```
bike.model = 'F21'

bike.model

vehicle.brand
```

numpy - numerical python

1- numpy is used for large group of mathematical operations .

```
import numpy as np

list(range(0,6))

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

list(range(0,6,2))

[0, 2, 4]

#multipy two list

list_1 =[2,4,5,3,8,9,5]
list_2 =[3,6,8,6,0,4,3]
lst =[]

for i in range(0,len(list_1)):
    x= list_1[i]*list_2[i]
    lst.append(x)

print('the product of list are:-',lst)


#or just by import numpy , we can easily product 2 list


import numpy as np


the product of list are:- [6, 24, 40, 18, 0, 36, 15]

x = np.array([2,4,5,3,8,9,5])
y= np.array([3,6,8,6,0,4,3])

print(x*y)

[ 6 24 40 18  0 36 15]

z = np.array([2,4,2,8.7,9])
print(z)                    #here the numpy will change all the
```

```
varible into float as one number is in float
x= np.array([2.5,5.4,3.4,7.8,2.4],dtype= 'int')
print(x)

[2.  4.  2.  8.7 9. ]
[2 5 3 7 2]
```

zeros() - this is use to make one , 2, 3 dimensional matrix in form of array

```
np.zeros((3,5))

array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])

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

array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]])

np.zeros(5)

array([0., 0., 0., 0., 0.])

np.zeros((3,4), dtype='int')

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

ones() - this is also makes matrix with assign number = 1

```
np.ones((2,3,4))

array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

full() - this will fill the matrix with only one number

```
np.full(19,3)
```

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

```
np.full((4,5),3)
```

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

arrange() - same as range it is use to range the values

```
str= np.arange(2,20,2, dtype='float')
```

```
str
```

```
array([ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18.])
```

eye() - used to make unit matrix

```
np.eye(3)
```

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

```
np.eye(5,4)
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 0.]])
```

```
np.eye(4,4, dtype='bool')
```

```
array([[ True, False, False, False],
       [False,  True, False, False],
       [False, False,  True, False],
       [False, False, False,  True]])
```

linspace() - making a relation between the set limits

```
np.linspace(10,50,15) #range is 10-50 and total number between them
will be 15
```

```
array([10.        , 12.85714286, 15.71428571, 18.57142857,
21.42857143,
       24.28571429, 27.14285714, 30.        , 32.85714286,
35.71428571,
       38.57142857, 41.42857143, 44.28571429, 47.14285714,
50.        ])
```

random() - to give the output of random number in a boundry

```
np.random.rand(4) #it will give 4 random number in set 0 to 1
```

```
array([0.89073296, 0.5206371 , 0.02142318, 0.98054205])
```

```
np.random.rand(4,3)
```

```
array([[0.29577613, 0.10065102, 0.96763045],
       [0.26857358, 0.016239  , 0.62018001],
       [0.44163637, 0.77754132, 0.99203536],
       [0.66289988, 0.19094433, 0.30455958]])
```

```
np.random.randn(10)
```

```
array([ 0.35601917, -1.04037304, -2.02288422, -0.01144324, -
0.35440766,
       -1.09017615,  1.06658991,  2.30974064,  0.10283347,
0.4530453 ])
```

```
np.random.normal(4,7,20) #normal(loc=0===mean.0, scale=1.0==standard
distribution, size=None) for
                                  #normal diatribution of  random
variable
```

```
array([ 3.97648251, -8.34884531,  0.08398691, 15.24112814,
12.92395858,
        5.39543781,  6.3740657 ,  4.24955048, -2.8469708 ,
0.33842966,
       -1.99577772,  5.26144628,  5.99947415,  6.12896637,
1.25892439,
       -5.08419039,  4.14197405, 10.54433535, 16.0866632 ,
9.59187682])
```

```
np.random.randint(5,8,(6,4)) #5 is the low level , 8 is the high level
, 6-4 is array
```

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

properties of numpy

```
 matrix =np.random.randint(0,8,(3,5))
```

```
matrix
```

```
array([[7, 6, 4, 3, 4],
       [4, 2, 7, 0, 7],
       [0, 4, 7, 4, 6]])
```

```
matrix.ndim  #ndim gives the dimension like it is of 2 dimensions
```

```
2
```

```
matrix.shape #shape will give the matrix
```

```
(3, 5)
```

```
matrix.size  # size give the total no of elements
```

```
15
```

```
matrix.dtype
```

```
dtype('int32')
```

RESHAPE

```
# reshape


matrix = np.arange(1,16)
matrix

array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])

matrix.reshape(5,3)

array([[ 1,  2,  3],
       [ 4,  5,  6],
       [ 7,  8,  9],
       [10, 11, 12],
       [13, 14, 15]])

matrix.reshape(3,5)

array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])

array = matrix.reshape((1,15))
array

array([[ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]])

array.ndim

2

matrix.ndim

1
```

MAX() - WILL give the largest number
ARGMAX() - will givethe index of largest number

```
mat = np.random.randint(1,34,10)
mat

array([24, 30, 30,  8,  4,  2, 28, 23, 33, 16])

mat.max()

33
```

```
mat.min()

2

mat.argmin()

5

mat.argmax()

8

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

array([[98, 88, 45,  5, 72],
       [ 7, 76, 46, 83, 39],
       [45, 52, 65, 53, 74],
       [81, 26, 99, 73, 43],
       [ 6, 45, 60, 55, 68]])

matr.max()

99

matr.argmax()

17

matr.argmin()

3
```

concatinate () - to combine 2 array

```
a1= np.array([1,2,3,4,5])
a2 = np.array([5,6,7,8,9])
result = np.concatenate([a1,a2])
result

array([1, 2, 3, 4, 5, 5, 6, 7, 8, 9])

a3 = np.arange(1,7).reshape((2,3))
a3

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

a4 = np.array([[21,22,23,24],[25,26,27,28]])
a4

array([[21, 22, 23, 24],
       [25, 26, 27, 28]])
```

```
np.concatenate([a3,a4], axis=1)

#the size of both the matrix should be same

array([[ 1,  2,  3, 21, 22, 23, 24],
       [ 4,  5,  6, 25, 26, 27, 28]])
```

split()

```
#spliting 1d array

array = np.array([1,2,7,21,4,5,6,733,44,22,32,45])

np.split(array,(3,10))

[array([1, 2, 7]), array([ 21,   4,   5,   6, 733,  44,  22]),
array([32, 45])]

x,y,z = np.split(array,(3,10))

x

array([1, 2, 7])

y

array([ 21,   4,   5,   6, 733,  44,  22])

z

array([32, 45])

np.split(array,3)

[array([ 1,  2,  7, 21]), array([  4,   5,   6, 733]), array([44, 22,
32, 45])]

#2 dimensional array

array_1 = np.arange(1,21).reshape(4,5)
array_1

array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])

np.split(array_1, (2,3))

[array([[ 1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10]]),
 array([[11, 12, 13, 14, 15]]),
 array([[16, 17, 18, 19, 20]])]
```

```
x,y,z =np.split(array_1, (2,3), )
```

```
x
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10]])
```

```
y
```

```
array([[11, 12, 13, 14, 15]])
```

```
z
```

```
array([[16, 17, 18, 19, 20]])
```

```
 #hsplit works for columns
```

```
array_1
```

```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20]])
```

```
np.hsplit(array_1,(2,3))
```

```
[array([[ 1,  2],
        [ 6,  7],
        [11, 12],
        [16, 17]]),
 array([[ 3],
        [ 8],
        [13],
        [18]]),
 array([[ 4,  5],
        [ 9, 10],
        [14, 15],
        [19, 20]])]
```

```
#vsplit for rows
```

```
np.vsplit(array_1,2)
```

```
[array([[ 1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10]]),
 array([[11, 12, 13, 14, 15],
        [16, 17, 18, 19, 20]])]
```

```
np.vsplit(array_1,(2,4))
```

```
[array([[ 1,  2,  3,  4,  5],
        [ 6,  7,  8,  9, 10]]),
 array([[11, 12, 13, 14, 15],
        [16, 17, 18, 19, 20]]),
 array([], shape=(0, 5), dtype=int32)]
```

sort

```
array = np.array([1,2,7,21,4,5,6,733,44,22,32,45])

array

array([  1,   2,   7,  21,   4,   5,   6, 733,  44,  22,  32,  45])

np.sort(array)

array([  1,   2,   4,   5,   6,   7,  21,  22,  32,  44,  45, 733])

 ar =np.random.randint(5,20,(6,4))
ar

array([[ 5,  8, 10, 10],
       [ 5,  5, 16, 17],
       [12,  7,  5, 19],
       [12, 14, 13,  5],
       [16, 12, 11, 14],
       [10, 12,  9, 10]])

sort = np.sort(ar , axis =0) #sorting by column
sort

array([[ 5,  5,  5,  5],
       [ 5,  7,  9, 10],
       [10,  8, 10, 10],
       [12, 12, 11, 14],
       [12, 12, 13, 17],
       [16, 14, 16, 19]])

sort_1 = np.sort(ar ) #sorting by row
sort_1

array([[ 5,  8, 10, 10],
       [ 5,  5, 16, 17],
       [ 5,  7, 12, 19],
       [ 5, 12, 13, 14],
       [11, 12, 14, 16],
       [ 9, 10, 10, 12]])
```

indexing

```
#indexing in 2d


variable = np.array([[2,4,2,7],
                     [9,7,5,0],
                     [4,5,3,8]])
variable
```

```
array([[2, 4, 2, 7],
       [9, 7, 5, 0],
       [4, 5, 3, 8]])
```

```
variable[2,3]   #2 is the row and 3 is the column
```

```
8
```

```
variable[0,0]
```

```
2
```

```
#negitive indexing
```

```
variable[-1,-3]
```

```
5
```

slicing

```
a = np.arange(1,20)
a
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19])
```

```
a[3:12]
```

```
array([ 4,  5,  6,  7,  8,  9, 10, 11, 12])
```

```
a[2:: 3]
```

```
array([ 3,  6,  9, 12, 15, 18])
```

```
#2 dim array
```

```
b= np.arange(20).reshape(4,5)
b
```

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

```
b[2, :]
```

```
array([10, 11, 12, 13, 14])
```

```
b[: ,1]
```

```
array([ 1,  6, 11, 16])
```

```
b[0:4, 0:3]
```

```
array([[ 0,  1,  2],
       [ 5,  6,  7],
       [10, 11, 12],
       [15, 16, 17]])
```

```
b[1::2,1:3]
```

```
array([[ 6,  7],
       [16, 17]])
```

```
#assigning value by indexing
```

```
m = np.arange(1,20)
m
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19])
```

```
m[2]=10
```

```
m
```

```
array([ 1,  2, 10,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19])
```

```
#assigning value by slicing



m[0:6]= 22,33,44,55,66,77
m
```

```
array([22, 33, 44, 55, 66, 77,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17,
       18, 19])
```

```
#assigning values in 2d

array_1 = np.array([[1,2,3],
           [2,4,5],
           [4,5,6] ])
array_1
```

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

```
array_1[2,0]=55
```

```
array_1
```

```
array([[ 1,  2,  3],
       [ 2,  4,  5],
       [55,  5,  6]])
```

```
array_1[: ,2]= 22
```

```
array_1
```

```
array([[ 1,  2, 22],
       [ 2,  4, 22],
       [55,  5, 22]])
```

```
array_1[0 ,:]=11
array_1
```

```
array([[11, 11, 11],
       [ 2,  4, 22],
       [55,  5, 22]])
```

```
array_1[0:2,0:2]
```

```
array([[11, 11],
       [ 2,  4]])
```

```
array_1[0:3,0:2] =0
array_1
```

```
array([[ 0,  0, 11],
       [ 0,  0, 22],
       [ 0,  0, 22]])
```

```
array_1[1]
```

```
array([ 0,  0, 22])
```

```
#fancy indexing in 1d
```

```
fancy = np.arange(1,50,5)
fancy
```

```
array([ 1,  6, 11, 16, 21, 26, 31, 36, 41, 46])
```

```
index = [2,3,5]
fancy[index]
```

```
array([11, 16, 26])
```

```
index = np.array([2,4,7])
index
```

```
array([2, 4, 7])
```

```
fancy[index]
```

```
array([11, 21, 36])
```

```
#fancy indexing in 2d array

array_3= np.zeros((10,10), dtype=int)
array_3
```

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

```
lst1= [2,3,4,5,6]
array_3[lst1]
```

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

numpy operations

```
#comparision operator - > < = != >= <=

array = np.arange(1,10)
array
```

array > 5

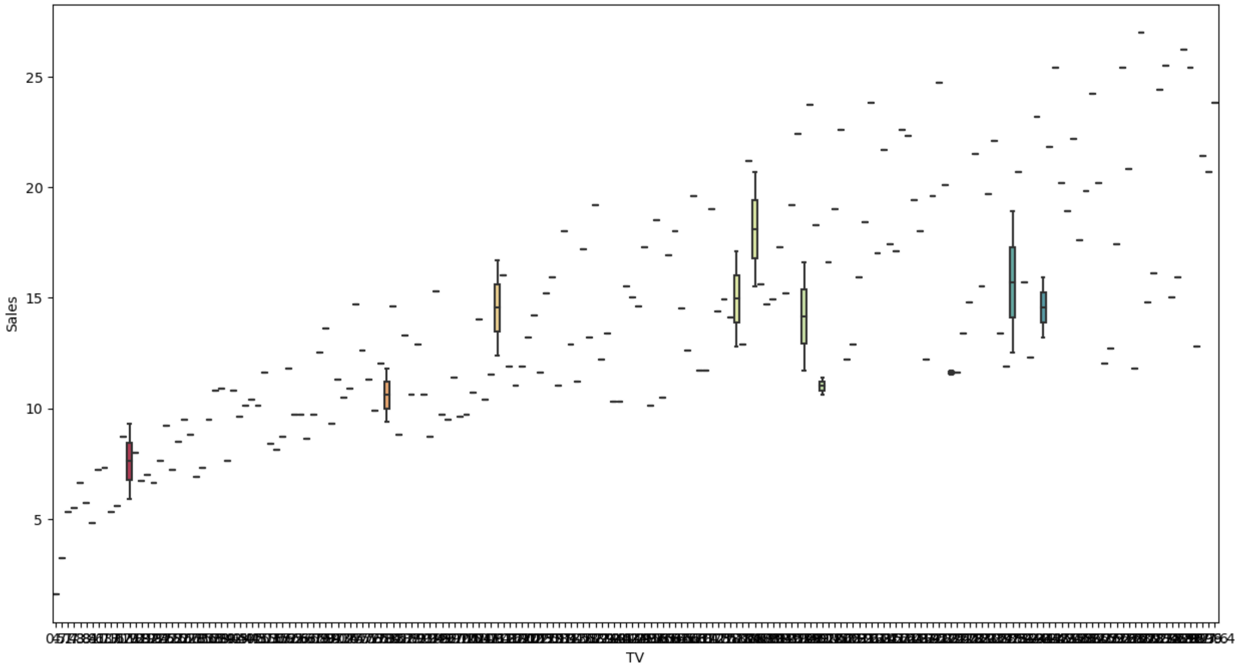
```
array[array > 5]

condition = array >=5
array[condition]

new_condition = (array!=8) & (array>6)
array[new_condition]

new_condition = (array!=8) | (array<6)
array[new_condition]
```

artimetic operation in numpy

```
a = np.arange(1,40)
a

a - 5

np.subtract(a ,5)

a + 5

np.add(a,5)

a /2

np.divide(a,2)

a * 3

np.multiply(a, 3)

((a*10) /2) -6

a *a

a ** 3

np.power(a,3)

a ** .5

np.sqrt(a)

a%5

np.mod(a,5)

np.sin(100)
np.cos(100)

np.log(a)

np.log10(a)

 #https://www.datacamp.com/cheat-sheet/numpy-cheat-sheet-data-analysis-in-python    for all the numpy library
```

aggregate functions

```
array = np.arange(1,27)
array

array([ 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])

np.sum(array)

351

np.mean(array)

13.5

np.median(array)

13.5

np.min(array)

1

np.max(array)

26

np.std(array)

7.5

np.var(array)

56.25

np.sqrt(np.var(array))

7.5
```



NUMPY ARRAY

```
list(range(0,6))

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

#multipy two list

list_1 =[2,4,5,3,8,9,5]
list_2 =[3,6,8,6,0,4,3]
lst =[]

for i in range(0,len(list_1)):
    x= list_1[i]*list_2[i]
    lst.append(x)

print('The product of list are:-',lst)


The product of list are:- [6, 24, 40, 18, 0, 36, 15]

arr1= np.array([1,2,3,4,5])
arr1 # 1d array

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

arr2=([[1,2,3,4,5],[6,7,8,9,1]])
arr2  #2d array
```

```
[[1, 2, 3, 4, 5], [6, 7, 8, 9, 1]]
```

```
arr3 = np.zeros([2,3])
arr3
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

```
np.zeros((2,3,4))
```

```
array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]])
```

```
arr4=np.ones([5,4])
arr4
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

```
np.full((4,5),3)
```

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

```
arr = np.identity(5)
arr  #diagonal matrix
```

```
array([[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.]])
```

```
arr6 = np.arange(5,16)
arr6
```

```
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
arr6 = np.arange(5,16,2)
arr6
```

```
array([ 5,  7,  9, 11, 13, 15])
```

```
arr7 = np.linspace(10,20,13)
arr7
```

```
array([10.        , 10.83333333, 11.66666667, 12.5       ,
13.33333333,
       14.16666667, 15.        , 15.83333333, 16.66666667,
17.5       ,
       18.33333333, 19.16666667, 20.        ])
```

```
np.linspace(10,50,15) #range is 10-50 and total number between them
will be 15
```

```
array([10.        , 12.85714286, 15.71428571, 18.57142857,
21.42857143,
       24.28571429, 27.14285714, 30.        , 32.85714286,
35.71428571,
       38.57142857, 41.42857143, 44.28571429, 47.14285714,
50.        ])
```

```
arr8 = arr7.copy()
arr8
```

```
array([10.        , 10.83333333, 11.66666667, 12.5       ,
13.33333333,
       14.16666667, 15.        , 15.83333333, 16.66666667,
17.5       ,
       18.33333333, 19.16666667, 20.        ])
```

```
np.random.rand(4) #it will give 4 random number in set 0 to 1
```

```
array([0.91545147, 0.25644956, 0.5047637 , 0.96960528])
```

```
np.random.rand(4,3)
```

```
array([[0.50371879, 0.81780023, 0.37006666],
       [0.59406737, 0.63154755, 0.79763987],
       [0.67317268, 0.17251936, 0.71894218],
       [0.53349766, 0.5496025 , 0.70760979]])
```

```
np.random.normal(4,7,20) #normal(loc=0===mean.0, scale=1.0==standard
distribution, size=None) for
                                  #normal diatribution of  random
variable
```

```
array([-0.24789155,  0.832509  , -1.80579103, 11.37987463,
2.12481202,
        4.27784755,  4.01332883, 12.4711903 , 11.89938887,
0.16903856,
       -0.73380088, 15.10091152,  2.15462166, 11.99890203,
0.96025511,
       -6.87767124, -0.91730628, -4.60704586,  5.62724555,
4.81254748])
```

NUMPY PROPERTIES AND ATTRIBUTES

```
arr1

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

arr1.shape

(5,)

arr2

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

arr3.shape

(2, 3)

arr9 = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])
arr9

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

       [[5, 6],
        [7, 8]]])

arr9.shape

(2, 2, 2)

arr9.ndim

3

arr3.ndim

2

arr1.ndim

1

arr1.size

5

arr9.itemsize
```

```
4

arr8

array([10.        , 10.83333333, 11.66666667, 12.5       ,
13.33333333,
       14.16666667, 15.        , 15.83333333, 16.66666667,
17.5       ,
       18.33333333, 19.16666667, 20.        ])

arr8.itemsize

8

# to change the data type
print(arr9.dtype)
arr9.astype('float')

int32

array([[[1., 2.],
        [3., 4.]],

       [[5., 6.],
        [7., 8.]]])
```

NUMPY - INDEXING , SLICING , ITERATION

```
    arr12= np.arange(24).reshape(6,4)
    arr12

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

#the 3rd column

arr12[:, 2]

array([ 2,  6, 10, 14, 18, 22])

#from 3rd to 4th column

arr12[:, 2:4]

array([[ 2,  3],
       [ 6,  7],
       [10, 11],
       [14, 15],
       [18, 19],
       [22, 23]])

#SLICING

#for 9,10
#    13,14


arr12[2:4,1:3]

array([[ 9, 10],
       [13, 14]])
```

INDEXING

```
arr12

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

 for i in arr12:
        print(i)

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

# if we want each item  indiviualy than

for i in  np.nditer(arr12):
    print(i)

0
1
2
3
4
5
6
```

```
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
```

NUMPY OPERATIONS

```
arr1 = np.array([1,2,3,4,5,6])
arr2 = np.array([4,5,6,7,8,9])

#VECTOR ADDITION

arr1+arr2

array([ 5,  7,  9, 11, 13, 15])

#VECTOR MULTIPLICATION

arr1*arr2

array([ 4, 10, 18, 28, 40, 54])

arr1*3

array([ 3,  6,  9, 12, 15, 18])

#comparision



arr2>3

array([ True,  True,  True,  True,  True,  True])

#dot product
```

```
arr3= np.arange(6).reshape(2,3)
arr4 = np.arange(6,12).reshape(3,2)
arr3.dot(arr4)
```

```
array([[ 28,  31],
       [100, 112]])
```

```
#maximum


arr4.max()
```

```
11
```

```
#minimum
```

```
arr4.min()
```

```
6
```

```
arr4
```

```
array([[ 6,  7],
       [ 8,  9],
       [10, 11]])
```

```
arr4.min(axis= 0) # row wise
```

```
array([6, 7])
```

```
arr4.min(axis= 1)   # column wise
```

```
array([ 6,  8, 10])
```

```
arr4.sum(axis=0)  #adding column wise
```

```
array([24, 27])
```

```
arr4.sum(axis=1)  # adding row wise
```

```
array([13, 17, 21])
```

```
np.mean(arr4)        # MEAN
```

```
8.5
```

```
np.std(arr4)         #STANDARD DEVIATION
```

```
1.707825127659933
```

```
np.median(arr4)      # MEDIAN
```

```
8.5
```

```
np.exp(arr4)         #EXPONENT
```

```
array([[  403.42879349,  1096.63315843],
       [ 2980.95798704,  8103.08392758],
       [22026.46579481, 59874.1417152 ]])
```

RESHAPING NUMPY ARRAY

```
arr4.transpose() #transpose
```

```
array([[ 6,  8, 10],
       [ 7,  9, 11]])
```

```
arr4 = np.arange(6,12).reshape(2,3)
arr3,arr4
```

```
(array([[0, 1, 2],
        [3, 4, 5]]),
 array([[ 6,  7,  8],
        [ 9, 10, 11]]))
```

```
np.hstack((arr3,arr4))             #  stacking - to combine 2 array

#hstack = for horizontally stacking
```

```
array([[ 0,  1,  2,  6,  7,  8],
       [ 3,  4,  5,  9, 10, 11]])
```

```
np.vstack((arr3,arr4))             #  stacking - to combine 2 array

#hstack = for vertically stacking
```

```
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

```
arr3
```

```
array([[0, 1, 2],
       [3, 4, 5]])
```

```
np.hsplit(arr3,3)  #HORIZONTAL SPLITTING
```

```
[array([[0],
        [3]]),
 array([[1],
        [4]]),
 array([[2],
        [5]])]

np.vsplit(arr3,2)      #         VERTICAL SPLLITING

[array([[0, 1, 2]]), array([[3, 4, 5]])]

arr7 = np.arange(24).reshape(6,4)   # FANCY INDEXING
arr7

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

arr7[[1,3,4]]      #2nd, 4th , 5th row

array([[ 4,  5,  6,  7],
       [12, 13, 14, 15],
       [16, 17, 18, 19]])

# INDEXING WITH BOOLEAN  ARRAY

arr = np.random.randint(low=1,high=100,size=20).reshape(4,5)
arr

array([[77, 50, 71, 40, 25],
       [35, 28, 63, 40, 38],
       [72, 50, 76, 74, 36],
       [52, 62, 58,  1,  7]])

arr[arr>50]

array([77, 71, 63, 72, 76, 74, 52, 62, 58])

arr[(arr>50) & ( arr%2!=0)]

array([77, 71, 63])
```

BROADCASTING

```
# IN NUMPY ARRAY  DESPITE OF DIFFERENT SIZES OF ARRAY , ADDITION IS
POSSIBLE

a1 = np.arange(4).reshape(4,1)
a2= np.arange(12).reshape(4,3)

a1,a2
```

```
(array([[0],
        [1],
        [2],
        [3]]),
 array([[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11]]))
```

```
a1+a2
```

```
array([[ 0,  1,  2],
       [ 4,  5,  6],
       [ 8,  9, 10],
       [12, 13, 14]])
```