NumPy

When we deal with numeric data, the library to use is NumPy.

→ **Arrays**

→ arr = np.array([1,2,3], dtype = np.float32)

Int will be upcasted to float and float will be upcasted to string in the case that the array has multiple data types.

np.int8, np.int16, int32, int64
float32, float64
np.bool

we use the .copy() function since references will change original variables

→ a = b.copy()

get the dtype using
→ arr.dtype

Casting
arr = arr.astype(np.float32)

NaN
When we dont want np array to contain a value at a particular index, we use np.nan as a placeholder / filler value.

→ arr = np.array([np.nan, 1, 2])

Note: np.nan cannot take an int dtype

→ infinity can be represented using np.inf & -np.inf
Note: cannot take int as dtype

arange

1 np.arange(n) returns array [0, n)
2 np.arange(m,n) returns array [m, n)
3 np.arange(m,n,s) returns array [m, n) with step s

Linspace

specify number of elements in returned array
arr = np.linspace(5, 11, num=4, endpoint=False,
dtype=np.int32)

reshape

New shape must contain exactly all the elements from the input array.

we can use -1 in at most one dimension
arr = np.arange(8)
reshaped_arr = np.reshape(arr, (2,4))

flatten

flatten an array into 1D array
flattened = arr.flatten()

transpose

```
transposed = np.transpose(arr)
transposed = np.transpose(arr, axes=(0,1,2))
```

axes is the permutation of dimensions

zeros and ones

```
arr = np.zeros(4)
arr = np.ones ((2,3))
arr = np.ones((2,3), dtype=np.int32)
```

```
arr = np.zeros_like(arrog)
```

to create array with same shape as another array

```
arr = np.ones_like(ogarr)
```

Arithmetic

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

```
arr+1   → add 1 to each element
arr-1.2 → subtract each by 1.2
arr*2   → mul each by 2
arr/2
arr//2  → Integer division
arr**2  } pow function
arr**0.5 }
```

eg. def farenToCel (tempe):
        return $\left(\frac{5}{9}\right)$ * (tempe - 32)

farens = np.array([32, 4, 14])
celsius = farenToCel (farens)
Note :- This creates a new array

np.π → pi
np.e → e

Other operations :-
np.exp(arr) →
np.exp2(arr) →
np.log(arr)
np.log10(arr)
np.power(3,arr) → raise 3 to each element in array
np.power(arr2, arr) → raise arr2 to power of each number in arr

matrix multiplication

np.matmul(mat1, mat2)
Note → Will result in ValueError in case of incorrect matrix dimensions

Note → Gives dot product in case of 2, 1-D arrays

Random

np.random.randint(5) → [0, n)
np.random.randint (5, high = 6) → [5, 6)
np.random.randint(-3, high = 14, size = (2, 2))
→ [[5, -3], [10, 10]]

np.random.seed (n) → set the random seed
np.random.shuffle(arr) → randomly shuffle an array. for a matrix, only the rows get shuffled

np.random.uniform()
np.random.uniform(low, high, size)
→ draw sample from uniform distributions

np.random.normal()
np.random.normal(loc, scale, size)
loc → mean
scale → standard deviation
draw samples from normal distribution

custom sampling
arr = [1, 2, 3]
np.random.choice(arr)
np.random.choice(arr, size)
np.random.choice(arr, size, p)
p → probability for each element

Slicing

```
arr = np.array ([1,2,3,4,5])
arr[:] → array ([1,2,3,4,5])
arr[1:] → array ([2,3,4,5])
arr[2:4] → array ([3,4]) → [2,4)
arr[:-1] → array ([1,2,3,4])
arr[-2:1] → array ([4,5])
```

We can use comma seperated values for multi dimensional arrays.

min max

```
np.argmin(arr)  } get min and max values index
np.argmax(arr)  }     from array
```

```
np.argmin(arr, axis)
     → axis → the dimension to run on
```

Note - flattens arrays by default

Filtering

```
arr = np.array ([[0, 2, 3], [1, 3, -6]])
arr == 3
arr > 0
arr != 1
~(arr != 1)
```

Note:- np.nan cannot be used

Use np.isnan to filter for location of np.nan

Where

returns ~~its~~ indices of elements matching the condition.

np.where(arr==3)
x_ind, y-ind = np.where(arr != 0)

~~positive and negative~~
True replacement values and False replacement values can be changed.

np.where(arr, positives, negatives)

Any and all
Any → OR condition
All → AND condition

arr = np.array([-2, -1, 3])
np.any(arr>0) ⇒ True
np.all(arr>0) ⇒ False

axis param can be passed like in argmin and argmax
np.any (arr>0, axis=0) ⇒ rows
np.any (arr>0, axis=1) ⇒ colums

Output is now an array

statistics in NumPy

min & max

arr.min() → Returns least element

arr.max() → returns greatest element

axis can be passed; output will now be an array.

→ np.mean(arr)
→ np.median(arr)
→ np.var(arr) ⇒ variance

} also take axis

Aggregation

np.sum(arr)
axis parameter can be passed
Note will return a flattened array.

np.cumsum(arr) → cumulative sum

np.concatenate([arr1, arr2])
axis param can be passed, default = 0
and concatenates vertically

Saving data

np.save('<filename>.npy', arr)
Note - will overwrite file with same name
Note - .npy will get appended if not in name

Loading

arr = np.load('arr.npy')
Note:- will NOT auto append .npy