URBAN EDGE

NUMPY: package for scientific computing.

→ Provides multi-dim array object, masked arrays & matrices, mathematical, logical, shape manipulation, sorting, selecting and many other opns.

→ Unlike list data structure in python numpy arrays size is a fixed criterion (i.e can't grow dynamically), changing size of an ndarray will create a new array & delete the original.

→ Elements in Numpy array are all req to be of same data type ∴ same size in memory.
exception: we can have arrays of Python, Numpy arrays, thereby allowing for arrays of diff sized elements.

→ NumPy is fast cuz its code is vectorized, (lot of stuff happens behind the scenes) making it more concise & easier to read.

Section 1 Creating Arrays:

→ list_1 = [1,2,3,4,5] → python list
np_arr_1 = np.array(list_1, dtype= np.int8) → convert to numpy array
numpy datatype integer
byte: -128 to 127
(2^8 = 256)
If it had been np.uint8 → unsigned integer
0 to 255
→ 1-D array.

→ md_arr_1 = [[1,2,3], [4,5,6]]
np_md_array_1 = np.array(md_list_1)

→ np. arange (1, 10) → creates array of nos. from 1 to 9
excluded

→ np. linespace (0, 5, 4) → returns array of 4 values equally spaced b/w. 0 & 5.

→ np. zeros (3) → creates an array of 3 zeroes.

→ np. zeros ((2,3)) → 2x3 multi dim array of 0's
→ np. ones ((2,3))

→ np. random. randint (10, 50, 5) → creates an array with 5 random values b/w [10, 50)
excluded

→ np. random. randint (10, 50 (2,3))

Data types :

→ Boolean : np. bool_ → Char : np. byte → short : np. short
→ Integer : np. short_ → Long int : ~~np. lon~~ np. int_ → Float : np. single & np. float32
→ double : np. float64 & np. double → np. int8 : -128 to 127.
→ np. int 16 : -2^{15} : $2^{15}-1$

Section 2 # Slicing & Indexing :

1D → np_arr_1 ([0]) = 6 → ~~setting~~ changing first element in the array from 1 to 6.

nD → np_md_array_1 [0] [1] = 7

other way → np_arr_1 . itemset (0, 7) → setting 1st element of 1D array to 7 np.put (np.arr_1 , [0,1,2] , [10,10,10]) ~~instead~~ changes value of elements at index 0,1,2 to 10

np.take (np.arr_1 , [0,1,2]) extracts val of arr at these indexes

URBAN EDGE

nd → np.md - .itemset((1,1), 7)

→ np_md_arr . size → outputs 6

→ np_md_arr.shape → outputs (rows, cols) = (2,3)

→ np_arr_1 [:5:2] → start at 1st thru 5th with step=2
(this is known as slicing)

→ np_md_arr [:,1] → value at index 1 from each row.

→ np_arr_1 [::-1] → flip the array

→ Get values from array less than 5
np_md_arr [np_md_arr < 5]

ex. array is [[10,2,3]
[10,5,6]
[10,8,9]]

np.unique (array)
= array ([2,3,5,6,8,9,10])

Section 3 # Reshaping Arrays :

→ a = np.arange (6) . reshape (3,2)
→ a = [[1,2], [3,4], [5,6]]

→ np.reshape (a, (2,3))
→ a = [[1,2,3], [4,5,6]]

→ np.reshape (a,6)
→ a = [1,2,3,4,5,6]

→ a.transpose() → matrice transpose

→ a.flatten()

→ np_md_arr . resize (2,5)
→ items are either lost or 0 is added.

np.put (np_arr_1 , [0,1,2] , [10,10,10]) ~~~~ changes value of elements at index 0,1,2 to 10

np.take (np_arr_1 , [0,1,2]) extracts val of arr at these indexes.

ad → np_md - .itemset((1,1), 7)

→ np_md_arr.size → outputs 6

→ np_md_arr.shape → outputs (rows, cols) = (2,3)

→ np_arr_1 [:5:2] → start at 1st thru 5th with step=2
(this is known as slicing)

→ np_md_arr [:,1] → value at index 1 from each row.

→ np_arr_1 [::-1] → flip the array

→ Get values from array less than 5
np_md_arr [np_md_arr < 5]

ex: array is : [[10,2,3]
[10,5,6]
[10,8,9]]

np.unique (array)
= array ([2,3,5,6,8,9,10])

ection3 # Reshaping Arrays :

→ a = np.arange (6) . reshape (3,2)
→ a = [[1,2], [3,4], [5,6]]

→ np.reshape (a, (2,3))
→ a = [[1,2,3], [4,5,6]]

→ np.reshape (a,6)
→ a = [1,2,3,4,5,6]

→ a.transpose() → matrice transpose

→ a.flatten()

→ np_md_arr.resize (2,5)
→ items are either lost or 0 is added

URBAN EDGE ___/___/___

→ np. nd_arr. sort (axis = 1) → sort rows

→ np nd. arr. sort (axis = 0) → sort cols. (accross rows)

section 4 # Stacking & splitting :

→ np. vstack ((arr1, arr2)) (vertical stack) (no. of cols must be same)

→ np. hstack " " (horizontal stack) (no. of rows must be sa)

→ np. delete (arr1, 1, 0) (delete first row)

axis = 0 for row

[1,3,5]

→ np. delete (arr1, ~~5~~, none) [np. delete () returns array after deletion)

eg arr1 = [[1, 2, 3, 4]
[5, 6, 7, 8]]

now axis = none
means np. delete (1D array version of arr1)
([1, 2, 3, 4, 5, 6, 7, 8])

element on indexes 1, 3, 5 deleted

→ np. hsplit (arr3, 5) lets say arr3 is a 2 x 10 array

split into 5 arrays.

→ np. hsplit (arr3, (2, 4)) split after 2nd & 4th column

sections # Copying.

→ cp_arr_1 = np. zeros ((2,2))
→ cp_arr_2 = cp_arr_1
→ cp_arr_1 [0,0] = 1 → even [0,0] pos for copy arr 2 will chan to 1.

Section 6 # Math

→ we can perform normal addn, subtraction mult & division on 2 or more numpy arrays.
* (shape must be same) *

→ arr1 = np.array([1, 2, 3, 4])

→ arr2 = np.array([5, 6, 7, 8])

arr1 + arr2 ✓ arr1 - arr2 ✓ arr1 x arr2 ✓

arr1 / arr2 ✓

→ If it encounters a division by 0 it throws up a runtime warning. (not error)

→ arr1.sum() → sum of all elements in arr1)

→ arr3 = ([1, 2, 3], [2, 4, 9], [7, 7, 6])

arr3.min(axis = 1)

gives [1, 2, 6] (across rows)

arr3.min (axis = 0)

gives [1, 2, 3] (across cols)

→ np.power (arr1, arr2) raises all element in arr1 to the power of elements of arr2.

→ np.exp (arr2) exponential of all element in array 2.

→ np.log2 (arr2) log of all " "

→ np.floor (arr2)

→ np.ciel (arr2)

Section 7 # Reading files → Very imp gonna use them multiple times

→ var_name = pd.csv (file name).to_numpy()

converts table of data in file to numpy arrays

→ we can also ~~read~~ read the data using numpy
from numpy import genfromtext

→ var_name = genfromtxt (file, delimeter = ';')

→ we can also remove NaNs (not a number) values
var_name = [row [~np.isnan(row)] for row in var_name]

lecture 8 # Statistics functions

→ np.mean(arr)
→ np.median(arr)
→ np.average (arr)
same ofe
→ np.std (arr)
→ np.var (arr)

→ np.corrcoef ($\underbrace{\text{var\_name [:, 0]}}_{\text{first col}}$, $\underbrace{\text{var\_name [:, 1]}}_{\text{2nd col.}}$)

↓

correlation coefficient (more its closer to 1 more correlated it is)

→ ~~Calc~~ regression line $\frac{\Sigma(x-\bar{x})\cdot(y-\bar{y})}{\Sigma(x-\bar{x})}$ = slope.

→ NumPy can be used for many more things:
→ Trigonometric calculation
→ Matrix Multiplication
→ Dot pdts, inner pdts
→ Tensor dot pdts etc