c=z, cmap='viridis')` # z is used for color mapping
Marker Size (Array `s`)	Sets different sizes for individual points.	`plt.scatter(x, y, s=[20, 40, 60, 80])`

3. Bar Plot:

Bar plots display data with rectangular bars, with the length of each bar proportional to the value.

Code example:

```python
import matplotlib.pyplot as plt

# Data
x = ['A', 'B', 'C', 'D']
y = [3, 7, 2, 5]

# Bar plot
plt.bar(x, y)
plt.show()
```



Formatting style methods for Bar Plot:

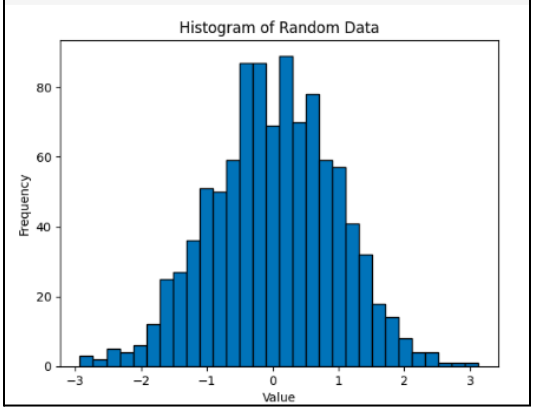
Method	Definition	Code Example
Height (`height`)	Specifies the height of the bars (y-values).	`plt.bar(x, height=[5, 10, 15])`
Width (`width`)	Sets the width of the bars.	`plt.bar(x, height=[5, 10, 15], width=0.5)`
Edge Color (`edgecolor`)	Defines the color of the bar edges.	`plt.bar(x, height=[5, 10, 15], edgecolor='black')`
Tick Labels (`tick_label`)	Specifies the labels for the x-ticks (categorical).	`plt.bar(x, height=[5, 10, 15], tick_label=['A', 'B', 'C'])`
Align	Sets the alignment of the bars (left, center, right).	`plt.bar(x, height=[5, 10, 15], align='center')`
Logarithmic Scaling (`log`)	Sets logarithmic scaling for the y-axis.	`plt.bar(x, height=[5, 10, 15], log=True)`

4. Hist Plot:

A **histogram plot** is used to represent the distribution of data by dividing it into bins and counting the number of data points that fall into each bin.

Code example:

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

# Generating random data
data = np.random.randn(1000)

# Create a histogram
plt.hist(data, bins=30, edgecolor='black')

# Add labels and title
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram of Random Data')

# Show the plot
plt.show()
```



Formatting style methods for Hist Plot:

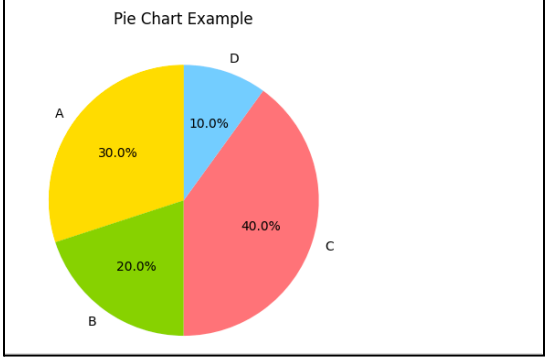
Method	Definition	Code Example
Bins (`bins`)	Specifies the number of bins or the bin edges.	`plt.hist(data, bins=10)`
Histtype (`histtype`)	Defines the type of histogram (e.g., 'bar', 'step', 'stepfilled').	`plt.hist(data, histtype='step')`
Stacked (`stacked`)	Stacks multiple histograms on top of each other if multiple datasets are provided.	`plt.hist([data1, data2], stacked=True)`
Rwidth (`rwidth`)	Controls the width of the bars relative to the available space.	`plt.hist(data, rwidth=0.8)`
Density (`density`)	Normalizes the histogram so that the total area under the histogram equals 1.	`plt.hist(data, density=True)`
Cumulative (`cumulative`)	Plots the cumulative histogram.	`plt.hist(data, cumulative=True)`

5. Pie Plot:

A **pie plot** is used to represent categorical data in a circular format, where each category is represented as a slice. The size of each slice is proportional to the quantity it represents.

Code example:

```python
# Data for the pie chart
labels = ['A', 'B', 'C', 'D']
sizes = [30, 20, 40, 10]

# Create a pie chart
plt.pie(sizes,
        labels=labels,
        autopct='%1.1f%%',
        startangle=90,
        colors=['gold', 'yellowgreen', 'lightcoral', 'lightskyblue']
        )

# Add title
plt.title('Pie Chart Example')

# Show the plot
plt.show()
```



Formatting style methods for Pie Plot:

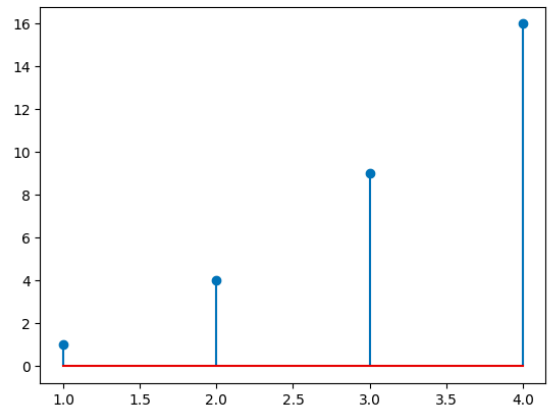
Method	Definition	Code Example
Labels (`labels`)	Specifies the labels for the wedges in the pie chart.	`plt.pie(sizes, labels=['A', 'B', 'C'])`
Explode (`explode`)	Offsets a wedge from the center, used for highlighting a slice.	`plt.pie(sizes, explode=(0.1, 0, 0))`
Autopct (`autopct`)	Displays the percentage value on each wedge.	`plt.pie(sizes, autopct='%1.1f%%')`
Shadow (`shadow`)	Adds a shadow effect behind the pie chart.	`plt.pie(sizes, shadow=True)`
Start Angle (`startangle`)	Rotates the start of the pie chart by the specified angle.	`plt.pie(sizes, startangle=90)`
Radius (`radius`)	Scales the size of the pie chart.	`plt.pie(sizes, radius=1.2)`
Colors (`colors`)	Defines the colors of the wedges.	`plt.pie(sizes, colors=['r', 'g', 'b'])`
Wedgeprops (`wedgeprops`)	Customizes the properties of the wedges (e.g., line width).	`plt.pie(sizes, wedgeprops={'linewidth': 2, 'edgecolor': 'black'})`
PCTdistance (`pctdistance`)	Sets the distance of the percentage labels from the center.	`plt.pie(sizes, pctdistance=0.85)`

6. Stem Plot:

Stem plots display data as stems and leaves, with lines extending from a baseline to represent values.

Code example:

```
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]

# Stem plot
plt.stem(x, y)
plt.show()
```



Formatting style methods for Stem Plot:

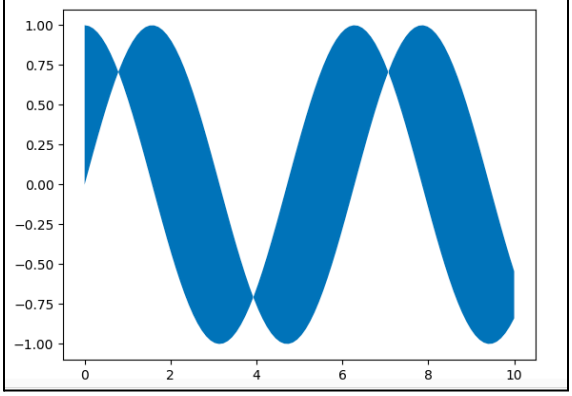
Method	Definition	Code Example
Line Style (`linefmt`)	Specifies the style of the lines that extend from the baseline.	`plt.stem(x, y, linefmt='r-')` # Red solid lines
Marker Style (`markerfmt`)	Defines the style of the markers (dots) at the end of the lines.	`plt.stem(x, y, markerfmt='go')` # Green circles
Base Line Style (`basefmt`)	Sets the style of the baseline (horizontal line from which stems extend).	`plt.stem(x, y, basefmt='k-')` # Black solid baseline
Orientation (`orientation`)	Adjusts whether the stem plot is vertical or horizontal.	`plt.stem(x, y, orientation='horizontal')`
Use Negative Values (`use_line_collection`)	Controls whether a `LineCollection` object is used to optimize performance for large datasets.	`plt.stem(x, y, use_line_collection=True)`

7. Fill Between Plot:

The `fill_between()` function fills the area between two horizontal curves (typically, the x-axis and a line).

Code example:

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

# Data
x = np.linspace(0, 10, 100)
y1 = np.sin(x)
y2 = np.cos(x)

# Fill between
plt.fill_between(x, y1, y2)
plt.show()
```



Formatting style methods for Fill Between Plot:

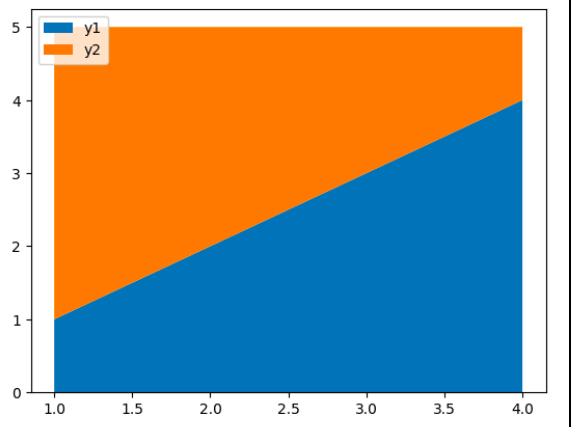
Method	Definition	Code Example
Y1 (`y1`)	Specifies the first set of y-values (lower boundary).	`plt.fill_between(x, y1=[2, 3, 4])`
Y2 (`y2`)	Specifies the second set of y-values (upper boundary).	`plt.fill_between(x, y1=[2, 3, 4], y2=[5, 6, 7])`
Where (`where`)	A condition to control which regions are filled.	`plt.fill_between(x, y1=[2, 3, 4], y2=[5, 6, 7], where=(y1 > 3))`
Color	Defines the color of the filled area.	`plt.fill_between(x, y1=[2, 3, 4], y2=[5, 6, 7], color='blue')`
Alpha (Transparency)	Controls the transparency of the filled area.	`plt.fill_between(x, y1=[2, 3, 4], y2=[5, 6, 7], alpha=0.5)`
Hatch	Adds patterns to the filled area (e.g., diagonal lines).	`plt.fill_between(x, y1=[2, 3, 4], y2=[5, 6, 7], hatch='//')`
Edge Color (`edgecolor`)	Defines the color of the border around the filled area.	`plt.fill_between(x, y1=[2, 3, 4], y2=[5, 6, 7], edgecolor='black')`
Line Width (`linewidth`)	Specifies the width of the border around the filled area.	`plt.fill_between(x, y1=[2, 3, 4], y2=[5, 6, 7], linewidth=2)`

8. Stackplot Plot:

A stack plot displays multiple data series stacked on top of each other, typically used for showing how parts contribute to the whole.

Code example:

```python
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4]
y1 = [1, 2, 3, 4]
y2 = [4, 3, 2, 1]

# Stack plot
plt.stackplot(x, y1, y2, labels=['y1', 'y2'])
plt.legend(loc='upper left')
plt.show()
```



Formatting style methods for Stack Plot:

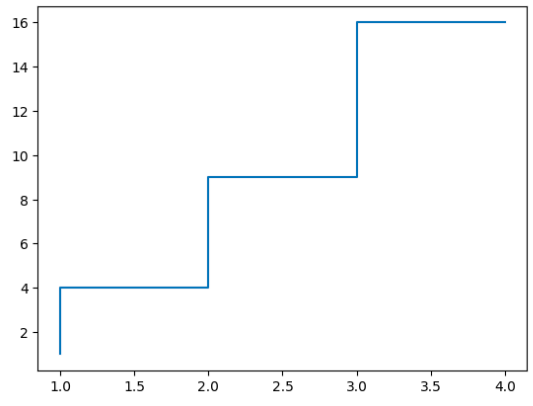
Method	Definition	Code Example
Values (`values`)	Defines the y-values for each stacked series.	`plt.stackplot(x, values1, values2)`
Labels	Specifies the labels for each stacked layer in the legend.	`plt.stackplot(x, values1, values2, labels=['Layer 1', 'Layer 2'])`
Colors	Sets the colors of the stacked areas.	`plt.stackplot(x, values1, values2, colors=['r', 'g'])`
Alpha (Transparency)	Controls the transparency of the stacked areas.	`plt.stackplot(x, values1, values2, alpha=0.6)`
Orientation (`orientation`)	Controls the direction of stacking (vertical/horizontal).	`plt.stackplot(x, values1, values2, orientation='horizontal')`
Edge Color (`edgecolor`)	Sets the color of the borders of the stacked areas.	`plt.stackplot(x, values1, values2, edgecolor='black')`
Line Width (`linewidth`)	Specifies the width of the borders around the stacked areas.	`plt.stackplot(x, values1, values2, linewidth=1.5)`

9. Stairs Plot:

A stairs plot creates a stepped line plot where the lines change value at discrete steps.

Code example:

```python
import matplotlib.pyplot as plt

# Data
x = [1, 2, 3, 4]
y = [1, 4, 9, 16]

# Stairs plot
plt.step(x, y)
plt.show()
```



Formatting style methods for Stairs Plot:

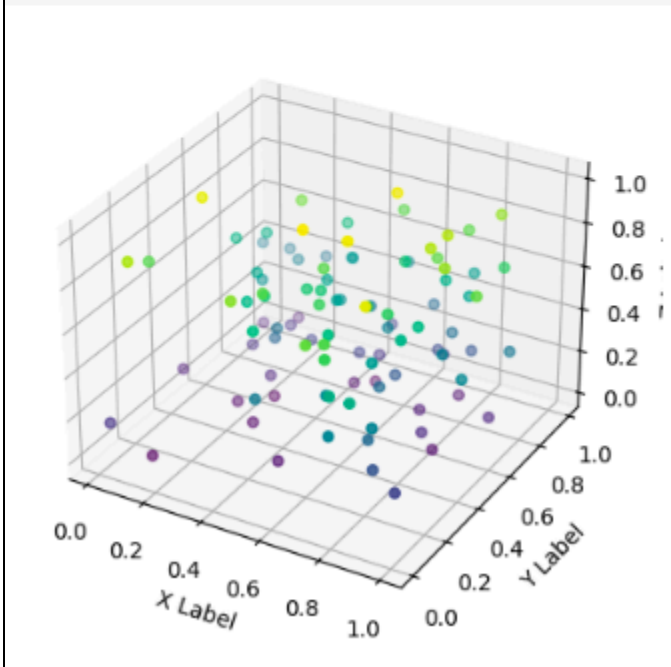
Method	Definition	Code Example
Direction (`where`)	Controls where the steps change (e.g., 'pre', 'post', 'mid').	`plt.step(x, y, where='mid')` # Step at the midpoint between points
Line Style (`linestyle`)	Specifies the line style for the steps.	`plt.step(x, y, linestyle='--')` # Dashed line steps
Color	Sets the color of the step lines.	`plt.step(x, y, color='purple')`
Alpha (Transparency)	Controls the transparency of the step lines.	`plt.step(x, y, alpha=0.7)`
Line Width (`linewidth`)	Controls the width of the step lines.	`plt.step(x, y, linewidth=2)`

3D Plotting:

Matplotlib also supports **3D plotting** through the `projection='3d`. You can create 3D plots such as scatter plots, surface plots, and wireframe plots.

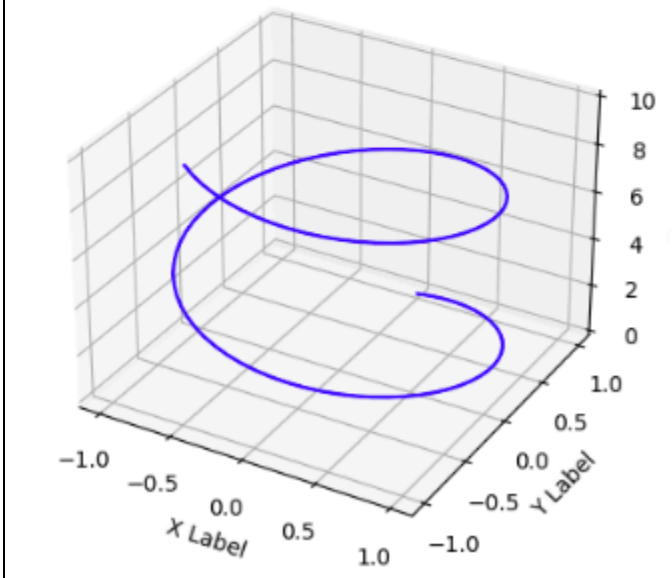
1. **3D Scatter Plot**

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

# Data for plotting
x = np.random.rand(100)
y = np.random.rand(100)
z = np.random.rand(100)

# Create a figure and 3D axis using 'projection=3d'
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Create a scatter plot
ax.scatter(x, y, z, c=z, cmap='viridis')

# Add labels
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

# Show the plot
plt.show()
```



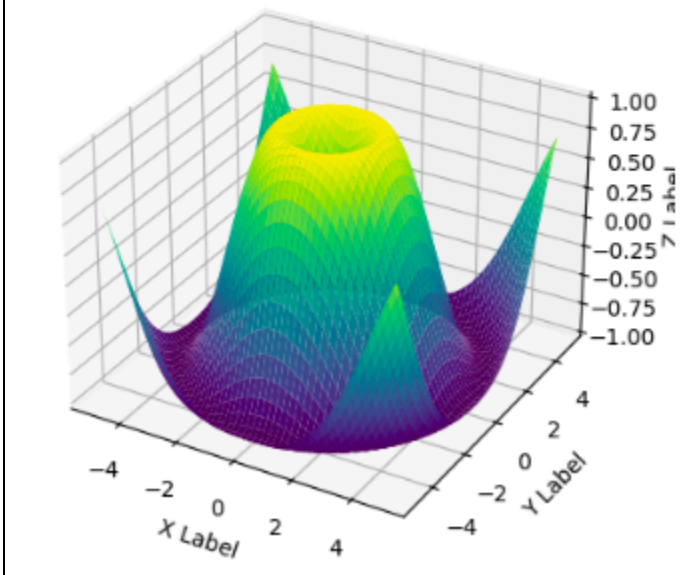
2. 3D Line Plot

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

# Data for plotting
t = np.linspace(0, 10, 100)
x = np.sin(t)
y = np.cos(t)
z = t

# Create a figure and 3D axis using 'projection=3d'
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Create a 3D line plot
ax.plot(x, y, z, label='3D Line', color='b')

# Add labels
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

# Show the plot
plt.show()
```



3. 3D Surface Plot

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

# Data for plotting
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))

# Create a figure and 3D axis using 'projection=3d'
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Create a 3D surface plot
ax.plot_surface(x, y, z, cmap='viridis')

# Add labels
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')

# Show the plot
plt.show()
```



Note for 3D Plots:

1. **Enabling 3D Plotting with `projection='3d'`**:
    a. By specifying `projection='3d'` when creating a subplot, you enable 3D plotting in Matplotlib. This allows you to create various types of 3D plots in the same figure.

2. **Single Axis Object for Multiple 3D Plots**:
    a. You can use the same axis (`ax`) object to create various types of 3D plots, such as **scatter plots**, **line plots**, **surface plots**, and **wireframe plots**. This makes it easy to manage and customize different types of 3D visualizations in a single figure.

3. **Interactive 3D Plots**:
    a. 3D plots can be interactive by default in environments like Jupyter Notebooks, allowing users to rotate, zoom, and pan the plot for better exploration of the data.