**NumPy**

Why is array faster than list?

In list address of elements are stored inside list but in np. array directly elements are stored.

arr1.shape             # Tells shape of the array.

arr1.itemsize          # Tells the bites space needed for each elements like 8 for integers

arr1.size              # Tells total elements of the array.

arr1.dtype             # Tells data type of the array.

arr1.astype(str)      # Converts all data type inside array -> use at data cleaning to save memory.

arr1.ndim              # Tells the number of dimensions of the array.

arr1.reshape(3,4,3)      # reshapes matrix into other form.

arr1.T                # gives transpose of that matrix.

res = np.where(arr1 > 5 , True , False)  # does masking basis of condition

arr1= np.array([range(1,35,3),range(80,115,3),range(40,75,3)])

arr1 # will generate a range of elements in array just like range

arr2 = np.zeros((4,6),dtype=bool)

arr2 # will generate a matrix of elements in array with ( 4 x 6 ) dimension and all elements as 0.

arr3 = np.ones((2,4),dtype=bool)

arr3 # will generate a matrix of elements in array with ( 2 x 4 ) dimension and all elements as 1.

arr4 = np.array([2,4,3,6,4,7])

arr4 # will generate a 1 D array

arr5 = np.identity(5,dtype=int)

arr5 # will generate a matrix of elements in array with all elements as 0 but diagonal elements as 1.

arr6 = np.arange(1,22,2,dtype=int)

arr6

arr7 = np.linspace(20.0, 60.0, num=4,  endpoint=False)

arr7 # Returns evenly spaced samples, linspace(start, stop, num=50(Default) [ no of elemens required], endpoint=True, retstep=False [ do we need to give any step], dtype=None, axis=0)

%timeit arr1 + arr2   # Calculate time for that operation

np.argmin(arr) # gives minimum elements index

np.argmax(arr) # gives maximum elements index

np.argsort(arr) # gives list of index

np.sum( arr, axis =0) #we can give axis to all aggregate functions

np.split(arr,k) # splits the array into k equal parts if possible

np.array\_split(arr,n) # splits the array into n equal parts and never shows any errors.

np.sort(a, axis = 0) # Will sort the array

**2D**

np.flatten() # converts any dimension array into 1D array

Masking = mask = arr > 5 # creating a mask of true or false values

m1[1, 1] #m1 [**row, column**]

m1[[0,1,2],[0,1,2]] # picking up element (0,0), (1,1) and (2,2) also known as fancy indexing

**Slicing**

Arr[:: , :: ] # first :: will represent rows and second :: represent column

Arr[:: , :: ] [:: , :: ] [:: , :: ] # second [:: , :: ] slices based on first [:: , :: ] result and third slices based on 2nd slices result and so onn. **Carry forward slicing .**

**Indexing and slicing same as list**

arr[:,:2][:2][[1,1,0,1]] #we can use slicing and indexing at same time and indexes can be repeated.

**Masking**

that matrix gets converted into a 1D array after masking because

To retain matrix shape, it **has to retain all the elements**

It **cannot retain its**3×4**with lesser number of elements**

So, this filtering operation **implicitly converts high-dimensional array into 1D array**

If we want, we can reshape the resulting 1D array into 2D

But, we need to know **beforehand** what is the **dimension or number of elements** in resulting 1D array

np.mean(a, axis=1)

**What if we want to find the mean of elements in each row or in each column?**

We can do **same thing with axis parameter** like we did for np.sum() function

Question: Now you tell What will np.mean(a, axis=0) give?

It will give **mean of values in DIFFERENT rows**

**axis = 0 ---> Changes will happen along the vertical axis**

Mean of values will be calculated **in the vertical direction**

Rows collapse/merge when we do axis=0

np.any()

any() returns True if **any of the corresponding elements** in the argument arrays follow the **provided condition**.

What if we want to check whether "all" the elements in our array are non-zero or follow the specified condition?

np.all()

np.where()

Function signature: **np.where(condition, [x, y])**

This functions returns an ndarray whose elements are chosen from x or y depending on condition.

Np.matmul( a , b ) # Matrix multiplication

A @ b # Matrix multiplication

Np.dot( a , b ) # Matrix multiplication

Np.split( array , 3(no.of parts to split) ) will split the array into equal parts

Np.stack ( (arr1 , arr2 ,arr3) , axis = 0 )

Np.concatenate( (arr1 , arr2 ,arr3) , axis = 0 )

Np.tile( array , (3,1) )