**import numpy as np**

**Array method and atribute:-**

Dtype = check data type

Ndim = find array dimension

shape = check array shape dimention

reshape = array dimention change

nditer() = iterate the advance method

( for x in np.nditer(arr) print(x)) any of dimention array

concatenate() = join the array

( np.concatenate(arr + arr1), axis=1)

stack() = join the array

( np.stack(arr + arr1), axis=1)

hstack() = join the array to stack along rows

( np.hstack(arr + arr1))

vstack() = join the array to stack along columns

( np.vtack(arr + arr1))

dstack() = join the array to stack along columns ( np.dtack(arr + arr1))

array\_split() = np.array\_split(arr, 3)

where() = seraching

( np.where(arr == 4))

searchsorted() = (np.searchsorted(arr, 7))

sort() = (np.sort(arr))

**Array random module:--**

random.randint(100)

random.rand()

random.randint(100, size=(5))

random.randint(100, size=(3, 5))

random.rand(3, 5)

random.choice([3, 5, 7, 9])

random.choice([3, 5, 7, 9], size=(3, 5))

random.shuffle(arr) = changing the array index

random.permutation(arr))

sns.distplot([0, 1, 2, 3, 4, 5],hist=False) plt.show()

random.normal(size=(2, 3))

random.normal(loc=1, scale=2, size=(2, 3))

random.binomial(n=10, p=0.5, size=10)

sns.distplot(random.binomial(n=10, p=0.5, size=1000), hist=True,

kde=False) plt.show()

random.poisson(lam=2, size=10)

sns.distplot(random.poisson(lam=2, size=1000), kde=False) plt.show()

sns.distplot(random.normal(loc=50, scale=7, size=1000), hist=False, label='normal')

sns.distplot(random.poisson(lam=50, size=1000), hist=False, label='poisson') plt.show()

Difference Between Normal and Binomial Distribution:-

The main difference is that normal distribution is continous whereas binomial is discrete,

but if there are enough data points it will be quite similar to normal distribution with certain loc and scale.

sns.distplot(random.normal(loc=50, scale=5, size=1000), hist=False, label='normal')

sns.distplot(random.binomial(n=100, p=0.5, size=1000), hist=False, label='binomial')

plt.show()

Difference Between Binomial and Poisson Distribution:-

Binomial distribution only has two possible outcomes, whereas poisson distribution can have unlimited possible outcomes.

But for very large n and near-zero p binomial distribution is near identical to poisson distribution such that n \* p is nearly equal to lam.

sns.distplot(random.binomial(n=1000, p=0.01, size=1000), hist=False, label='binomial')

sns.distplot(random.poisson(lam=10, size=1000), hist=False, label='poisson')

plt.show()

x = random.uniform(size=(2, 3))

sns.distplot(random.uniform(size=1000), hist=False)

x = random.logistic(loc=1, scale=2, size=(2, 3))

sns.distplot(random.logistic(size=1000), hist=False)

plt.show()

Difference Between Logistic and Normal Distribution:-

Both distributions are near identical, but logistic distribution has more area under the tails,

meaning it represents more possibility of occurrence of an event further away from mean.

sns.distplot(random.normal(scale=2, size=1000), hist=False, label='normal')

sns.distplot(random.logistic(size=1000), hist=False, label='logistic')

plt.show()

x = random.multinomial(n=6, pvals=[1/6, 1/6, 1/6, 1/6, 1/6, 1/6])

Relation Between Poisson and Exponential Distribution:-

Poisson distribution deals with number of occurences of an event in a time period

whereas exponential distribution deals with the time between these events.

sns.distplot(random.exponential(size=1000), hist=False)

plt.show()

Chi Square Distribution:-

Chi Square distribution is used as a basis to verify the hypothesis.

It has two parameters:

df - (degree of freedom).

size - The shape of the returned array.

x = random.chisquare(df=2, size=(2, 3))

sns.distplot(random.chisquare(df=1, size=1000), hist=False)

plt.show()

Rayleigh Distribution:-

Rayleigh distribution is used in signal processing.

It has two parameters:

scale - (standard deviation) decides how flat the distribution will be default 1.0).

size - The shape of the returned array.

x = random.rayleigh(scale=2, size=(2, 3))

sns.distplot(random.rayleigh(size=1000), hist=False)

plt.show()

Pareto Distribution:-

A distribution following Pareto's law i.e. 80-20 distribution (20% factors cause 80% outcome).

x = random.pareto(a=2, size=(2, 3))

sns.distplot(random.pareto(a=2, size=1000), kde=False)

plt.show()

Zipf distributions are used to sample data based on zipf's law.

x = random.zipf(a=2, size=(2, 3))

x = random.zipf(a=2, size=1000)

sns.distplot(x[x<10], kde=False)

plt.show()

What are ufuncs:-

ufuncs stands for "Universal Functions" and they are NumPy functions that operate on the ndarray object.

Why use ufuncs:-

ufuncs are used to implement vectorization in NumPy which is way faster than iterating over elements.

They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation.

ufuncs also take additional arguments, like:

where = boolean array or condition defining where the operations should take place.

dtype = defining the return type of elements.

out = output array where the return value should be copied.

What is Vectorization:-

Converting iterative statements into a vector based operation is called vectorization.

It is faster as modern CPUs are optimized for such operations.

example without unfun method:-

x = [1, 2, 3, 4]

y = [4, 5, 6, 7]

z = []

for i, j in zip(x, y):

z.append(i + j)

print(z)

x = [1, 2, 3, 4]

y = [4, 5, 6, 7]

z = np.add(x, y)

How To Create Your Own ufunc:-

To create your own ufunc, you have to define a function,

like you do with normal functions in Python, then you add it to your NumPy ufunc library with the frompyfunc() method.

The frompyfunc() method takes the following arguments:

function - the name of the function.

inputs - the number of input arguments (arrays).

outputs - the number of output arrays.

create:-

def myadd(x, y):

return x+y

myadd = np.frompyfunc(myadd, 2, 1)

print(myadd([1, 2, 3, 4], [5, 6, 7, 8]))

Check if a function is a ufunc:-

print(type(np.add))

print(type(np.concatenate))

print(type(np.blahblah))

if type(np.add) == np.ufunc:

print('add is ufunc')

else:

print('add is not ufunc')

math func :-

newarr = np.add(arr1, arr2)

newarr = np.sum([arr1, arr2])

np.subtract(arr1, arr2)

newarr = np.multiply(arr1, arr2)

newarr = np.divide(arr1, arr2)

newarr = np.power(arr1, arr2)

newarr = np.mod(arr1, arr2)

newarr = np.remainder(arr1, arr2)

newarr = np.divmod(arr1, arr2)

newarr = np.absolute(arr)

arr = np.trunc([-3.1666, 3.6667])

arr = np.fix([-3.1666, 3.6667])

arr = np.around(3.1666, 2)

arr = np.floor([-3.1666, 3.6667])

arr = np.ceil([-3.1666, 3.6667])

Log method:-

arr = np.arange(1, 10)

print(np.log2(arr))

arr = np.arange(1, 10)

print(np.log10(arr))

arr = np.arange(1, 10)

print(np.log(arr))

nplog = np.frompyfunc(log, 2, 1)

print(nplog(100, 15))

x = np.prod(arr)

x = np.prod([arr1, arr2])

newarr = np.prod([arr1, arr2], axis=1)

newarr = np.cumprod(arr)

newarr = np.diff(arr)

newarr = np.diff(arr, n=2)

num1 = 4

num2 = 6

x = np.lcm(num1, num2)

arr = np.array([3, 6, 9])

x = np.lcm.reduce(arr)

x = np.gcd(num1, num2)

x = np.sin(np.pi/2)

arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])

x = np.sin(arr)

arr = np.array([90, 180, 270, 360])

x = np.deg2rad(arr)

arr = np.array([np.pi/2, np.pi, 1.5\*np.pi, 2\*np.pi])

x = np.rad2deg(arr)

x = np.arcsin(1.0)

x = np.arcsin(arr)

base = 3

perp = 4

x = np.hypot(base, perp)

x = np.sinh(np.pi/2)

arr = np.array([np.pi/2, np.pi/3, np.pi/4, np.pi/5])

x = np.cosh(arr)

x = np.arcsinh(1.0)

arr = np.array([0.1, 0.2, 0.5])

x = np.arctanh(arr)

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

x = np.unique(arr)

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

arr2 = np.array([3, 4, 5, 6])

newarr = np.union1d(arr1, arr2)

newarr = np.intersect1d(arr1, arr2, assume\_unique=True)

newarr = np.setdiff1d(set1, set2, assume\_unique=True)

newarr = np.setxor1d(set1, set2, assume\_unique=True)