CHAPTER 4

Presented by
Afreen Banu & Ashwin R G

Agenda

- Anaconda
- Jupyter Notebooks
- Numpy Basics
- Pandas Basics
- Matplotlib Basics

NUMPY BASICS

- NumPy, which stands for Numerical Python,
- NumPy is a Python library used for working with arrays.
- NumPy was created in 2005 by Travis Oliphant
- The most important object defined in NumPy is an N-dimensional array type called **ndarray**

How to define

- **numpy.array**

  It creates a ndarray from any object exposing an array interface, or from any method that returns an array.

- **Checking NumPy Version**

```
import numpy as np

print(np.__version__)
```

- **type()**

  This built-in Python function tells us the type of the object passed to it.

NUMPY.ARRAY(OBJECT, DTYPE = NONE, COPY = TRUE, ORDER = NONE, SUBOK = FALSE, NDMIN = 0)

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array or any (nested) sequence.
2 3	**dtype** The desired data type of array, optional**copy**Optional. By default (true), the object is copied
4	**order**C (row-major) or F (column-major) or A (any) (default)
5	**subok** By default, returned array forced to be a base class array. If true, sub-classes passed through
6	**ndmin** Specifies minimum dimensions of the resultant array

Creating arrays

Ex1:

#0D array

- import numpy as np

  arr = np.array(42)

  print(arr)
- Output:

  42

Ex 2:

#1 dimensional array

```
import numpy as np
    a = np.array([1,2,3])
    print a
```

The output is as follows –

```
[1, 2, 3]
```

Ex 3:

2 dimensional arrays

```
import numpy as np
   a = np.array([[1, 2], [3, 4]])
    print a
```

The output is as follows –

```
[[1, 2]
 [3, 4]]
```

- Ex 3

#3D array

- import numpy as np

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

print(arr)
```

- Output:

```
[[[1 2 3]
[4 5 6]]
[[1 2 3]
 [4 5 6]]]
```

#Creating more than 3d array

- We can create by using **ndim** argument

Ex4

#Create array of 5D

- import numpy as np

```
arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)
```

- Output

```
[[[[[1 2 3 4]]]]]
```

How to check Dimensions

- Using **ndim** keyword

Ex 5:

```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5])
print(a.ndim)
print(b.ndim)
```

Output:

```
0
1
```

Checking Shape of the array

- Shape is the number of elements in the array
- **syntax:** numpy.shape(array_name)
  Parameters: Array is passed as a Parameter.
  Return: A tuple whose elements give the lengths of the corresponding array dimensions.

Ex6:

```
#find the shape of array
arr1 = np.array([1, 3, 5, 7])
print(arr1.shape)
```

Output:

```
(1,4)
```

- Besides creating an array from a sequence of elements, you can easily create an array filled with 0's:

>>> np.zeros**(2)**

Output:

array([0., 0.])

- Or an array filled with 1's:

>>> np.ones**(2)**

Output:

array([1., 1.])

- You can create an array with a range of elements:

>>> np.arange**(4)**

Output:

array([0, 1, 2, 3])

- And even an array that contains a range of evenly spaced intervals. To do this, you will specify the **first number**, **last number**, and the **step size**.

>>> np.arange**(2, 9, 2)**

Output:

array([2, 4, 6, 8])

Sorting arrays

- Sorting an element is simple with np.sort()

arr = np.array**([2, 1, 5, 3, 7, 4, 6, 8])**

>>> np.sort**(**arr**)**

Output:

array([1, 2, 3, 4, 5, 6, 7, 8])

- *Sorting alphabetical data*

```
import numpy as np

   arr = np.array(['banana', 'cherry', 'apple'])

   print(np.sort(arr))
```

Output:

['apple' 'banana' 'cherry']

Sorting 2D array

```
import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])

print(np.sort(arr))
```

Output:

```
[[2 3 4]
[0 1 5]]
```

Access Array Elements

- You can access an array element by referring to its index number.

1.import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])

Output:

1

2.data = np.array**([1, 2, 3])**

>>> data**[1]**

2

>>> data**[0:2]**

array([1, 2])

>>> data**[1:]**

array([2, 3])

>>> data**[-2:]**

array([2, 3])

- import numpy as np

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('2nd element on 1st row: ', arr[0, 1])
```

Output:

2

- import numpy as np

```
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr[0, 1, 2])
```

Output:

6

- import numpy as np

```
arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])
print('Last element from 2nd dim: ', arr[1, -1])
```

Output:

10

- import numpy as np

```
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5:2])
```

- Output:

[2,4]

- import numpy as np
  arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
  print(arr[1, 1:4]
- Output:

[7,8,9]

Searching in array

By using 2 methods

- numpy.where()
- numpy.searchsorted()
- **numpy.where:()** It returns the indices of elements in an input array where the given condition is satisfied.
- **numpy.searchsorted():** The function is used to find the indices into a sorted array arr such that, if elements are inserted before the indices, the order of arr would be still preserved

WHERE()

- import numpy as np
  arr = np.array([1, 2, 3, 40, 5, 40, 40])
  x = np.where(arr == 40)
  print(x)

Output:

(array([3, 5, 6],)

- import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])

x = np.where(arr%4 == 0)

print(x)

Output:

(array([3, 7]),)

SEARCHSORTED()

- import numpy as np
  arr = np.array([6, 7, 9, 19])
  x = np.searchsorted(arr, 8)
  print(x)

Output:

2

- np.searchsorted**([1,2,3,4,5], 3)**

Output:2

- >>> np.searchsorted**([1,2,3,4,5], 3,** side='right'**)**

Output:3

- >>> np.searchsorted**([1,2,3,4,5], [-10, 10, 2, 3])**
  Output:*array([0, 5, 1, 2])*

Reshaping array: reshape()

- import numpy as np
  arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
  newarr = arr.reshape(4, 3)
  print(newarr)

Output:

[[1 2 3]

[4 5 6]

[7 8 9]

[10 11 12]]

- Convert the array into a 1D array:(-1)
- import numpy as np
  arr = np.array([[1, 2, 3], [4, 5, 6]])
  newarr = arr.reshape(-1)
  print(newarr)
- Output:[1 2 3 4 5 6]

Filtering array: filter()

- In NumPy, you filter an array using a *boolean index list*.
- If the value at an index is True that element is contained in the filtered array, if the value at that index is False that element is excluded from the filtered array.
- import numpy as np
  arr = np.array([41, 42, 43, 44])
  x = [True, False, True, False]
  newarr = arr[x]
  print(newarr)

Output:[41 43]

- seq = [0, 1, 2, 3, 5, 8, 13]

```
  # result contains odd numbers of the list
result = filter(lambda x: x % 2 != 0, seq)
print(list(result))
 # result contains even numbers of the list
result = filter(lambda x: x % 2 == 0, seq)
print(list(result))
```

Output:

- [1, 3, 5, 13]
- [0, 2, 8]

JOINING ARRAY:

- concatenate(): It simply joins 2 arrays.
- stack():Stacking is same as concatenation, the only difference is that stacking is done along a new axis.
- hstack(): this joins along rows.
- vstack() :this joins along columns.
- dstack():this is to stack along height, which is the same as depth.

CONCATENATE()

- a=np.arange(6).reshape(2,3)

Print a

Output:([[0,1,2]

[3,4,5]])

- b=np.arange(7,13).reshape(2,3

Print b

Output: ([[7,8,9]

[10,11,12]])

- np.concatenate((a,b))

Output:

Array([[0,1,2]

[3,4,5]

[7,8,9]

[10,11,12]])

- np.concatenate((a,b),axis=1)

Output:

Array([[0,1,2 ,7,8,9],

[3,4,5,10,11,12]])

- np.stack((a,b))

Output:([[[0,1,2]

[3,4,5]],

[[7,8,9]

[10,11,12]]])

VSTACK()

- np.vstack((a,b))

Output:Array([[0,1,2],
[3,4,5],
[7,8,9],
[10,11,12]])

- np.hstack((a,b))

Output:Array([[0,1,2, 7,8,9],
[3,4,5, 10,11,12]])

- np.dstack((a,b))

Output: ([[[0,7],
[1,8],
[2,9]],

[3,10],
[4,11],
[5,12]]])

Pandas Basics

PANDAS

- Pandas is **an open source Python package that is most widely used for data science/data analysis**
- **It works with large data sets**
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis" and was created by Wes McKinney in 2008.
- It uses 3 Datastructures 1.Series
  2. Data frames
  3. Panel
- Series: deals with 1D array
- Data frames: deals with 2D array
- Panel:deals with multi dimensions

Importing Pandas

- import pandas as pd
- Syntax: pd.Series(data,index)
- Syntax:pd.DataFrame(data)
- Syntax:pd.Panel(data,major axis,minor axis,dtype)

Creating series

- Empty series
- Series using arrays
- Series using lists
- Series using dictionary

- Creating empty series

```
import pandas as pd
S=pd.Series( )
Series( ( ) )
```

- Creating series using array

```
a=np.array([10,20,30,40])
Output:0  10
          1  20
          2   30
          3   40
```

- import pandas as pd
  a = [10, 70, 20]
  myvar = pd.Series(a, index = ["x", "y", "z"])
  print(myvar)

Output:x 10

y 70

z 20

- Accessing by referring variables

```
import pandas as pd
a = [10, 70, 20]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar["y"])
```

Output:70

Creating series using lists

- L=[10,20,30,40]

```
pd.Series(L)
Output:0 10
       1  20
       2  30
       3  40
```

- Series using dictionary

```
import pandas as pd
sales = {"day1": 42, "day2": 38, "day3": 39}
myvar = pd.Series(calories)
print(myvar)
Output:day1 42
       day2 38
       day3 39
```

Creating Data frames

- import pandas as pd
  data = {
    "calories": [420, 380, 390],
    "duration": [50, 40, 45]
  }
  myvar = pd.DataFrame(data)
  print(myvar)

Output: calories duration
0 420 50
1 380 40
2 390 45

- Accesing rows:by using loc keyword

print(df.loc[0])

Output:calories 420
duration 50

print(df.loc[[0, 1]])

Output: calories duration
0 420 50
1 380 40

- import pandas as pd
  sales = {
    “price": [120, 280, 390],
    “pieces": [50, 40, 45]
  }
  myvar =
  pd.DataFrame(data,index=["day1", "day2", "day3"])
  print(myvar)

print(myvar.loc["day2"])

Output:calories 280

duration 40

Loading Files Into a DataFrame

- Loading data from excel file: read_excel
- Loading data from CSV file: read_csv

- Loading excel file

```
import pandas as pd
Df=pd.read_excel("path")
```

- Loading csv file

```
import pandas as pd
Df=pd.read_csv("path")
```

- import pandas as pd
  df = pd.read_csv('data.csv')
  print(df.to_string())

Output: it will print entire data

- Print the DataFrame without the to_string() method:

import pandas as pd

df = pd.read_csv('data.csv')

print(df)

Output:it will print only starting 5 rows and ending 5 rows

Reading json

- Big data sets are often stored, or extracted as JSON.

```
import pandas as pd
df = pd.read_json('data.json')
print(df.to_string())
```

Output:prints entire data

Dictionary as JSON

```
import pandas as pd

data = {
  "Duration":{
    "0":60,
    "1":60,
    "2":60,
    "3":45,
    "4":45,
    "5":60
  },
  "Pulse":{
    "0":110,
    "1":117,
    "2":103,
    "3":109,
    "4":117,
    "5":102
  },
  "Maxpulse":{
    "0":130,
    "1":145,
    "2":135,
    "3":175,
    "4":148,
    "5":127
  },
  "Calories":{
    "0":409,
    "1":479,
    "2":340,
    "3":282,
    "4":406,
    "5":300
  }
}

df = pd.DataFrame(data)

print(df)
```

- Output:

	Duration	Pulse	Maxpulse	Calories
0	60	110	130	409.
1	60	117	145	479.0
2	60	103	135	340.0
3	45	109	175	282.4
4	45	117	148	406.0
5	60	102	127	300.5

Viewing the Data : slicing

- Head( )
- tail( )
- One of the most used method for getting a quick overview of the top rows, is the head() method.
- One of the most used method for getting a quick overview of the bottom rows, is the tail() method.

- import pandas as pd
  df = pd.read_csv('data.csv')
  print(df.head(10))

Output:returns top 10 rows

print(df.head())

Output:returns only 5 top rows

- import pandas as pd
  df = pd.read_csv('data.csv')
  print(df.tail(10))

Output:returns 10 bottom rows

print(df.tail())

Output:returns only 5 bottom rows

Other built in operations in dataframes

- Df.shape:returns no of rows and columns
- Df.loc( ):returns rows
- Df.loc[df[col name]==‘val’]:returns particular row with the given condition
- Df.columns:returns the names of coulmns
- Df.[‘col name’].head():returns particular column
- Df.[‘col name’].tail( ):
- Df.[start index:end index:step size]

Handling missing data:handling null values

- Sometimes our dataset contain some empty cells and those cells are known as null values

WHY TO HANDLE NULL VALUES

- Because we cant provide null values to our machine learning model

What can we do

- Either fill the null values with some values
- Drop such values
- Either replace them

- isnull( ): it checks the data and returns in the form of Boolean values
- isnull( ).sum( ):it returns the number of null values in the particular rows or coulumns in the data set.
- isnull( ).sum( ).sum( ):it returns total number of null values in data set.

FILLING NULL VALUES:

- df.fillna(value=‘ ’)
- df.fillna(method=‘pad’):filling the previous value
- df.fillna(method=‘bfill’):filling the next value
- df.fillna(method=‘pad’, axis=1):
- df.fillna(method=‘bfill’, axis=1)
- df.fillna({‘name’: ‘xyz’, ‘rno’: ‘123’}) :filling different values in ull in different columns
- df.fillna(value=df[‘name’].mean( )):filling with the mean value of that column
- df.fillna(value=df[‘name’].min( )):filling with the minimum value of that column
- df.fillna(value=df[‘name’].max( )):filling with the maximum value of that column

Dropping null values:dropna( )

- df.dropna( ):it drops all the null values
- df.dropna(how='all'):it drops only if all the values of that column are null
- df.dropna(how='any'):it drops only if any of the values of that column are null

Replacing null values:replace( )

- Df.replace(to_replace=np.nan,value=1234):
  it replaces null values into specified value
- Df.replace(to_replace=3.0,value=4.0):
  it replaces any values into specified value

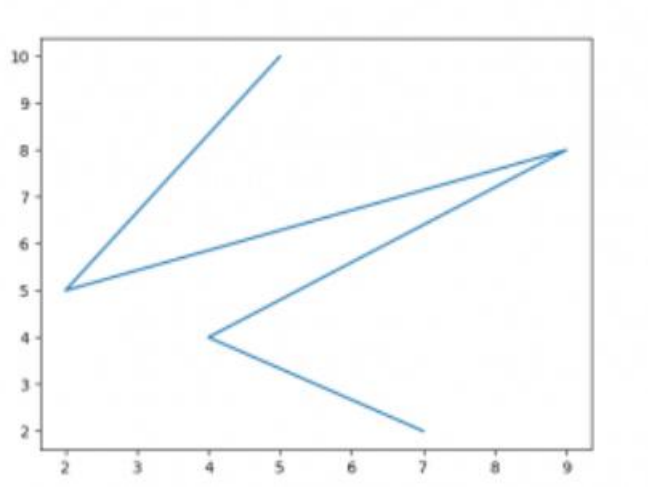
MATPLOTLIB BASICS

- Matplotlib is an amazing visualization library in Python for 2D plots of arrays
- Matplotlib was created by John D. Hunter.
- Matplotlib is open source and we can use it freely.
- Module we are going to use is PYPLOT
- The alias we are using for Pyplot is plt
- Matplotlib consists of several plots like line, bar, scatter, histogram etc.

BASIC PLOTS: LINE PLOT

```
# importing matplotlib module
from matplotlib import pyplot as plt

# x-axis values
x = [5, 2, 9, 4, 7]

# Y-axis values
y = [10, 5, 8, 4, 2]

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

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

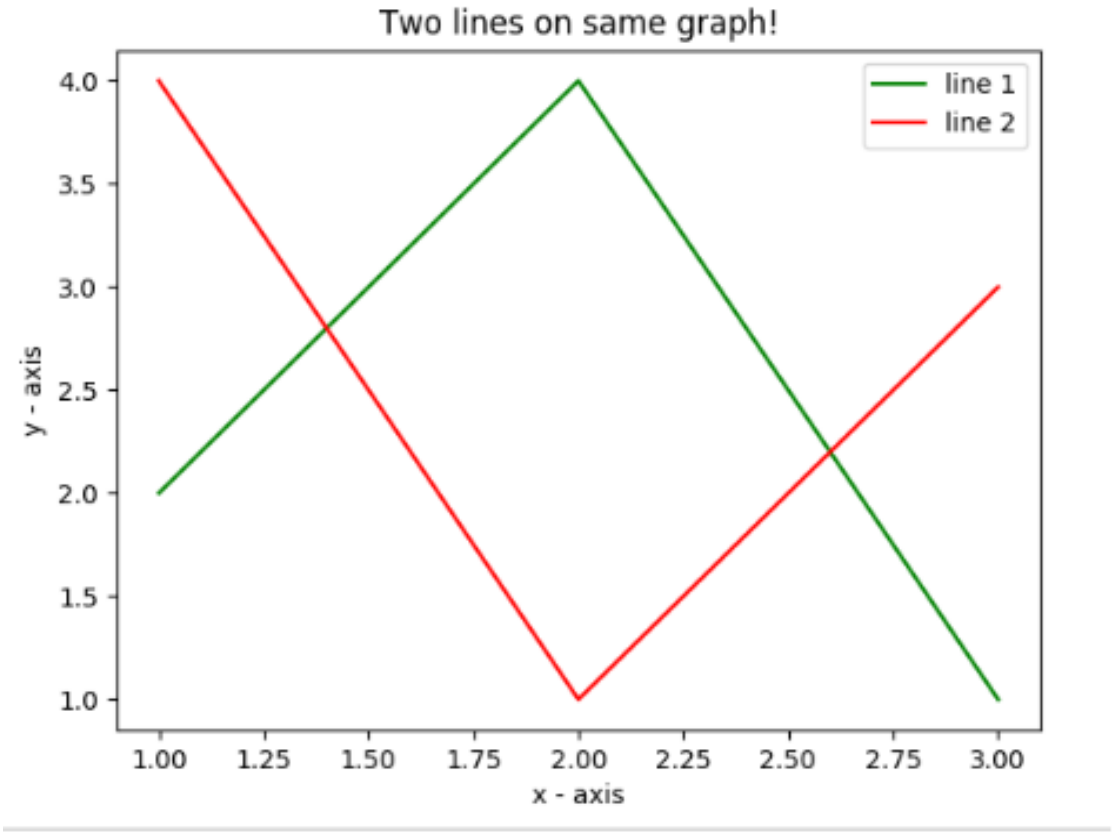
O



Plotting 2 or more lines in same graph

```
import matplotlib.pyplot as plt

# line 1 points
x1 = [1,2,3]
y1 = [2,4,1]
# plotting the line 1 points
plt.plot(x1, y1, label = "line 1")

# line 2 points
x2 = [1,2,3]
y2 = [4,1,3]
# plotting the line 2 points
plt.plot(x2, y2, label = "line 2")

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')
# giving a title to my graph
plt.title('Two lines on same graph!')

# show a legend on the plot
plt.legend()

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

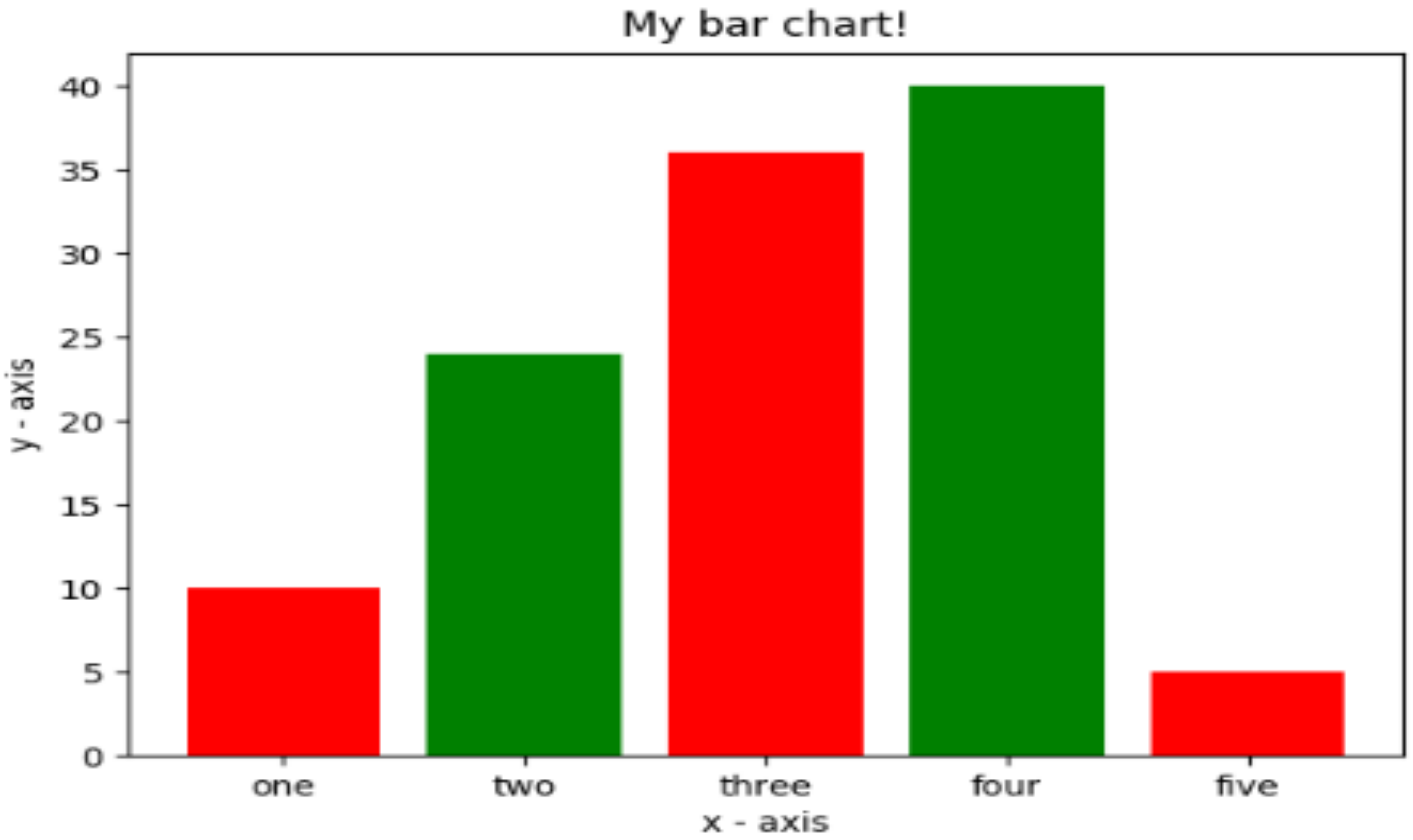
OUTPUT:


Two lines on same graph!
line 1
line 2
4.0
3.5
3.0
2.5
2.0
1.5
1.0
y - axis
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
x - axis

Bar graph

```
import matplotlib.pyplot as plt

# x-coordinates of left sides of bars
left = [1, 2, 3, 4, 5]

# heights of bars
height = [10, 24, 36, 40, 5]

# labels for bars
tick_label = ['one', 'two', 'three', 'four', 'five']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label,
        width = 0.8, color = ['red', 'green'])

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')

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

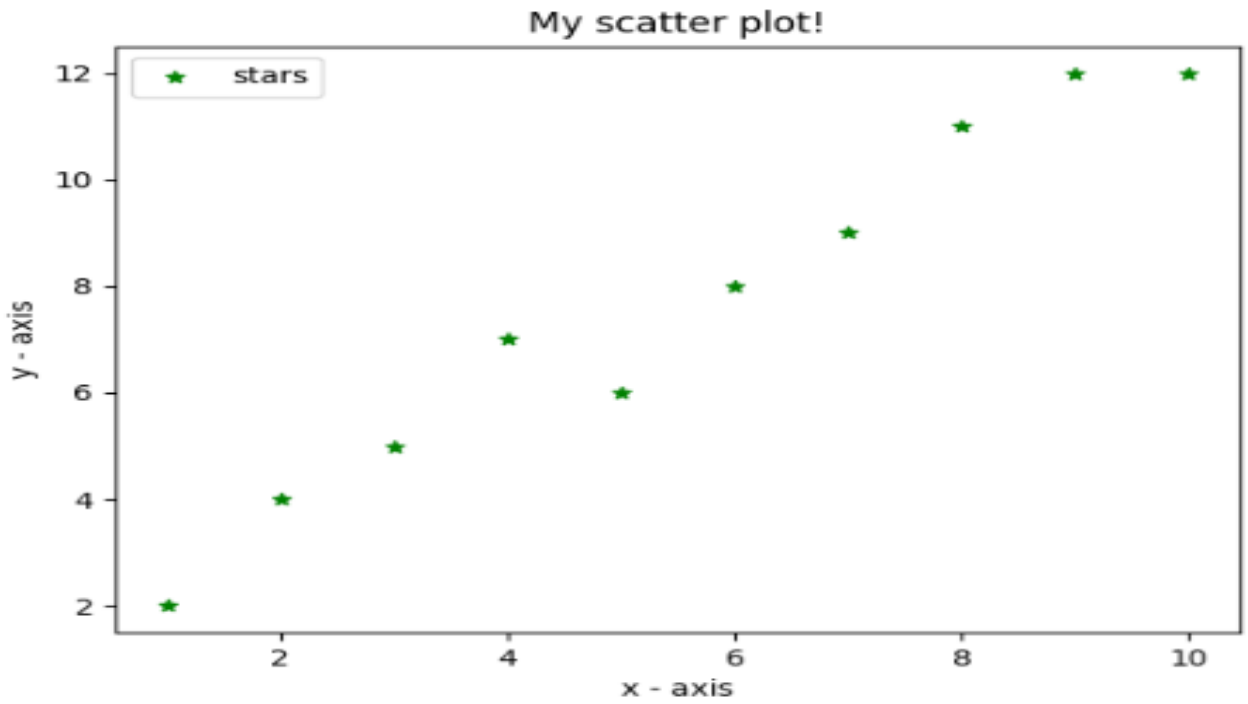
OUTPUT


My bar chart!
y - axis
40
35
30
25
20
15
10
5
0
one
two
three
four
five
x - axis

Scatter graph

```
import matplotlib.pyplot as plt

# x-axis values
x = [1,2,3,4,5,6,7,8,9,10]
# y-axis values
y = [2,4,5,7,6,8,9,11,12,12]

# plotting points as a scatter plot
plt.scatter(x, y, label= "stars", color= "green",
            marker= "*", s=30)

# x-axis label
plt.xlabel('x - axis')
# frequency label
plt.ylabel('y - axis')
# plot title
plt.title('My scatter plot!')
# showing legend
plt.legend()

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

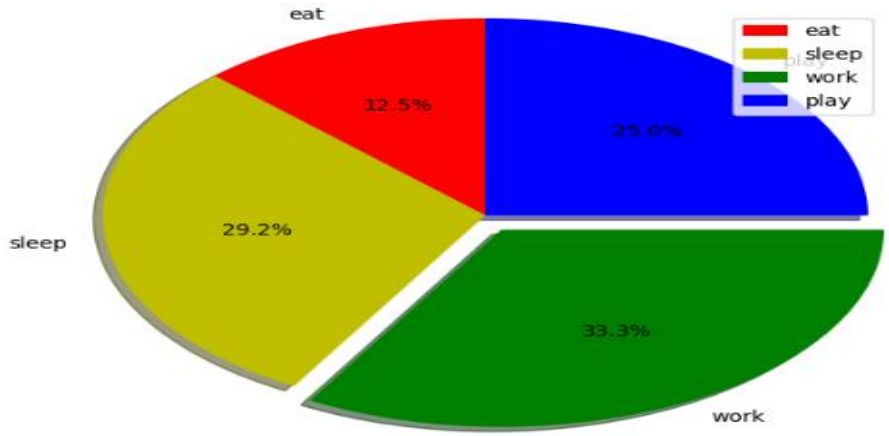
OUTPUT



Pie chart

```
import matplotlib.pyplot as plt

# defining labels
activities = ['eat', 'sleep', 'work', 'play']

# portion covered by each label
slices = [3, 7, 8, 6]

# color for each label
colors = ['r', 'y', 'g', 'b']

# plotting the pie chart
plt.pie(slices, labels = activities, colors=colors,
        startangle=90, shadow = True, explode = (0, 0, 0.1, 0),
        radius = 1.2, autopct = '%1.1f%%')

# plotting legend
plt.legend()

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

OUTPUT

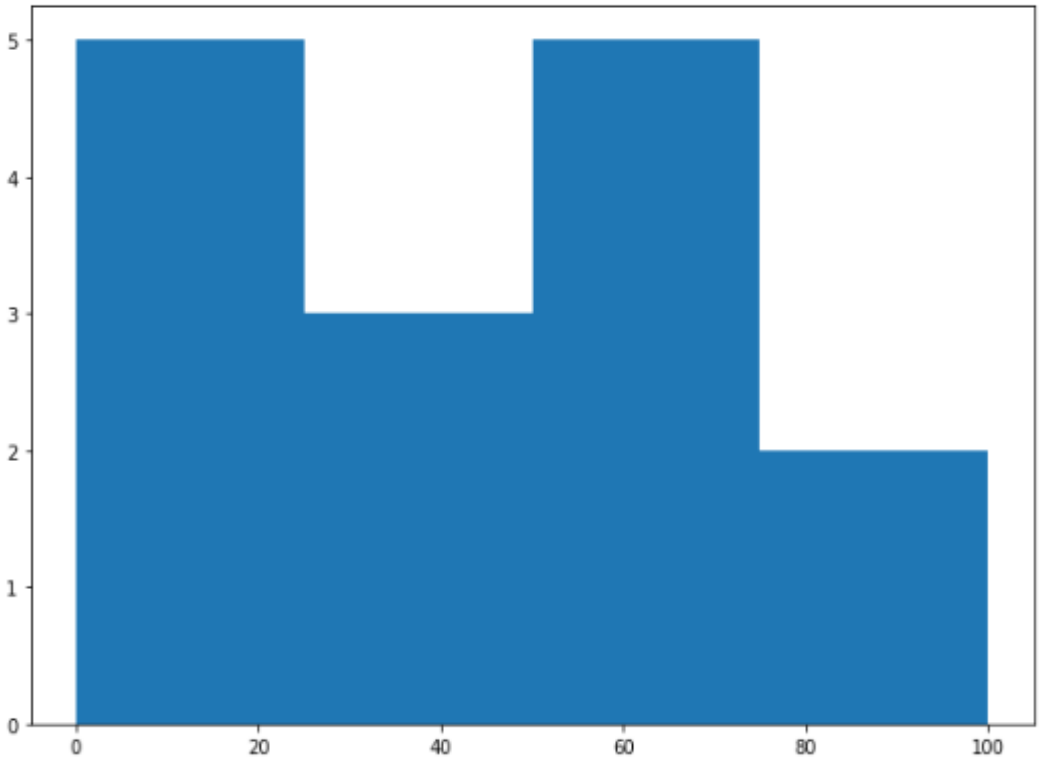


•Here, we plot a pie chart by using **plt.pie()** method.
•First of all, we define the **labels** using a list called **activities**.
•Then, a portion of each label can be defined using another list called **slices**.
•Color for each label is defined using a list called **colors**.
•**shadow = True** will show a shadow beneath each label in pie chart.
•**startangle** rotates the start of the pie chart by given degrees counterclockwise from the x-axis.
•**explode** is used to set the fraction of radius with which we offset each wedge.
•**autopct** is used to format the value of each label. Here, we

HISTOGRAM

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


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

# Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(a, bins = [0, 25, 50, 75, 100])

# Show plot
plt.show()
```

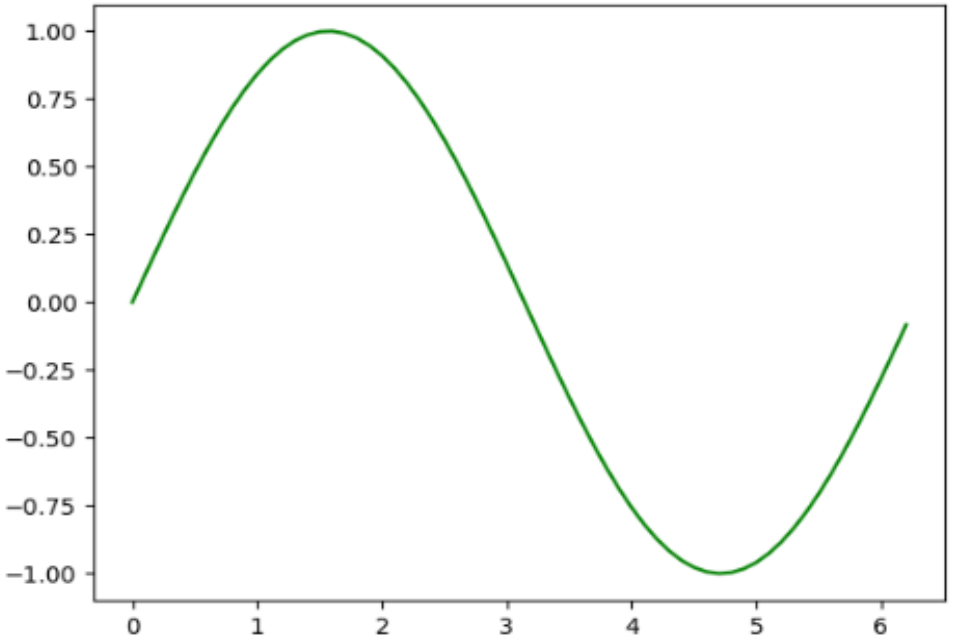
OUTPUT



Plotting graph of given equation

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

# setting the x - coordinates
x = np.arange(0, 2*(np.pi), 0.1)
# setting the corresponding y - coordinates
y = np.sin(x)

# plotting the points
plt.plot(x, y)

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

OUTPUT



•To set the x-axis values, we use the **np.arange()** method in which the first two arguments are for range and the third one for step-wise increment. The result is a NumPy array.
•To get corresponding y-axis values, we simply use the predefined **np.sin()** method on the NumPy array.
•Finally, we plot the points by passing x and y arrays to the **plt.plot()** function.