Matplotlib

Matplotlib is one of the most popular data visualization libraries in Python. It provides a wide range of tools to create high-quality charts and graphs, making it an essential tool for data scientists and analysts. One of the key components of Matplotlib is pyplot, which provides a simple interface for creating plots. In this guide, we will explore the features of Matplotlib pyplot in detail and provide step-by-step instructions for creating different types of plots.

What are plots (graphics)

Visualization is the perfect form of representing data so that developers can understand what the data wants to express. It also gives a clear insight into the data demonstrating approaches. But not all data requires the same format of representation. That is where Matplotlib comes with different ways of generating visuals against data. This tutorial will explain the different types of two-dimensional plotting systems that Matplotlib pyplot can render

commonly used Plots come under Matplotlib. These are:

- Line Plot
- Bar Plot
- Scatter Plot
- Pie Plot
- Area Plotf
- Histogram Plot

Basic Example of plotting Graph

in Matplotlib, a "plot" is a graphical representation of data. It is a key feature used for visualizing data in a variety of forms such as line charts, bar charts, scatter plots, histograms, and more. Here are some of the most common types of plots in Matplotlib:

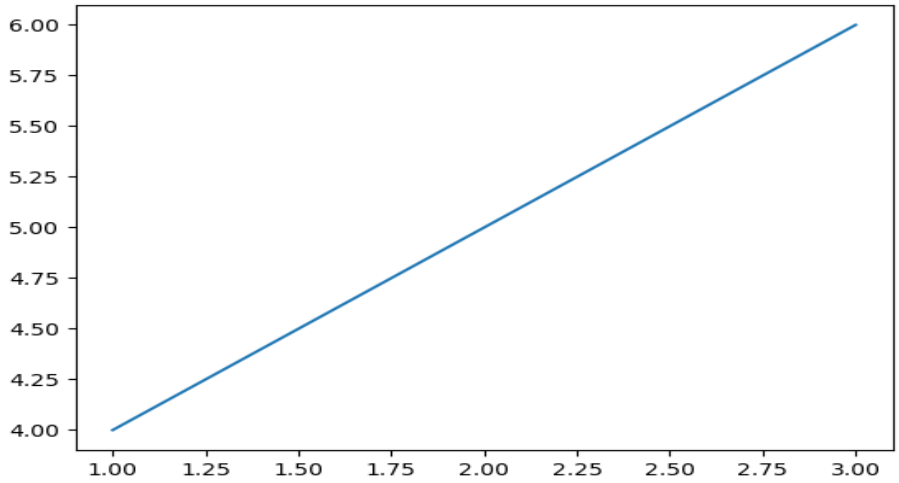
The basic example of generating a simple graph; the program

Line Plot: The simplest and most common type of plot. It displays data points connected by straight lines.

Example:1

```
from matplotlib import pyplot as plt
x=[1,2,3]
y=[4,5,6]
plt.plot(x,y)
plt.show()
```

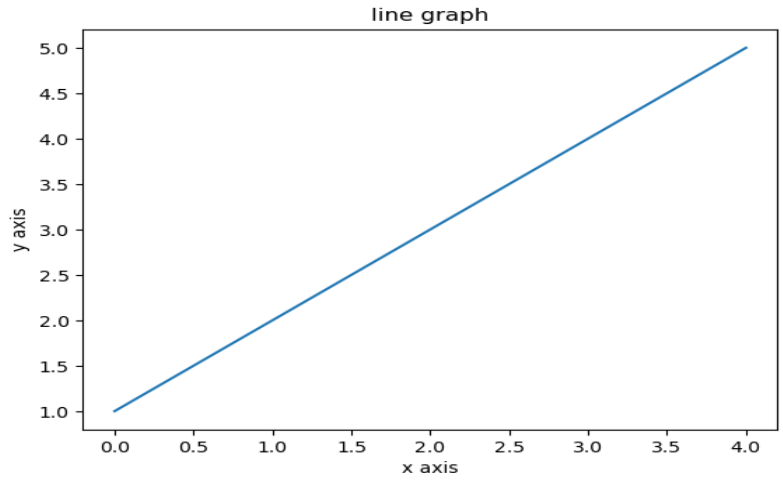
output:



Example:2

```
from matplotlib import pyplot as plt
x=[1,2,3,4,5]
# y=[4,5,6]
plt.xlabel("x axis")
plt.ylabel("y axis")
plt.title("line graph")
plt.plot(x)
plt.show()
```

output:



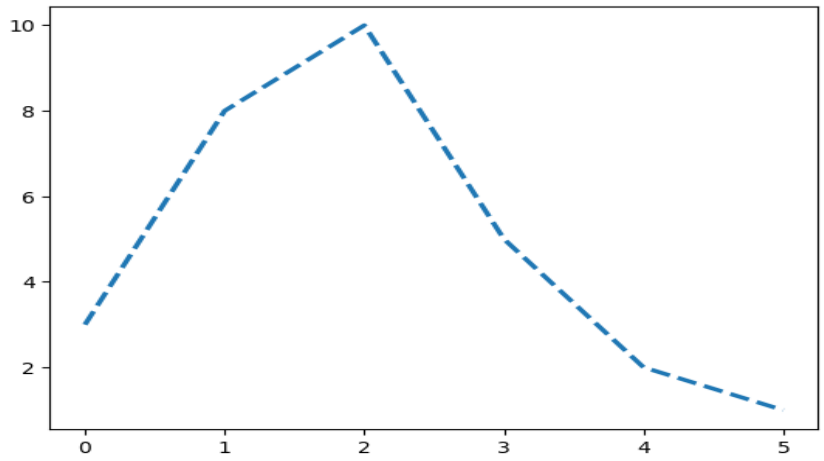
in the above program, it plots the graph x-axis ranges from 0-4 and the y-axis from 1-5. If we provide a single list to the plot(), matplotlib assumes it is a sequence of y values, and automatically generates the x values. Since we know that python index starts at 0, the default x vector has the same length as y but starts at 0. Hence the x data are [0, 1, 2, 3, 4].

Linestyle:graph

Ex:1

```
from matplotlib import pyplot as plt

xpoints=[3,8,10,5,2,1]

plt.plot(xpoints,linestyle="dashed",linewidth="2.5")

plt.show()
```

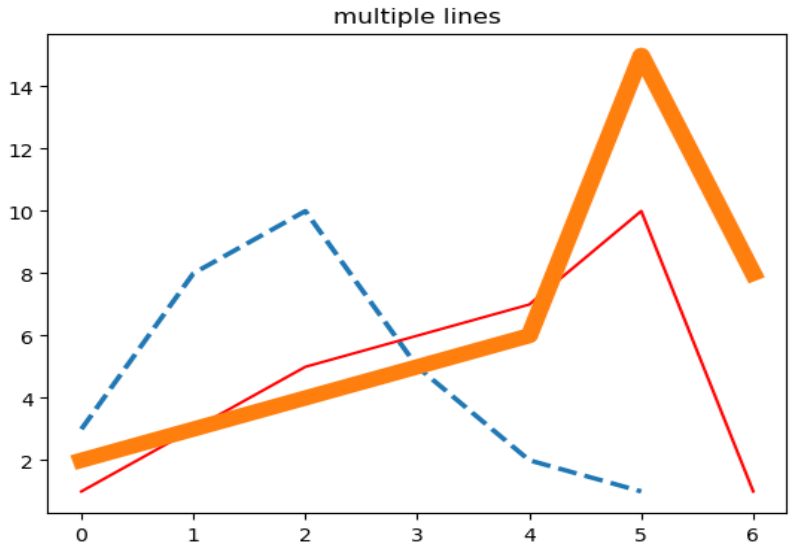
output:



Ex:2

```
from matplotlib import pyplot as plt

x=[3,8,10,5,2,1]
y=[1,3,5,6,7,10,1]
z=[2,3,4,5,6,15,8]
plt.plot(xpoints,linestyle="dashed",linewidth="2.5")

plt.plot(ypoints,color="red")
plt.plot(zpoints,linewidth="8.5")
plt.title("multiple lines")
plt.xlabel=("xaxis")

plt.show()
```

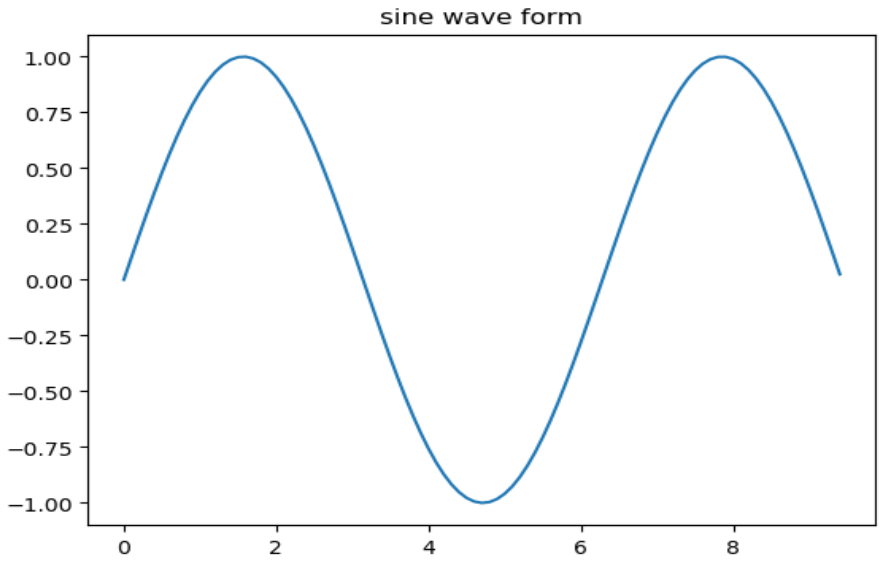
output:



Sine Wave Plot

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

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.title("sine wave form")

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```

output:



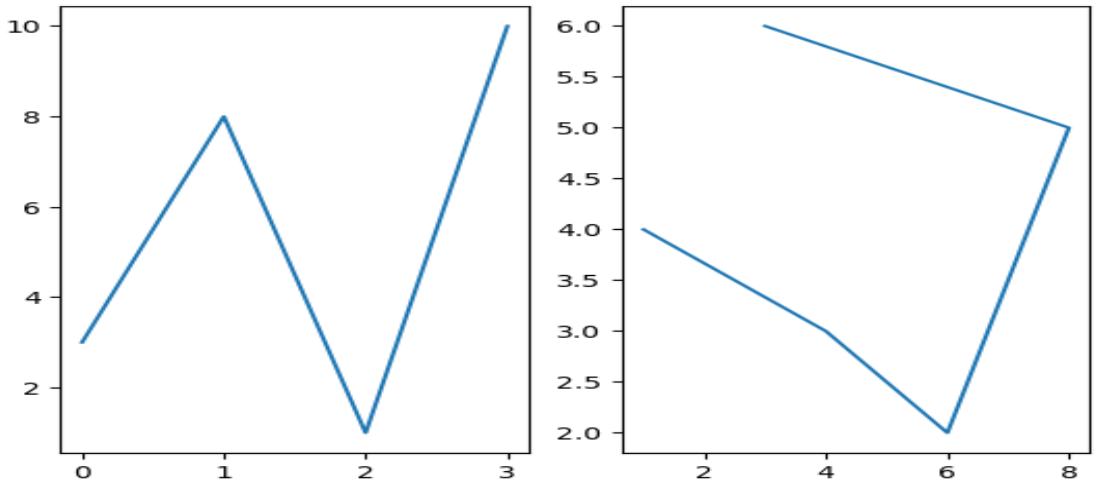
Subplots

Ex:1

```
from matplotlib import pyplot as plt

x=[0,1,2,3]

y=[3,8,1,10]


plt.subplot(1,2,1)

plt.plot(x,y)


x=[1,4,6,8,3]

y=[4,3,2,5,6]


plt.subplot(1,2,2)

plt.plot(x,y)
```

plt.show()

output:



Ex:2

import numpy as np

import matplotlib.pyplot as plt

Compute the x and y coordinates for points on sine and cosine curves

x = np.arange(0, 3 * np.pi, 0.1)

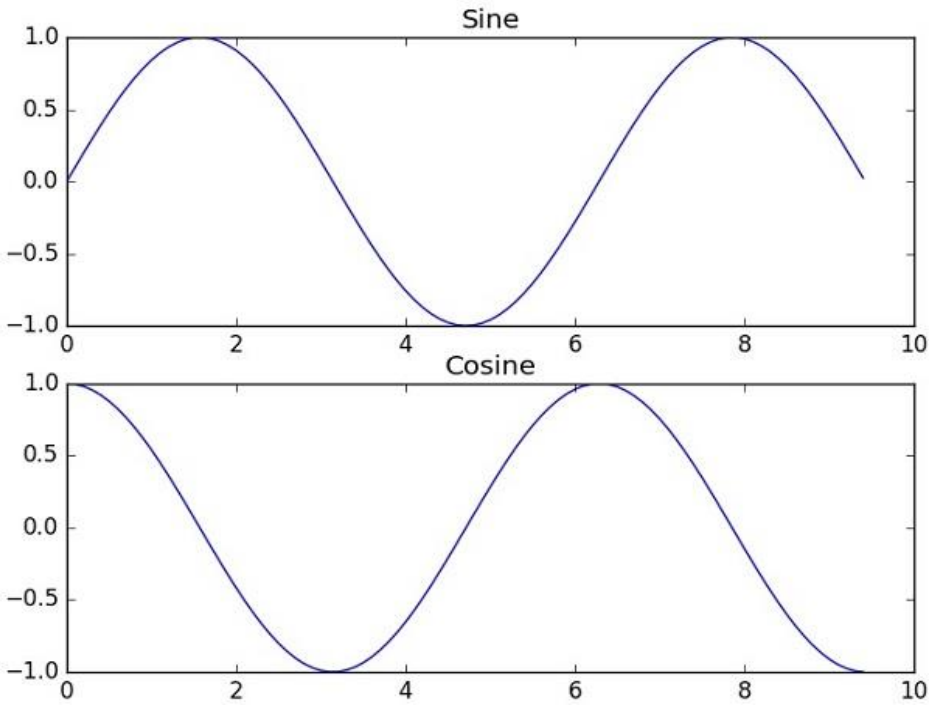
y_sin = np.sin(x)

y_cos = np.cos(x)

Set up a subplot grid that has height 2 and width 1,

```
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

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

output:

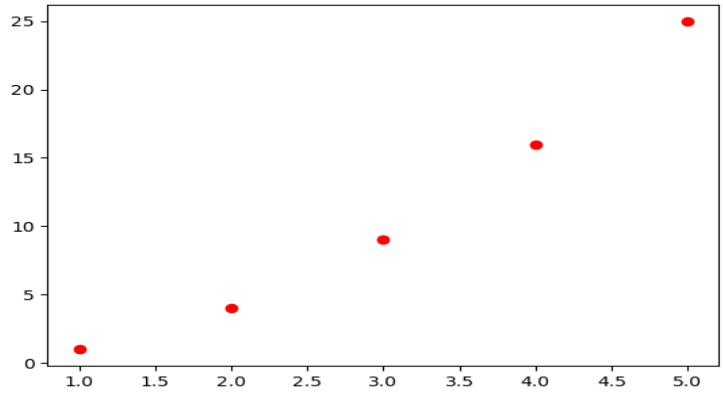


Scatter plot

The scatter plots are mostly used for comparing variables when we need to define how much one variable is affected by another variable. The data is displayed as a collection of points. Each point has the value of one variable, which defines the position on the horizontal axes, and the value of other variable represents the position on the vertical axis.

```
from matplotlib import pyplot as plt

x=[1,2,3,4,5]

y=[1,4,9,16,25]

plt.scatter(x,y,color=”red”)

plt.show()
```

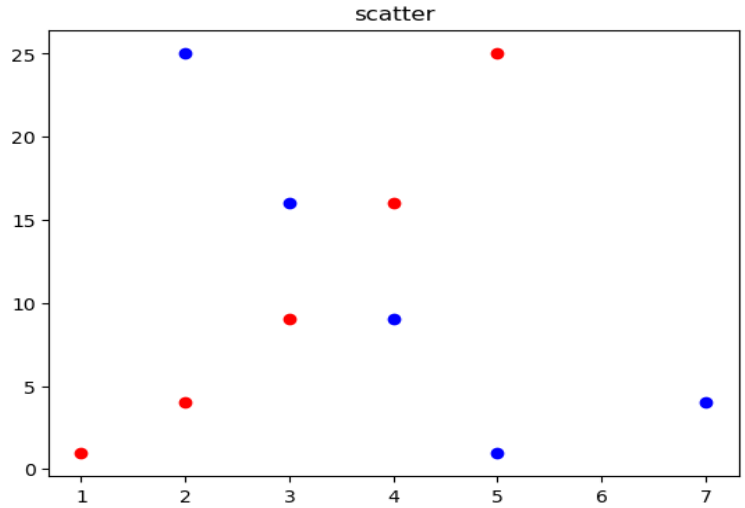
output:



Ex:2

```
from matplotlib import pyplot as plt
x=[1,2,3,4,5]
y=[1,4,9,16,25]
plt.scatter(x,y,color="red")
plt.title("scatter")

x=[5,7,4,3,2]
y=[1,4,9,16,25]
plt.scatter(x,y,color="blue")
plt.show()
```

output:



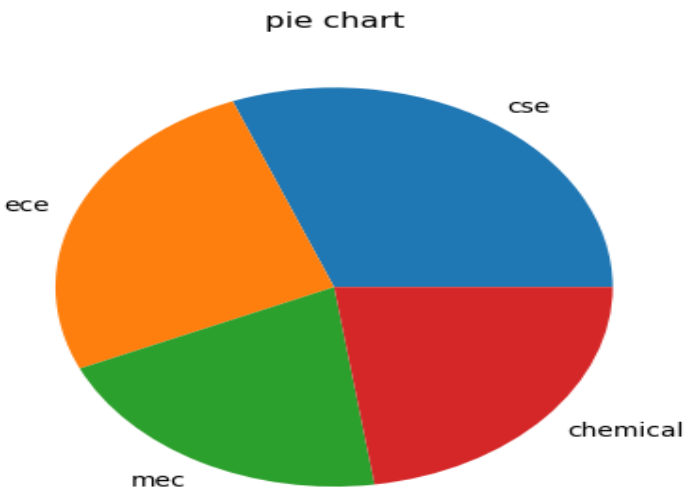
Pie chart

A pie chart is a circular graph that is broken down in the segment or slices of pie. It is generally used to represent the percentage or proportional data where each slice of pie represents a particular category. Let's have a look at the below example:

Ex:1

```
from matplotlib import pyplot as plt


rgukt_branches=["cse","ece","mec","chemical"]
students=[300,250,200,220]
plt.title("pie chart")


plt.pie(students,labels=rgukt_branches)


plt.show()
```

output:

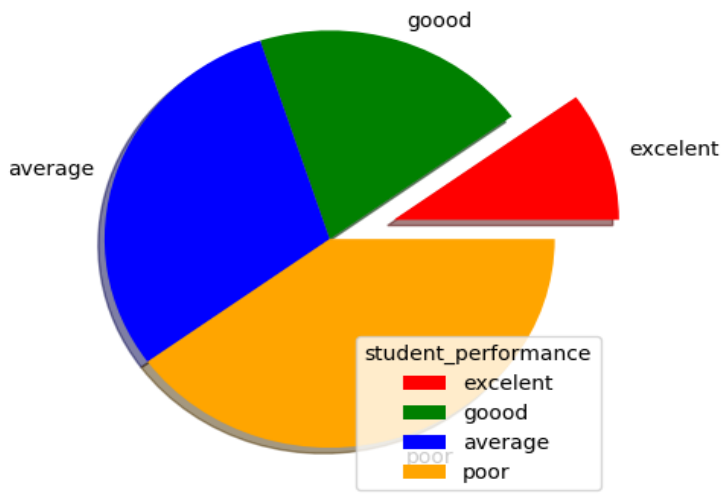


Ex:2

```
from matplotlib import pyplot as plt
```

```
student_performance=["excelent","goood","average","poor"]
student_values=[10,20,30,40]
c=["red","green","blue","orange"]

plt.pie(student_values,labels=student_performance,explode=[0.3,0,0,0],shadow
=True,colors=c)

plt.legend(title="student_performance")
plt.show()
```

output:



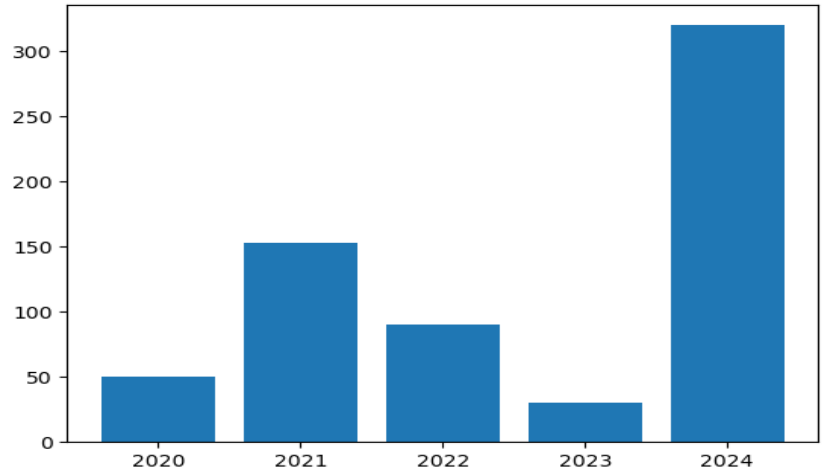
Bar graph

Bar graphs are one of the most common types of graphs and are used to show data associated with the categorical variables. Matplotlib provides a **bar()** to make bar graphs which accepts arguments such as: categorical variables, their value and color.

Ex:1

```
from matplotlib import pyplot as plt

x=[2020,2021,2022,2023,2024]
y=[50,153,90,30,320]

plt.bar(x,y)

plt.show()
```

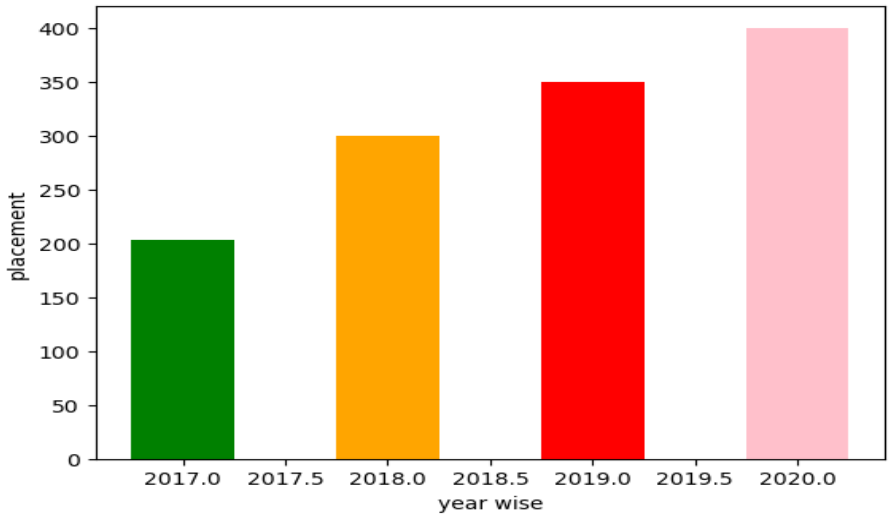
output:



Ex:2

```
from matplotlib import pyplot as plt

x=[2017,2018,2019,2020]
y=[203,300,350,400]
c=["green","orange","red","pink"]
plt.bar(x,y,color=c,width=0.5)

plt.xlabel("year wise")
plt.ylabel("placement ")
plt.show()
```

output:

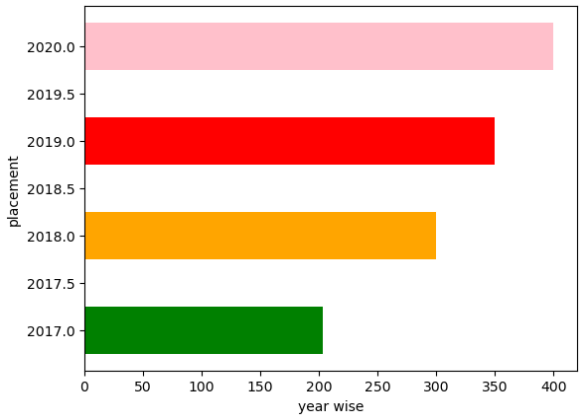


q2

Ex:3

```
from matplotlib import pyplot as plt

x=[2017,2018,2019,2020]
y=[203,300,350,400]
c=["green","orange","red","pink"]
plt.barh(x,y,color=c,height=0.5)

plt.xlabel("year wise")
plt.ylabel("placement")
plt.show()
```

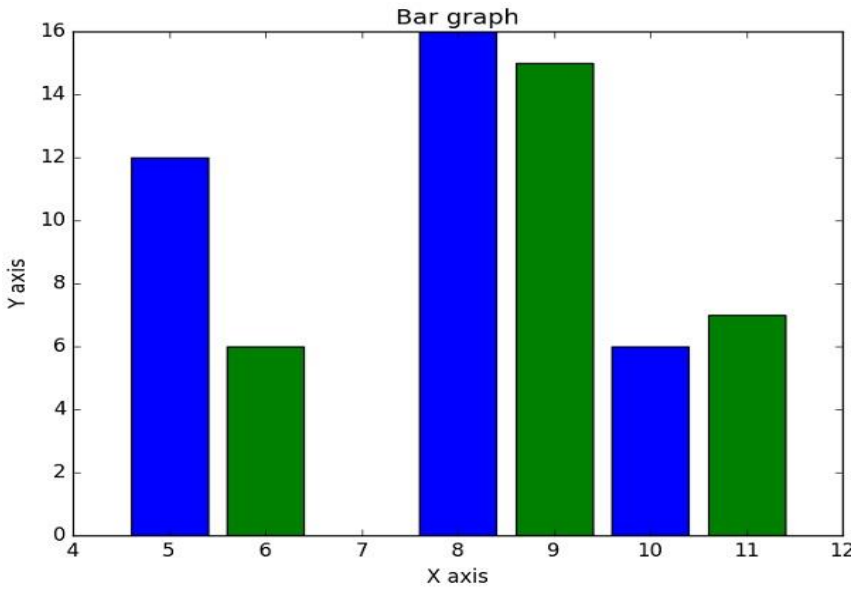
output:



Ex:4

```
from matplotlib import pyplot as plt
x = [5,8,10]
y = [12,16,6]

x2 = [6,9,11]
y2 = [6,15,7]
plt.bar(x, y, align = 'center')
plt.bar(x2, y2, color = 'g', align = 'center')
plt.title('Bar graph')
plt.ylabel('Y axis')
plt.xlabel('X axis')

plt.show()
```

output:



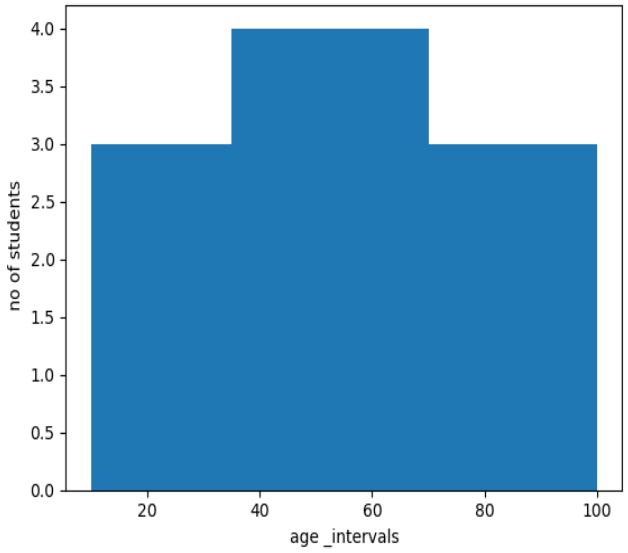
Histogram

First, we need to understand the difference between the bar graph and histogram. A histogram is used for the distribution, whereas a bar chart is used to compare different entities. A histogram is a type of bar plot that shows the frequency of a number of values compared to a set of values ranges.

Ex:1

```
from matplotlib import numpy as np


students=[1,10,20,35,50,90,83,85,13,48,55]

age_intervals=[10,35,70,100]


plt.xlabel("age _intervals")

plt.ylabel(" no of students")


plt.hist(students,age_intervals)


plt.show
```

output:

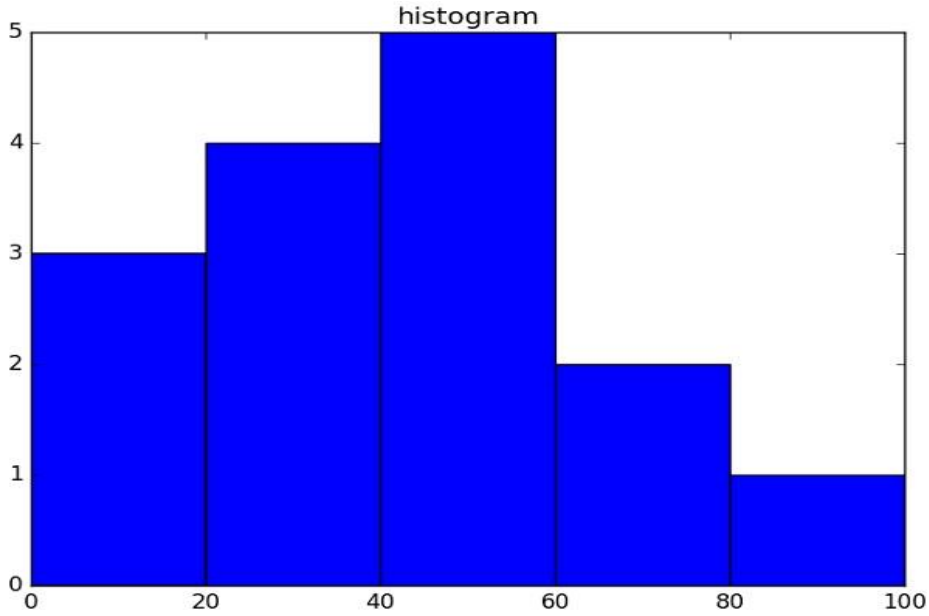


Matplotlib can convert this numeric representation of histogram into a graph. The **plt() function** of pyplot submodule takes the array containing the data and bin array as parameters and converts into a histogram.

Ex:1

```
from matplotlib import pyplot as plt

import numpy as np


a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])

plt.hist(a, bins = [0,20,40,60,80,100])

plt.title("histogram")

plt.show()
```

output:



Grid:

In Matplotlib, a grid is a set of horizontal and vertical lines that help to better visualize the data by aligning points and making it easier to read the plot. You can add a grid to your plot using the grid() function. Here are some examples of how to use grids in Matplotlib:

Ex:1from matplotlib import pyplot as plt

x = [80, 85, 90, 95, 100, 105, 110, 115, 120, 125]

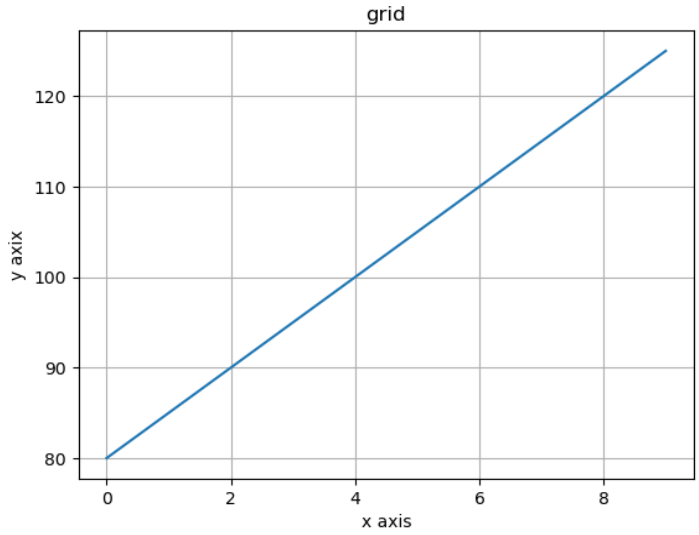
y = [240, 250, 260, 270, 280, 290, 300, 310, 320, 330]

plt.plot(x)

plt.grid()

```
plt.title("grid")
plt.xlabel("x axis")
plt.ylabel("y axix")

plt.show()
```

output:



Ex:2

- which: Can be 'major', 'minor', or 'both' to specify which grid lines to apply the style to.
- linestyle: The style of the grid lines (e.g., '-', '--', '-.', ':').
- linewidth: The width of the grid lines.
- color: The color of the grid lines.

```
from matplotlib import pyplot as  plt

x=[1,2,3,4,5]
y=[2,3,5,7,11]
```

```
plt.plot(x,y)
plt.grid(True,linestyle="dashed",linewidth=0.5,color="red")
plt.title("grid data")
plt.xlabel("x axix")
plt.ylabel("y axis")

plt.show()
```

output:

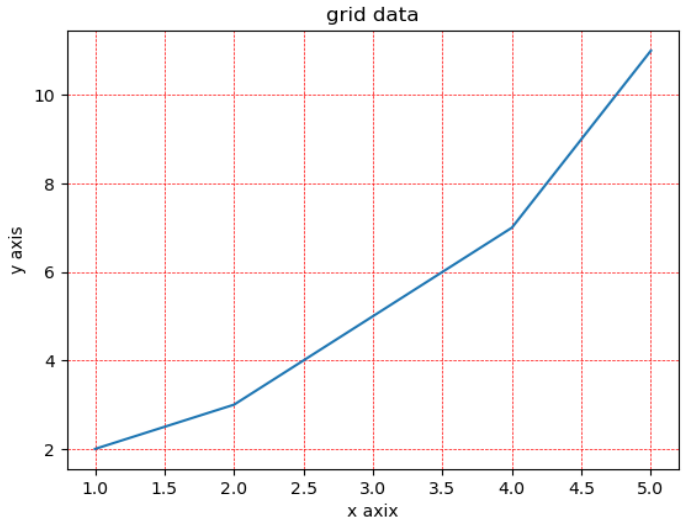


Image insertation

Inserting images into a Matplotlib plot can be done using the imshow function. Here's a basic example to demonstrate how to do this:

The extent parameter of **imshow** is used to set the limits of the image in data coordinates.

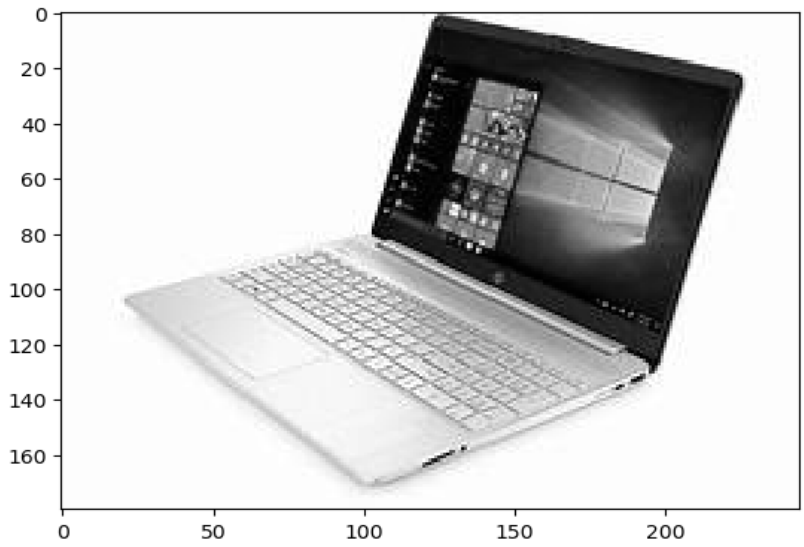
The **ax.plot** function is used to plot data on top of the image.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import convolve
```

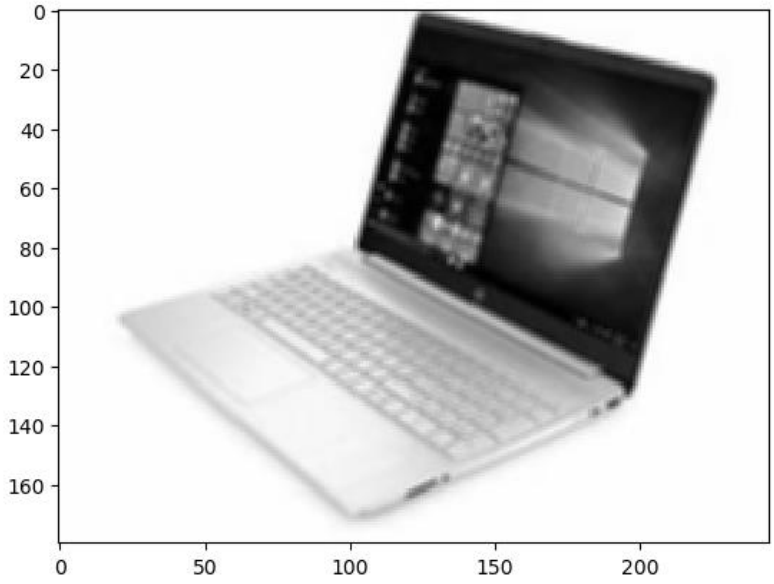
```
image = plt.imread("C:/Users/venky blenders/OneDrive/Pictures/laptop.jpeg")

plt.imshow(image, cmap='gray')
plt.show()
```

output:


```
if image.ndim == 3:
 image2 = np.mean(image, axis=2)

image2.shape
(180, 245)

plt.imshow(image2, cmap='gray')
plt.show()
```




```
kernel = np.array([[1,1,1],[1,1,1],[1,1,1]])/9
image3 = convolve(image2, kernel)
plt.imshow(image3, cmap='gray')
plt.show()
```



Ex:2

```
import matplotlib.pyplot as plt

# Load an image from file
```

```
img = plt.imread("C:/Users/venky blenders/OneDrive/Pictures/pexels-photo.jpg")

# Create a plot
fig, ax = plt.subplots()

# Display the image
ax.imshow(img)

# Hide the axes
ax.axis('off')

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

output:

ex:3

```
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
```

```
import matplotlib.image as mpimg
import numpy as np

# Load an image
image = mpimg.imread('C:/Users/venky blenders/OneDrive/Pictures/image 1 - Copy.jpg')

# Create some data to plot
x = np.linspace(0, 10, 100)
y = np.sin(x)

fig, ax = plt.subplots()

# Display the image as the background
ax.imshow(image, extent=[0, 10, -1, 1])

# Plot the data
ax.plot(x, y, 'r-', linewidth=2)

# Adjust the limits and add labels
ax.set_xlim(0, 10)
ax.set_ylim(-1, 1)
ax.set_xlabel('X-axis')
ax.set_ylabel('Y-axis')

plt.show()
```

output:


1
0
−1
Y-axis
0
2
4
6
8
10
X-axis