**NumPy:**

**1.Introduction to NumPy:**

NumPy is an abbreviation for Numerical Python. It’s basically used for scientific computing in Python and has built in math library’s highly useful for making numerical computation which contains Statistics, Linear and Non-Linear equations. At the core of NumPy is the **ndarray**, which stands for n-dimension array.

# There are several ways to create ndarray in NumPy, below given:

# 1.Using regular Python lists.

# 2.Using built-in NumPy function. (We are using this approach)

**2. Working with NumPy:**

# 1. Creating and Saving NumPy ndarray:

# a) Importing NumPy:

# Code:

#importing pandas liberary and naming as pd to use

import pandas as pd

# b) Creating 1-Dimensional(1D) ndarray and Creating 2-Dimensional(2D) ndarray:

# Code:

# We create a 1D ndarray that contains only integers  
list1 = np.array([1, 2, 3, 4, 5])  
#We create a 2-D ndarray  
List2 = np.array([[6,7,8,9,1]])  
  
# We print list1 and list2  
print()  
print('list1 = ', list1)  
print()  
print(‘list2 =’, list2)  
print()

c) Using Built-in Functions on ndarray:

List of Built-in Functions and working:

* np.array()
* np.shape
* np.size
* np.dtype
* type()
* np.zeros(shape)
* np.ones(shape)
* np.arange(start,stop,size)
* np.eye(N)

All the above functions are used in below code along with there uses mentioned in comments:

**Code:**

# We import NumPy into Python  
import numpy as np  
# We create a rank 2 ndarray that only contains integers  
list1 = [1,2,3,4,5]  
list2 = [6,7,8,9,10]  
array1 = np.array([list1,list2])  
  
# We print list1  
print()  
print('list1 = \n', list1)  
print()  
#We print list2  
print()  
print('list2 = \n', list2)  
print()  
  
# We print information about both-lists  
  
print('Both lists has dimensions:', array1.shape)  
  
#The [np.shape] :method print Size along each of list's dimension  
  
print()  
  
print('Both list has a total of', array1.size, 'elements')  
  
#[np.size] : method tells about the total number of elements of the array

print()  
print('list2 is an object of type:', type(array1))  
  
#[type] : This function tells us that both lists are actually are NumPy ndarray  
  
  
  
print()  
print('The elements in array lists(list1 and 2) are of type:', array1.dtype)  
  
#[dtype] : This attribute tells us that the elements of list 1 and list2 are stored  
#in memory as signed 64-bit integers  
  
  
  
print()  
  
# We create a (3 x 4) ndarray full of zeros.  
array1 = np.zeros((3,4))  
# [np.zeros(shape)]: function is used to create array which is full of zeros with # given shape,shape contain value in the form of (rows,colums) i.e(3,4)  
  
# We print array1 after. zero function applied  
print()  
print('array1 after zero function: = \n', array1)  
print()  
  
  
  
  
# We create a 3 x 2 ndarray full of ones.  
array1 = np.ones((3,2))  
  
#[np.ones(shape)]: function is used to create array which is full of ones with given #shape,shape contain value in the form of (rows,colums) i.e(3,2)  
  
  
# We print array1 after performing function ones  
print()  
print('array1 after ones function: = \n', array1)  
print()  
  
  
  
  
# We create a 5 x 5 Identity matrix.  
array1 = np.eye(5)  
  
#[np.eye(N)] : This function is used to create an Identity matrix(is a square matrix #that has 1s in its main diagonal and zero at other indexes) N for N\*N matrix

# We print array1 after .eye function applied  
print()  
print('array1 after eye function: = \n', array1)  
print()  
  
  
  
  
# We create a rank 1 ndarray that has sequential integers from 0 to 8  
  
array1 = np.arange(9)  
  
# [np.arange(start,stop,step)] : This create a 1D array with evenly spaced values # here[start and stop] its tels about starting point and ending with step being # the distance between two adjecent values."""  
  
  
# We print the ndarray  
print()  
print('array1 after arange function applied = ', array1)  
print()

**Output:**

Final output of Code 1:

list1 = [1 2 3 4 5]

list2 = [[ 1 2 3]

[ 4 5 6]

[ 7 8 9]

[10 11 12]]

Final Output of above Code 2 with all its functions contains:

list1 =

[1, 2, 3, 4, 5]

list2 =

[6, 7, 8, 9, 10]

Both lists has dimensions: (2, 5)

Both list has a total of 10 elements

list2 is an object of type: <class 'numpy.ndarray'>

The elements in array lists(list1 and 2) are of type: int32

array1 after zero function: =

[[0. 0. 0. 0.]

[0. 0. 0. 0.]

[0. 0. 0. 0.]]

array1 after ones function: =

[[1. 1.]

[1. 1.]

[1. 1.]]

array1 after eye function: =

[[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.]]

array1 after arange function applied = 2.718281828459045

**d) Scalar Operations on NumPy:**

Scalar Operation on NumPy are also called as Numeric Operations like addition, subtraction etc. perform on arrays we have lots of in-built function available but we perform basic element-wise operations using arithmetic symbols and functions.

**Code:**

from \_\_future\_\_ import division

# The above \_future\_ useful in version 2 but in 3 onward not much useful but steal works

# We import NumPy into Python  
import numpy as np  
array1 = np.array([[1,2,3,4],[5,6,7,8]])  
print(array1)

#multiplication of array1  
print()  
array2 = array1 \* array1  
print(array2)  
print()  
#Division of array1  
array3 = array1 / array1  
print(array3)  
print()

#Addition of array1   
array4 = array1 + array1  
print(array4)  
print()  
a = array1 + array4  
print(a)

#Do Subtraction of array1  
array5 = array1 - array1  
print(array5)  
ar =array4 - array5  
print(ar)  
print()

#This print reciprocal of array1  
array6 = 1/array1  
print(array6)

**Output:**

[[1 2 3 4]

[5 6 7 8]]

multiplication: [[ 1 4 9 16]

[25 36 49 64]]

Division: [[1. 1. 1. 1.]

[1. 1. 1. 1.]]

Addition : [[ 2 4 6 8]

[10 12 14 16]]

[[ 3 6 9 12]

[15 18 21 24]]

Substraction: [[0 0 0 0]

[0 0 0 0]]

substraction: [[ 2 4 6 8]

[10 12 14 16]]

reciprocal: [[1. 0.5 0.33333333 0.25 ]

[0.2 0.16666667 0.14285714 0.125 ]]

e) **Indexing NumPy arrays:**

Elements can be accessed using indices inside square brackets, [ ].

In order to access individual elements (single) at a time, NumPy provides a way known as *slicing.*

*Indexing NumPy array (part 1):*

**Code *:***

# Following is the Types of Slicing

1. ndarray[start: end]

2. ndarray[start:]

3. ndarray[:end]

import numpy as np

# Creating a 1D array that contain integer from 0 to 9  
arr = np.arange(0,10)

# anathor way of creating 1D array of integer from 0 to 9  
arr1 = np.array([0,1,2,3,4,5,6,7,8,9])  
print("1-D Array:\n", arr)  
print("1-D Array:\n", arr1)  
  
#We can select elements start at 0 th index and end at 5th or elemnt from zero to 4 in array

print("Element from 0 to 5th index:\n", arr[0:5])  
#We can select elements from index 2 to 6

print("Element from index[2:6]:\n",arr[2:6])  
print()  
#We change elements from index 0 to 5 with value 20 and print it

arr[0:5] = 20  
print("After change index value[0 to 5]:\n", arr)  
  
# Here we select all elements from index 0 to 6 from array having name arr and assign to new array name as arr2

# Slicing only creates a view of the original array  
arr2 = arr[0:6]  
print('New Created array:\n', arr2)  
#all elements in array(arr2) are modified with value 29  
arr2[:] = 29

#It also effect original array [i.e arr]  
print("After change in New Array Elements : \n", arr2)  
print("Original Array Effected Due to new array modifications:\n", arr)

# creating new array copy, by using copy command we create new ndarrays that are completely independent

arrcopy = arr.copy()  
#Printing copy  
print("New array using copy function:\n",arrcopy)

**Output:**

1-D Array:

[0 1 2 3 4 5 6 7 8 9]

1-D Array:

[0 1 2 3 4 5 6 7 8 9]

Element from 0 to 5th index:

[0 1 2 3 4]

Element from index[2:6]:

[2 3 4 5]

After change index value[0 to 5]:

[20 20 20 20 20 5 6 7 8 9]

New Created array:

[20 20 20 20 20 5]

After change in New Array Elements :

[29 29 29 29 29 29]

Original Array Effected Due to new array modifications:

[29 29 29 29 29 29 6 7 8 9]

New array using copy function:

[29 29 29 29 29 29 6 7 8 9]

*Indexing NumPy array (part 2):*

Performing Indexing on 2-D array also we can access elements in array using for loop shown in code.

**Code:**

import numpy as np  
#creating 2-Dimensional array  
D2array = np.array([[1,2,3],[4,5,6],[7,8,9]])  
print("Print 2-D array:", D2array)  
print()  
# Printing Element from array index 2 of 2nd row  
print("2-D Elements ",D2array[1][2])  
print()  
#Slices of 2-D array  
#We select all the elements that are in 1st through third row and in the 1st to 3rd columns  
slice1 = D2array[0:2,0:2]  
print(slice1)  
print()  
#We replace element that are in 1st through second row and in the 2nd to 3rd column with 15  
D2array[:2,1:] = 15  
print("After Replacement 2-D Array:\n", D2array)  
print()  
#We use .shape[0] to get shape of array but here create array having values(0,1,2)  
arr\_len = D2array.shape[0]  
print("2Darray:\n", arr\_len)  
print()  
# We are using loops to indexing and accesing elemnts of arr\_len  
for i in range(arr\_len):  
 D2array[i] = i;  
print("For loop:",D2array)  
print()  
#one more way of accessing the rows  
print(D2array[[0,1]])  
print()  
print(D2array[[1,0]])

**Output:**

Print 2-D array: [[1 2 3]

[4 5 6]

[7 8 9]]

2-D Elements 6

[[1 2]

[4 5]]

After Replacement 2-D Array:

[[ 1 15 15]

[ 4 15 15]

[ 7 8 9]]

2Darray:

3

For loop: [[0 0 0]

[1 1 1]

[2 2 2]]

[[0 0 0]

[1 1 1]]

[[1 1 1]

[0 0 0]]

**f) Premium Array Operations:**

We can also apply mathematical functions, such as (sqrt), to all elements of a ndarray

**Code:**

# arange :Return an array with evenly spaced element

# sqrt : Use to find the square root of elements

# exp : Perform exponent operation of maths

# random :It creates a random numbers

# addition : Perform addition of elements

# maximum : Find maximum value of all elements in array

import numpy as np

A = np.arange(15)  
print(A)  
A = np.arange(1,15,2)  
print(A)  
  
# We apply different mathematical functions to all elements of A  
print()  
print('EXP(A) =\n', np.exp(A))  
print()  
B = np.sqrt(A)  
print('SQRT(A) =\n', B)  
print()  
  
# Addition  
C = np.add(A,B)  
print("Addition of Array Elements is:\n", B)  
print()  
print()  
#Maximum of Elements  
print('Maximum value of all elements in C:', C.max())  
print()

**Output:**

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]

[ 1 3 5 7 9 11 13]

EXP(A) =

[2.71828183e+00 2.00855369e+01 1.48413159e+02 1.09663316e+03

8.10308393e+03 5.98741417e+04 4.42413392e+05]

SQRT(A) =

[1. 1.73205081 2.23606798 2.64575131 3. 3.31662479

3.60555128]

Addition of Array Elements is:

[1. 1.73205081 2.23606798 2.64575131 3. 3.31662479

3.60555128]

Maximum value of all elements in C: 16.605551275463988

**g) Saving and Loading arrays from external memory:**

We can save NumPy arrays into external memory in different format like text etc. Also, we can load the same from memory using different function ex.(load). Below code explain such functions.

**Code:**

import numpy as np  
  
# Create single arrays from 0-9  
arr = np.arange(20)  
print("Array",arr)  
  
#.save function save a single array file with some name  
np.save('saved\_array',arr)  
#'saved\_array' is a array file name  
  
#New file is created - saved\_array.npy  
#np.load function load the previous saved file having extension .npy  
new\_array = np.load('saved\_array.npy')  
print("Loaded File:\n",new\_array)  
  
# Multiple arrays  
array\_1 = np.arange(30)  
array\_2 = np.arange(35)  
# .savez function use to save multidimension arrays  
np.savez('saved\_archive.npz',x = array\_1, y = array\_2)  
  
load\_archive = np.load('saved\_archive.npz')  
# Loading first from saved file  
print('load\_archive[x] is:\n')  
print(load\_archive['x'])  
# Loading second array from saved file  
print('load\_archive[y] is')  
print(load\_archive['y'])  
  
  
#saving array to txtfile  
  
#While saving array in text file we use savetxt function  
np.savetxt('arrayfile.txt', array\_1, delimiter=',')  
# delimiter use to seprate array elements   
# loading of txt files  
load\_txt\_file = np.loadtxt('notepadfile.txt',delimiter=',')  
# loadtxt function is use to load text file array   
print("load\_txt\_file is")  
print(load\_txt\_file)

**Output:**

Array [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

Loaded File:

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

load\_archive[x] is:

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

24 25 26 27 28 29]

load\_archive[y] is

[ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

24 25 26 27 28 29 30 31 32 33 34]

load\_txt\_file is

[ 0. 1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12. 13. 14. 15. 16. 17.

18. 19. 20. 21. 22. 23. 24. 25. 26. 27. 28. 29.]

**h) Statistical Processing and Array Graphical Sketches:**

Here we are performing some statistical and other mathematical Operations on Array and presenting it in the form of graphs: We are using a new library called ***‘matplotlib’*** it’s a library us for graphical uses and representations.

**Code:**

import numpy as np  
# matplotlib.pyplot is uses for ploting point and making changes in figure  
import matplotlib.pyplot as plt  
#Creats array from -100 to +100 having interver of 10 in between  
axes\_values = np.arange(-100,100,10)  
# meshgrid function use for representing X and Y co-ordinates in graph  
dx, dy = np.meshgrid(axes\_values,axes\_values)  
print("Printing X\_Coordinate(dx):")  
print(dx)  
  
print("Y\_Coordinate(dy):")  
print(dy)  
# Finding derivation of given elements in array  
function1 = 2\*dx+3\*dy  
#plt.imshow(function1)  
#plt.title("function of Derivation")  
#plt.colorbar()  
#plt.savefig("figa.png")  
# Addition of cos values on x and y axis  
function2 = np.cos(dx)+np.cos(dy)  
  
print("Derivation:", function1)  
print("Addition of cos values:", function2)  
  
#replace function2 by function1 to get graph of function1  
# .imshow is used to plotting the function on graph  
plt.imshow(function2)  
# .title : function prints the title for graph   
plt.title("function cos plot")  
# colorbar produces a colorbar  
plt.colorbar()  
# it saves the output to a png file  
plt.savefig('myfig2.png')

**OUTPUTE:**

Printing X\_Coordinate(dx):

[[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]

[-100 -90 -80 -70 -60 -50 -40 -30 -20 -10 0 10 20 30

40 50 60 70 80 90]]

Y\_Coordinate(dy):

[[-100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 -100

-100 -100 -100 -100 -100 -100]

[ -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 -90 -90

-90 -90 -90 -90 -90 -90]

[ -80 -80 -80 -80 -80 -80 -80 -80 -80 -80 -80 -80 -80 -80

-80 -80 -80 -80 -80 -80]

[ -70 -70 -70 -70 -70 -70 -70 -70 -70 -70 -70 -70 -70 -70

-70 -70 -70 -70 -70 -70]

[ -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60 -60

-60 -60 -60 -60 -60 -60]

[ -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50 -50

-50 -50 -50 -50 -50 -50]

[ -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40 -40

-40 -40 -40 -40 -40 -40]

[ -30 -30 -30 -30 -30 -30 -30 -30 -30 -30 -30 -30 -30 -30

-30 -30 -30 -30 -30 -30]

[ -20 -20 -20 -20 -20 -20 -20 -20 -20 -20 -20 -20 -20 -20

-20 -20 -20 -20 -20 -20]

[ -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10 -10

-10 -10 -10 -10 -10 -10]

[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0]

[ 10 10 10 10 10 10 10 10 10 10 10 10 10 10

10 10 10 10 10 10]

[ 20 20 20 20 20 20 20 20 20 20 20 20 20 20

20 20 20 20 20 20]

[ 30 30 30 30 30 30 30 30 30 30 30 30 30 30

30 30 30 30 30 30]

[ 40 40 40 40 40 40 40 40 40 40 40 40 40 40

40 40 40 40 40 40]

[ 50 50 50 50 50 50 50 50 50 50 50 50 50 50

50 50 50 50 50 50]

[ 60 60 60 60 60 60 60 60 60 60 60 60 60 60

60 60 60 60 60 60]

[ 70 70 70 70 70 70 70 70 70 70 70 70 70 70

70 70 70 70 70 70]

[ 80 80 80 80 80 80 80 80 80 80 80 80 80 80

80 80 80 80 80 80]

[ 90 90 90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90]]

Derivation: [[-500 -480 -460 -440 -420 -400 -380 -360 -340 -320 -300 -280 -260 -240

-220 -200 -180 -160 -140 -120]

[-470 -450 -430 -410 -390 -370 -350 -330 -310 -290 -270 -250 -230 -210

-190 -170 -150 -130 -110 -90]

[-440 -420 -400 -380 -360 -340 -320 -300 -280 -260 -240 -220 -200 -180

-160 -140 -120 -100 -80 -60]

[-410 -390 -370 -350 -330 -310 -290 -270 -250 -230 -210 -190 -170 -150

-130 -110 -90 -70 -50 -30]

[-380 -360 -340 -320 -300 -280 -260 -240 -220 -200 -180 -160 -140 -120

-100 -80 -60 -40 -20 0]

[-350 -330 -310 -290 -270 -250 -230 -210 -190 -170 -150 -130 -110 -90

-70 -50 -30 -10 10 30]

[-320 -300 -280 -260 -240 -220 -200 -180 -160 -140 -120 -100 -80 -60

-40 -20 0 20 40 60]

[-290 -270 -250 -230 -210 -190 -170 -150 -130 -110 -90 -70 -50 -30

-10 10 30 50 70 90]

[-260 -240 -220 -200 -180 -160 -140 -120 -100 -80 -60 -40 -20 0

20 40 60 80 100 120]

[-230 -210 -190 -170 -150 -130 -110 -90 -70 -50 -30 -10 10 30

50 70 90 110 130 150]

[-200 -180 -160 -140 -120 -100 -80 -60 -40 -20 0 20 40 60

80 100 120 140 160 180]

[-170 -150 -130 -110 -90 -70 -50 -30 -10 10 30 50 70 90

110 130 150 170 190 210]

[-140 -120 -100 -80 -60 -40 -20 0 20 40 60 80 100 120

140 160 180 200 220 240]

[-110 -90 -70 -50 -30 -10 10 30 50 70 90 110 130 150

170 190 210 230 250 270]

[ -80 -60 -40 -20 0 20 40 60 80 100 120 140 160 180

200 220 240 260 280 300]

[ -50 -30 -10 10 30 50 70 90 110 130 150 170 190 210

230 250 270 290 310 330]

[ -20 0 20 40 60 80 100 120 140 160 180 200 220 240

260 280 300 320 340 360]

[ 10 30 50 70 90 110 130 150 170 190 210 230 250 270

290 310 330 350 370 390]

[ 40 60 80 100 120 140 160 180 200 220 240 260 280 300

320 340 360 380 400 420]

[ 70 90 110 130 150 170 190 210 230 250 270 290 310 330

350 370 390 410 430 450]]

Addition of cos values: [[ 1.72463774 0.41424526 0.75193163 1.49563808 -0.09009411 1.8272849

0.19538081 1.01657032 1.27040093 0.02324734 1.86231887 0.02324734

1.27040093 1.01657032 0.19538081 1.8272849 -0.09009411 1.49563808

0.75193163 0.41424526]

[ 0.41424526 -0.89614723 -0.55846086 0.18524559 -1.4004866 0.51689241

-1.11501168 -0.29382217 -0.03999155 -1.28714515 0.55192638 -1.28714515

-0.03999155 -0.29382217 -1.11501168 0.51689241 -1.4004866 0.18524559

-0.55846086 -0.89614723]

[ 0.75193163 -0.55846086 -0.22077449 0.52293196 -1.06280022 0.85457878

-0.77732531 0.04386421 0.29769482 -0.94945877 0.88961276 -0.94945877

0.29769482 0.04386421 -0.77732531 0.85457878 -1.06280022 0.52293196

-0.22077449 -0.55846086]

[ 1.49563808 0.18524559 0.52293196 1.26663841 -0.31909378 1.59828523

-0.03361886 0.78757065 1.04140126 -0.20575233 1.6333192 -0.20575233

1.04140126 0.78757065 -0.03361886 1.59828523 -0.31909378 1.26663841

0.52293196 0.18524559]

[-0.09009411 -1.4004866 -1.06280022 -0.31909378 -1.90482596 0.01255305

-1.61935104 -0.79816153 -0.54433092 -1.79148451 0.04758702 -1.79148451

-0.54433092 -0.79816153 -1.61935104 0.01255305 -1.90482596 -0.31909378

-1.06280022 -1.4004866 ]

[ 1.8272849 0.51689241 0.85457878 1.59828523 0.01255305 1.92993206

0.29802797 1.11921748 1.37304809 0.1258945 1.96496603 0.1258945

1.37304809 1.11921748 0.29802797 1.92993206 0.01255305 1.59828523

0.85457878 0.51689241]

[ 0.19538081 -1.11501168 -0.77732531 -0.03361886 -1.61935104 0.29802797

-1.33387612 -0.51268661 -0.258856 -1.50600959 0.33306194 -1.50600959

-0.258856 -0.51268661 -1.33387612 0.29802797 -1.61935104 -0.03361886

-0.77732531 -1.11501168]

[ 1.01657032 -0.29382217 0.04386421 0.78757065 -0.79816153 1.11921748

-0.51268661 0.3085029 0.56233351 -0.68482008 1.15425145 -0.68482008

0.56233351 0.3085029 -0.51268661 1.11921748 -0.79816153 0.78757065

0.04386421 -0.29382217]

[ 1.27040093 -0.03999155 0.29769482 1.04140126 -0.54433092 1.37304809

-0.258856 0.56233351 0.81616412 -0.43098947 1.40808206 -0.43098947

0.81616412 0.56233351 -0.258856 1.37304809 -0.54433092 1.04140126

0.29769482 -0.03999155]

[ 0.02324734 -1.28714515 -0.94945877 -0.20575233 -1.79148451 0.1258945

-1.50600959 -0.68482008 -0.43098947 -1.67814306 0.16092847 -1.67814306

-0.43098947 -0.68482008 -1.50600959 0.1258945 -1.79148451 -0.20575233

-0.94945877 -1.28714515]

[ 1.86231887 0.55192638 0.88961276 1.6333192 0.04758702 1.96496603

0.33306194 1.15425145 1.40808206 0.16092847 2. 0.16092847

1.40808206 1.15425145 0.33306194 1.96496603 0.04758702 1.6333192

0.88961276 0.55192638]

[ 0.02324734 -1.28714515 -0.94945877 -0.20575233 -1.79148451 0.1258945

-1.50600959 -0.68482008 -0.43098947 -1.67814306 0.16092847 -1.67814306

-0.43098947 -0.68482008 -1.50600959 0.1258945 -1.79148451 -0.20575233

-0.94945877 -1.28714515]

[ 1.27040093 -0.03999155 0.29769482 1.04140126 -0.54433092 1.37304809

-0.258856 0.56233351 0.81616412 -0.43098947 1.40808206 -0.43098947

0.81616412 0.56233351 -0.258856 1.37304809 -0.54433092 1.04140126

0.29769482 -0.03999155]

[ 1.01657032 -0.29382217 0.04386421 0.78757065 -0.79816153 1.11921748

-0.51268661 0.3085029 0.56233351 -0.68482008 1.15425145 -0.68482008

0.56233351 0.3085029 -0.51268661 1.11921748 -0.79816153 0.78757065

0.04386421 -0.29382217]

[ 0.19538081 -1.11501168 -0.77732531 -0.03361886 -1.61935104 0.29802797

-1.33387612 -0.51268661 -0.258856 -1.50600959 0.33306194 -1.50600959

-0.258856 -0.51268661 -1.33387612 0.29802797 -1.61935104 -0.03361886

-0.77732531 -1.11501168]

[ 1.8272849 0.51689241 0.85457878 1.59828523 0.01255305 1.92993206

0.29802797 1.11921748 1.37304809 0.1258945 1.96496603 0.1258945

1.37304809 1.11921748 0.29802797 1.92993206 0.01255305 1.59828523

0.85457878 0.51689241]

[-0.09009411 -1.4004866 -1.06280022 -0.31909378 -1.90482596 0.01255305

-1.61935104 -0.79816153 -0.54433092 -1.79148451 0.04758702 -1.79148451

-0.54433092 -0.79816153 -1.61935104 0.01255305 -1.90482596 -0.31909378

-1.06280022 -1.4004866 ]

[ 1.49563808 0.18524559 0.52293196 1.26663841 -0.31909378 1.59828523

-0.03361886 0.78757065 1.04140126 -0.20575233 1.6333192 -0.20575233

1.04140126 0.78757065 -0.03361886 1.59828523 -0.31909378 1.26663841

0.52293196 0.18524559]

[ 0.75193163 -0.55846086 -0.22077449 0.52293196 -1.06280022 0.85457878

-0.77732531 0.04386421 0.29769482 -0.94945877 0.88961276 -0.94945877

0.29769482 0.04386421 -0.77732531 0.85457878 -1.06280022 0.52293196

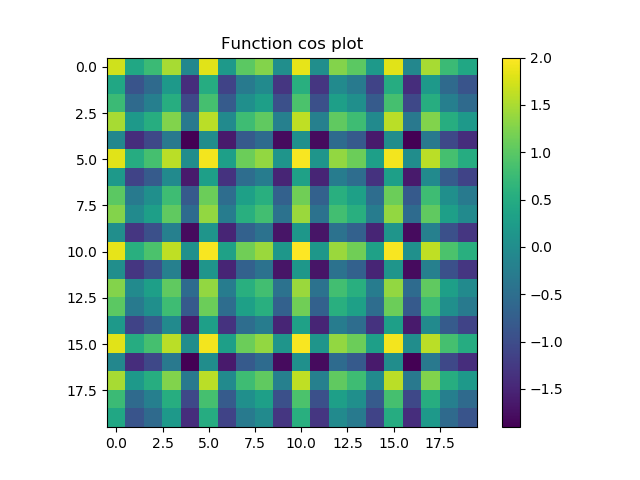
-0.22077449 -0.55846086]

[ 0.41424526 -0.89614723 -0.55846086 0.18524559 -1.4004866 0.51689241

-1.11501168 -0.29382217 -0.03999155 -1.28714515 0.55192638 -1.28714515

-0.03999155 -0.29382217 -1.11501168 0.51689241 -1.4004866 0.18524559

-0.55846086 -0.89614723]]



myfig.png

**i) Conditional Clauses and Boolean Operations on NumPy:**

Here contains study about logical operations in NumPy like (and) & (or).

Boolean operation contains and standard functions like sorting, sum.

**Code:**

import numpy as np

# creating 1D arrays  
x = np.array([100,400,500,600])  
print("Array List\_1 :\n",x)  
y = np.array([100,15,20,25])  
print("Array List\_2 :\n",y)

# Another array list having boolean values  
condition = np.array([True,True,False,False])  
print("Condition array list:\n",condition)

#Using loop to compare values of x,y and conditons

# It check first 2 value of c and last two of y True for x and False for Y and print

z = [a if cond else b for a,cond,b in zip(x,condition,y)]  
print("For loop output:\n",z)

print()  
  
# .where checks where the given condition is satisfied  
z2 = np.where(condition,x,y)  
print("First two value of x and last two of y:\n",z2)  
print()

#Checking where value of x is greate than (0,0,1)  
z3 = np.where(x>0,0,1)  
print(z3)  
  
  
#Standard functions operation of numpy  
  
#sum of elements in x  
print("Enter the sum of Elements in x:\n",x.sum())  
print()

#2-D array  
n = np.array([[1,2],[3,4]])  
#Using sum function to add two column  
print("Columns sum:\n",n.sum(0))  
print()

# We write mean value of array elements using mean()  
print("Elements mean value:\n",x.mean())  
print()

# Writing standard deviations  
print("Elements standard Deviation:\n",x.std())  
print()  
# reserving some space in memory for dictionary elements  
print(x.var())  
print()  
  
#logical operations and / or operations  
  
condition2 = np.array([True,False,True])  
#Performing or operation  
print("Logical or operation :\n",condition2.any())

# Performing or operation on array element  
print("Logical and operation:\n",condition2.all())  
  
# Performing sorting on numpy arrays

#creating simple 1-D array  
simple\_array = np.array([1,2,8,10,7,3])  
print("Before Sorting:\n",simple\_array)  
print()

# performing sorting on unsorted array list  
simple\_array.sort()  
print("After Sorting:\n",simple\_array)  
print()

# Creating another array  
array\_2 = np.array(['solid','solid','solid','liquid','liquid','gas','gas'])  
print("Array\_2:\n",array\_2)  
print()

# Finding unique values from array which are non repeated  
print("Unique values from array\_2:\n",np.unique(array\_2))  
print()

# Finding if element present in array using boolen result True if present else false  
print("Is this Element's present in array [solid,gas,plasma]:\n", np.in1d(['solid','gas','plasma'],array\_2))

**OUTPUTE:**

Array List\_1 :

[100 400 500 600]

Array List\_2 :

[100 15 20 25]

Condition array list:

[ True True False False]

For loop output:

[100, 400, 20, 25]

First two value of x and last two of y:

[100 400 20 25]

[0 0 0 0]

Enter the sum of Elements in x:

1600

Columns sum:

[4 6]

Elements mean value:

400.0

Elements standard Deviation:

187.08286933869707

35000.0

Logical or operation :

True

Logical and operation:

False

Before Sorting:

[ 1 2 8 10 7 3]

After Sorting:

[ 1 2 3 7 8 10]

Array\_2:

['solid' 'solid' 'solid' 'liquid' 'liquid' 'gas' 'gas']

Unique values from array\_2:

['gas' 'liquid' 'solid']

Is Element present in array [solid,gas,plasma]:

[ True True False]

**Pandas:**

1. **Introduction to Pandas:**

Pandas is a package for data manipulation and analysis in Python. It incorporates two additional data structure into Python, namely Pandas Series and Pandas DataFrame. These data structure allows us to work with labeled and relational data in an easy and intuitive manner.

1. **Working with Pandas *Series*:**

Series in pandas is a Data Structure or we can say it as a 1-D labeled array, capable of holding point numbers, python objects etc.

**CODE:**

import pandas as pd  
from pandas import Series  
import numpy as np  
groceries = pd.Series(data = [30, 6, 'Yes', 'No'],index = ['eggs', 'apples', 'milk', 'bread'])  
print(groceries)  
print()  
# We print some information about Groceries  
print('Groceries has shape :', groceries.shape)  
print()  
print('Dimension', groceries.ndim)  
print()  
print('Groceries has a Total of :', groceries.size, 'elements')  
# We print the index and data of Groceries  
print()  
print('The data in Groceries is : ', groceries.values)  
print()  
print('The index of Groceries is:', groceries.index)  
print()  
# We check whether bananas is a food item (an index) in Groceries  
x = 'bananas' in groceries  
print()  
print("Bananas in groceries :", x)  
# We access elements in Groceries using index labels:  
  
# We use a single index label  
print()  
print('How many eggs do we need to buy:', groceries['eggs'])  
print()  
# we use loc to access multiple index labels  
print('How many eggs and apples do we need to buy:\n', groceries.loc[['eggs', 'apple']])  
print()  
  
# We access elements in Groceries using numerical indices:  
  
# we use multiple numerical indices  
print('How many eggs and apples do we need to buy:\n', groceries[[0,1]])  
print(groceries[0] + groceries[1])  
print()  
  
# We use a negative numerical index  
print('Do we need bread:\n', groceries[[-1]])  
print()  
# We use a single numerical index  
print('How many eggs do we need to buy:', groceries[0])  
print()  
# we use iloc to access multiple numerical indices  
print('Do we need milk and bread:\n', groceries.iloc[[2, 3]])  
print()  
# We display the original grocery list  
print('Original Grocery List:\n', groceries)  
print()  
# We change the number of eggs to 2  
groceries['eggs'] = 2  
  
# We display the changed grocery list  
print()  
print('Modified Grocery List:\n', groceries)  
print()  
print('Modified Grocery List:\n', groceries.drop('apples'))  
print(groceries)  
print()  
# We remove apples from our grocery list in place by setting the inplace keyword to True  
groceries.drop('apples', inplace=True)  
print(groceries)  
print()  
index1 = groceries.index  
print(index1)  
print()  
# another example on Series  
#seriease creation  
object = Series([5, 10, 15, 20])  
print("Seriease :\n",object)  
print()  
# Assesing Seriease Values  
print("According to series Values:\n",object.values)  
print()  
# Assesing Series index  
print("According to series index:\n",object.index)  
print()  
#use numpy arrays to series  
data\_array = np.array(['a', 'b', 'c'])  
s = Series(data\_array)  
print("Numpy array into Series",s)  
print()  
#custom index :--  
s = Series(data\_array, index=[100, 101, 102])  
print("Custome index Values Series: \n",s)  
print()  
s = Series(data\_array, index=['index1', 'index2', 'index3'])  
print("Custom Index Series :\n",s)  
print()  
#using real life example  
revenue = Series([20, 80, 40, 35], index = ['ola', 'uber', 'grab', 'gojeck'])  
print("Revenue of Companies:",revenue)  
print()  
#condition using in series  
print("Companies having revenue greater than 35",revenue[revenue >= 35])  
print()  
#use boolean conditions  
print('lyft' in revenue)  
#converting Series into Disctionary  
print()  
r = revenue.to\_dict()  
print("After converting Series into Dictionary",r)  
print(revenue.to\_dict())  
print()  
#nan values  
index2 = ['ola', 'uber', 'grab', 'gojek', 'lyft']  
revenue2 = Series(revenue,index2)  
print("Boolen Values as True or False:\n",revenue2)  
print()  
# .isnull() method returns a Boolean DataFrame of the same size as store\_items  
# and indicates with True the elements that have NaN values and with False the elements that are not.  
print("Returning Boolean DataFrame:\n",pd.isnull(revenue2))  
#notnull: stores True for ever NON-NULL value and False for a null value  
print()  
print("Showing Not null values as true:\n",pd.notnull(revenue2))  
#addition of series(+)  
print()  
print("Adding revinues:\n",revenue+revenue2)  
#assigning name to revinues  
print()  
revenue2.name = 'Company Revenues'  
# changing index-name  
revenue2.index.name='Company name'  
#revenue2.value.name='values'  
print("Final Revenue",revenue2)

**OUTPUTE:**

eggs 30  
apples 6  
milk Yes  
bread No  
  
Groceries has shape : (4,)  
  
Dimension 1  
  
Groceries has a Total of : 4 elements  
  
The data in Groceries is : [30 6 'Yes' 'No']  
  
The index of Groceries is: Index(['eggs', 'apples', 'milk', 'bread'], dtype='object')  
  
  
Bananas in groceries : False  
  
How many eggs do we need to buy: 30  
  
How many eggs and apples do we need to buy:  
 eggs 30  
apple NaN  
dtype: object  
  
How many eggs and apples do we need to buy:  
 eggs 30  
apples 6  
dtype: object  
36  
  
Do we need bread:  
 bread No  
dtype: object  
  
How many eggs do we need to buy: 30  
  
Do we need milk and bread:  
 milk Yes  
bread No  
dtype: object  
  
Original Grocery List:  
 eggs 30  
apples 6  
milk Yes  
bread No  
dtype: object  
  
Modified Grocery List:  
 eggs 2  
apples 6  
milk Yes  
bread No  
dtype: object  
  
Modified Grocery List:  
 eggs 2  
milk Yes  
bread No  
dtype: object  
eggs 2  
apples 6  
milk Yes  
bread No  
dtype: object  
  
eggs 2  
milk Yes  
bread No  
dtype: object  
  
Index(['eggs', 'milk', 'bread'], dtype='object')  
  
Seriease :  
 0 5  
1 10  
2 15  
3 20  
dtype: int64  
  
According to series Values:  
 [ 5 10 15 20]  
  
According to series index:  
 RangeIndex(start=0, stop=4, step=1)  
  
Numpy array into Series 0 a  
1 b  
2 c  
dtype: object  
  
Custome index Values Series:   
 100 a  
101 b  
102 c  
dtype: object  
  
Custom Index Series :  
 index1 a  
index2 b  
index3 c  
dtype: object  
  
Revenue of Companies: ola 20  
uber 80  
grab 40  
gojeck 35  
dtype: int64  
  
Companies having revenue greater than 35 uber 80  
grab 40  
gojeck 35  
dtype: int64  
  
False  
  
After converting Series into Dictionary {'ola': 20, 'uber': 80, 'grab': 40, 'gojeck': 35}  
{'ola': 20, 'uber': 80, 'grab': 40, 'gojeck': 35}  
  
Boolen Values as True or False:  
 ola 20.0  
uber 80.0  
grab 40.0  
gojek NaN  
lyft NaN  
dtype: float64  
  
Returning Boolean DataFrame:  
 ola False  
uber False  
grab False  
gojek True  
lyft True  
dtype: bool  
  
Showing Not null values as true:  
 ola True  
uber True  
grab True  
gojek False  
lyft False  
dtype: bool  
  
Adding revinues:  
 gojeck NaN  
gojek NaN  
grab 80.0  
lyft NaN  
ola 40.0  
uber 160.0  
dtype: float64  
  
Final Revenue Company name  
ola 20.0  
uber 80.0  
grab 40.0  
gojek NaN  
lyft NaN  
Name: Company Revenues, dtype: float64

**3.Working with Pandas DataFrame:**

DataFrame are 2-Dimensional Data Structure that can hold many data types. This are similar to a spreadsheet.

**CODE:**

import pandas as pd  
from pandas import Series, DataFrame  
import numpy as np  
#Reding Copied-Data from clipboard  
revenue\_df = pd.read\_clipboard()  
print("Clipboard\_data:\n",revenue\_df)  
  
#index and columns data accessing  
print("Columns Data From Clipboard Copyed Data", revenue\_df.columns)  
print()  
print("Rows Data From Clipboard Copyed Data:", revenue\_df['Company'])  
print()  
#multiple columns accessing  
print("Accessing Multiple Columns at a time:\n",DataFrame(revenue\_df, columns=["Rank (2018)", "Company", "Country"]))  
print()  
#NaN Values assessing  
revenue\_df = DataFrame(revenue\_df,columns=['Ranks (2018)', 'Company', 'Profit'])  
print("Accessing NaN values:\n", revenue\_df)  
#Accessin startin 2 rows i.e head(2)  
print("First two rows:",revenue\_df.head(2))  
print()  
# Accessing last 2 rows i.e tail(2)  
print("Accessing last two rows:\n",revenue\_df.tail(2))  
  
# access rows in DataFrame  
print("Accessing rows :",revenue\_df.ix[3])  
print()  
  
#deletion of column  
del revenue\_df['Profit']  
print("After Deletion:\n",revenue\_df)  
print()  
#dictionary function to dataframe  
sample = {  
 'company':['A','B'],  
 'Profit':[1000,5000]  
}  
  
print("Dictionary Data:\n",sample)  
  
sample\_df = DataFrame(sample)  
print("After converting Dictionary to DataFrame",sample\_df)

**OUTPUTE:**

Clipboard\_data:  
 Rank (2018) Company Country Revenue  
0 #1 Walmart USA $500.3 billion  
1 #2 State Grid China $348.9 billion  
2 #3 Sinopec Group China $327.0 billion  
3 #4 China National Petroleum China $326.0 billion  
4 #5 Royal Dutch Shell Netherlands $311.9 billion  
5 #6 Toyota Motor Japan $265.2 billion  
6 #7 Volkswagen Germany $260.0 billion  
7 #8 BP UK $244.6 billion  
8 #9 Exxon Mobil USA $244.4 billion  
9 #10 Berkshire Hathaway USA $242.1 billion  
Columns Data From Clipboard Copyed Data Index(['Rank (2018)', 'Company', 'Country', 'Revenue'], dtype='object')  
  
Rows Data From Clipboard Copyed Data: 0 Walmart  
1 State Grid  
2 Sinopec Group  
3 China National Petroleum  
4 Royal Dutch Shell  
5 Toyota Motor  
6 Volkswagen  
7 BP  
8 Exxon Mobil  
9 Berkshire Hathaway  
Name: Company, dtype: object  
  
Accessing Multiple Columns at a time:  
 Rank (2018) Company Country  
0 #1 Walmart USA  
1 #2 State Grid China  
2 #3 Sinopec Group China  
3 #4 China National Petroleum China  
4 #5 Royal Dutch Shell Netherlands  
5 #6 Toyota Motor Japan  
6 #7 Volkswagen Germany  
7 #8 BP UK  
8 #9 Exxon Mobil USA  
9 #10 Berkshire Hathaway USA  
  
Accessing NaN values:  
 Ranks (2018) Company Profit  
0 NaN Walmart NaN  
1 NaN State Grid NaN  
2 NaN Sinopec Group NaN  
3 NaN China National Petroleum NaN  
4 NaN Royal Dutch Shell NaN  
5 NaN Toyota Motor NaN  
6 NaN Volkswagen NaN  
7 NaN BP NaN  
8 NaN Exxon Mobil NaN  
9 NaN Berkshire Hathaway NaN  
First two rows: Ranks (2018) Company Profit  
0 NaN Walmart NaN  
1 NaN State Grid NaN  
  
Accessing last two rows:  
 Ranks (2018) Company Profit  
8 NaN Exxon Mobil NaN  
9 NaN Berkshire Hathaway NaN  
  
Accessing rows : Ranks (2018) NaN  
Company China National Petroleum  
Profit NaN  
Name: 3, dtype: object  
  
Dictionary Data:  
 {'company': ['A', 'B'], 'Profit': [1000, 5000]}  
After converting Dictionary to DataFrame company Profit  
0 A 1000  
1 B 5000  
  
Process finished with exit code 0

**4.Learning to use indexing in pandas:**

Indexing improves searching and accessing data elements capabilities in pandas also help in performing other operations on data as speed is increases to access data. Indexes are immutable.

#importing numpy and pandas library  
import numpy as np  
import pandas as pd  
#importing pandas library for Data Structure,series and DataFrame  
from pandas import Series, DataFrame  
#Creating Series  
series1 = Series([10,20,30,40],index=['a','b','c','d'])  
print("Data Series:\n",series1)  
print()  
index1 = series1.index  
print("Index of Series:\n",index1)  
print()  
#Accessing index from row 2nd to last column  
print("Accessing index:\n",index1[2:])  
print()  
#negative indexes  
#Accessing index 'c' to 'd' or accessing last two index element  
print("Last two index values :\n",index1[-2:])  
print()  
#Accessing first two index elements  
print("First two index elements are :\n",index1[:-2])  
print()  
# Accessing element from index 2 and 4  
print("Element at index 2 and 4 are:\n",index1[2:4])  
print()

**OUTPUTE:**

Data Series:

a 10

b 20

c 30

d 40

dtype: int64

Index of Series:

Index(['a', 'b', 'c', 'd'], dtype='object')

Accessing index:

Index(['c', 'd'], dtype='object')

Last two index values :

Index(['c', 'd'], dtype='object')

First two index elements are :

Index(['a', 'b'], dtype='object')

Element at index 2 and 4 are:

Index(['c', 'd'], dtype='object')

**5.Reindexing Capabilities pandas:**

Reindexing helps in changing row column labels without affecting internal structure and data of datasets.

**Code:**

import numpy as np  
import pandas as pd  
from pandas import Series,DataFrame  
#importing random package from numpy library for random number generation  
from numpy.random import randn  
  
#create new series series1  
series1 = Series([1,2,3,4],index=['e','f','g','h'])  
print("Series\_1:",series1)  
print()  
#creating new indexes using reindex  
series2 = series1.reindex(['e','f','g','h','i','j'])  
print("Series\_2 after reindexing:\n",series2)  
print()  
#using fillvalue to fill NaN Values with 10  
series2 = series2.reindex(['e','f','g','h','i','j','k'],fill\_value=10)  
print("After Performing fill\_value function:\n",series2)  
print()  
#using reindex methods => ffill  
cars = Series(['Audi','Merc','BMW'], index=[0,4,8])  
print("Cars\_series:\n",cars)  
print()  
# usig range function to create array list  
ranger = range(13)  
print("New List\_array:\n",ranger)  
print()  
#Reindexing Cars data set Series using forward fill method  
cars = cars.reindex(ranger,method="ffill")  
print("Reindex with next values:\n",cars)  
print()  
#create new dataframe using randn(randomnumbers)  
dara\_frame = DataFrame(randn(25).reshape(5,5),index=['a','b','c','d','e'], columns=['c1','c2','c3','c4','c5'])  
print("New\_DataFrame:\n",dara\_frame)  
print()  
  
dara\_frame1 = dara\_frame.reindex(['a','b','c','d','e','f'])  
print("After Reindexing data\_frame:\n",dara\_frame1)  
#reindex rows of dataframe  
#reindex columns of dataframe  
df\_3 = dara\_frame1.reindex(columns=['c1','c2','c3','c4','c5','c6'])  
print("data\_frame1 after reindexing:",df\_3)  
#using .ix[] to reindex column and rows  
df\_4 = df\_3.ix[['a','b','c','d','e','f'],['c1','c2','c3','c4','c5','c6']]  
print("Reindexing rows and cloumns:\n",df\_4)

**OUTPUTE:**

Series\_1: e 1  
f 2  
g 3  
h 4  
dtype: int64  
  
Series\_2 after reindexing:  
 e 1.0  
f 2.0  
g 3.0  
h 4.0  
i NaN  
j NaN  
dtype: float64  
  
After Performing fill\_value function:  
 e 1.0  
f 2.0  
g 3.0  
h 4.0  
i NaN  
j NaN  
k 10.0  
dtype: float64  
  
Cars\_series:  
 0 Audi  
4 Merc  
8 BMW  
dtype: object  
  
New List\_array:  
 range(0, 13)  
  
Reindex with next values:  
 0 Audi  
1 Audi  
2 Audi  
3 Audi  
4 Merc  
5 Merc  
6 Merc  
7 Merc  
8 BMW  
9 BMW  
10 BMW  
11 BMW  
12 BMW  
dtype: object  
  
New\_DataFrame:  
 c1 c2 c3 c4 c5  
a -0.366388 -0.086927 -0.129307 1.031146 1.141705  
b -0.354327 0.237158 1.742396 -0.015683 0.180838  
c -0.528691 -0.313097 0.102760 0.476952 -1.069719  
d 1.353525 1.137602 -1.423114 0.782055 1.316864  
e 0.812630 -1.325269 0.328737 -0.647960 -0.387672  
  
After Reindexing data\_frame:  
 c1 c2 c3 c4 c5  
a -0.366388 -0.086927 -0.129307 1.031146 1.141705  
b -0.354327 0.237158 1.742396 -0.015683 0.180838  
c -0.528691 -0.313097 0.102760 0.476952 -1.069719  
d 1.353525 1.137602 -1.423114 0.782055 1.316864  
e 0.812630 -1.325269 0.328737 -0.647960 -0.387672  
f NaN NaN NaN NaN NaN  
  
data\_frame1 after reindexing:  
 c1 c2 c3 c4 c5 c6  
a -0.000313 0.454155 -0.837340 -1.065383 0.073335 NaN  
b 1.085367 0.887801 -0.289063 -0.332770 1.079381 NaN  
c -1.046876 1.380820 0.714574 0.030257 0.665012 NaN  
d 0.808538 -0.011694 0.624444 1.370789 -0.839120 NaN  
e 0.610212 0.188165 -0.865215 1.022317 -0.899449 NaN  
f NaN NaN NaN NaN NaN NaN  
Reindexing rows and cloumns:  
 c1 c2 c3 c4 c5 c6  
a -0.000313 0.454155 -0.837340 -1.065383 0.073335 NaN  
b 1.085367 0.887801 -0.289063 -0.332770 1.079381 NaN  
c -1.046876 1.380820 0.714574 0.030257 0.665012 NaN  
d 0.808538 -0.011694 0.624444 1.370789 -0.839120 NaN  
e 0.610212 0.188165 -0.865215 1.022317 -0.899449 NaN  
f NaN NaN NaN NaN NaN NaN  
  
Process finished with exit code 0

**6.Droping Entries from Datatypes:**

Dropping missing values in panda or dropping NaN values can be done under different case, it’s important because creates ambiguity and incomplete data to work on ,below code tells about different methods to perform this task.

**Code:**

#importing pandas liberary and naming as pd to use  
import pandas as pd  
#importing numpy and using as np  
import numpy as np  
#importing Series and DataFrame from pandas liberary  
from pandas import Series,DataFrame  
# Creating Series DataStructure  
cars = Series(['BMW','Audi','Merc'],index=['a','b','c'])  
print("Series Data:\n",cars)  
print()  
# Droping Entries at index a from DataSet  
cars = cars.drop('a')  
print("After Droping values at index[a]:\n",cars)  
  
#Creating DataFrames  
print()  
cars\_data = DataFrame(np.arange(9).reshape(3,3),index=['BMW','Audi','Merc'],columns=['rev','pro','exp'])  
print("DataFrame:\n",cars\_data)  
print()  
# Droping NaN Datas from DataFrame from rows  
cars\_data = cars\_data.drop('BMW',axis=0)  
print("Droping NaN Data From rows:\n",cars\_data)  
print()  
# Droping NaN values from columns  
cars\_data = cars\_data.drop('pro',axis=1)  
print("Droping Coloumn NaN values:\n",cars\_data)  
print()  
  
# We create a list of Python dictionaries  
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35, 'shirts': 15, 'shoes':8, 'suits':45},  
{'watches': 10, 'glasses': 50, 'bikes': 15, 'pants':5, 'shirts': 2, 'shoes':5, 'suits':7},  
{'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4, 'shoes':10}]  
  
# We create a DataFrame and provide the row index  
store\_items = pd.DataFrame(items2, index = ['store 1', 'store 2', 'store 3'])  
  
# We display the DataFrame  
print("Store\_items:\n", store\_items)  
# We print the number of non-NaN values in our DataFrame  
print()  
print('Number of non-NaN values in the columns of our DataFrame:\n', store\_items.count())  
# We count the number of NaN values in store\_items here we count replace with True,False and after sum up the total count of NaN values.  
x = store\_items.isnull().sum().sum()  
# We print x  
print('Number of NaN values in our DataFrame:', x)

**OUTPUTE:**

Series Data:  
 a BMW  
b Audi  
c Merc  
dtype: object  
  
After Droping values at index[a]:  
 b Audi  
c Merc  
dtype: object  
  
DataFrame:  
 rev pro exp  
BMW 0 1 2  
Audi 3 4 5  
Merc 6 7 8  
  
Droping NaN Data From rows:  
 rev pro exp  
Audi 3 4 5  
Merc 6 7 8  
  
Droping Coloumn NaN values:  
 rev exp  
Audi 3 5  
Merc 6 8  
  
Store\_items:  
 bikes pants watches shirts shoes suits glasses  
store 1 20 30 35 15.0 8 45.0 NaN  
store 2 15 5 10 2.0 5 7.0 50.0  
store 3 20 30 35 NaN 10 NaN 4.0  
  
Number of non-NaN values in the columns of our DataFrame:  
 bikes 3  
pants 3  
watches 3  
shirts 2  
shoes 3  
suits 2  
glasses 2  
dtype: int64  
Number of NaN values in our DataFrame: 3

**7.Handling Null Data in Pandas:**

In case when there is large number of data NaN is not easily visualized. For this case we have to handle that Null Data intelligently.

**Code:**

#importing numpy liberarys  
import numpy as np  
#importing pandas library  
import pandas as pd  
#importing Series and DataFrame From pandas library  
from pandas import Series,DataFrame  
#Creating Series DataStructure also having Nan Values  
series1 = Series(['A','B','C','D',np.nan])  
  
#validate  
# isnull method returns a Boolean DataFrame of the same size as series1 and indicates with True the elements that have NaN values and with False the elements that are not  
print("Checking NaN Elements:\n", series1.isnull())  
#The .dropna(axis) method eliminates any rows with NaN values when axis = 0 is used and will eliminate any columns with NaN values when axis = 1 is used.  
  
print("We drop any rows with NaN values:\n", series1.dropna(axis=0))  
  
# Creating DataFrame  
DataFrame\_1 = DataFrame([[1,2,3],[5,6,np.nan],[7,np.nan,10],[np.nan,np.nan,np.nan]])  
print("DataFrame:\n",DataFrame\_1)  
print()  
#We drop DataFrame with NaN Values  
  
print("Droping DataFrame with NaN Values:\n",DataFrame\_1.dropna())  
print()  
# We drop DataFrame with NaN Values  
"""  
How takes string value of two kinds only (‘any’ or ‘all’). ‘any’ drops the row/column if ANY value is Null and ‘all’ drops only if ALL values are null  
"""  
print("Drops only if ALL values are null:\n",DataFrame\_1.dropna(how='all'))  
print()  
# Droping NaN Values according to columns  
print("Droping NaN Values according to Columns:\n",DataFrame\_1.dropna(axis=1))  
print()  
#Creating Another DataFram  
  
# We create a list of Python dictionaries  
items2 = [{'bikes': 20, 'pants': 30, 'watches': 35, 'shirts': 15, 'shoes':8, 'suits':45},  
{'watches': 10, 'glasses': 50, 'bikes': 15, 'pants':5, 'shirts': 2, 'shoes':5, 'suits':7},  
{'bikes': 20, 'pants': 30, 'watches': 35, 'glasses': 4, 'shoes':10}]  
  
# We create a DataFrame and provide the row index  
store\_items = pd.DataFrame(items2, index = ['store 1', 'store 2', 'store 3'])  
  
# We display the DataFrame  
print("Store DataFrame:\n",store\_items)  
# We print the number of non-NaN values in our DataFrame  
print()  
print('Number of non-NaN values in the columns of our DataFrame:\n', store\_items.count())  
print()  
print("(Minimum amount)3 Values to drop:\n",store\_items.dropna(thresh=3))  
print()  
  
print("Droping minimum 2 values:\n",store\_items.dropna(thresh=2))  
print()  
#ForwardFilling  
# We replace all NaN values with 0  
store\_items.fillna(0)  
print("After Filling all NaN values with 0:\n",store\_items.fillna(0))  
print()  
# We replace NaN values with the previous value in the column  
print("Replacing NaN Values with previous column values:\n",store\_items.fillna(method = 'ffill', axis = 0))  
print()  
# We replace NaN values with the previous value in the row  
print("Replacing NaN values with previous row values:\n",store\_items.fillna(method = 'ffill', axis = 1))  
#Backfill  
# We replace NaN values with the next value in the row  
print()  
print("Replacing NaN values with next values in the row but backward :\n",store\_items.fillna(method = 'backfill', axis = 1))  
print()  
# We replace NaN values by using linear interpolation using column values  
print("Replacing NaN values using linear interpolation:\n", store\_items.interpolate(method = 'linear', axis = 0))

**OUTPUTE:**

Checking NaN Elements:  
 0 False  
1 False  
2 False  
3 False  
4 True  
dtype: bool  
We drop any rows with NaN values:  
 0 A  
1 B  
2 C  
3 D  
dtype: object  
DataFrame:  
 0 1 2  
0 1.0 2.0 3.0  
1 5.0 6.0 NaN  
2 7.0 NaN 10.0  
3 NaN NaN NaN  
  
Droping DataFrame with NaN Values:  
 0 1 2  
0 1.0 2.0 3.0  
  
Drops only if ALL values are null:  
 0 1 2  
0 1.0 2.0 3.0  
1 5.0 6.0 NaN  
2 7.0 NaN 10.0  
  
Droping NaN Values according to Columns:  
 Empty DataFrame  
Columns: []  
Index: [0, 1, 2, 3]  
  
Store DataFrame:  
 bikes pants watches shirts shoes suits glasses  
store 1 20 30 35 15.0 8 45.0 NaN  
store 2 15 5 10 2.0 5 7.0 50.0  
store 3 20 30 35 NaN 10 NaN 4.0  
  
Number of non-NaN values in the columns of our DataFrame:  
 bikes 3  
pants 3  
watches 3  
shirts 2  
shoes 3  
suits 2  
glasses 2  
dtype: int64  
  
(Minimum amount)3 Values to drop:  
 bikes pants watches shirts shoes suits glasses  
store 1 20 30 35 15.0 8 45.0 NaN  
store 2 15 5 10 2.0 5 7.0 50.0  
store 3 20 30 35 NaN 10 NaN 4.0  
  
Droping minimum 2 values:  
 bikes pants watches shirts shoes suits glasses  
store 1 20 30 35 15.0 8 45.0 NaN  
store 2 15 5 10 2.0 5 7.0 50.0  
store 3 20 30 35 NaN 10 NaN 4.0  
  
After Filling all NaN values with 0:  
 bikes pants watches shirts shoes suits glasses  
store 1 20 30 35 15.0 8 45.0 0.0  
store 2 15 5 10 2.0 5 7.0 50.0  
store 3 20 30 35 0.0 10 0.0 4.0  
  
Replacing NaN Values with previous column values:  
 bikes pants watches shirts shoes suits glasses  
store 1 20 30 35 15.0 8 45.0 NaN  
store 2 15 5 10 2.0 5 7.0 50.0  
store 3 20 30 35 2.0 10 7.0 4.0  
  
Replacing NaN values with previous row values:  
 bikes pants watches shirts shoes suits glasses  
store 1 20.0 30.0 35.0 15.0 8.0 45.0 45.0  
store 2 15.0 5.0 10.0 2.0 5.0 7.0 50.0  
store 3 20.0 30.0 35.0 35.0 10.0 10.0 4.0  
  
Replacing NaN values with next values in the row but backward :  
 bikes pants watches shirts shoes suits glasses  
store 1 20.0 30.0 35.0 15.0 8.0 45.0 NaN  
store 2 15.0 5.0 10.0 2.0 5.0 7.0 50.0  
store 3 20.0 30.0 35.0 10.0 10.0 4.0 4.0  
  
Replacing NaN values using linear interpolation:  
 bikes pants watches shirts shoes suits glasses  
store 1 20 30 35 15.0 8 45.0 NaN  
store 2 15 5 10 2.0 5 7.0 50.0  
store 3 20 30 35 2.0 10 7.0 4.0

**8.Selecting and Modifying data in Pandas:**

Pandas allows us to modify and select specific data and make changes to them using some method’s and functionality as needed.

**Code:**

#importing numpy library  
import numpy as np  
#importing pandas library  
import pandas as pd  
# from pandas library importing Series and DataFrame DataStructure  
from pandas import Series,DataFrame  
# Creating Series DataStructure  
series1 = Series([100,200,300],index=['A','B','C'])  
print("Series1:\n", series1)  
print()  
#Accessing elements with reference to index as a Variable from Series data  
  
# Accessing elemnt at index A  
print("Accessing Index [A] Data:\n",series1['A'])  
print()  
# Accessing Element at index B  
print("Accessing data at Index[B]:\n",series1['B'])  
print()  
# Multiple Index Accessing  
print("Accessing Elements From Index A and B:\n", series1[['A','B']])  
print()  
#Accessing Elements at indexes using index number references  
print("Index Elements at Index 0:\n",series1[0])  
print()  
print("Index Elements at Index 0 and 2 :\n", series1[0:2])  
print()  
  
#Accessing indexes using some condition based  
print("Index in Series1 having value greater than 150:\n", series1[series1>150])  
#Index having values exactly same as 300  
print("Index having value equal 300:\n",series1[series1==300])  
#Operations on Data Frames  
#Creating DataFrame and accesing it  
DataFrame\_1 = DataFrame(np.arange(9).reshape(3,3),index=['car','bike','cycle'],columns=['A','B','C'])  
print("DataFrame\_:\n",DataFrame\_1)  
print()  
# Accessing Indexes using Variable References  
print("Value at index A in DataFrame",DataFrame\_1['A'])  
print()  
# Multiple index accessing  
print("Values at index A and B of DataFrame:\n",DataFrame\_1[['A','B']])  
print()  
#Assecing Index and data using Condition where DataFrame has Value greater than 15  
print("DataFrame having value greater than 15 :\n",DataFrame\_1 > 5)  
print()  
# ix : function use to access and perform slicing based on labels and integers  
#accessing values of index name bike  
print("Element of Index Bike :\n", DataFrame\_1.ix['bike'])  
print()  
#accessing elemnts at index 1  
print("Elements at index 1:")  
print(DataFrame\_1.ix[1])

**OUTPUTE:**

Series1:  
 A 100  
B 200  
C 300  
dtype: int64  
  
Accessing Index [A] Data:  
 100  
  
Accessing data at Index[B]:  
 200  
  
Accessing Elements From Index A and B:  
 A 100  
B 200  
dtype: int64  
  
Index Elements at Index 0:  
 100  
  
Index Elements at Index 0 and 2 :  
 A 100  
B 200  
dtype: int64  
  
Index in Series1 having value greater than 150:  
 B 200  
C 300  
dtype: int64  
Index having value equal 300:  
 C 300  
dtype: int64  
DataFrame\_:  
 A B C  
car 0 1 2  
bike 3 4 5  
cycle 6 7 8  
  
Value at index A in DataFrame car 0  
bike 3  
cycle 6  
Name: A, dtype: int32  
  
Values at index A and B of DataFrame:  
 A B  
car 0 1  
bike 3 4  
cycle 6 7  
  
DataFrame having value greater than 15 :  
 A B C  
car False False False  
bike False False False  
cycle True True True  
  
Element of Index Bike :  
 A 3  
B 4  
C 5  
Name: bike, dtype: int32  
  
Elements at index 1:  
A 3  
B 4  
C 5  
Name: bike, dtype: int32

**9.Co-ordinate and Regulate Data:**

Arranging data is must when we are working on chunks of data, so we can utilize it properly as per needed and effectively, coordination and regulation has important part in arrangements of data.

**Code:**

import numpy as np  
import pandas as pd  
from pandas import Series,DataFrame  
  
series\_1 = Series([100,200,300],index=['a','b','c'])  
series\_2 = Series([300,400,500,600],index=['a','b','c','d'])  
  
#Performing sum of two series  
print("Addition of series:\n", series\_1+series\_2)  
  
#Creating Dataframe  
# We create 2\*2 matrix having elements from 0 to 5  
dataframe\_1 = DataFrame(np.arange(4).reshape(2,2),columns=['a','b'],index=['car','bike'])  
print(dataframe\_1)  
print()  
dataframe\_2 = DataFrame(np.arange(9).reshape(3,3),columns=['a','b','c'],index=['car','bike','cycle'])  
print(dataframe\_2)  
print()  
# Summing two DataFrames  
print(dataframe\_1+dataframe\_2)  
print()  
#We add dataframe2 values to dataframe\_1 where the dataframe\_1 has null values  
dataframe\_1 = dataframe\_2.add(dataframe\_2,fill\_value=0)  
print("Replacing dataframe\_1 NaN values with dataframe\_2:\n",dataframe\_1)  
print()  
#Creating new dataSeries and assigning row 0 elements from dataframe\_2 to new Series  
new\_series = dataframe\_2.ix[0]  
print("New Series:\n", new\_series)  
#Substracting new created series from dataframe\_2  
print("After deletation of new series from dataframe:\n",dataframe\_2 - new\_series)

**OUTPUTE:**

Addition of series:  
 a 400.0  
b 600.0  
c 800.0  
d NaN  
dtype: float64  
 a b  
car 0 1  
bike 2 3  
  
 a b c  
car 0 1 2  
bike 3 4 5  
cycle 6 7 8  
  
 a b c  
bike 5.0 7.0 NaN  
car 0.0 2.0 NaN  
cycle NaN NaN NaN  
  
Replacing dataframe\_1 NaN values with dataframe\_2:  
 a b c  
car 0 2 4  
bike 6 8 10  
cycle 12 14 16  
  
New Series:  
 a 0  
b 1  
c 2  
Name: car, dtype: int32  
After deletation of new series from dataframe:  
 a b c  
car 0 0 0  
bike 3 3 3  
cycle 6 6 6

**10.Ranking and Sorting characteristics:**

Simple representation of data cannot say its important and not distinguish data according to their importance manly when data is large, so there is need to do Ranking and Sorting of Data which has a significant important, following function for doing this shown in code:

**Code:**

#importing numpy and pandas library's  
import numpy as np  
import pandas as pd  
from pandas import Series  
#numpy library for random number generation operations  
from numpy.random import randn  
#Creating series datastructure  
series\_1 = Series([500,1000,1500],index=['a','c','b'])  
print("Series Data:\n",series\_1)  
#sorting by index references  
print("Sorting Based on index:\n",series\_1.sort\_index())  
print()  
  
#sorting according to values not by index  
print("Sorting based on values :\n",series\_1.sort\_values())  
print()  
  
#rank() method returns a rank of every respective index of a series passed  
print("Ranking of Series\_1 data:\n",series\_1.rank())  
  
###ranking of series  
# we create a Series having random numbers from 0 to 10  
series\_2 = Series(randn(10))  
print("Series\_2:\n",series\_2)  
print()  
print("Series\_2 ranking :",series\_2.rank())  
#sorting values of series data  
series\_2 = series\_2.sort\_values()  
#ranking sorted datav alues  
print("Ranking \_sorted data:\n",series\_2.rank())

**OUTPUTE:**

Series Data:  
 a 500  
c 1000  
b 1500  
dtype: int64  
Sorting Based on index:  
 a 500  
b 1500  
c 1000  
dtype: int64  
  
Sorting based on values :  
 a 500  
c 1000  
b 1500  
dtype: int64  
  
Ranking of Series\_1 data:  
 a 1.0  
c 2.0  
b 3.0  
dtype: float64  
Series\_2:  
 0 -1.637046  
1 0.259139  
2 -0.011552  
3 1.067627  
4 -1.380449  
5 -1.558696  
6 1.527309  
7 0.375792  
8 -2.044987  
9 1.001571  
dtype: float64  
  
Series\_2 ranking : 0 2.0  
1 6.0  
2 5.0  
3 9.0  
4 4.0  
5 3.0  
6 10.0  
7 7.0  
8 1.0  
9 8.0  
dtype: float64  
Ranking \_sorted data:  
 8 1.0  
0 2.0  
5 3.0  
4 4.0  
2 5.0  
1 6.0  
7 7.0  
9 8.0  
3 9.0  
6 10.0  
dtype: float64

**11.Statistics and graph sketches with Pandas:**

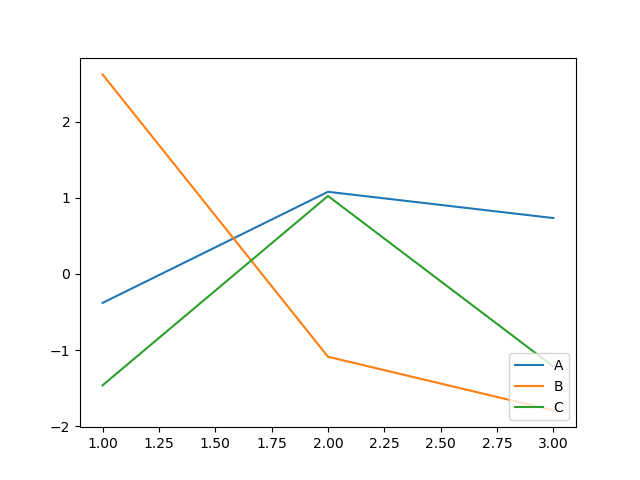
While there is large number of data and need to find patterns between them, for making decisions and scientific analysis the best way is to represent them in form of graph and statistics.

**Code:**

from pandas import Series,DataFrame  
import numpy as np  
from numpy.random import randn  
import matplotlib.pyplot as plt  
# Creating 2-Dimension array having NaN values also  
array\_1 = np.array([[10, np.nan,20],[30,40,np.nan]])  
print(array\_1)  
print()  
# Converting 2-D array list into DataFrame DataStructure  
data\_frame = DataFrame(array\_1,index=[1,2],columns=list('ABC'))  
print("DataFrame :\n",data\_frame)  
print()  
#Arithmetaic operations on DataStructure DataFrame  
  
#Performing Addition on each column elements between two columns in DataFrame  
print("Addition:", data\_frame.sum())  
print()  
# Performing Addition along indexes axis indicate indexs axis=1 means rows  
print("Addition on row elemnts:\n",data\_frame.sum(axis=1))  
print()  
#Printing minimum value from ata DataFrame datasets  
print("Printing minimum values ::\n",data\_frame.min())  
  
  
print()  
#Printing maximum value from DataFrame Datasets elements  
print("Elemnts of Maximum value:\n",data\_frame.max())  
print()  
# Printing index number where element has maximum value  
print("Writing index number of max values:\n", data\_frame.idxmax())  
print()  
#finding cumulative sum which means finding partial sum of given sequences  
print("Cumulative sum:\n",data\_frame.cumsum())  
#Describe function perform all basic stastic operations ata once on data  
print("(Describe)Mathematical Operation on a data at once :\n",data\_frame.describe())  
print()  
#Creating DataFrame using random numbers  
data\_frame1 = DataFrame(randn(9).reshape(3,3),index=[1,2,3],columns=list('ABC'))  
print("Randomly generated DataFrame:\n",data\_frame1)  
# Ploting DataFrame data on Stasticall graph  
plt.plot(data\_frame1)  
# Showing small index-column on graph for more details which is in lower right part  
plt.legend(data\_frame1.columns,loc="lower right")  
# saving the statistical graph in image  
plt.savefig('samplepic.png')  
#showing the same graph  
plt.show()  
#creating Series DataStructure  
series1 = Series(list('abcccaabd'))  
# Printing unique non repeated values from series data structure  
print("Unique values from :\n",series1.unique())  
  
# Printing Counts of Unique values  
print("Counts of unique values:\n",series1.value\_counts())

**OUTPUTE:**

[[10. nan 20.]  
 [30. 40. nan]]  
  
DataFrame :  
 A B C  
1 10.0 NaN 20.0  
2 30.0 40.0 NaN  
  
Addition: A 40.0  
B 40.0  
C 20.0  
dtype: float64  
  
Addition on row elemnts:  
 1 30.0  
2 70.0  
dtype: float64  
  
Printing minimum values ::  
 A 10.0  
B 40.0  
C 20.0  
dtype: float64  
  
Elemnts of Maximum value:  
 A 30.0  
B 40.0  
C 20.0  
dtype: float64  
  
Writing index number of max values:  
 A 2  
B 2  
C 1  
dtype: int64  
  
Cumulative sum:  
 A B C  
1 10.0 NaN 20.0  
2 40.0 40.0 NaN  
(Describe)Mathematical Operation on a data at once :  
 A B C  
count 2.000000 1.0 1.0  
mean 20.000000 40.0 20.0  
std 14.142136 NaN NaN  
min 10.000000 40.0 20.0  
25% 15.000000 40.0 20.0  
50% 20.000000 40.0 20.0  
75% 25.000000 40.0 20.0  
max 30.000000 40.0 20.0  
  
Randomly generated DataFrame:  
 A B C  
1 0.679981 -0.127400 -0.918216  
2 -0.128138 0.260528 -1.458202  
3 0.767245 -0.829230 -0.334639  
  
Unique values from :  
 ['a' 'b' 'c' 'd']  
 Counts of unique values:  
 c 3  
a 3  
b 2  
d 1

****

samplepic.png

**Flowchart for NumPy and Pandas:**

Start

Arithmetic operations and Sorting ndarray

Want Multidimensional arrays?

Creating ,accessing, deleting ndarrays

yes

Import NumPy

Indexing, slicing ndarray

Accessing and deleting data

Quick Plot for visualization?

yes

Import pandas

Arithmetic operations

DataFrame

Pandas has Data Structure?

yes

Dealing with NaN values

Series

Creating panda series and DataFrame

Customized of unique plot?

yes

Import matplotlib

Plotting DataFrame

Data on graph

plt.savefig()

(saving graph)

Stop