Numpy:-

Perle Python no library for numerical computation used to create Array & matrix

Numpy → Numerical + Python.

→ import numpy as np

l = [1,2,3,4]

np.array(l)

⇒ array([1,2,3,4])

type(ar)

⇒ numpy.ndarray

↑

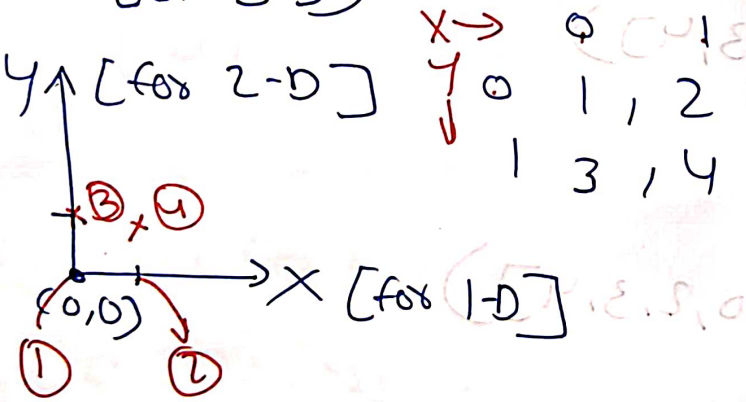
n dimentional.

colarful grage (RGB).

⇒ np.array([1,2],[3,4]]) → Two dimentional

array([[1,2],

[3,4]])



np.asarray(l) ⇒ convert into array.

⇒ array([1,2,3,4])

np.asanyarray(a)

⇒ array([[3,4]])

```
b= np.matrix(1)
b
⇒ matrix([[1,2,3,4]])   always two 2-D matrix
np.asanyarray(b)
matrix([[1,2,3,4]]) → subset of array matrix
                      two dimentional.
```

```
L= [1,2,3,4
a= np.array(L)
a
⇒ array([1,2,3,4])
c=a → shallow copy.
c
⇒ array([1,2,3,4])
a
⇒ array([1,2,3,4])
c[0]=100
c
⇒ array([100,2,3,4])
a
⇒ array([100,2,3,4])
d= np.copy(a) ← deep copy.
d
⇒ array([100,2,3,4])
a
⇒ array([100,2,3,4])
```

a[1] = 400

a

⇒ array([100, 400, 3, 4])

d

⇒ array([[100, 2, 3, 4]])

np.fromfunction(lambda i, j : i == j, (3,3))

i \ j →	0	1	2
0	T	F	F
1	F	T	F
2	F	F	T

3×3

np.fromfunction(lambda i, j : i*j, (3,3))

array([[0, 0, 0],
[0, 1, 2]
[0, 2, 4]])

a = (i*i for i in range(5))

np.fromiter(a, float)

⇒ array(0, 1, 4, 9, 16)

np.fromstring('234 234', sep=' ')

⇒ array([234, 234])

np.fromstring('4,5', sep=',')

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

ar = np.array(l)

ar.

⇒ array([2,3,4,5,6])

ar.ndim.

ar2 = np.array([[1,2,3,4], [2,3,4,5]])

ar2

ar2.ndim

2

ar.size (Kithe element present hai].

⇒5

ar2.size

⇒ 8

ar.shape

⇒ (5)

(2, 4)
rows columns

ar.dtype

⇒ dtype(int)

ar22 = np.array([(1.4,45,45), (23,45,66)])

ar22

ar22.dtype

list(range(5)) → point no ~~negative~~ no not possible.

[0,1,2,3,4]

list(range(0.1,5))

np.arange(2.3,5.6)

array([2.3, 3.3, 4.3, 5.3])

np.arange(2.3, 5.6, .3)

array([[2.3, 2.6 ... - - - - 5.3]])

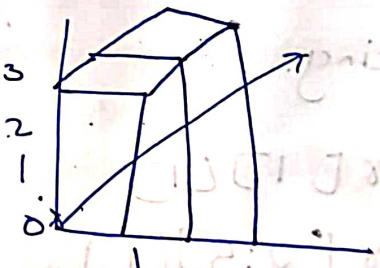
list(np.arange(2.3, 5.6, .3)) → agar list mai data chiy.

np.linspace(1,5) → 10 data produce. → Between particular interval.

⇒ data in fixed range.

np.zeros(5)

np.zeros((3,4)) → row, column.



np.zeros((3,4,2)) → array, row, column

ary = np.zeros((3,4,2,3))

ary.ndim.

np.ones(4)

np.ones((2,3))

on = np.ones((2,3,2))

on*4

np.empty((3,4))

np.eye(4) → identity.

np.linspace(2,4,20).

np.logspace(2,5,10, base=2)
↑ zaroori nhi hai

np.random(randn(3,4)
low ↑ colum

np.random.rand(3,4).

np.random.randint(1,110;(3,4))

arr = np.random.rand(3,4)

arr.

arr.reshape(6,2) → -1 khud se smjh lega.

Slicing.

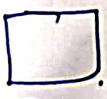
arr[1][1]

arr[2:5][1]

arr = np.random.randint(1,100,(5,5))

arr>50

arr[arr>50]



arr[2:4, [1,2]]

```
arr[0][0] = 5000

arr1 = np.random.randint(1,3, (3,3))
arr2 = np.random.randint(1,3, (3,3))

arr1 + arr2
arr1 - arr2
arr1 / arr2
arr1 * arr2 simple multiplication.
arr1 @ arr2 -> matrix multiplication.
arr1 | 10
arr1 + 100
arr1 ** 2
```

%. modulus or, and, ^ >

```
np.zeros((4,4))        arr = np.zeros((4,4))
low = np.arr([1,2,3,4])
arr + low
array [1,2,3,4]
      [1,2,3,4]
      [1,2,3,4]
      [1,2,3,4]]
low.T
col = np.array([[1,2,3,4]])
col.T + arr.
arr1 = np.random.randint(1,4, (3,4))
arr1
np.sqrt(arr1)
np.exp(arr1)
np.log10(arr1)
```

⇒ arr = np.random.randint (1,10, (3,4))

arr

arr [[[4,4,8,6],

_____

_____]])

⇒ arr.reshape (6,2)

⇒ arr.reshape (2,6)

⇒ arr.T (Transpose)

⇒ arr.flatten () 1D array.

arr1 = np.array ([1,2,3,4])

arr1.ndim

⇒ np.expand_dims (arr1, axis=0 / axis=1)

⇒ data = np.array ([[1],[2],[3]])

~~data~~ np.squeeze (data) → 1D

⇒ array ([1,2,3,4])

⇒ np.repeat (arr1,3)

⇒ np.roll (arr1,2)

⇒ np.diag (arr1) → diagonal.

arr = np.array (['sudh', 'Kumar'])

np.char.upper (arr)

np.char.title(arr)

np.char.capitalize (arr)

np.sin (arr1)

np.cos (arr1)

np.tan (arr1)

np.log10(arr1)

np.exp(arr1)

np.sqrt (arr1)

np.power(arr1,2)

np.mean (arr1)

np.median (arr1)

np.std (arr1)

np.var (arr1)

np.min (arr1)

np.max (arr1)

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

np.sort (arr)

np.searchsorted (arr, 34)

np.count_nonzero (arr1)

np.where (arr>6) → index.

np.extract (arr>3, arr)

[Pandas]

Introduction: Pandas is a Python library used for working with data sets.

⇒ The name "pandas" has a reference for both "Panel Data", & "Python Data Analysis" and was created by Wes McKinney in 2008.

⇒ has a functions for analyzing, cleaning, exploring, and manipulating data.

Installation: (command) → pip install pandas

Import: Import pandas as pd.

Data Model:

Pandas is build around data structures called Series and Data frames.

Data for these collections can be imported from various file formats such as comma - separated values, JSON, Parquet,

SQL database tables or queries and Microsoft Excel.

Pandas is a popular Python library for data analysis and visualization. It offers two main data structures Dataframe and Series.

Dataframe : A dataframe is a two-dimensional tabular data structure that can store multiple columns of diff. data types.

A dataframe has a row index and column index, which can be labels, numbers, or a combination of both. Dataframe can be created from various sources, such as lists, dictionaries, arrays, or files.

Series : A Series is a one-dimensional, labeled array that can store a single column of any data type. A series has an index, which can be labels, numbers, or a combination of both. Series can also be created as various sources, such as lists, Dictionaries, Scalars, or files.

—2. Operations) ~

⊙ To read files in pandas

Command → pd.read_csv("filename").

Here, pandas only read ~~structured~~ structural data ~~data~~ structural data.

Structural data means data in Tabular form.

. let it store in a variable df.

i.e.

→ df = pd.read_csv("filename")

→ df.head()

> output : It will give starting 5 records by default.

→ df.head(3) [If you want to read only 3 records from starting]

→ df.tail(5) [If you want to read last 5 records]

→ type(df) [To check the type]

> output : pandas.core.frame.DataFrame

→ df.columns [To check the column in Dataframe]

→ list(df.columns) [To convert column in the form of list]

→ df['columnname'] [To check the data of particular column]

→ type(df['columnname'])

output : pandas.core.series.Series

Note :

Dataframe means multiple columns & multiple records whereas Series is equivalent to a list

→ list(df['columnname']) [To convert series into a list]

Note :

slightest diff. b/w list and series is that list didn't show indexing in output although it have indexing no. but do not show : and series shows indexing number in output.

→ df[['columnname']] [To pass a similar column as a list]

→ type(df[['columnname']])

output : pandas.core.frame.DataFrame

{If you pass a single column name without list then, its type will come as series but If pass with list then, the type will shown as Dataframe.}

→ ~~df [['email', keyword~~.

→ df [['columnname1', columnname2']]

[To select a multiple columns, pass the columns inside a list unless it will give an error]

→ df.dtypes [To show the data types]

{To Read an Excel sheet}.

→ pd.read_excel("Excel filename") [To read Excel file]
→ df1 = pd.read_excel [Store in a variable]
→ type (df1) [To check the type]

→ df1.dtypes [To check the datatype]

→ df1.columns [To show the columns inside Excel file]

→ df1 [['columname1', 'columname2']] [To check multiple columns]

{To Read a data which is Available on a certain link}

** The data which is available in the link in formate of comma means The separator is a comma "

→ pd.read_csv("Linkname") [To read the data available in the link]

→ df2 = pd.read_csv('Linkname') [Store in a variable]

→ df2.head(3) [To show first 3 records]

→ df2.columns [To show columns]

→ type(df2) [To show the type]

↳ dataframe

→ df2['columnname'] [To show a particular column]

→ type(df2['columnname'])

↳ series

→ df2.tail(3) [To show last 3 records]

NOTE:

Dataframe can have multiple columns, while Series can only have one. means that Dataframe can store more complex and heterogeneous data, while a Series can store more simple and Homogenous data.

(New operation)

↳ import pandas as pd

→ df = pd.read_csv("linkname")

→ df.head()

→ type(df) → (dataframe)

→ ~~dtype~~ df.dtypes

(Call the function) → df.describe()

[It only analyse statistical information. means only describe numerical data]

→ df.dtypes [To show the data types of the columns]

→ df.[['column1', 'column2', 'column3']]

[Column names whose datatype is catagorical means String format]

What if there are multiple columns ~~for~~ in catagorical format then. we can filter out the ~~condition~~ Columns whose data type is object [string] by a given condition.

→ df.dtypes == 'object'

output :

column 1	True →	where datatype is an object
Column 2	False }	where datatype is not an object
column 3	False }	

→ ~~df. data~~

→ df. dtypes [df. dtypes == 'object']

output :

~~#~~ Columnames such as :

Here's / These are indexes

Name	obj	datatype
Sex	obj	
Cabin	obj	
Embarked	obj	

dtype : object

→ df.dtypes[df.dtypes == 'object'].index

output :

Index(['Name', 'sex', 'Cabin', 'Embarked'],
dtype = 'object')

Note: [Here indexes are column's name whose datatype is an object.]

→ type(df.dtypes[df.dtypes == 'object'])

output :

pandas.core.series.Series

Now, For fetching out the data from the above indexes.

→ df[df.dtypes[df.dtypes == 'object'].index]

[To read the data of columns ~~in~~ which is. ~~categori~~ in ~~se~~ categorical format]

→ df[df.dtypes[df.dtypes == 'object'].index].describe()

[gives description for the catagorical column-s (String based) such as. That name columns has 890 records, cabin column has 204 records & show on ——]

→ df[df.dtypes[df.dtypes == 'float64'].index]

[To select the columns whose data type is float]

→ df[df.dtypes[df.dtypes == 'int64'].index]

[To select the columns whose datatype is int]

→ df[['columnname']][4:11]

[To select the data from 4th row to 11th row in the column]

→ df['new_col'] = 0 [To create a new column]

→ df [To show the data]

→ df["new_col1"] = df['column1'] + df['column2']

[To form a new column by adding two columns]

→ df [To show]

→ Pd.Categorical(df['column_name'])

is a function in the pandas which represent the unique catorigies.

→ pd. categorical (df['columnname']) [To show the unique categories of the column]

→ df['columnname'].unique() [To show the unique categories of particular/ desired column]

{ for conditional operation }.

→ df['Columnname'] // condition.

Output: will show In the form of True or false

→ df[df['Columnname'] // condition]

output: Record acc. to the condition will show.

→ len(df[df['columnname'] // condition])

[To show the total length]

→ df[(df['column1'] // cond① 1) & (df['column2'] // cond② 2)]

[For when two conditions are applied]

→ max (df['columnname']) [To show the maximum number of column]
is a function in pandas.

→ df[0:2] .[To select the row from 0 index to 2 index from the data].

→ df[0::2] [To select the even data from even indexes]

→ df.iloc[0:2] [To show the rows of 0 index to 2 index]

[Loc() and iloc() in Pandas Dataframe]

Loc() and iloc() are used in slicing data from Python Dataframe.

function .loc is primarily used for label indexing while function .iloc is mainly used for integer indexing.

for instance:

indexes of rows

→ df.loc[0:2, ['column1', 'column2', 'column3']]

[To show the desired records of data]

→ df.iloc [0:2, [0, 1, 2]]

Row's index

Inbuild index of column

→ df= pd.read_csv("#any csv link")

→ df.columns [To show the columns]

Drop function → used for deleting any rows and columns.

e.g. :

→ df.drop('columname', axis=1)

Output:	desired column will drop from the data. But the column will delete only temporary not permanently.

→ df.[To show the data]

→ df.drop('columname', axis=1, inplace=True)

output:	column permanently deleted from the data.

→ df.drop(3) [To drop the row] (but temporary)

{Row name or 3rd Row}

→ df. drop(3, inplace = True) [To permanently delete the 3rd row]

Note:

Axis: refers to how a function or operation is applied to the Data frame or the Series, pandas only takes two values, either 0 or 1, as an argument to the axis property.

- Axis 0 → refers to the rows of a data frame or a Series. means That the operation is applied to each rows of the Data frame or Series.
- Axis 1 → refers to the columns of a data frame or Series, means. that the operation is applied to each column of them.

inplace: inplace is a parameter in pandas. inplace By default is false means the operation will not possess permanently.

~~For~~ ~~per~~ To possess permanent change, → [inplace = True].

→ df.set_index('columnname') [To change indexes into columnname]

→ df.set_index('columnname', inplace = True), (for permanent change)

→ df.reset_index() [To reset the change]

→ ~~df~~

"Let's make a dictionary"

→ d = {'Key 1' : [3, 4, 5, 6, 7],
'Key 2' : [1, 2, 3, 4, 5],
'Key 3' : [3, 5, 4, 2, 1]
}

(must be of equal length)
[These key will become the column]

→ pd.dataframe(d)

output :

	Key 1	Key 2	Key 3
0	3	1	3
1	4	2	5
2	5	3	4
3	6	4	2
4	7	5	1

→ pd.read_csv('# any csv data')

→ df1 = pd.read_csv('# csv data') [To store into df1 variable]

→ df1.dropna() [Drop the rows from the data, where Nan/na is stored]
(Null values)

Note:

since by default axis = 0 (along the rows) in pandas.

→ df1.dropna(inplace = True) [To drop permanently]

→ df1.dropna(axis = 1, inplace = True) [To drop the Column where Null values is stored]

→ df1.fillna('#newname', inplace = True)

[To fill the row with the #new name where Na/null values is stored]
Or
(replace Na to the new name)

[groupby() Method in pandas]

groupby() is used for grouping the data according to the categories and applying function to the categories.

→ used to split the data into groups based on some criteria.

→ df [To show the first data]

→ df.reset_index(inplace=True) [To reset the changes permanently]

→ df.groupby('columnname')

→ g = df.groupby('columnname') [g is a variable]

→ g → { g is an object of groupby where the data of the column is split according to the categories. }

→ g.sum() [Total sum of column A/c to the different categories]

→ g.mean() [to find Average of the column A/c to diff' categories]

→ g.max()

→ g.max().T [To transpose rows into columns or columns into row]

(or)

→ g.max().transpose()

(Row's index)

→ df[['column1', 'column2', 'column3']][0:5]

[To fetch a small data from 0 index to 5 index of]

→ df2 = df[['column1', 'column2', 'column3']][0:5]

[To store in a df2 variable]

Similarly

→ df3 = df[['column4', 'column5', 'column6']][5:10]

→ df2 [To show the fetching data]

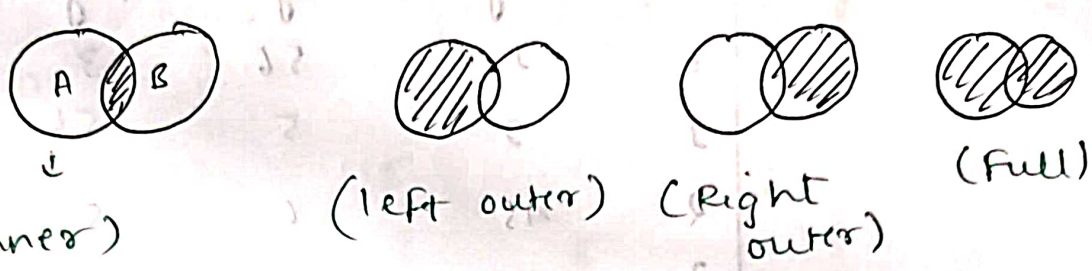
→ df3 [To show the fetch data]

→ pd.concat([df2, df3]) [to merge/concat]

[along vertically since by default value of axis = 0 (along row)]

→ pd.concat([df2, df3], axis=1) [To concat horizontally
↳ along the columns]

NOTE: Venn Diagram: (~~Inner~~ Intersection)



(Inner) (left outer) (Right outer) (Full)

let suppose,

→ data1 = pd.dataframe({'key1': [1, 2, 4, 5, 6],
'key2': [4, 5, 6, 7, 8],
'key3': [3, 4, 5, 6, 6]
})

→ data1

output

	key1	key2	key3
0	1	4	3
1	2	5	4
2	4	6	5
3	5	7	6
4	6	8	6

→ data2 = pd.Data-frame({'key1': [1, 2, 45, 63, 67]
'key4': [56, 5, 6, 7, 8],
'key5': [3, 56, 5, 6, 6]
})

→ data 2

output :

	Key 1	Key 4	Key 5
0	1	56	3
1	2	5	51
2	45	1	5
3	6	7	6
4	67	8	6

Note :

[Merge Operation]

(object to merge with)

how : {'left', 'right', 'outer', 'inner', 'cross'}, default: 'inner'

(Type of merge to be performed)

- left : use only keys from left frame.
- right : use only keys from right frame.
- outer : use union of keys from both frames.
- inner : use intersection of keys from both frames.
- cross : creates the cartesian product from both frames.

→ pd. merge (data1, data2) [# Perform 'inner' Merge operation]

output : # default how: 'inner' (follows the cond^n of inner)

	Key 1	Key 2	Key 3	Key 4	Key 5
0	1	4	3	56	3
1	2	5	4	5	56
2	6	8	5	7	6

→ pd. merge(data1, data2 , how = 'left')
[# Perform left merge operation]

output

	Key 1	Key 2	Key 3	Key 4	Key 5
0	1	4	3	56	3
1	2	5	4	5.0	56
2	4	6	5	NaN	NaN
3	5	7	6	NaN	NaN
4	6	8	6	7	6

Here, left means all the data will come from left frame.

If there is no key common in both dataframe [Data1, Data2] then, NaN will show.

↳ pd. merge (data1, data2, how = 'right')

output :

	Key 1	Key 2	Key 3	Key 4	Key 5
0	1	4.0	3.0	56	3
1	2	5.0	4.0	5	56
2	45	NaN	NaN	6	5
3	6	8.0	6.0	7	6
4	67	NaN	NaN	8	6

Here : 'right → Data will fetch from the right frame.

↳ pd. merge (data1, data2, how = 'outer')

output :

	Key 1	Key 2	Key 3	Key 4	Key 5
0	1	4	3	56	3
1	2	5	4	5	56
2	4	6	5	NaN	NaN
3	5	7	6	NaN	NaN
4	6	8	6	7	6
5	45	NaN	NaN	6	5
6	67	NaN	NaN	8	6

outer : Data fetched from both frames. ~~where~~ NaN will show where key doesn't match in both frames.

→ pd.merge (data1, data2, how='cross')

[Creates the cartesian product from both frames]

[Join operation].

↳ data1 = pd.DataFrame ({'Key1': [1, 2, 4, 5, 6],
'Key2': [4, 5, 6, 7, 8],
'Key3': [3, 4, 5, 6, 6],
}, index = ['a', 'b', 'c', 'd', 'e'])

By default index is 0, 1, 2, 3, 4

↳ Data2 = pd.DataFrame ({'Key4': [1, 2, 4, 5, 6],
'Key5': [4, 5, 6, 7, 8],
'Key6': [3, 4, 5, 6, 6]},
index = ['a', 'b', 'h', 'i', 'j'])

↳ Data1

output :

	Key1	Key2	Key3
a	1	4	3
b	2	5	4
c	4	6	5
d	5	7	6
e	6	8	6

↳ Data2

output :

	Key4	Key5	Key6
a	1	4	3
b	2	5	4
h	4	6	5
i	5	7	6
j	6	8	6

→ data1.join(data2)

output :

	Key1	Key2	Key3	Key4	Key5	Key6
a	1	4	3	1	4	3
b	2	5	4	2	5	4
c	4	6	5	NaN	NaN	NaN
d	5	7	6	NaN	NaN	NaN
e	6	8	6	NaN	NaN	NaN

NOTE :-

Join operation is similar to merge. however Join works on index while merge works on column. and in Join operation. by default it perform left join operation.

→ data1.join(data2, how='cross')

→ data1.join(data2, how='right')

→ data1.join(data2, how='outer').

↳ data = {"a" : [1, 2, 3, 4],
"b" : [4, 5, 6, 7]
"c" : ["pqr", "xyz", "abc", 'rst']}

→ df = pd. Dataframe (data].

→ df.

output :

	a	b	c
0	1	4	pqr
1	2	5	xyz
2	3	6	abc
3	4	7	rst

(by default Index)

↳ df. set_index ("a") [column 'a' become index]

output :

a	b	c
1	4	pqr
2	5	xyz
3	6	abc
4	7	rst

→ df. reset_index () [To reset the change]

→ df. reindex (['2', '1', '0', '3'])

~~→ df. reindex (['b', 'c', 'd', 'a'])~~

Output :

	a	b	c
2	3	6	abc
1	2	5	xyz
0	1	4	pqr
3	4	7	rst

→

[Numpy]

Numpy is a python library used for working with arrays., also has functions for working in domain of linear algebra, fourier transform, and matrices.

Numpy was created in 2005 by Travis Oliphant.

Numpy stand for → Numerical python.

#[Installation of Numpy :]

If python and PIP already installed on a system, then installation is Numpy is easy using command: `pip install numpy`

#[Import Numpy] :

By adding the 'import' keyword:

```
import numpy as np
```

Commands :-

→ import numpy as np.

→ l = [1, 2, 3, 4]

→ np. array (l)

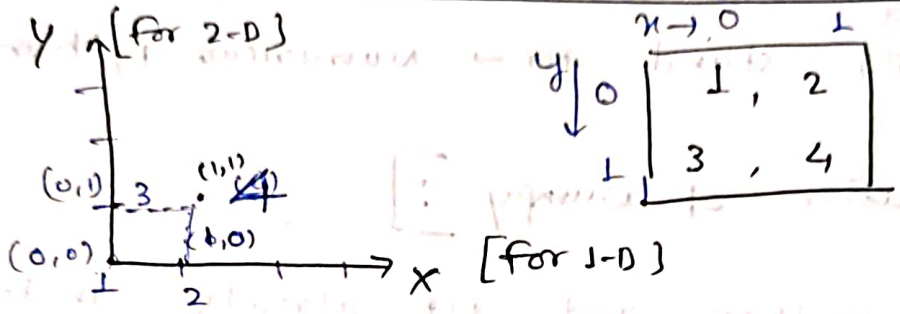
output : array ([1, 2, 3, 4])

↳ type (ar) output : numpy.ndarray
(n-dimensional)

↳ np.array([[1,2],[3,4]])

output : array([[1,2],
[3,4]])

There two brackets means two dimensional array



→ np.asarray(l) [To convert into ndarray]

→ a = [2,3,4]

→ np.asanyarray(a) [Convert list into array]

output : array([2,3,4])

→ b = np.matrix(l) [matrix subset of array, and matrix by default is 2-dimensional]

→ b [To convert list into array]

output : matrix([[1,2,3,4]])

→ np.asanyarray(b)

output : matrix([[1,2,3,4]])

Since b is already an array so it will remain same

→ a = np. array(l)

→ a

output : array ([1,2,3,4])

→ c = a (swallow copy)

→ c

output : array ([1,2,3,4])

→ c[0] [To access 0 index element]

→ c[0] = 100

→ c

output : array ([100, 2, 3, 4])

→ a

output : array ([100, 2, 3, 4])

→ d = np. copy (a)

→ d

output : array ([100, 2, 3, 4])

→ a

output : array ([100, 2, 3, 4])

→ a[1] = 400

→ a

output : array ([100, 400, 3, 4])

→ d

output : array ([100, 2, 3, 4]) (No change in d)

→ np. fromfunction (lambda i, j : i==j , (3,3))

- i, j, are two parameter
- i==j is a condition. If condn matched then
- It will True unless false.

(3,3) → matrices 3X3, means. 3 rows and 3 columns.

output :

```
                i→ 0     1     2
array ([[0[ True,  False, False],
        j|  1[False, True,  False],
         ↓  2[False, False, True]])
```

→ np. fromfunction (lambda i, j : i*j , (3,3))

output :

```
array ([[[ 0 , 0 , 0 ],
         [ 0 , 1 , 2 ],
         [ 0 , 2 , 4 ]]])
```

fromfunction():

fromfunction() is a function which helps in terms of generating and producing the arrays in any point of time.

→ a = (i*i for i in range(5))

→ np.fromiter(a, float)

output :
array([0., 1., 4., 9., 16.])

→ np.fromstring('234 234', sep=' ') [To separate string and convert into array]

output :
array([234., 234.])

Numpy - Data Types :

↳ L = [2, 3, 4, 5, 6]

↳ ar = np.array(L) [To convert list into array].

↳ ar [to show the array]

output : array([2, 3, 4, 5, 6])

↳ ar.ndim [To check the dimension of array]

↳ ar2 = np.array([[1, 2, 3, 4], [2, 3, 4, 5]])

↳ ar2.ndim

↳ ar.size [To show the elements available in the array].

↳ ar2.size[ ]

→ ar.shape

output : (5,)

→ ar2.shape.

output : (2, 4) i.e. 2 - rows
4 - columns

→ ar.dtype (To find data type)

→ ar3 = np.array([(1.4, 45, 45), (23, 45, 66)])

→ ar3

output array([[1.4, 45, 45,]
[23, 45, 66]])

→ ar3.dtype

→ range(5) (produce data from 0 to 5 except 5)

output : range(0,5)

→ list(range(5))

output : [0, 1, 2, 3, 4]

→ np.arange(2.3, 5.6) (Produce float data from 2.3 to 5.6)

output : array([2.3, 3.3, 4.3, 5.3])

→ np.arange(2.3, 5.6, .3) [If .3 jump Krna Hai]

→ list (np.arange(2.3, 5.6, .3)) [To convert array to list]

→ np.linspace (1, 5, 10) [To produce 10 data inbetween 1 to 5 scale]

output :

```
array ([ 1 , 1.44444 , 1.888889 , 2.33333 , 2.77778 ,
         3.2222 , 3.666667 , 4.111111 , 4.555556 , 5.  ])
```

→ np.zeros (5) [Produce an array having zero values in it]

```
output : array ([ 0 , 0 , 0 , 0 , 0 ])
```

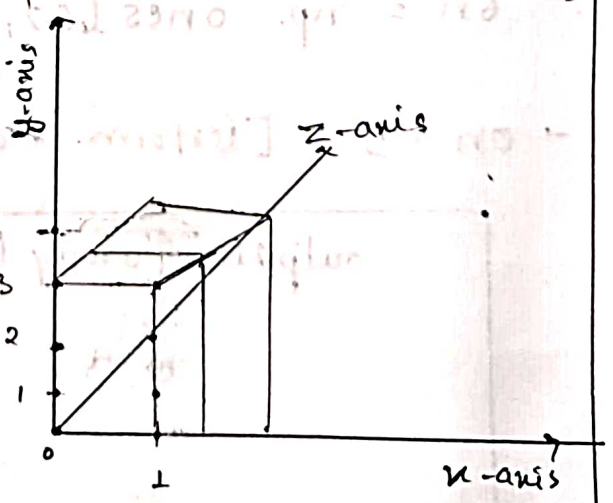
→ np.zeroes ((3, 4)) [3 - rows, 4 columns] (2-D)

```
output : array ([[ [ 0 , 0 , 0 , 0 ],
                  [ 0 , 0 , 0 , 0 ],
                  [ 0 , 0 , 0 , 0 ] ])
```

→ np.zeroes ((3, 4, 2)) [Produce 4X2, 3 such kind of matrices].
(array) (rows) (columns)

```
output : ([[[0[ 0 , 0 ],
            1[ 0 , 0 ],
            2[ 0 , 0 ],
            3[ 0 , 0 ]],
           [[ 0 , 0 ],
            [ 0 , 0 ],
            [ 0 , 0 ],
            [ 0 , 0 ]],
           [[ 0 , 0 ],
            [ 0 , 0 ],
            [ 0 , 0 ],
            [ 0 , 0 ]]])
```

(Representation in a 3-D co-ordinate)



→ ar4 = np. zeroes ((3,4,2,3)) [To produce 3 sets of 4 matrices with 2×3].

→ ar4.ndim [Tells the dimension]

→ np.ones(4).

output : array ([1, 1, 1, 1])

→ np.ones((2,3)) [Produce 2×3 matrix] (2-dimen. Matrix)

output : array ([[1, 1, 1],
[1, 1, 1]])

→ np.ones((2,3,2)) [Produce 3×2 matric]
(Type, rows, column)

output : array ([[[1, 1],
[1, 1],
[1, 1]],
[[1, 1],
[1, 1],
[1, 1]]])
→ (3-bracket 3-dim.)

→ on = np.ones((2,3,2)) [To store in a variable]

→ on + 5 [Perform arithmetical operation].

output : array ([[[6, 6],
[6, 6],
[6, 6]],
[[6, 6],
[6, 6],
[6, 6]]])

→ on *4

→ np. empty ((3,6))

```
output : array([[ 0 , 0, 0, 1, 1, 1 ],
                [ 2 , 2 , 2, 0, 1, 2 ],
                [ 0 , 1, 2, 0, 1, 2 ]])
```

→ np. eye (4) [To create an identity matrix]

```
output : array([[[1, 0, 0, 0],
                 [ 0, 1, 0, 0],
                 [0, 0, 1, 0],
                 [0, 0, 0, 1]]])
```

→ np. linspace (2, 4, 20) [To produce 20 data between 2 to 4]

→ np. logspace (2, 5, 10) [To find log of 10 data b/w 2 to 5] (by default log base = 10)

→ np. logspace (2, 5, 10, base = 2)

→ np. random. randn (3, 4) [To generate a random data of 3 X 4 matrices]. (3 rows, 4 columns)

→ np. random. rand (3, 4) [Also generate random data of 3 X 4 matrices]

→ np.random.randint(1, 110, (3, 4)) (3 rows, 4 column)

(Range b/w 1 to 110)

output : array([[47, 105, 60, 19],
[10, 82, 4, 92],
[12, 81, 14, 68]])

→ pd.DataFrame(np.random.randint(1,110,(3,4)))
[To convert into dataframe]

→ arr = np.random.rand(3,4) [To create random array of 3 rows & 4 columns]

→ arr. [To show]

→ arr.reshape(6,2) [It will rearrange the above data/array]
[But should be multiple of above data].

→ [slicing concept]

→ arr[1] [for 1st index row]

→ arr[1][1]
(column index)
row index

→ arr[2:5] [row from 2nd index row to 5th index row]

→ arr[2:5][1] (column index)

→ arr = np.random.randint(1, 50, (5,5))

[To create random data, Range from 1 to 50 and (5 rows & 5 column) 5×5 matrices]

→ arr

output: arr([[20, 30, 40, 15, 10],
[10, 15, 20, 25, 30],
[5, 10, 8, 18, 19],
[10, 15, 20, 21, 22],
[25, 28, 39, 49, 50]])

→ arr > 20 [Conditional query; output will shown in the form of True & false].

{If arr > 20, then, True will appear unless false}.

→ arr[arr > 20]

output: array([30, 40, 25, 30, 21, 22, 25, 28, 39, 49, 50])

→ arr

→ arr [2:4, [1, 2]] [slice data from 2nd & 3rd index row to 1st and 2nd index column]

output : array([[10, 8],
[15, 20]])

(row index)

→ arr [0][0] = 28 [Manipulate data through indexing]

(column index)

→ arr [To show]

→ arr1 = np. random. randint (1,3, (3,3))

[create random data b/w 1 and 3 of 3 x 3 matrices]

→ arr2 = np. random. randint (1,3 , (3,3))

→ arr1.

output : array ([[[2, 2, 1],
[2, 1, 2],
[1, 2, 1]]])

→ arr2

```
output : array([[1, 2, 2],
                [2, 1, 1],
                [1, 1, 1]])
```

→ arr1 + arr2 [Arthmetic operation]

→ arr1 - arr2

→ arr1 / arr2

→ arr1 * arr2 [* for inder multiplication]

```
output : array([[2, 4, 2],
                [4, 1, 2],
                [1, 2, 1]])
```

→ arr1 @ arr2 [@ is for matrix multiplication]

```
output : array([[7, 7, 7],
                [6, 7, 7],
                [6, 5, 5]])
```

→ arr1 + 100 [100 will be add in arr1]

→ arr1 ** 2

{Numpy - Broadcasting}

→ arr = np.zeros((4,4)) [Array of 4×4 matrices, every element is zero].

→ arr

```
output : array([[[0,0,0,0],
                 [0,0,0,0],
                 [0,0,0,0],
                 [0,0,0,0]]])
```

→ row = np.array([1,2,3,4])

→ row

```
output : array([1,2,3,4])
```

→ arr + row [Row wise addition] [which is called Broadcasting]

```
output : array([[1,2,3,4],
                [1,2,3,4],
                [1,2,3,4],
                [1,2,3,4]])
```

→ row.T (Transpose) [output will remain same since it only has 1 row]

```
output ([1,2,3,4])
```

→ col = np.array([[[1,2,3,4]]])

→ col.T

```
array([[1],
       [2],
       [3],
       [4]])
```

→ col

output : array([[[1,2,3,4]]])

→ col.T + arr [This time Broadcasting will occur column wise]

→ arr1 = np.random.randint(1,4,(3,4))

→ arr1

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

→ np.sqrt(arr1) [square root of every element in arr1]

→ np.exp(arr1) [To find exponential]

→ np.log10(arr)

[Numpy - Array Manipulation]

→ arr = np.random.randint(1,10,(3,4))

→ arr.

```
array([[4, 4, 8, 6],      # Total 12 no. of data
       [8, 2, 5, 8],
       [5, 2, 8, 9],
```

→ arr.reshape(6,2) [Rearrange into 6 rows and 2 columns]

→ arr.reshape(2,7) [error will show. since data should be same. Here is 2×7 ⇒ 14 data]

→ arr.T {Rotate the actual array}.

→ ~~arr.flat~~

→ arr.flatten() {flatten, which convert 2 dim. array into one dimension}

```
array([4, 4, 8, 6, 8, 2, 5, 8, 5, 2, 8, 9])
```

⓪ ① ② ③ ④

→ arr1 = np.array([1, 2, 3, 3, 4])

→ arr1.ndim [To show the dim]

→ np.expand_dims(arr1, axis=1)

[expand_dims converts one dim. into two dimension;
axis=1 means along the column]

```
array([[1],
       [2],
       [3],
       [4]])
```

→ np.expand_dims(arr1, axis=0) [expand along row]

```
array([[1, 2, 3, 4]])
```

→ arr1.

```
array([1, 2, 3, 3, 4])  # one-dim. array
```

→ np.repeat(arr1, 2)

```
array([1, 1, 2, 2, 3, 3, 3, 3, 4, 4])
```

→ np.roll(arr1, 3) [It will shift the element (shift in the interval of 3)]

```
array([3, 3, 4, 1, 2])
      ⓪ ① ② ③ ④ ← (indexes)
```

→ np. diag (arr1) [Put every element of arr1 in the diagonal]

```
array ([[[ 1, 0, 0, 0, 0],
         [0, 2, 0., 0, 0],
         [0, 0, 3, 0, 0],
         [0, 0, 0, 3, 0],
         [0, 0, 0, 0, 4]]])
```

#{Numpy - Binary Operators}

→ arr1 = np. random. randint (1, 10, (3, 4))

→ arr2 = np. random. randint (1, 10, (3, 4))

→ arr1 + arr2 [Index wise operation]

→ arr1 * arr2 [Index wise multiplication operation]

→ arr1 / arr2 [Division op.]

→ arr1 - arr2 [substraction op.]

→ arr1 % arr2 [Modulus op.]

→ arr1 ** arr2 [power operation]

→ arr1 & arr2 [Binary AND (&) operation b/w arr1 and arr2]

→ arr1 | arr2 [OR operation]

→ arr1 > arr2 [conditional query]

{# If fullfill the condn then True will apear Unless false}

[# Numpy - String functions]

→ arr = np.array (['boy', 'girl'])

→ np.char.upper (arr) [convert strings in upperxase]

→ np.char.title (arr) [convert into title] ~~means~~

```
array (['Boy', 'Girl'])
```

→ np.char.capitalize (arr)

[# Numpy - Mathematical functions]

→ arr1

→ np.sin (arr1)

→ np.cos (arr1)

→ np.tanh(arr1)

→ np.log10 (arr1)

→ np.exp(arr1)

→ np.sqrt(arr1)

→ np.power(arr1, 2)

→ np.mean (arr1)

→ np.median (arr1)

→ np.std (arr1)
{# standard deviation)

→ np.var(arr1)
(# variances)

[# Sort, Search & Counting functions.]

→ arr = np.array([4,3,4,5,48,90])

→ arr

→ np.sort(arr) [sort data in ascending order]

→ arr1 = np.array([1,2,0,4,0,0,9])

→ np.count_nonzero(arr1) [will show How many non-zero data is available]

→ np.where(arr>6) [It will show the index location according to the condition]

output : (array([4,5]),)
index

→ np.extract(arr>6, arr) [To fetch the data out A/c to the condn from the array]

output :
array([48,90])

~~# Numpy.~~

{# Numpy - Byte Swapping}

→ arr.byteswap() [# convert elements into byte]

{# Numpy - Copies & Views}

⇒ The changes made to the copy, will not affect original array, and any changes made to the original array will not affect the copy.

⇒ Any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

→ arr = np.array([1,2,3,4,5])

→ X = arr.copy()

→ arr[0] = 23

→ arr

output : array([23, 2, 3, 4, 5])

→ X

output array([1, 2, 3, 4, 5])

→ arr = np. array ([1,2,3,4,5])

→ y = arr. view()

→ arr[0] = 33

→ arr

output : array ([33,2,3,4,5])

→ y

output : array ([33,2,3,4,5])

Numpy - Matrix Library

→ import numpy. matlib as nm

→ nm. zeros (5) [will generate matrix]

output : matrix ([[0, 0, 0, 0, 0]])

→ nm. ones ((3,4))
(rows)
(column)

matrix ([[1, 1, 1, 1],
[1, 1, 1, 1],
[1, 1, 1, 1]])
→ 2 dimension

→ `hm.eye(5)` [Generate identity matrix].

```
matrix ([[1, 0, 0, 0, 0],
         [0, 1, 0, 0, 0],
         [0, 0, 1, 0, 0],
         [0, 0, 0, 1, 0],
         [0, 0, 0, 0, 1]])
```

{# Linear Algebra ← Numpy}

→ `arr1 = np.random.randint([[2,3], [4,5]])`

→ `arr2 = np.random.randint([[5,3], [2,5]])`

→ `np.dot(arr1, arr2)`

```
array([[ 3, 2],
       [6, 4]])
```

→ `arr1 @ arr2` [@ for Matrix Multiplication].

```
array([[3, 2],
       [6, 4]])
```

Matplotlib

⇒ Matplotlib is a popular plotting library in python used for creating high-quality visualizations and graphs. It was originally developed by John D. Hunter in 2003

⇒ It is a versatile and powerful library for creating high-quality plots and visualizations in python.

Advantages of matplotlib	disddvantages
• **Versatility**: can create a wide range of plots	**Steep learning Curve**: sometimes complex syntax
• **Customization**: offers extensive customization options to control every aspect of the plot	**Verbose syntax**: less intuitive compared to other plotting libraries like seaborn or Plotly.
• **Integration with Numpy**	**Default asthetics**
• **Extensible**	**Limited Interactivity**
• **Interactive Plots**	**Dependency on External libraries:**

{# Import Matplotlib}

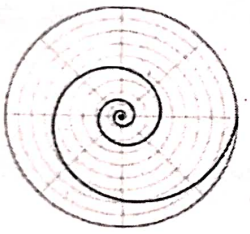
Using command → `import matplotlib.pyplot as plt`

⇒ import numpy by using command:

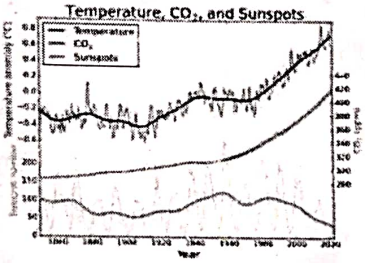
`import numpy as np`

Pyplot : Most of Matplotlib utilis lies under the pyplot submodule, and are usually imported under plt alias:

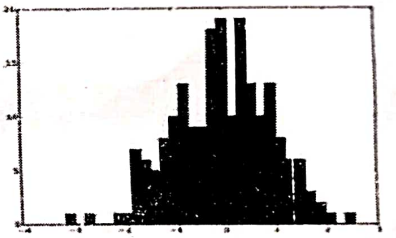
[There are following plots in Matplotlib]



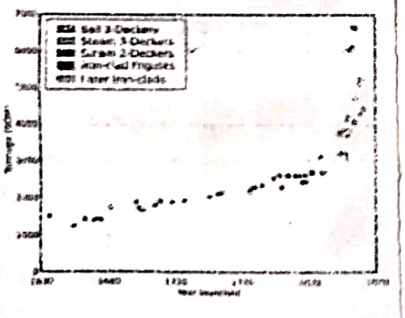
Polar plot



Line plot



Histogram



Scatter plot

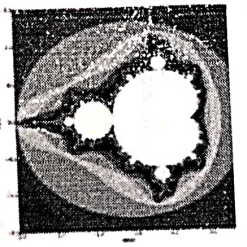
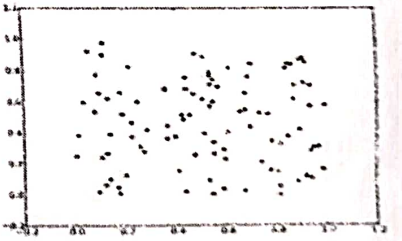


Image plot



Scatter plot

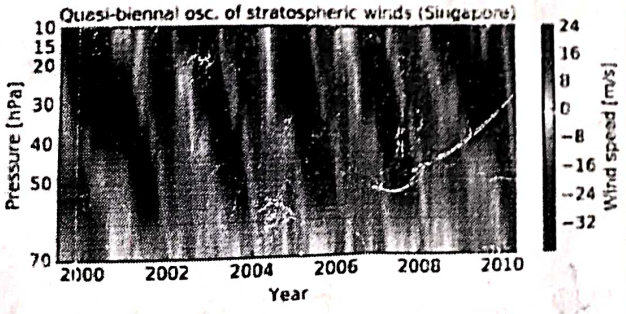
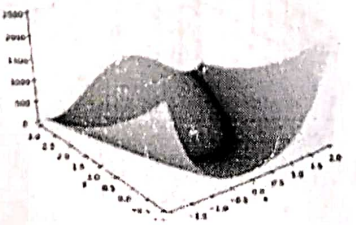
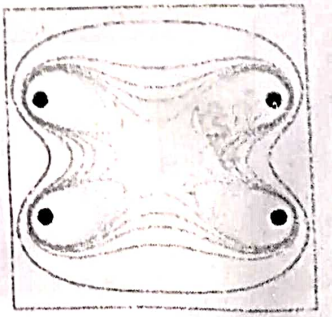


Image plot



3-D plot



Contour plot

Commands for generating and plotting data through matplotlib.

→ import matplotlib.pyplot as plt

→ import numpy as np

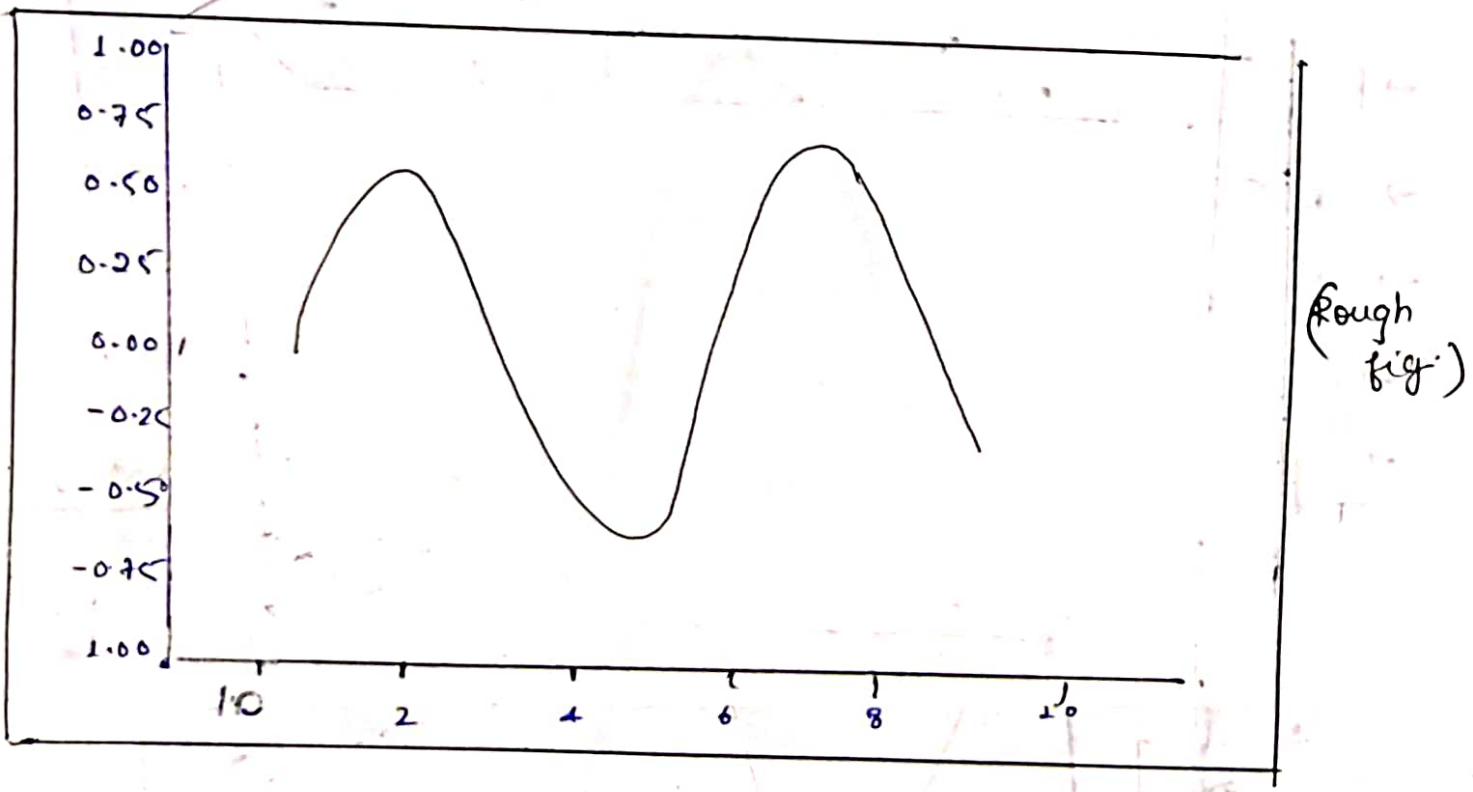
→ x = np.linespace (0, 10, 200) [To generate 200 data from 0 to 10]

→ x [To show the data]

→ y = np.sin(x) [To generate y with the help of sin function]

→ y [To show]

→ plt.plot(x,y)



(Rough fig.)

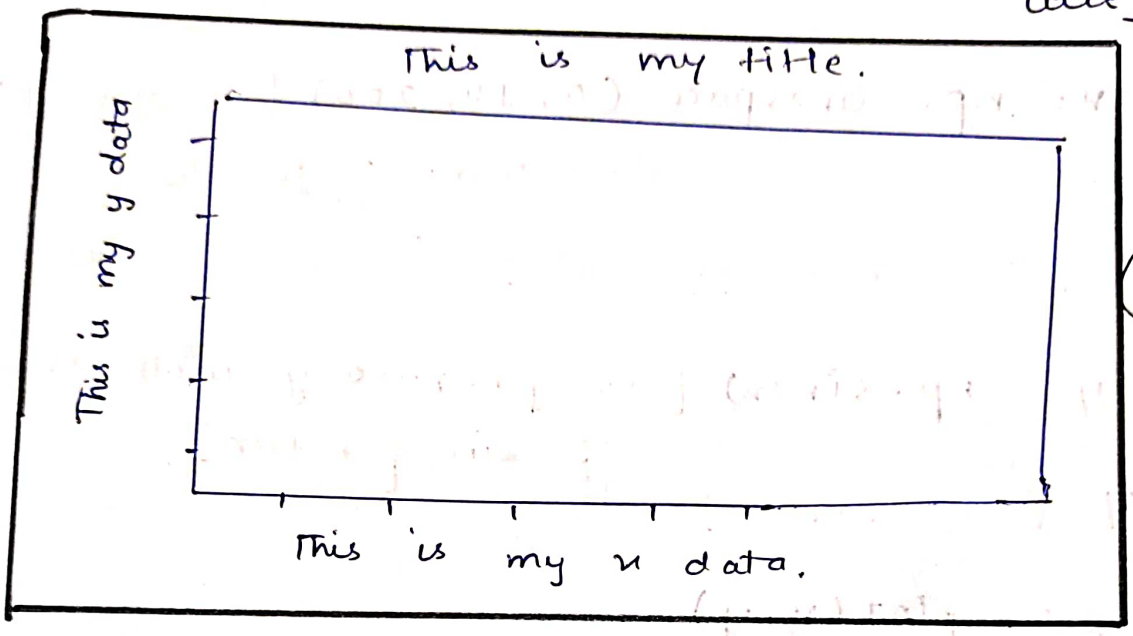
sin ∯ <sin wave data>

→ plt. xlabel ("this is my x data")

→ plt. ylabel ("this is my y data")

↳ [for labeling x and y-axis]

→ plt. title ("this is my title") [for labelling title]



(Rough fig.)

→ plt. plot (x, y) [Call]

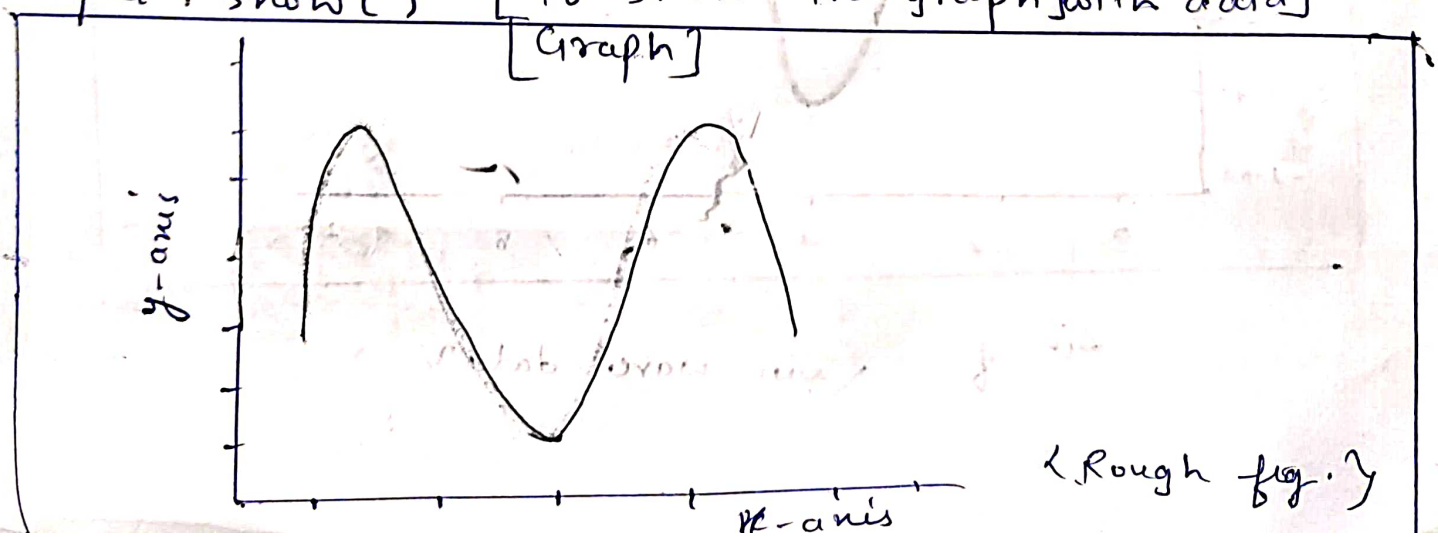
→ plt. xlabel ("x-axis")

→ plt. ylabel ("y-axis")

→ plt. title ("Graph")

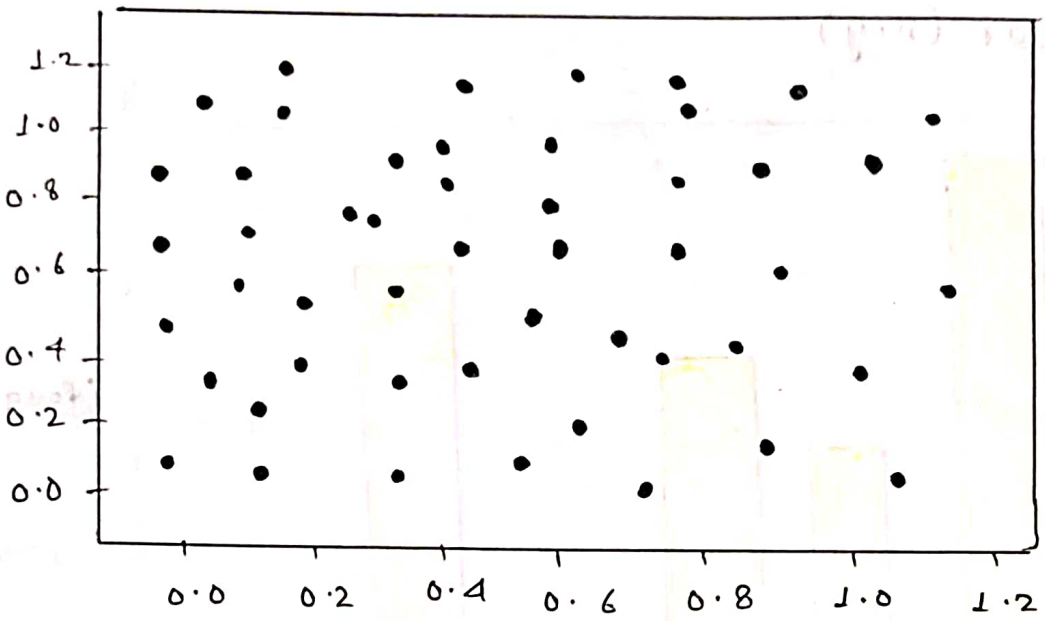
→ plt. show() [To show the graph with data]

[Graph]



(Rough fig.)

→ x = np. random. rand(50) } [Generate 50 random data]
y = np. random. rand(50)
plt. scatter (x, y)



{# The data above is completely distributive}
(Rough fig.)

→ colours = np. random. rand(50)

→ sizes = 1000 * np. random. rand(50)

→ plt. scatter (x, y, c = colours, s = sizes, alpha=.5)
[alpha determines the intensity of the data in graph, c and s are the parameters]

→ plt. xlabel ("x-axis")

→ plt. ylabel ("y-axis")

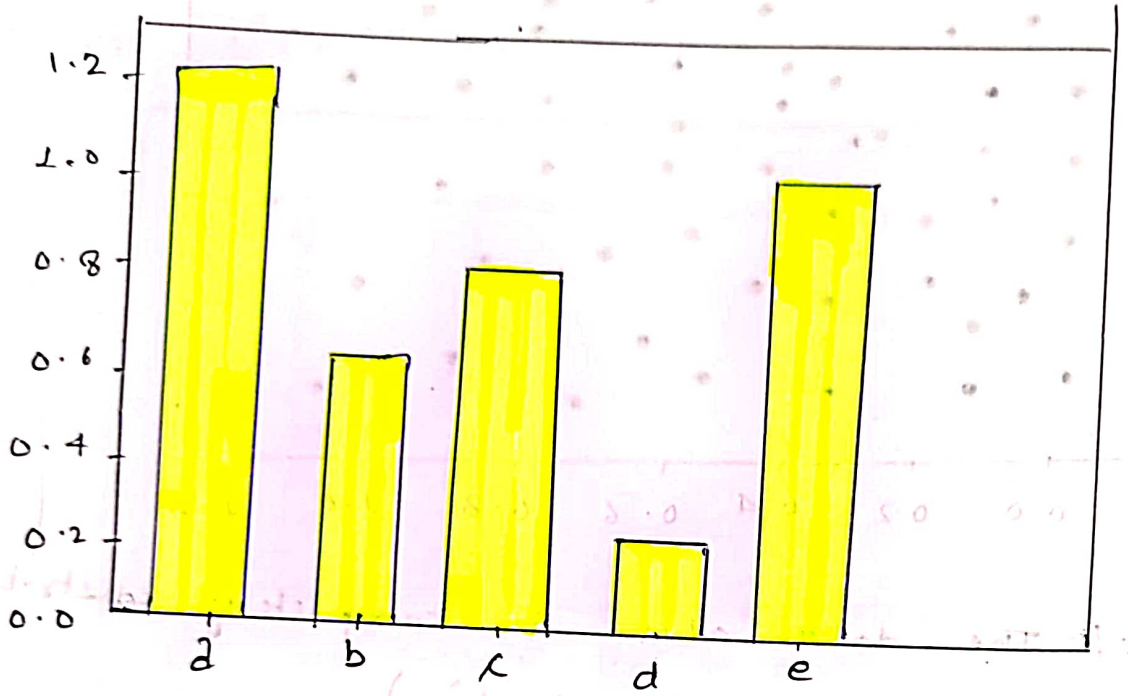
→ plt. title ("random data")

Graph b/w ~~cat~~ catagorical V/S Numerical

→ x = ["a", 'b', 'c', 'd', 'e'] [To generate 5 catagorical data]

→ y = np.random.randn(5) [To generate 5 numerical data].

→ plt.bar(x,y)



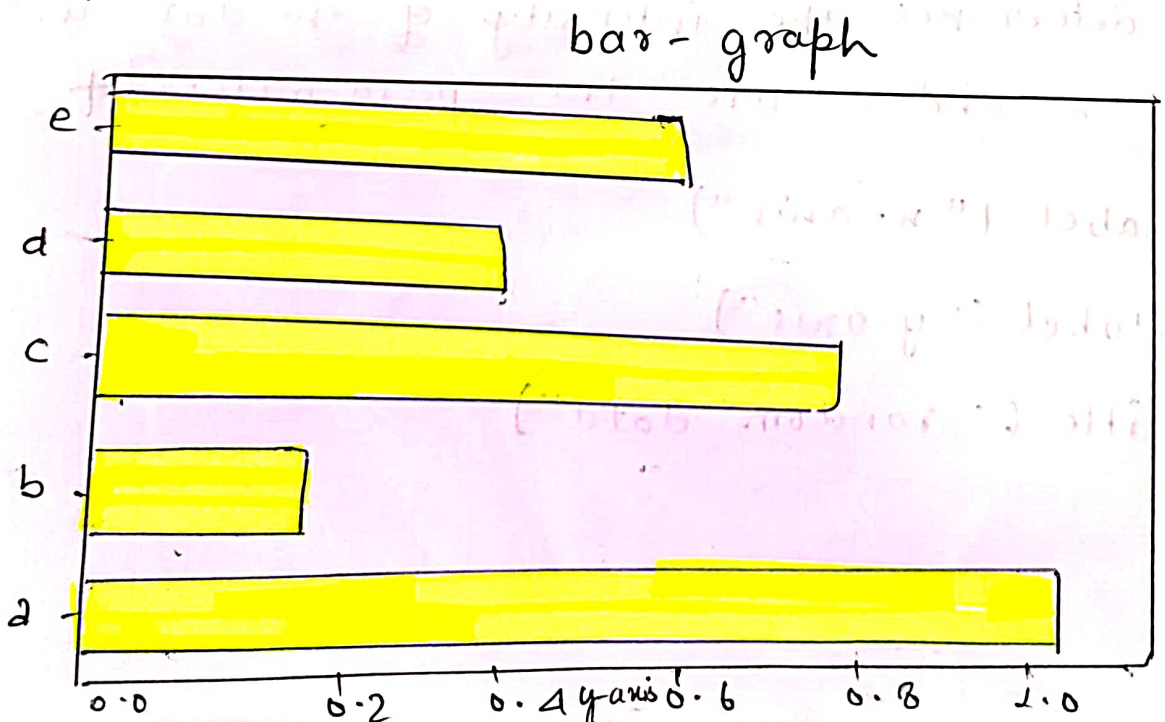
(Rough fig.)

→ plt.barh(x,y) [For Horizontal bar plot/graph]

→ plt.xlabel("x-axis")

→ plt.ylabel("y-axis")

→ plt.title("bar-graph")



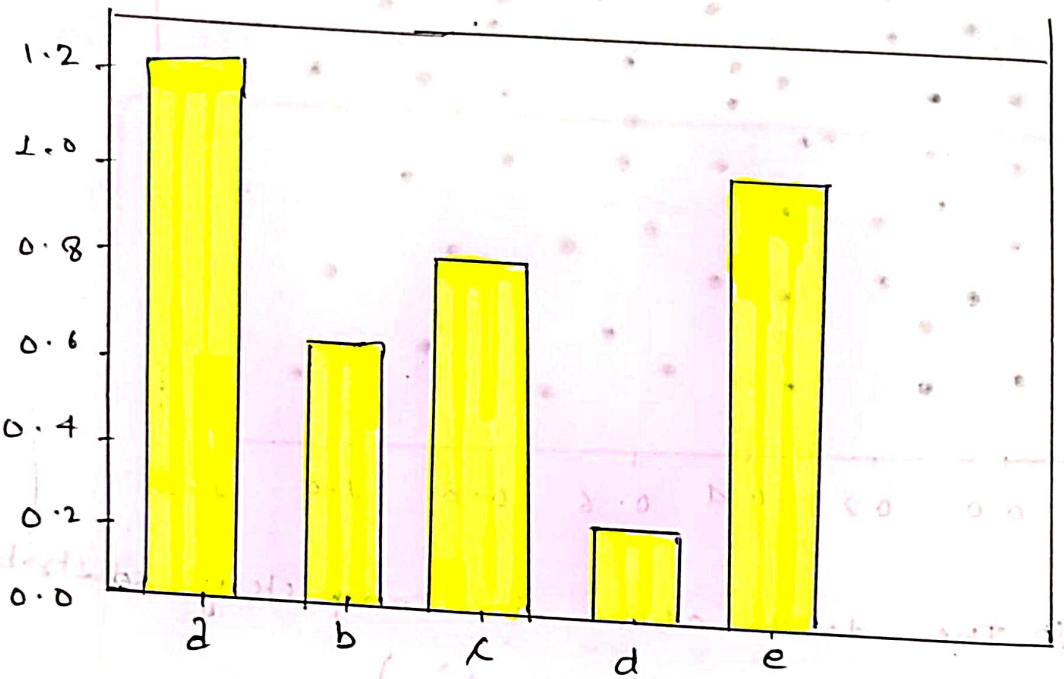
(Rough fig.)

Graph b/w ~~bal~~ catagorical V/S Numerical data

→ x = ["a", 'b', 'c', 'd', 'e'] [To generate 5 catagorical data]

→ y = np.random.rand(5) [To generate 5 numerical data].

→ plt.bar(x,y)



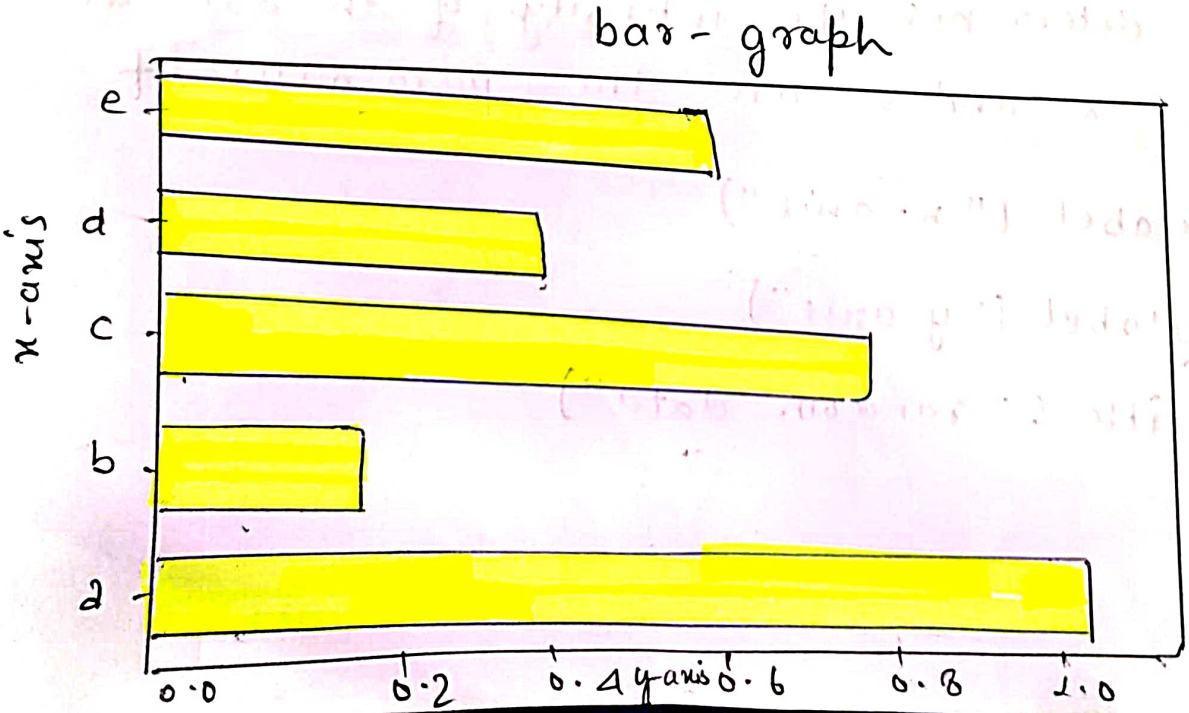
(Rough fig.)

→ plt.barh(x,y) [for Horizontal bar plot/graph]

→ plt.xlabel("x-axis")

→ plt.ylabel("y-axis")

→ plt.title("bar-graph")

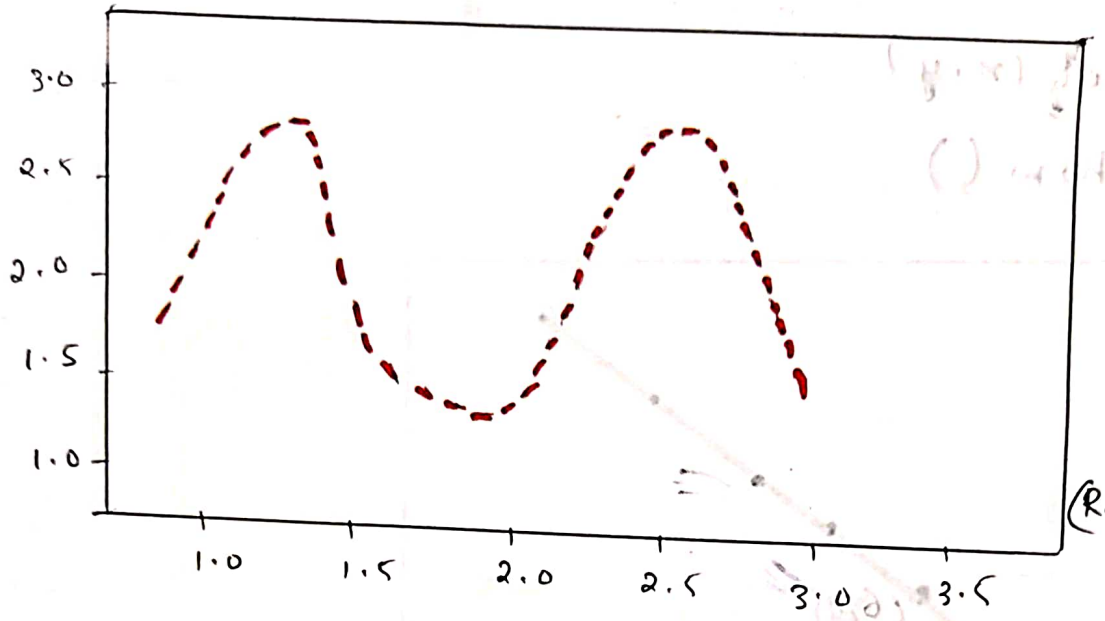


(Rough fig.)

→ plt. figure (figsize = (6,2)) [for fig. size]

→ plt . plot (x, y, '--r') [r is red in colour with -- design

→ plt. show ()

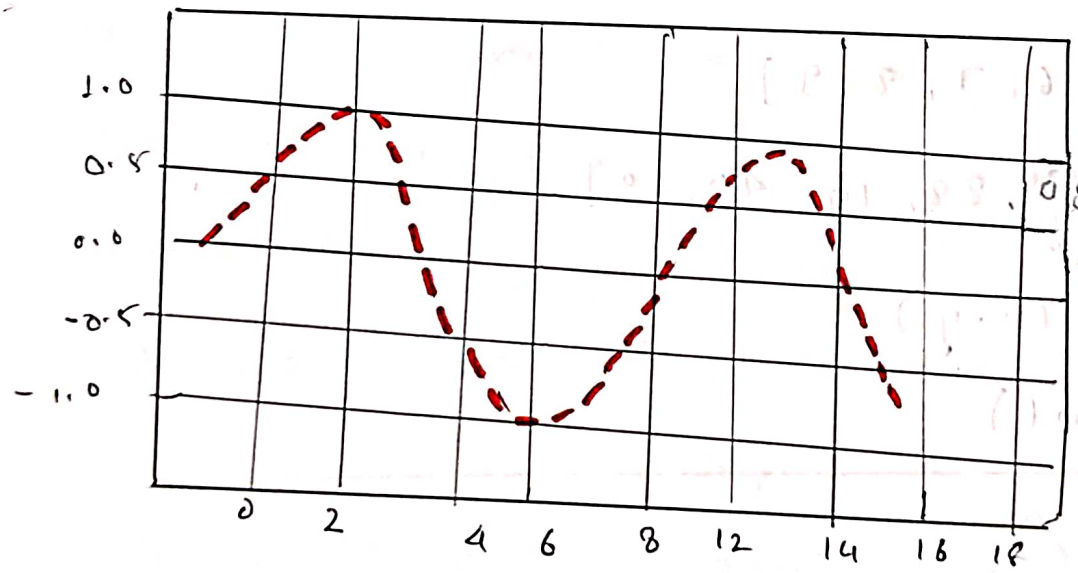


(Rough fig.)

→ plt . plot (x, y, '--r')

→ plt. grid () {for adding grid lines}

→ plt. show()



(Rough figure)

→ plt. grid (axis = 'x') [Display grid-lines only for x-axis]

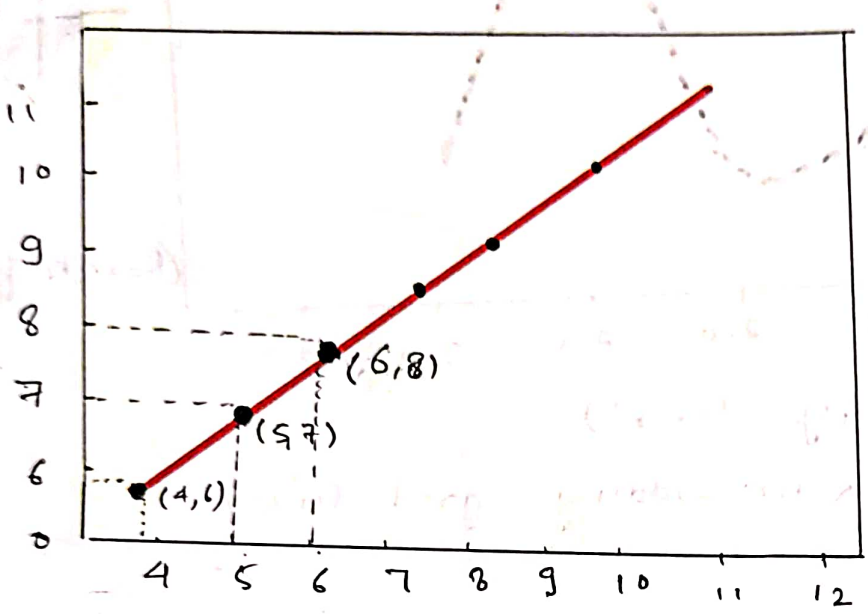
→ ~~plt~~ plt. grid (axis = 'y') [Display grid lines only for y-axis]

→ x : [4, 5, 6, 7, 8, 9]

→ y = [6, 7, 8, 9, 10, 11].

→ plt. plot (x, y)

→ plt. show ()



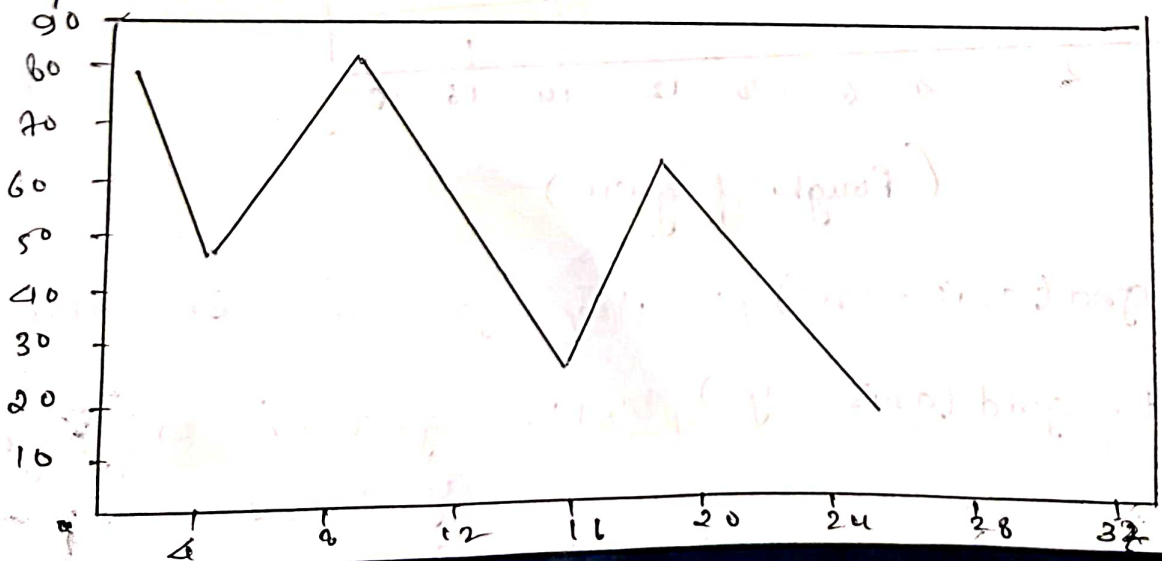
(Rough fig.)

→ x = [4, 5, 6, 7, 8, 9]

→ y = [60, 30, 88, 19, 40, 10]

→ plt. plot (x, y)

→ plt. show ()

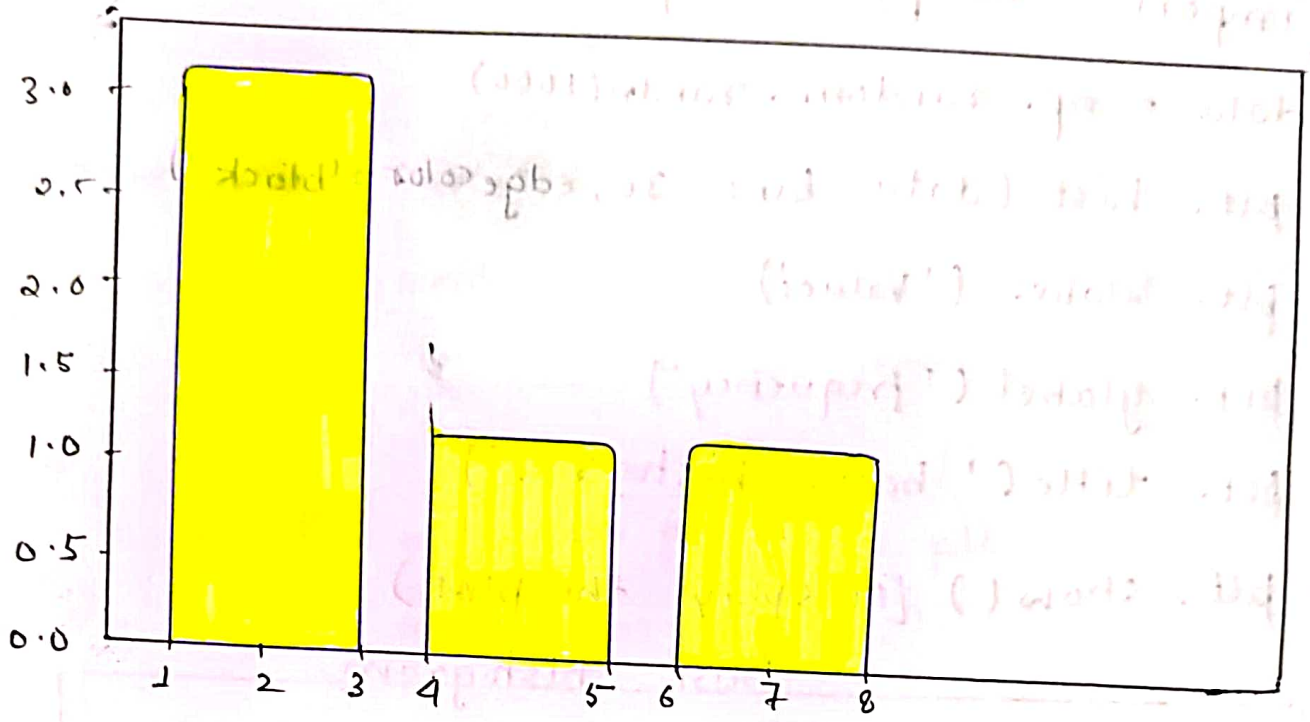


(Rough fig.)

→ data [1, 2, 3, 4, 5, 1, 2, 3, 6, 7, 8, 1, 2, 3]

→ plt. hist(data) [hist → histogram]

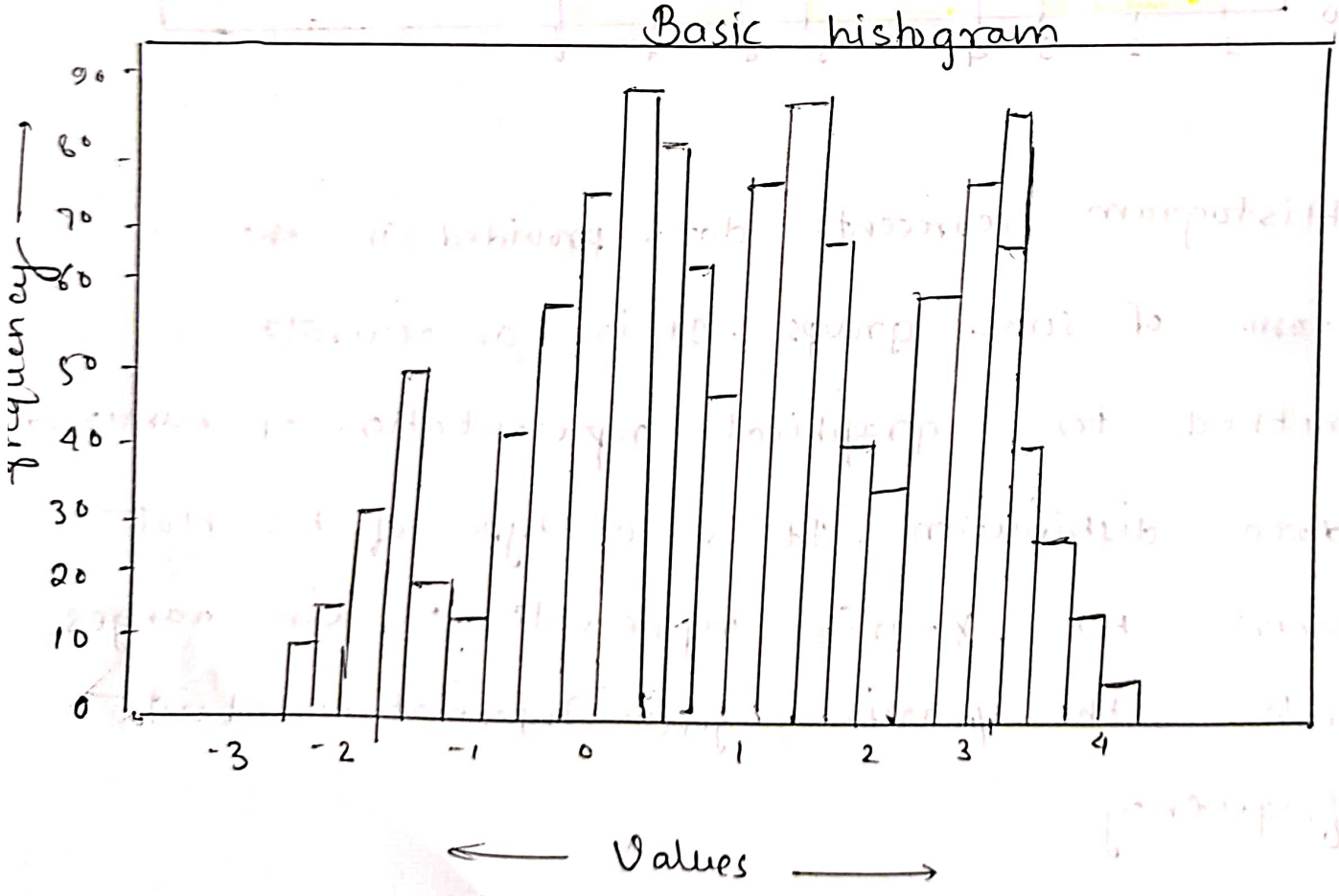
→ plt.show ()



Histogram : represents data provided in the form of some groups. It is an accurate method for graphical representation of numerical data distribution. It is a type of bar-plot where the X-axis represents the bin ranges while the Y-axis gives information about frequency

Create a Basic Histogram in Matplotlib.

```
→ import matplotlib.pyplot as plt
→ import numpy as np
→ data = np.random.randn(1000)
→ plt.hist(data, bins=30, edgecolor='black')
→ plt.xlabel('Values')
→ plt.ylabel('Frequency')
→ plt.title('basic histogram')
→ plt.show() [#Display the plot)
```



Matplotlib Subplot

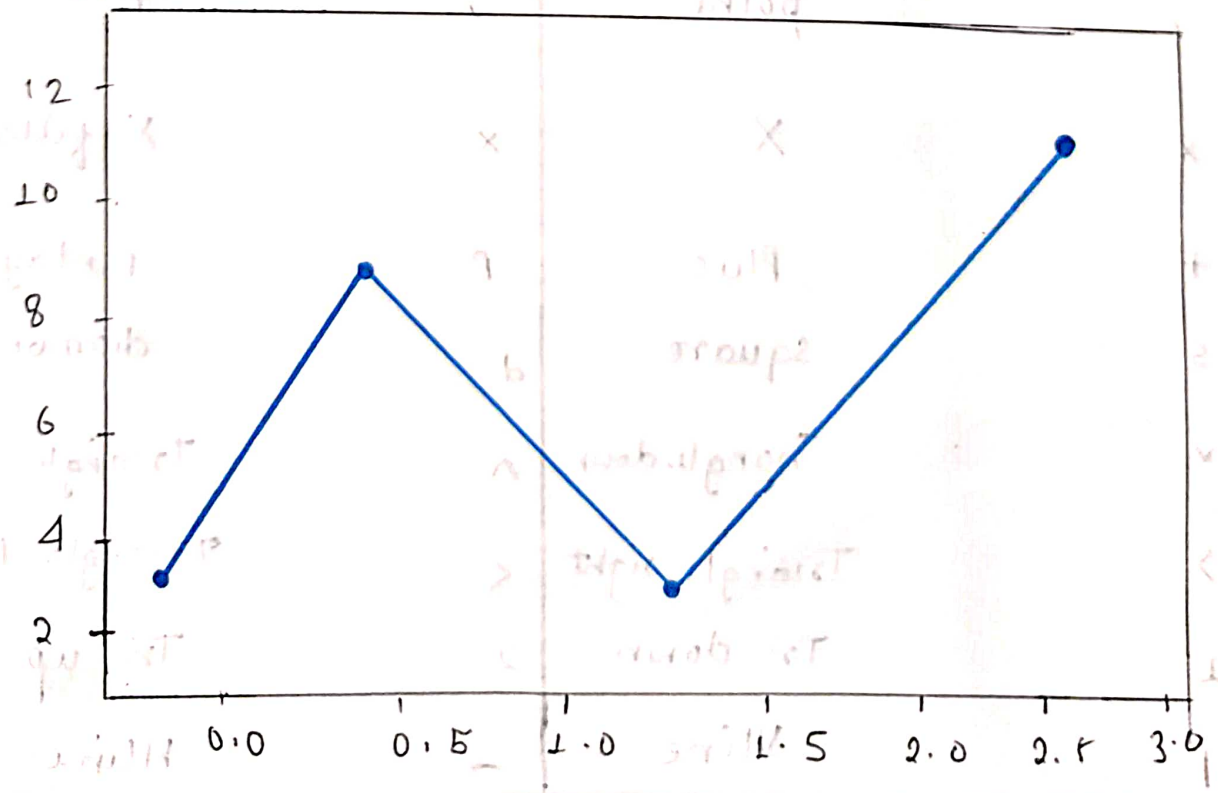
With subplot() function we can draw multiple plots in one figure.

Matplotlib Markers.

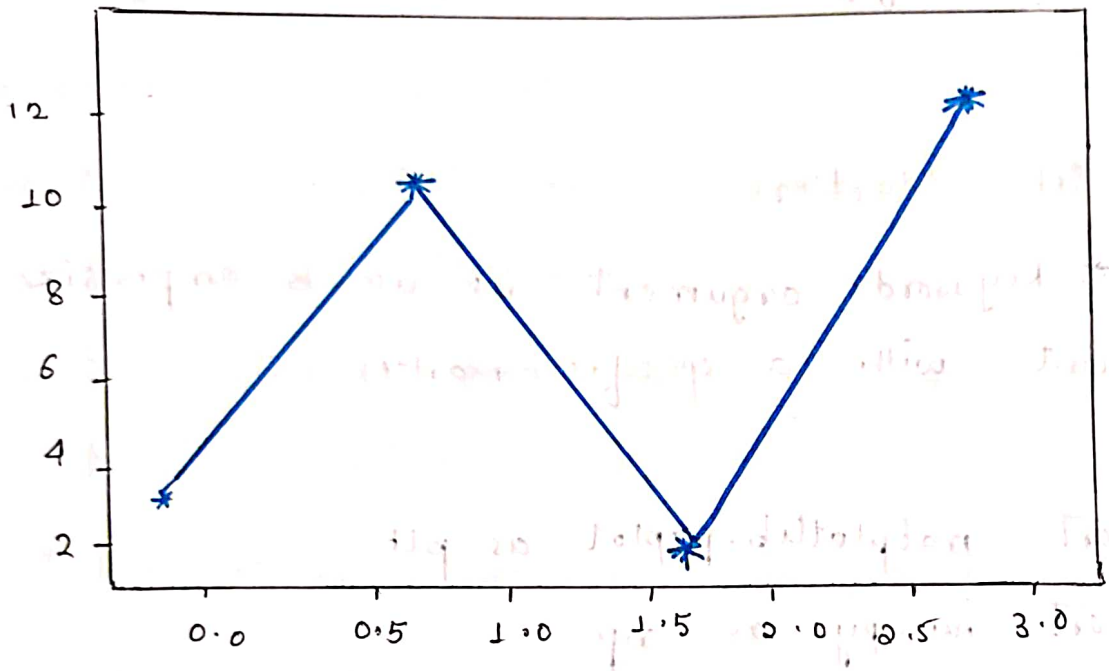
marker Keyword argument is use to emphasize each point with a specific marker.

E.g :

```
→ import matplotlib.pyplot as plt.
→ import numpy as np
→ ypoints = np.array([3,8,1,10])
→ plt.plot(ypoints, marker = 'o')
→ plt.show()
```



```
...
plt.plot(ypoints, marker = '*')
...
```



There are many kinds of markes:

Marker	Description	Marker	Description
o	circle	*	Star
.	point	,	pixel
x	X	x	X (filled)
+	Plus	p	Pentagon
s	square	d	diamond (thin)
v	Triangle down	^	Triangle up
>	Triangle right	<	Triangle left
1	Tri down	2	Tri up
|	Vline	_	Hline