MATPLOTLIB

Introduction

What is Matplotlib?

Matplotlib is a popular plotting library in Python used for creating high-quality visualizations and graphs. It offers various tools to generate diverse plots, facilitating data analysis, exploration, and presentation. Matplotlib is flexible, supporting multiple plot types and customization options, making it valuable for scientific research, data analysis, and visual communication. It can create different types of visualization reports like line plots, scatter plots, histograms, bar charts, pie charts, box plots, and many more different plots. This library also supports 3-dimensional plotting.

Installation of Matplotlib

Let's check how to set up the Matplotlib in Google Colab. Colab Notebooks are similar to Jupyter Notebooks except for running on the cloud. It is also connected to our Google Drive, making it much easier to access our Colab notebooks anytime, anywhere, and on any system. You can install Matplotlib by using the PIP command.

```
!pip install matplotlib
```

Copy Code

```
!pip install matplotlib

Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (3.2.2)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (0.10.0)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.19.5)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib) (1.3.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from cycler>=0.10->matplotlib) (1.15.0)
```

Source: Local

To verify the installation, you would have to write the following code chunk:

```
import matplotlib
print(matplotlib.__version__)
```

Copy Code

```
#First import the library.
import matplotlib
#print the version of the mtploltib
print(matplotlib.__version__)

3.2.2
```

Source: Local

Pyplot

Introduction to pyplot

matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current Axes (please note that we use uppercase Axes to refer to the Axes concept, which is a central part of a figure and not only the plural of *axis*).

Generating visualizations with pyplot is very quick:

```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4])
plt.ylabel('some numbers')
plt.show()
```

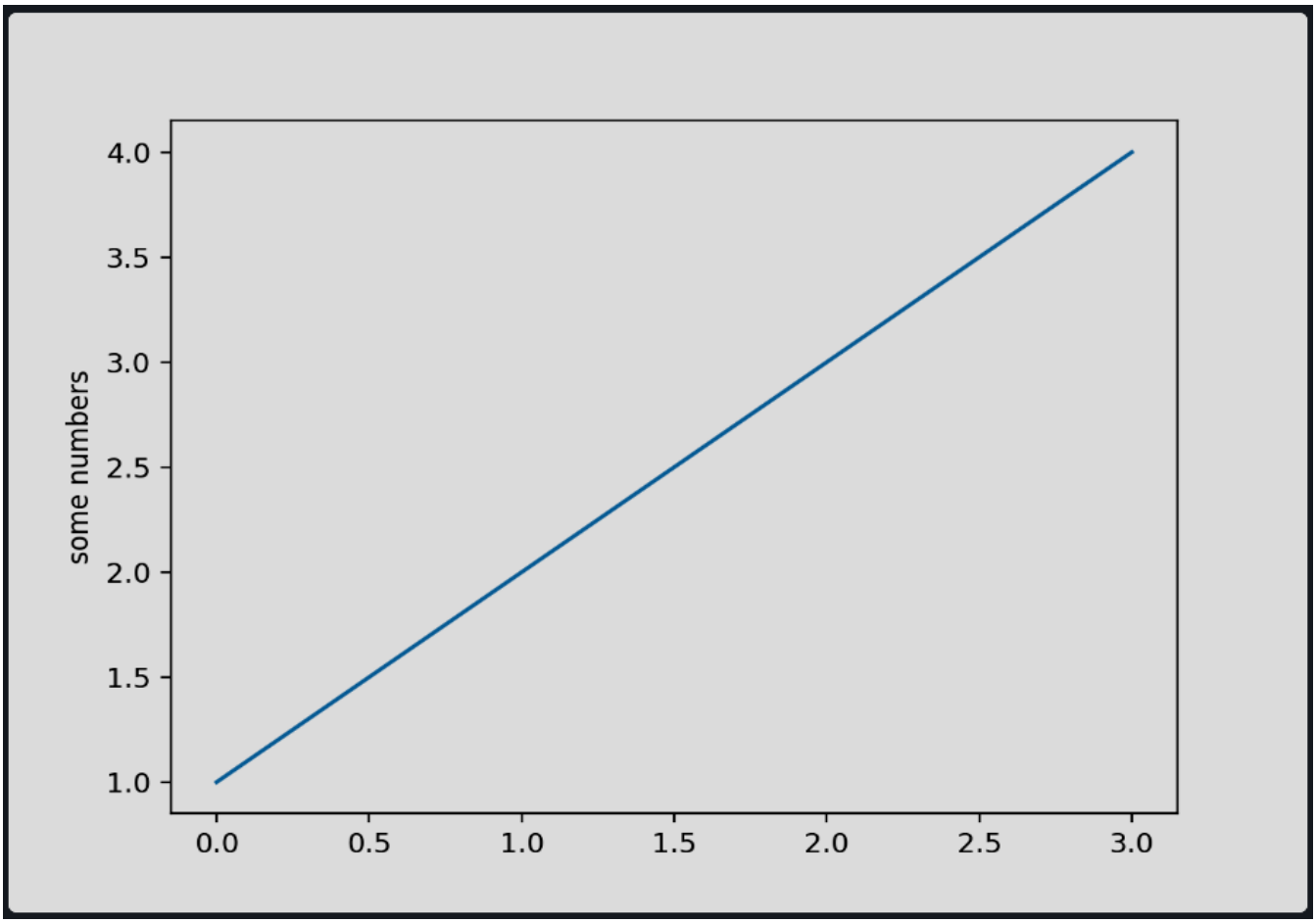


Figure Class

Introduction to Figure Class

The **Figure()** class in Matplotlib is a top-level artist that acts as the primary container for all plot elements. It holds everything together, including subplots, axes, titles, legends, and other artistic elements.

This class is available in the **matplotlib.figure** module with several customization options, in addition to the Figure() class, the module also contains classes related to creating and managing figures.

Creating a Figure

A Figure instance is typically created using pyplot methods such as figure, subplots, and subplot_mosaic. These methods return both a Figure instance and a set of Axes, providing a convenient way to create and work with visualizations.

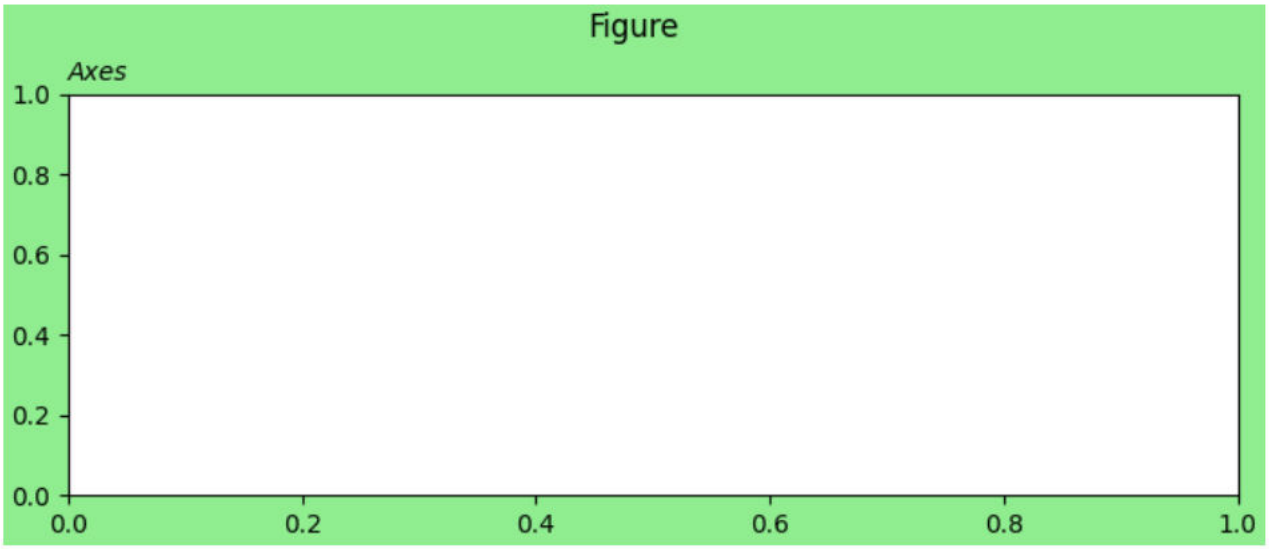
Example

Here is an example that uses the **pyplot.figure()** method to create a figure.

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

# Creating the Figure instance
fig = plt.figure(figsize=[7, 3], facecolor='lightgreen',
layout='constrained')

# Adding a title to the Figure
fig.suptitle('Figure')

# Adding a subplot (Axes) to the Figure
ax = fig.add_subplot()

# Setting a title for the subplot
ax.set_title('Axes', loc='left', fontstyle='oblique', fontsize='medium')

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

Output

On executing the above code we will get the following output –



Axes Class

Introduction to Axes Class

Axes is the most basic and flexible unit for creating sub-plots. Axes allow placement of plots at any location in the figure. A given figure can contain many axes, but a given axes object can only be in one figure. The axes contain two axis objects 2D as well as, three-axis objects in the case of 3D. Let's look at some basic functions of this class.

add_axes() function

Alternatively, you can also add the axes object to the figure by calling the **add_axes()** method. It returns the axes object and adds axes at position [left, bottom, width, height] where all quantities are in fractions of figure width and height.

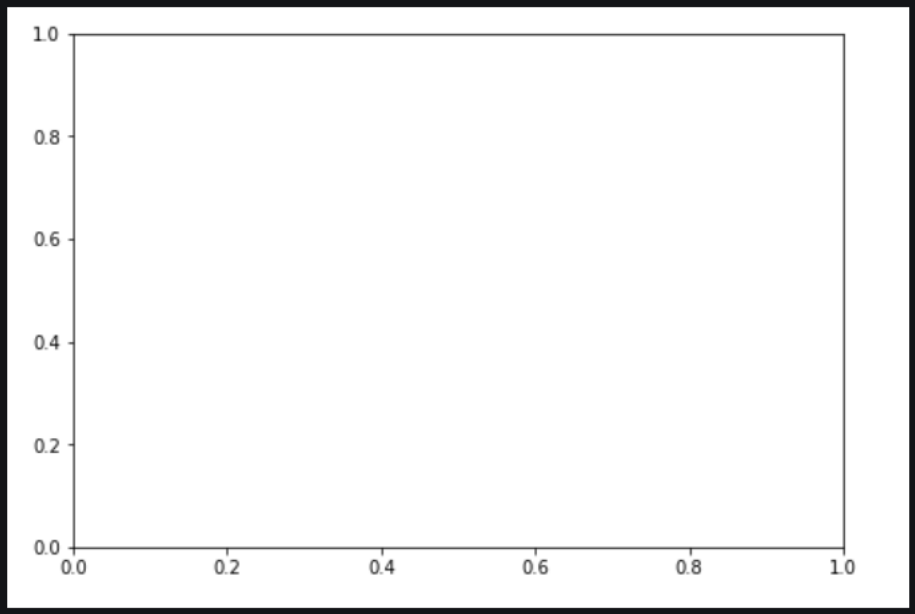
Syntax :

```
add_axes([left, bottom, width, height])
```

Example :

```
import matplotlib.pyplot as plt


fig = plt.figure()

#[left, bottom, width, height]
ax = fig.add_axes([0, 0, 1, 1])
```

Output:



Matplotlib – Setting Ticks and Tick Labels

Introduction to Setting Ticks and Tick Labels

Matplotlib has the ability to customize ticks and tick labels on axes, which enhances the readability and interpretability of graphs. This article will explore setting ticks and tick labels, providing a clear example to illustrate the core concepts.

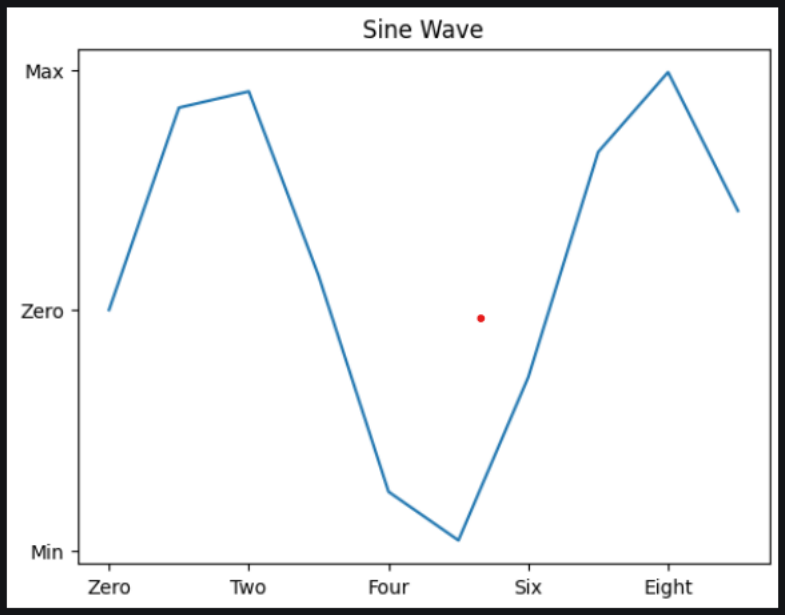
Setting Ticks and tick labels – using set_xticks() and set_yticks()

In Matplotlib, ticks are markers on the axes of a plot that denote specific data points. Tick labels are the textual representations associated with each tick. Customizing both ticks and tick labels can make your plots more intuitive and easier to interpret.\

Inputs

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

x = np.arange(0, 10, 1)
y = np.sin(x)

plt.plot(x, y)
plt.xticks([0, 2, 4, 6, 8], ['Zero', 'Two', 'Four', 'Six', 'Eight'])
plt.yticks([-1, 0, 1], ['Min', 'Zero', 'Max'])
plt.title('Sine Wave')
plt.show()
```

Output:



Setting Ticks and tick labels in matplotlib

Understanding Ticks and Tick Labels

What are Ticks?

Ticks are markers on the axes of a plot that denote specific data points. By default, Matplotlib automatically generates these ticks based on the data being plotted. However, in

many cases, users may want to customize these ticks to better reflect the data or improve clarity.

What are Tick Labels?

Tick labels are the textual representations associated with each tick. They provide context for what each tick represents in terms of data values. Customizing tick labels can make a plot more informative and easier to understand.

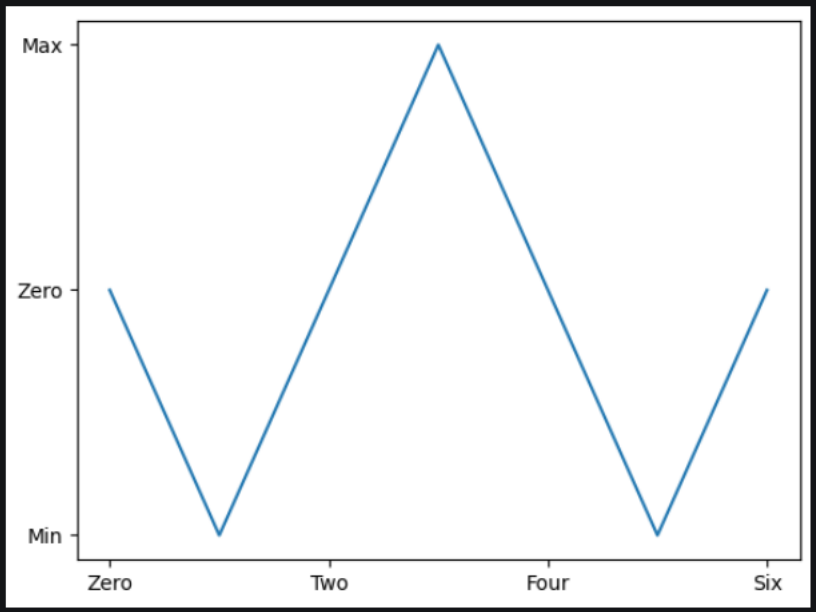
Methods for Setting Ticks and Tick Labels

1. set_xticklabels() and set_yticklabels()

set_xticks() and set_yticks() set the locations of the ticks on the x-axis and y-axis, respectively. Once the tick positions are set, you can use set_xticklabels() and set_yticklabels() to assign custom labels to the ticks.

```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4, 5, 6]
y = [0, -1, 0, 1, 0, -1, 0]

fig, ax = plt.subplots()
ax.plot(x, y)

ax.set_xticks([0, 2, 4, 6])
ax.set_yticks([-1, 0, 1])

ax.set_xticklabels(['Zero', 'Two', 'Four', 'Six'])
ax.set_yticklabels(['Min', 'Zero', 'Max'])

plt.show()
```

Output:



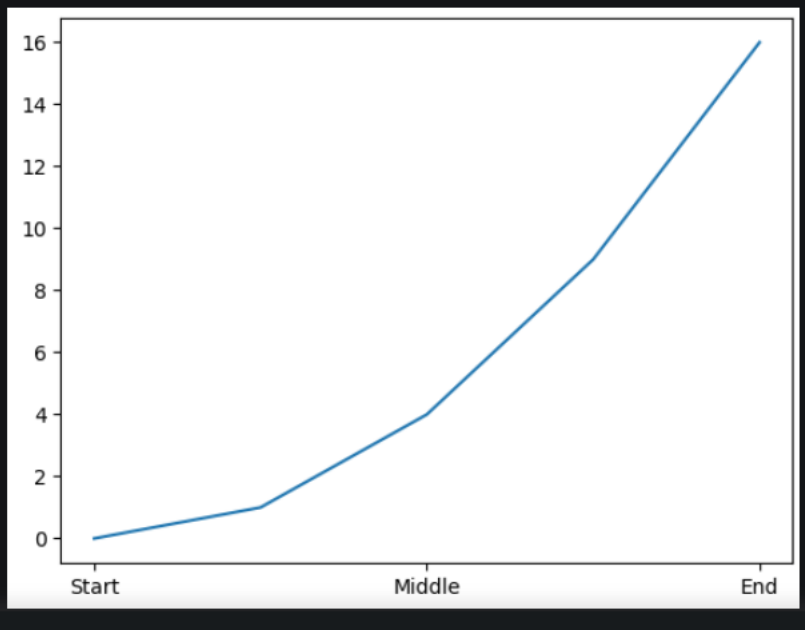
The plot will display customized tick labels for both the x and y axes.

2. xticks() and yticks()

xticks() and yticks() functions combine the functionality of set_xticks() and set_yticks() into one call, allowing you to set both tick positions and labels at the same time.

```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]
y = [0, 1, 4, 9, 16]

plt.plot(x, y)
plt.xticks([0, 2, 4], ['Start', 'Middle', 'End'])
plt.show()
```

Output:



Multiple Plots

Introduction to Multiple Plots

In Matplotlib, we can draw multiple graphs in a single plot in two ways. One is by using subplot() function and other by superimposition of second graph on the first i.e, all graphs will appear on the same plot. We will look into both the ways one by one.

In Matplotlib, there is another function very similar to subplot which is subplot2grid (). It is same almost same as subplot function but provides more flexibility to arrange the plot objects according to the need of the programmer.

This function is written as follows:

Syntax: *matplotlib.pyplot.subplot2grid(shape, loc, rowspan=1, colspan=1, fig=None, **kwargs)*

Parameter:

1. ***shape***
   This parameter is a sequence of two integer values which tells the shape of the grid for which we need to place the axes.The first entry is for row, whereas the second entry is for column.

2. ***loc***
   Like shape parameter, even loc is a sequence of 2 integer values, where first entry remains for the row and the second is for column to place axis within grid.

3. ***rowspan***
   This parameter takes integer value and the number which indicates the number of rows for the axis to span to or increase towards right side.

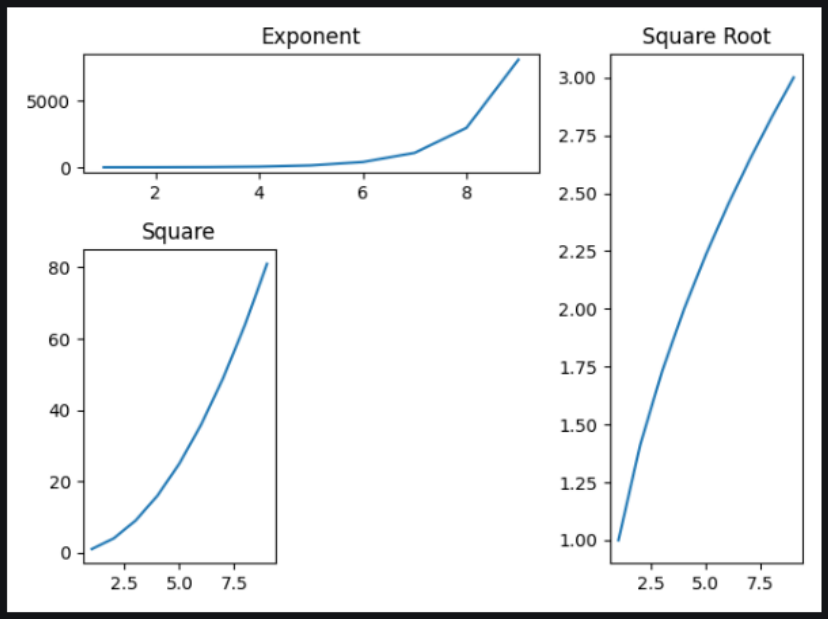
4. ***colspan***
   This parameter takes integer value and the number which indicates the number of columns for the axis to span to or increase the length downwards.

5. ***fig***
   This is an optional parameter and takes Figure to place axis in. It defaults to current figure.

6. *****kwargs***
   This allows us to pass any other additional keyword argument to the function call and has a default value of None.

```
# Importing libraries
import matplotlib.pyplot as plt
import numpy as np
import math

# Placing the plots in the plane
plot1 = plt.subplot2grid((3, 3), (0, 0), colspan=2)
plot2 = plt.subplot2grid((3, 3), (0, 2), rowspan=3, colspan=2)
plot3 = plt.subplot2grid((3, 3), (1, 0), rowspan=2)

# Using Numpy to create an array x
x = np.arange(1, 10)

# Plot for square root
plot2.plot(x, x**0.5)
plot2.set_title('Square Root')

# Plot for exponent
plot1.plot(x, np.exp(x))
plot1.set_title('Exponent')

# Plot for Square
plot3.plot(x, x*x)
plot.set_title('Square')

# Packing all the plots and displaying them
plt.tight_layout()
plt.show()
```

Output



Multiple Plots using subplot2grid() function

Legend

Introduction to Legend

A legend is an area describing the elements of the graph. In the Matplotlib library, there's a function called **legend()** which is used to place a legend on the axes. In this article, we will learn about the Matplotlib Legends.

Python Matplotlib.pyplot.legend() Syntax

***Syntax**: matplotlib.pyplot.legend(["blue","green"], bbox_to_anchor=(0.75, 1.15), ncol=2)*

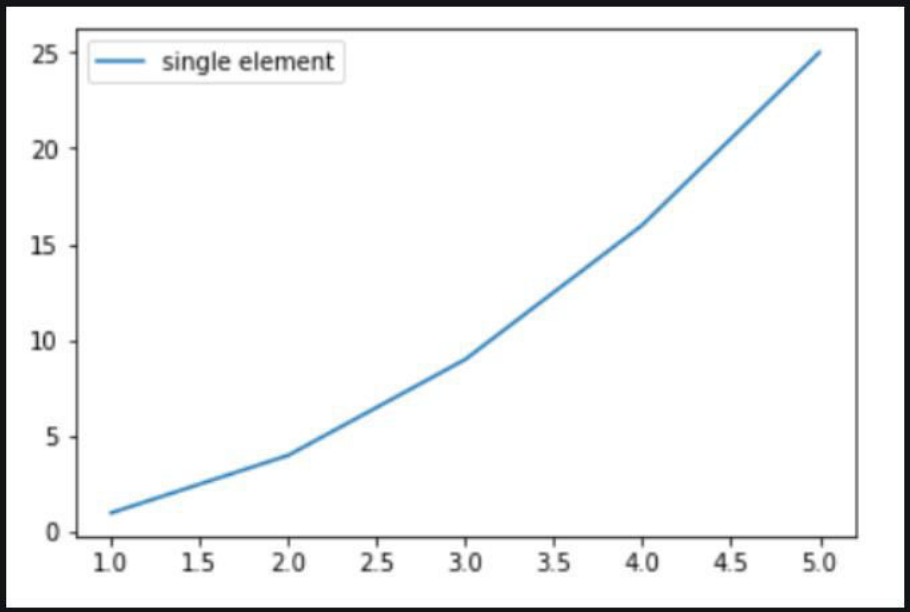
Attributes:

- ***shadow**: [None or bool] Whether to draw a shadow behind the legend.It's Default value is None.*
- ***markerscale**: [None or int or float] The relative size of legend markers compared with the originally drawn ones.The Default is None.*
- ***numpoints**: [None or int] The number of marker points in the legend when creating a legend entry for a Line2D (line).The Default is None.*
- ***fontsize**:The font size of the legend.If the value is numeric the size will be the absolute font size in points.*
- ***facecolor**: [None or "inherit" or color] The legend's background color.*

- ***edgecolor**: [None or "inherit" or color] The legend's background patch edge color.*

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

# X-axis values
x = [1, 2, 3, 4, 5]

# Y-axis values
y = [1, 4, 9, 16, 25]

# Function to plot
plt.plot(x, y)

# Function add a legend
plt.legend(['single element'])

# function to show the plot
plt.show()
```

Output :



Types of Plots in Matplotlib

Now that you know what Matplotlib function is and how you can install it in your system, let's discuss different kinds of plots that you can draw to analyze your data or present your findings. Also, you can go to the article to master in matplotlib

- Line Graph
- Bar Chart
- Histograms,
- Scatter Plot
- Pie Chart
- 3D Plots
- Box Plots

Line Graph

A line plot shows the relationship between the x and y-axis.

The plot() function in the **Matplotlib** library's Pyplot module creates a 2D hexagonal plot of x and y coordinates. plot() will take various arguments like **plot(x, y, scalex, scaley, data, **kwargs)**.

x, y are the horizontal and vertical axis coordinates where x values are optional, and its default value is range(len(y)).

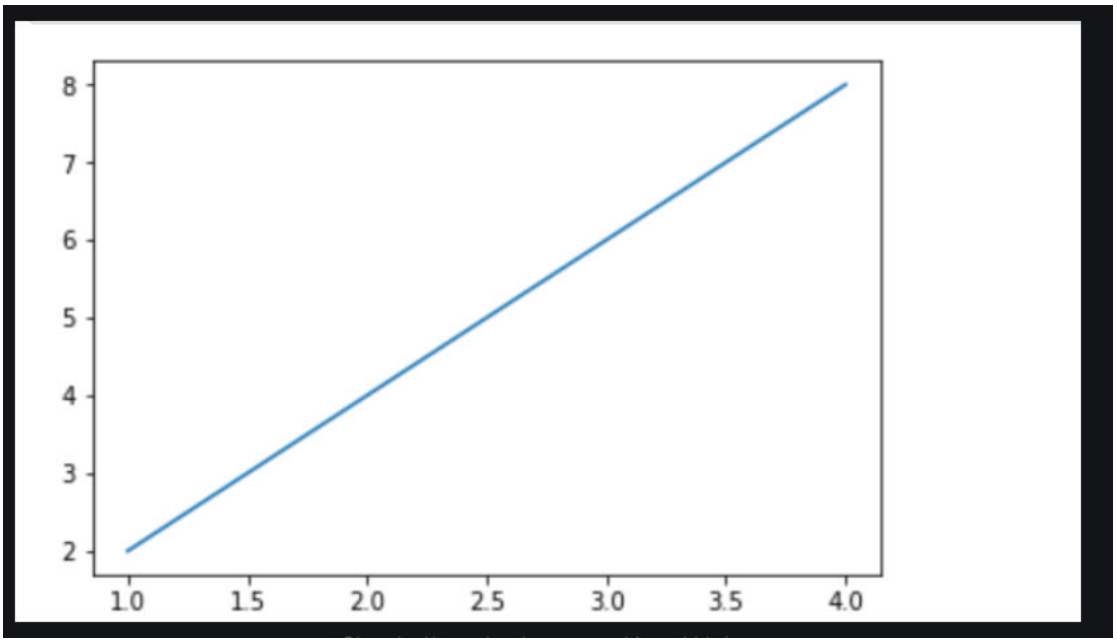
scalex, scaley parameters are used to autoscale the x-axis or y-axis, and its default value is actual.

****kwargs** specify the property like line label, linewidth, marker, color, etc.

Code:

```
# importing the required libraries
import matplotlib.pyplot as plt
import numpy as np

# define data values
x = np.array([1, 2, 3, 4])  # X-axis points
y = x*2  # Y-axis points

plt.plot(x, y)  # Plot the chart
plt.show()  # display
```

Output:



Bar Chart

Mainly, the barplot shows the relationship between the numeric and categoric values. In a bar chart, we have one axis representing a particular category of the columns and another axis representing the values or counts of the specific category. Barcharts can be plotted both vertically and horizontally using the following line of code:

plt.bar(x,height,width,bottom,align)

x: representing the coordinates of the x-axis

height: the height of the bars

, width: width of the bars. Its default value is 0.8

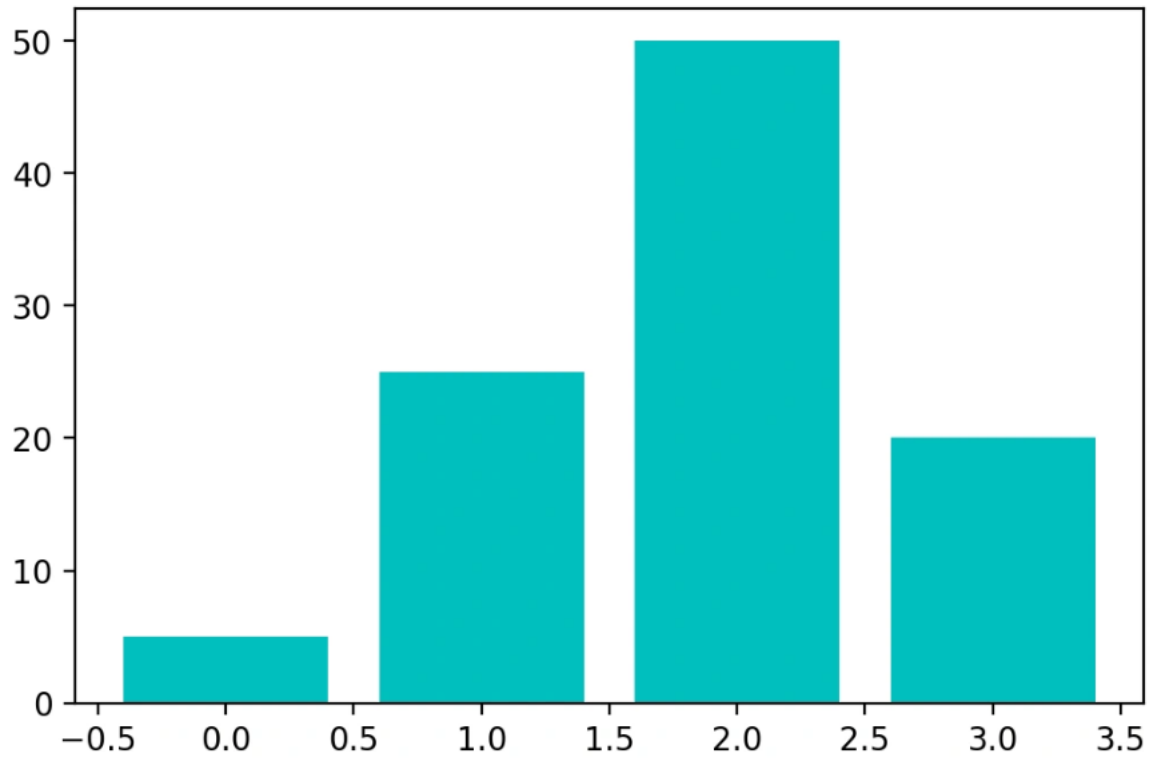
bottom: It's optional. It is a y-coordinate of the bar. Its default value is None

align: center, edge its default value is center

Code:

```
#define array
data= [5. , 25. , 50. , 20.]
plt.bar(range(len(data)), data,color='c')
plt.show()
```

Copy Code



Source: Local

Histograms

The most common graph for displaying frequency distributions is a histogram. To create a histogram, the first step is to create a bin of ranges, then distribute the whole range of values into a series of intervals and count the value that will fall in the given interval. We can use plt.Hist () function plots the histograms, taking various arguments like data, bins, color, etc.

x: x-coordinate or sequence of the array

bins: integer value for the number of bins wanted in the graph

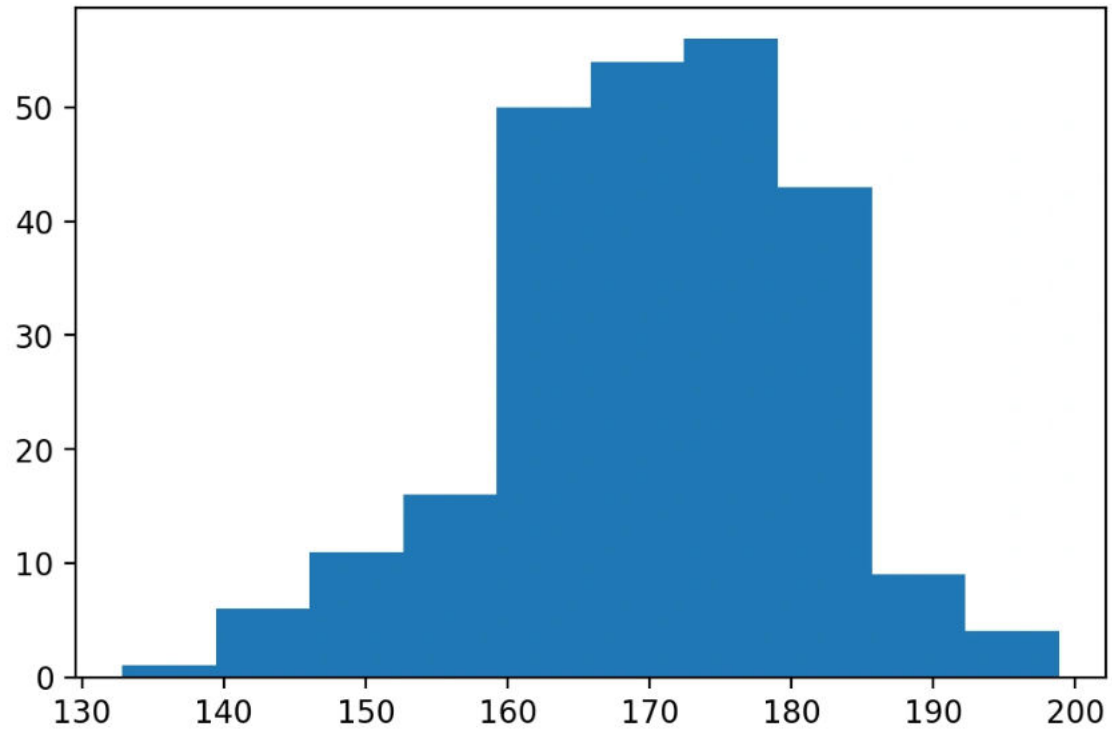
range: the lower and upper range of bins

density: optional parameter that contains boolean values

histtype: optional parameter used to create different types of histograms like:-bar bar stacked, step, step filled, and the default is a bar

Code:

```
#draw random samples from random distributions.
x = np.random.normal(170, 10, 250)
#plot histograms
plt.hist(x) plt.show()
```



Source: Local

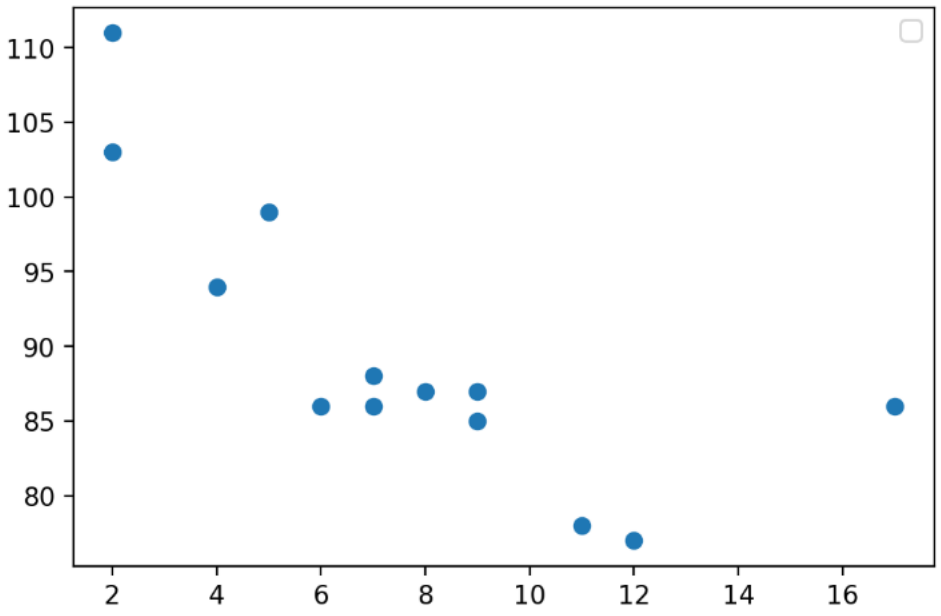
Scatter Plot

Scatter plots show the relationships between variables by using dots to represent the connection between two numeric variables.

The **scatter()** method in the **Matplotlib** library is used for plotting.

Code:

```
#create the x and y axis coordinates
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)
plt.legend()
plt.show()
```



Source: Local

Pie Chart

A pie chart (or circular chart) is used to show the percentage of the whole. Hence, it is used to compare the individual categories with the whole. **Pie()** will take the different parameters such as:

x: Sequence of an array

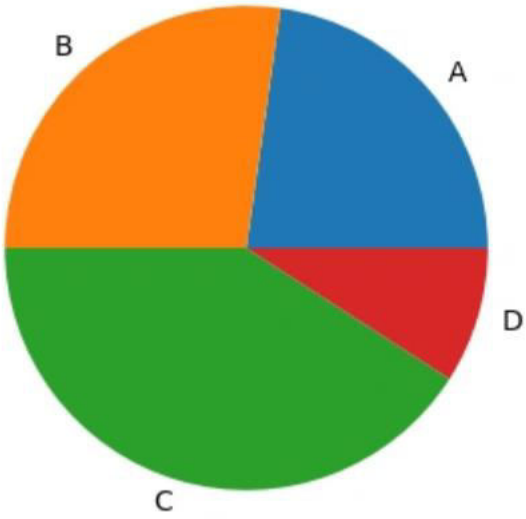
labels: List of strings which will be the name of each slice in the pie chart

Autopct: It is used to label the wedges with numeric values. The labels will be placed inside the wedges. Its format is %1.2f%

Code:

```
#define the figure size
plt.figure(figsize=(7,7))
x = [25,30,45,10]
#labels of the pie chart
labels = ['A','B','C','D']
plt.pie(x, labels=labels)
plt.show()
```

Copy Code



Source: Local

Box Plots

A Box plot in Python Matplotlib showcases the dataset's summary, encompassing all numeric values. It highlights the minimum, first quartile, median, third quartile, and maximum. The median lies between the first and third quartiles. On the x-axis, you'll find the data values, while the y-coordinates represent the frequency distribution.

Parameters used in box plots are as follows:

data: NumPy array

vert: It will take boolean values, i.e., true or false, for the vertical and horizontal plot. The default is True

width: This will take an array and sets of the width of boxes. Optional parameters

Patch_artist: It is used to fill the boxes with color, and its default value is false

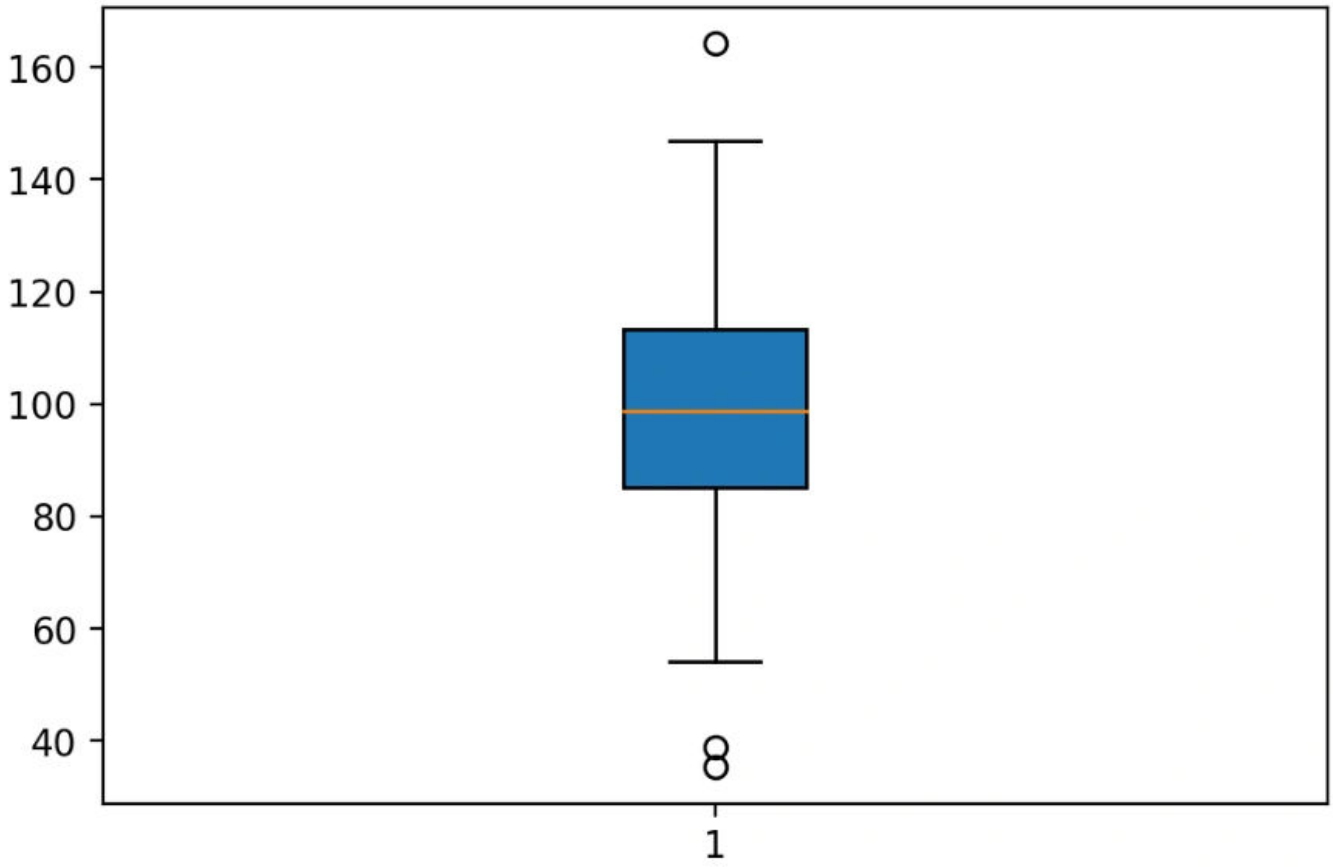
labels: Array of strings which is used to set the labels of the dataset

Code:

```
#create the random values by using numpy
values= np.random.normal(100, 20, 300)
#creating the plot by boxplot() function which is avilable in matplotlib
plt.boxplot(values,patch_artist=True,vert=True)
plt.show()
```

Copy Code

Output:



Source: Local

3D Plots

Now that you have seen some simple graphs, it's time to check some complex ones, i.e., 3-D graphs. Initially, Matplotlib was built for 2-dimensional graphs, but later, 3-D graphs were added. Let's check how you can plot a 3-D graph in Matplotlib.

Code:

```
from mpl_toolkits import mplot3d
```

Copy Code

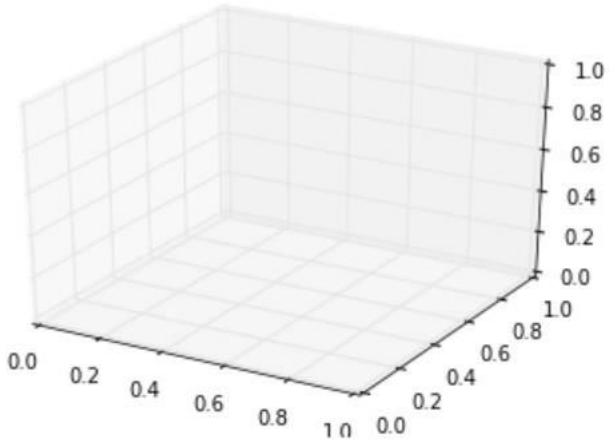
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

Copy Code

```
fig = plt.figure()
ax = plt.axes(projection='3d')
```

Copy Code

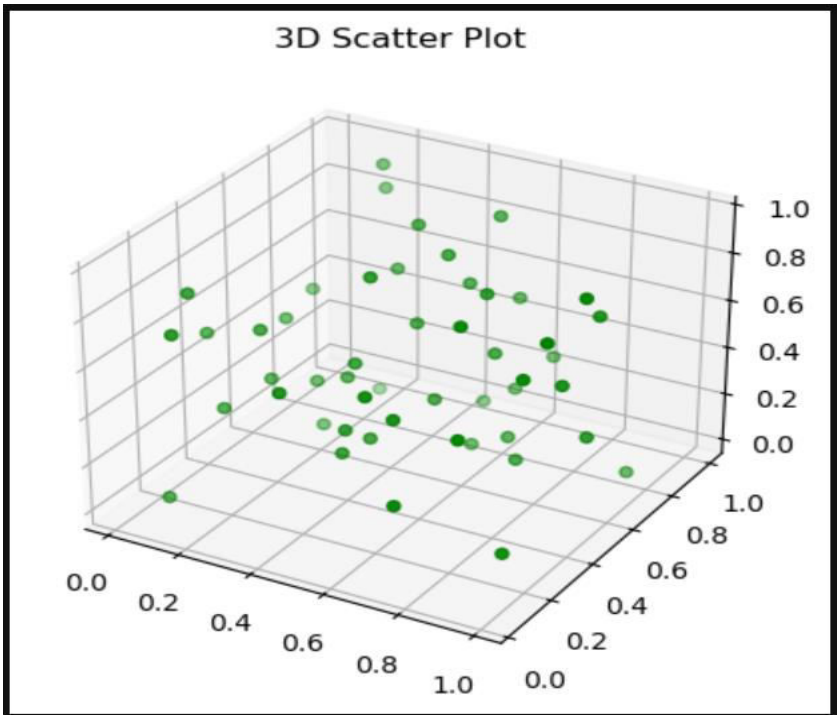
The above code is used to create the 3-dimensional axes.



Source: Local

Each plot that we have seen in 2-D plotting through Matplotlib can also be drawn as 3-D graphs. For instance, let's check a line plot in a 3-D plane.

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x = np.random.rand(50)
y = np.random.rand(50)
z = np.random.rand(50)
ax.scatter(x, y, z, color='green')
plt.title('3D Scatter Plot')
plt.show()
```



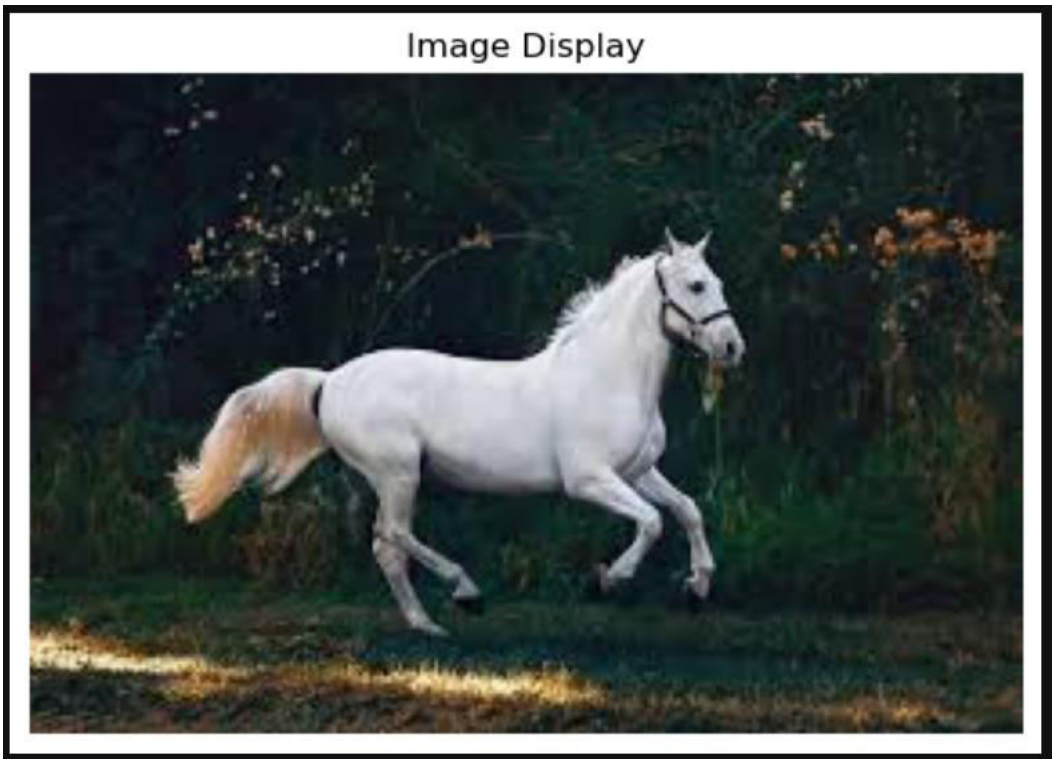
Working with Images

Matplotlib allows displaying images using imshow(), which can be used for image processing.

Code:

```
import matplotlib.image as mpimg
img = mpimg.imread(r"C:\Users\chitt\OneDrive\Pictures\image\horse image.jpg")
plt.imshow(img)
plt.title('Image Display')
plt.axis('off')  # Hide axes
plt.show()
```

Output:



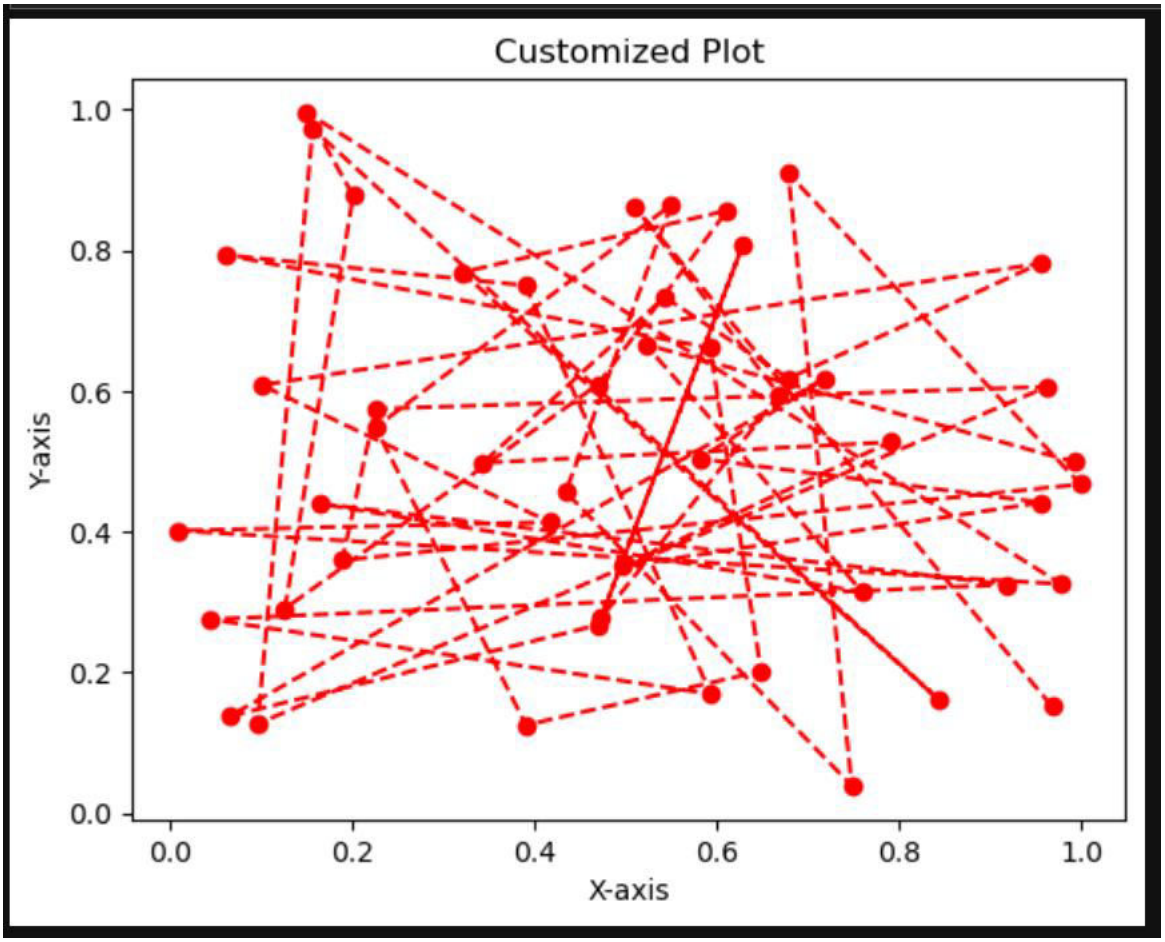
Customizing Plots

matplotlib uses matplotlibrc configuration files to customize all kinds of properties, which we call rc settings or rcparameters .You can control the defaults of almost every property in matplotlib: figure size and dpi, line width, color and style, axes, axis and grid properties, text and font properties and so on.

Code:

```
plt.plot(x, y, color='r', linestyle='--', marker='o')
plt.title('Customized Plot')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

Output:



Conclusion

In this article, we discussed Matplotlib, i.e., the basic plotting library in Python, and the basic information about the charts commonly used for statistical analysis. Also, we have discussed how to draw multiple plots in one figure using the subplot function.

As we explore Python Matplotlib, we've covered customizing figures and resizing plots with various arguments. Now, equipped with fundamental plotting skills, feel free to experiment with diverse datasets and mathematical functions.

Interview Question

Basic Questions

1. **What is Matplotlib?**
   Answer: Matplotlib is a Python library used for creating static, interactive, and animated visualizations. It provides an object-oriented API for embedding plots into applications.

2. **What is pyplot in Matplotlib?**
   Answer: pyplot is a module in Matplotlib that provides a MATLAB-like interface for plotting. It simplifies creating figures, axes, labels, and legends.
3. **How do you install Matplotlib?**
   Answer: You can install Matplotlib using pip:

```
pip install matplotlib
```

4. **How do you create a basic line plot in Matplotlib?**
   Answer:

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.show()
```

5. **What is the difference between Figure and Axes in Matplotlib?**
   Answer:
   - Figure: The entire plotting area or canvas.
   - Axes: The actual plotting area inside the Figure where data is drawn.

Customization & Features

6. **How do you set labels and titles for a plot?**
   Answer:

```
plt.xlabel("X-axis Label")
plt.ylabel("Y-axis Label")
plt.title("Title of the Plot")
```

7. **How do you add a legend in Matplotlib?**
   Answer:

```
plt.plot([1,2,3], [4,5,6], label="Line 1")
plt.legend()
```

8. **How do you change line style and color in a plot?**
   Answer:

```
plt.plot(x, y, linestyle='dashed', color='red', marker='o')
```

9. **How do you create multiple subplots in Matplotlib?**
   Answer:

```
fig, axes = plt.subplots(2, 2)  # 2x2 grid of subplots
axes[0,0].plot([1,2,3], [4,5,6])
plt.show()
```

10. **What function is used to display an image in Matplotlib?**
    Answer:

```
import matplotlib.image as mpimg
img = mpimg.imread('image.jpg')
plt.imshow(img)
plt.show()
```

Advanced Questions

11. **What are some commonly used markers in Matplotlib?**
    Answer: Some markers are:
    - '.' (dot)
    - 'o' (circle)
    - 's' (square)
    - '^' (triangle)

12. **How do you create a bar chart in Matplotlib?**
    Answer:

```
plt.bar(['A', 'B', 'C'], [10, 20, 15])
plt.show()
```

13. **How do you create a histogram in Matplotlib?**
    Answer:

```
import numpy as np
data = np.random.randn(1000)
plt.hist(data, bins=30)
plt.show()
```

14. How do you create a scatter plot in Matplotlib?

Answer:

```
x = np.random.rand(50)
y = np.random.rand(50)
plt.scatter(x, y)
plt.show()
```

15. How do you create a pie chart in Matplotlib?

Answer:

```
labels = ['A', 'B', 'C']
sizes = [30, 20, 50]
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.show()
```

3D & Miscellaneous

16. How do you create a 3D plot in Matplotlib?

Answer:

```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter([1,2,3], [4,5,6], [7,8,9])
plt.show()
```

17. How do you adjust figure size in Matplotlib?

Answer:

```
plt.figure(figsize=(10,5))
```

18. How do you save a plot as an image file?

Answer:

```
plt.savefig("plot.png")
```

19. What is the difference between show() and savefig()?

Answer:

- show(): Displays the plot interactively.
- savefig(): Saves the plot as an image file.

20. How do you add grid lines to a plot?

Answer:

```
plt.grid(True)
```

THANKYOU