**Introduction to Numpy**

Numpy is a module of python. The name is an acronym of “Numerical

Python” or “Numeric Python”. It is a fundamental package in Python used

for scientific computational operations.

**Installation of Numpy-**

* pip install numpy / sudo pip install numpy

**Examples on Numpy-**

* **Creating an array**

**Code :**

import numpy as np

my\_list1 = [1,2,3,4]

my\_list2 = [5,6,7,8]

my\_array = np.array(my\_list1)

my\_array = np.array([my\_list1,my\_list2])

print (my\_array)

print (my\_array.shape) //dimension of an array / list

print (my\_array.dtype) //datatype of the members of the array

new\_array1 = np.zeros(5) //array(1x5). All elements are zero

print (new\_array1)

new\_array2 = np.ones([5,5]) //array(5x5) all elements as 1

print (new\_array2)

new\_array3 = np.eye(5) //array(5x5) as identity matrix

print (new\_array3)

new\_array4 = np.arange(5,50,3)

print (new\_array4)

**Output :**

[[1 2 3 4] //combined array output

[5 6 7 8]]

(2, 4) //dimension of array output

Int64 //dtype of array output

[0. 0. 0. 0. 0.] //zeros() array output

[[1. 1. 1. 1. 1.] //ones() array output

[1. 1. 1. 1. 1.]

[1. 1. 1. 1. 1.]

[1. 1. 1. 1. 1.]

[1. 1. 1. 1. 1.]]

[[1. 0. 0. 0. 0.] // eye() array output

[0. 1. 0. 0. 0.]

[0. 0. 1. 0. 0.]

[0. 0. 0. 1. 0.]

[0. 0. 0. 0. 1.]]

[ 5 8 11 14 17 20 23 26 29 32 35 38 41 44 47] //arrange() output

Above code creates two arrays **my\_list1** and **my\_list2** and combines them to form a single **my\_array** using **np.array(my\_list1,my\_list2).**

**arange()-** used to create an array in sequential order  
**Syntax - a=np.arange(length)**

By default, the array starts from 0 and increments by 1

**syntax- a=np.arange(start,end,increment)**

**shape -** indicates the dimension of an array / list

**dtype -** indicates the type of data stored in an array

**zeros()**- creates an array of specified length with all elements as 0.

**ones()**- creates an array of specified length with all elements as 1.

**eye()**- creates an identity matrix of specified length.

* **Operations on array**

**Code:**

import numpy as np

array1 = np.array([[1,2,3,4],[5,6,7,8]])

print (array1)

array2 = array1\*array1 //multiplication

print (array2)

array3 = array1 \*\* 3 //exponential multiplication

print (array3)

array4 = array1 - array1; //subtraction

print (array4)

array5 = array2 - array1;

print (array5)

array6 = 1/array1 ; //reciprocal

print (array6)

**Output:**

[[1 2 3 4] //combined array output

[5 6 7 8]]

[[ 1 4 9 16] // array1 x array1 multiplication output

[25 36 49 64]]

[[ 1 8 27 64] // power of array1 exp multiplication output

[125 216 343 512]]

[[0 0 0 0] // array1 - array1 subtraction output

[0 0 0 0]]

[[ 0 2 6 12]

[20 30 42 56]]

[[1. 0.5 0.33333333 0.25 ] //reciprocal of array1

[0.2 0.16666667 0.14285714 0.125 ]]

* **Array indexes**

**Code :**

import numpy as np

arr = np.arange(0,12)

print (arr)

print (arr[0:5])

print (arr[2:6])

arr[0:5] = 20

print (arr)

arr2 = arr[0:6]

arr2[:] = 29 //all elements are modified

print (arr2) **// important thing**

print (arr) **// important thing**

array\_copy = arr.copy() //for copying the array

print (array\_copy)

arr2d = np.array([[1,2,3],[4,5,6],[7,8,9]])

print (arr2d)

print (arr2d[0][0])

slice1 = arr2d[0:3,0:2] //slices of 2d array

print (slice1)

arr2d[:2,1:] = 15

print (arr2d)

arr\_len = arr2d.shape[0]

print(arr\_len)

for i in range(arr\_len): //using loops to index

arr2d[i] = i;

print (arr2d);

print (arr2d[[0,2]]) //one more way of accessing the rows

print (arr2d[[1,0]])

**Output :**

[ 0 1 2 3 4 5 6 7 8 9 10 11] //output of arr = np.arange(0,12)

[0 1 2 3 4] //output of print (arr[0:5])

[2 3 4 5] //output of print (arr[2:6])

[20 20 20 20 20 5 6 7 8 9 10 11] //output of arr[0:5] = 20

[29 29 29 29 29 29] //output of arr2= arr[0:6] ; arr2[;]=29

[29 29 29 29 29 29 6 7 8 9 10 11] **// output of the copied array**

[29 29 29 29 29 29 6 7 8 9 10 11] **// output of the original array**

[[1 2 3] //output of the combined array

[4 5 6]

[7 8 9]]

1 //value at the index (0,0)

[[1 2] //slices of the array arr2d[0:3,0:2]

[4 5]

[7 8]]

[[ 1 15 15] //output of the sliced arr2d[:2,1:] = 15

[ 4 15 15] // making it’s value to 15

[ 7 8 9]]

3 // arr2d.shape[0] here 0 for rows 1 for columns

[[0 0 0] //created using loops

[1 1 1]

[2 2 2]]

[[0 0 0] //accessing the rows using arr2d[[0,2]]

[2 2 2]]

[[1 1 1]

[0 0 0]]

In **a:b** here **a-row, b-column**. Accesses all the elements from a to b.

While using Numpy we cannot create the copy of the array implicitly. We have to make the copy of the array using compy() function

**a.copy() -** creates the copy of the array **a** .

In the above code you can see a important thing where you made the changes on **arr2** which are reflected on the original array **arr**. To avoid this type of changes we have to use the copy function.

**arr2d.shape[0]** returns the number of rows in the array whereas **arr2d.shape[1]** returns the number of columns in the array.

* **Universal Array functions**

**Code :**

import numpy as np

A = np.arange(15)

print (A)

A = np.arange(1,15,2)

print (A)

B = np.sqrt(A) //square root

print (B)

C = np.exp(A) //exponential

print (C)

D = np.add(A,B) //add

print (D)

E = np.maximum(A,B) //maximum

print (E)

**Output :**

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

[ 1 3 5 7 9 11 13]

[1. 1.73205081 2.23606798 2.64575131 3. 3.31662479

3.60555128] //square root

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

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

[ 2. 4.73205081 7.23606798 9.64575131 12. 14.31662479 //addition

16.60555128]

[ 1. 3. 5. 7. 9. 11. 13.] //maximum

**sqrt()** - returns the square root

**exp()** - returns e^ to the element where e is the Euler’s number

**add()** - returns the addition of two elements

**max()** - returns the maximum value

* **Saving and loading elements from external libraries**

**Code :**

import numpy as np

arr = np.arange(10)

print (arr)

np.save('saved\_array',arr) //saving single array

new\_array = np.load('saved\_array.npy') //loading single array

print (new\_array)

array\_1 = np.arange(25)

array\_2 = np.arange(30)

np.savez('saved\_archive.npz',x = array\_1,y = array\_2)

//save multiple arrays

load\_archive = np.load('saved\_archive.npz')

//loading the multiple arrays

print ('load\_archive[x] is')

print (load\_archive['x'])

//loading one of the array from multiple arrays

print ('load\_archive[y] is')

print (load\_archive['y'])

np.savetxt('notepadfile.txt',array\_1,delimiter=',')

//saving the array in text file with delimiter **“ , ”**

load\_txt\_file = np.loadtxt('notepadfile.txt',delimiter=',')

//loading the array from text file with delimiter **“ , ”**

print ("load\_txt\_file is")

print (load\_txt\_file)

**Output :**

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

[0 1 2 3 4 5 6 7 8 9] //saved array output after loading

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] //loading one of the array from multiple saved arrays

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]

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.] //loaded from the text file

**np.save()**- saves an array

**np.load()**-loads an array

**np.savez()**- saves multiple arrays

**np.savetxt()**- saves an array in text format

**np.loadtxt()**- loads the array stored in text format

Saving and loading the files from external memory **helps to save the memory**.

* **Conditional clauses, standard functions and boolean expressions**

**Code :**

import numpy as np

x = np.array([100,400,500,600]) //each member 'a'

y = np.array([10,15,20,25]) //each member 'b'

condition =np.array([True,True,False,False]) //each member cond

z = [a if cond else b for a,cond,b in zip(x, condition, y)]

print (z) //loop condition

//np.where(#condition,#value for yes, #value for No)

z2 = np.where(condition,x,y)

print (z2)

z3 = np.where(x>0,0,1)

print (z3)

print (x.sum()) // x sum

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

print (n.sum(0)) // column sum

print (x.mean()) // mean

print (x.std()) // standard deviation

print (x.var()) // variance

condition2 = np.array([True,False,True])

print (condition2.any()) // or operator

print (condition2.all()) // and operator

unsorted\_array = np.array([1,2,8,10,7,3])

unsorted\_array.sort() // sorting in numpy arrays

print (unsorted\_array)

arr2 = np.array(['solid','solid','solid','liquid','liquid','gas','gas'])

print (np.unique(arr2))

print (np.in1d(['solid','gas','plasma'],arr2))

**Output :**

[100, 400, 20, 25] //loop condition output

[100 400 20 25] //output using where

[0 0 0 0] //output of z3

1600 // x sum

[4 6] // column sum

400.0 // mean

187.08286933869707 // standard deviation

35000.0 // variance

True // .any() output

False // .all() output

[ 1 2 3 7 8 10] // sorted array using numpy

['gas' 'liquid' 'solid'] // .unique() output

[ True True False] // .in1d() output

**.in1d()** - returns boolean value if the given element is present in the array

**.unique()** - returns the sorted array removing the repeated elements

**.all()** - returns True when all elements in the given iterable are true

**.any()** - returns True when all elements in the given iterable are true

**.sort()** - returns the sorted array

**.sum()** - retuns the array sum

**.mean() , .std() , .var()** - returns the mean, standard deviation and variance respectively

**.where(condition,Yes,No)** - returns the indices when condition satisfies

**Introduction to Matplotlib**

Matplotlib is a library of python for making 2d plots of arrays. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatterplots, etc., with just a few lines of code

**Installation of matplotlib-**

* pip install matplotlib / sudo pip install matplotlib

**Example :**

* **Plotting a function**

**Code :**

import numpy as np

import matplotlib.pyplot as plt

a=np.arange(-100,100,10)

print(a)

dx, dy = np.meshgrid(a,a) #groups the point with every other point

print("dx ",dx)

print("dy ",dy)

function1=2\*dx+3\*dy

function2=np.cos(dx)+np.cos(dy)

print(function1)

print(function2)

plt.imshow(function1)

plt.title('Function of 2\*dx+3\*dy')

plt.colorbar()

plt.savefig('myfig1.png')

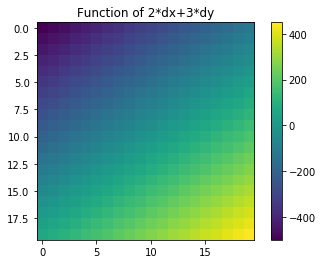
plt.imshow(function2)

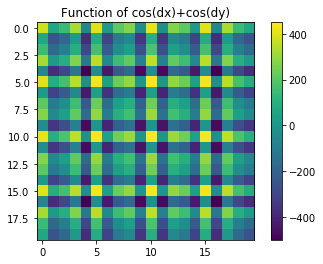
plt.title('Function of Cos(dx)+Cos(dy)')

plt.colorbar()

plt.savefig(‘myfig2.png')

**Output :**





In the above code

**.meshgrid()** - used to create a rectangular grid out of two given dimensional arrays

**imshow()** - used for displaying values/images

**title()** - assigns a title for the result

**colorbar()** - Adds a colorbar to a plot

**savefig()** - saves the plot as a png file

**Introduction to Pandas**

pandas is a software library written for the Python programming language for data manipulation and analysis.

**Installation of Pandas -**

* pip install pandas / sudo pip install pandas

**Examples on Pandas**

* **Series in Pandas**

**Code :**

Import pandas as pd

s=pd.Series([5,10,15,20])

print("Series=",s)

print("Values=",s.values)

print("Index=",s.index)

**Output :**

Series= 0 5 //printing the data and index and data type

1 10

2 15

3 20

dtype: int64

Values= [ 5 10 15 20] // values of the series

Index= RangeIndex(start=0, stop=4, step=1) // indexes of the series

In the above code

syntax- pandas.Series(data,index)

.values()-prints the values of the series

.index()- prints the indexes of the series

We can also create Series by passing an array to it and we can create our own set of indexes

**Code :**

import pandas as pd

data=np.array(['a','b','c'])

s1=pd.Series(data)

print(s1)

s2=pd.Series(data,index=[100,101,102])

print(s2)

s3=pd.Series(data,index=['A','E','I'])

print(s3)

**Output :**

0 a // prints the default index

1 b

2 c

dtype: object

100 a // prints the customized index

101 b

102 c

dtype: object

A a // prints the customized index

E b

I c

dtype: object

We can also use conditions within the series to manipulate the data. We can also convert the series into a dictionary using **.to\_dict()**.

**Code :**

import pandas as pd

revenue=pd.Series([20,80,40,35],index=['Ola','Uber','grap','gojek'])

print(revenue)

print(revenue['Ola'])

print(revenue[revenue>=35])

print( 'Ola' in revenue)

print('Car' in revenue)

revenue\_dict=revenue.to\_dict()

print(revenue\_dict)

**Output :**

Ola 20 //created series

Uber 80

grap 40

gojek 35

dtype: int64

20 // returned the data with respect to the given index

Uber 80 // applying conditions to the given index

grap 40

gojek 35

dtype: int64

True // returns **true** as Ola present in revenue

False // returns **false** as Ola not present in revenue

{'Ola': 20, 'Uber': 80, 'grap': 40, 'gojek': 35} //converted to dictionary

**Code :**

index2=['Ola','Uber','grap','gojek','lyft']

revenue2=pd.Series(revenue,index2)

print(revenue2)

print(pd.isnull(revenue2))

print(pd.notnull(revenue2))

**Output :**

Ola 20.0 // printing the index and data of revenue2

Uber 80.0

grap 40.0

gojek 35.0

lyft NaN

dtype: float64

Ola False // output of **.isnull()**

Uber False

grap False

gojek False

lyft True

dtype: bool

Ola True // output of **.notnull()**

Uber True

grap True

gojek True

lyft False

dtype: bool

**.isnull()**- returns true if null value is present else false.

**.notnull()**- returns true if the value is not present else false.

We can add two series by using **“ + ”** operator and we can also assign names to the series.

**Code :**

index2=['Ola','Uber','grap','gojek','lyft']

revenue2=pd.Series(revenue,index2)

print(revenue2)

print(revenue+revenue2)

revenue2.name="Company Revenues"

revenue2.index.name="Company Name"

print(revenue2)

**Output :**

Ola 40.0 //adding the two series

Uber 160.0

gojek 70.0

grap 80.0

lyft NaN

dtype: float64

Company Name // assigned the names

Ola 20.0

Uber 80.0

grap 40.0

gojek 35.0

lyft NaN

Name: Company Revenues, dtype: float64

* **DataFrame in Pandas**

**Code :**

import pandas as pd

import numpy as np

from pandas import Series,DataFrame

df=pd.read\_clipboard()

print(df)

#access indexes and columns

print(df.columns)

print(df['Industry'])

#multiple columns

print(DataFrame(df,columns=['Rank','Industry','Name']))

#NaN values

df2=DataFrame(df,columns=['Rank','Industry','Name','Profit'])

print("New dataFrame=")

print(df2)

#head and tail

print(df2.head(4)) #prints first 5 rows

print(df2.tail(4)) #prints last 5 rows

#access rows in dataframe

#print(df.ix[0]) #does not work

print(df.iloc[0]) #first row

print(df.loc[5]) #5th row

#assign values to dataframe using numpy

a1=np.array([1,2,3,4,5,6,7,8])

df2['Profit']=a1

print(df2)

#using series

profit=Series([900,100],index=[3,5])

df2['Profit']=profit

print(df2)

#deletion

del df2['Profit']

print(df2)

#use dictionary with dataframe

sample= {

'Company':['A','B'],

'Profit':[1000,5000]

}

print(sample)

dict\_df=DataFrame(sample)

print(dict\_df)

**read\_clipboard()** - read the value for the data frame copied on the clipboard

df=pd.read\_clipboard()

print(df)

**df.columns()** - displays all the columns in the dataframe

We can also access the columns by specifying the name of the columns

#access indexes and columns

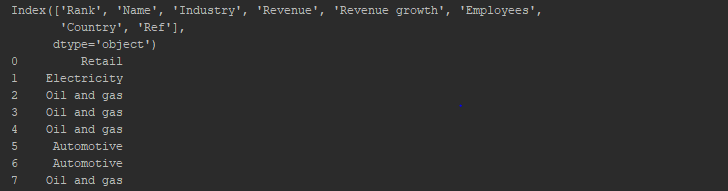
print(df.columns)

print(df['Industry'])

We can also access multiple columns

#multiple columns

print(DataFrame(df,columns=['Rank','Industry','Name']))



**.head(n)** - return the first n values in the dataframe.

**.tail(n)** - returns the last n values in the dataframe.

We can also access individual rows using iloc() or loc() function

#head and tail

print(df2.head(4)) #prints first 5 rows

print(df2.tail(4)) #prints last 5 rows

#access rows in dataframe

#print(df.ix[0]) #does not work

print(df.iloc[0]) #first row

print(df.loc[5]) #5th row

#assign values to dataframe

#using numpy

a1=np.array([1,2,3,4,5,6,7,8])

df2['Profit']=a1

print(df2)

The above code assigns the values of array a1 to the Profit column.

#using series

profit=Series([900,100],index=[3,5])

df2['Profit']=profit

print(df2)

Values 900 and 100 are assigned at the index 3 and 5.

#deletion

del df2['Profit']

print(df2)

**del**- deletes the entire column from the dataframe.

#use dictionary with dataframe

sample= {

'Company':['A','B'],

'Profit':[1000,5000]

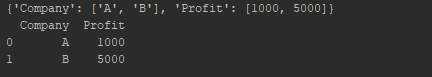
}

print(sample)

dict\_df=DataFrame(sample)

print(dict\_df)

The above code converts a dictionary sample into a dataframe. This can be done by passing dict in the DataFrame()



* **Indexes in Pandas**

s1=pd.Series([10,20,30,40],index=['A','B','C','D'])

print(s1)

index1=s1.index

print(index1)

**.index** – returns the index values

#index operations

print(index1[2])

print(index1[2:])

Returns the value at index 2 and value from index 2 to the last index, respectively.

Negative Indexes

#negative indexes

print(index1[-2:]) #ignores the first 2 elements

print(index1[:-2]) #ignores the last 2 elements

Range of Indexes

print(index1[1:4])

prints all the indexes from 1 to 3(index starts from 0)

Note- index once assigned cannot be changed using = operator.

#create a new series

s1=Series([1,2,3,4],index=['e','f','g','h'])

print(s1)

#creating new indexes using reindex

s2=s1.reindex(['e','f','g','h','i','j'])

print(s2)

**reindex()** - is used to change the index values of Series.

#using fill value

s2=s2.reindex(['e','f','g','h','i','j','k'],fill\_value=10)

print(s2)

#using ffill

cars=Series(['Audi','BMW','Honda'],index=[0,4,8])

print(cars)

ranger=range(13)

print(ranger)

cars=cars.reindex(ranger,method="ffill")

print(cars)

**fill()**- fills in the given value to the new index

**ffill()**- forward fill is used to fill the missing values.

#create new df suing random

#reindex rows of data frame

df1=DataFrame(randn(25).reshape(5,5),index=['a','b','c','d','e'],columns=['c1','c2','c3','c4','c5'])

print(df1)

df2=df1.reindex(['a','b','c','d','e','f'])

print(df2)

**.randn(n)** - provides n random numbers.

**.reshape(m,n)** - reshapes the given values into m rows and n columns.

The above code creates a new dataframe using random values.

We can also re-index the column values using reindex()

#reindex cols

df3=df2.reindex(columns=['c1','c2','c3','c4','c5','c6'])

print(df3)

We can re-index both rows and cols simultaneously by using ix[] or loc[] or iloc[]

df4=df1.loc[['a','b','c','d','e','f'],['c1','c2','c3','c4','c5','c6']]

print(df4)

* **Dropping entries in Series and DataFrame in Pandas**

**Code :**

import pandas as pd

import numpy as np

from pandas import Series,DataFrame

cars=pd.Series(['BMW','Audi','Honda'],index=['a','b','c'])

print(cars)

#drop entries

cars=cars.drop('a')

print(cars)

#df

cars\_df=DataFrame(np.arange(9).reshape(3,3),index=[ 'BMW' , 'Audi' , 'Honda' ] , columns=['rev','profit','expenses'])

print(cars\_df)

#drop rows

cars\_df=cars\_df.drop('BMW',axis=0)

print(cars\_df)

#drop cols

cars\_df=cars\_df.drop('profit',axis=1) #axis=0 for index

print(cars\_df) #axis=1 for columns

drop series values

syntax- **series\_name.drop(‘index\_name’)**

drops the elements of the given index

drop rows and cols in a dataframe

dataframe\_name.drop(‘index\_name’,axis=0) // for index

dataframe\_name.drop(‘column\_name’,axis=1) //for columns

* **Handling null values in Pandas**

**Code :**

import numpy as np

import pandas as pd

from pandas import Series,DataFrame

s1=Series(['A','B','C','D','E',np.nan])

print(s1)

#validate

print(s1.isnull())

#drop for unavailbale values

print(s1.dropna())

df=DataFrame([[1,2,3], [4,5,np.nan], [7,np.nan,10], [np.nan,np.nan,np.nan]])

print(df)

#drona in dataframe

print(df.dropna()) //deletes the entire row that has at least one NaN entry

print(df.dropna(how="all"))

#dropna corresponding to columns

print(df.dropna(axis=1)) //column wise deletion

df2=DataFrame([[1,2,3,np.nan],[4,5,6,7],[8,9,np.nan,np.nan],[12,np.nan,np.nan,np.nan]])

print(df2)

print(df2.dropna(thresh=3)) #should contain at least 3 data values (for rows)

print(df2.dropna(thresh=3,axis=1)) #for cols

#fillna

print(df2.fillna('-')) #fills all the null values by 0

print(df2.fillna({0:0,1:50,2:100,3:200})) #replaces null value in col 1 by 0, col2 by 50, col3 by 100 and col4 by 200

**.isnull()**- returns true if the entry is null else false

**.dropna()**- drops null values

Note- .dropna() drops every row with at least one null entry

**np.nan()**-creates a null entry.

**.dropna(how="all")**- drops only those rows in which all the entries are null.

**.dropna(axis=1)**- drops the elements column wise.

**.dropna(thresh=3)**- drops the row if it contains less than 3 values

**.dropna(thresh=3,axis=1)**- works column wise.

**.fillna(' ')**- fills the null values with the specified value.

* **Accessing the elements of Series and DataFrame in Pandas  
  Code :**

import pandas as pd

import numpy as np

s1=pd. Series([100,200,300],index=['A','B','C'])

print(s1)

#access element of series

print(s1['A'])

#access multiple elements

print(s1[['A','B']])

elements of a series can be accessed by specifying the index of the series. Multiple elements of a series can also accessed by passing multiple index values.

#number indexes

print(s1[0]) //equivalent to s1['A']

#accessing multiple elements

print(s1[0:2])

print(s1[0:4])

#conditional indexes

print(s1[s1>150])

print(s1[s1==300])

We can also pass the number as index instead of index values to save time. Conditional statements can also be used with Series as shown above.

df1=DataFrame(np.arange(9).reshape(3,3),index=['Car','Bike','Cycle'],columns=['A','B','C'])

print(df1)

#col wise

print(df1['A'])

print(df1[['A','B']]) //multiple values

print(df1>5)

print(df1.loc['Bike']) //access elements using ix function

elements of a dataframe can be accessed by in a similar way as that of Series by passing column values.

**df1[]>5**- returns true if the values are greater than 5 else false

* **Alignment of elements in Pandas  
  Code:**

import pandas as pd

import numpy as np

from pandas import Series,DataFrame

s1= Series([100,200,300],index=['A','B','C'])

s2= Series([300,400,500,600],index=['A','B','C','D'])

print("Sum of Series=",s1+s2)

#dataframe

df1=DataFrame(np.arange(4).reshape(2,2),index=['Car','Bike'],columns=['A','B'])

df2=DataFrame(np.arange(9).reshape(3,3),index=['Car','Bike','Boat'],columns=['A','B','C'])

print("Df1\n",df1)

print("Df2\n",df2)

print("Sum of DataFrames\n",df1+df2)

df1=df1.add(df2,fill\_value=0)

print(df1)

s3=df2.iloc[0]

print(df2-s3)

**‘+’** operator when used on the 2 series adds the corresponding indexes of the series. Null values are not considered and output is – **nan**

**‘+’** operator when used on the 2 dataframes adds the corresponding indexes of the series. Null values are not considered as **0**.

* **Ranking and Sorting in Pandas**  
  import numpy as np

import pandas as pd

from pandas import Series,DataFrame

from numpy.random import randn

s1=Series([500,1000,1500],index=['a','c','b'])

print(s1)

#sorting by index

print(s1.sort\_index())

#sort by values

print(s1.sort\_values())

#ranking of series

print(s1.rank())

s2=Series(randn(4))

print(s2)

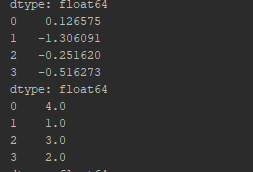
print(s2.rank())

**.sort\_index**- sorts the series by its index value

**.sort\_value**- sorts the series by the value of elements

**.rank()**- first sorts the series by its value(elements) and then assigns rank accordingly

Ex:



* **Statistics in Pandas**   
  from pandas import Series,DataFrame

import numpy as np

from numpy.random import randn

import matplotlib.pyplot as plt

array1 = np.array([[10,np.nan,20],[30,40,np.nan]])

print (array1)

df1 = DataFrame(array1,index=[1,2],columns=list('ABC'))

print (df1)

print (df1.sum()) #sums along each column

print(df1.sum(axis=1)) #sum along indexes

print (df1.min())

print (df1.max())

print(df1.idxmax())

print (df1.cumsum())

print (df1.describe())

df2 = DataFrame(randn(9).reshape(3,3), index=[1,2,3], columns=list('ABC'))

print (df2)

plt.plot(df2)

plt.legend(df2.columns,loc="lower right")

plt.savefig('samplepic.png')

plt.show()

**.min() / .max()** - returns min and max value in each column of the dataframe respectively.

.**idxmax()** - returns the index containing highest value in the dataframe.

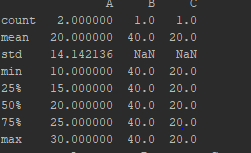
**.cumsum()**- returns the cumulative sum of the columns of dataframe.

**.describe()**- returns values such as count, mean , min, max etc. As shown below

**.plot()**- used to plot the dataframe values.

Plots can be made as **line, bar , box, hist etc**.

Ex-



**.plot(hist)**-plots a histogram

**Legend**- is used to identify each data element uniquely.

**legend(df2.columns,loc="lower right")**- displays the legend object at the lower right corner.

**.show()**-displays the plot.

ser1 = Series(list('abcccaabd'))

print (ser1.unique())

print (ser1.value\_counts())

**.unique()**-prints only the distinct values in the series.

**.value\_counts()**- prints the count of each element in the series.