NumPy

NumPy

NumPy = Numerical Python

- a library for working with array of data
- very memory efficient in handling big array of numbers
  - particularly fast with numerical computation
- provide a lot of supporting functions/methods compared to Python built-ins
- used extensively in data science and AI developments

Many other libraries use NumPy under the hood

- E.g., Scikit-learn, Pandas, TensorFLow etc

Aside: Installation of NumPy

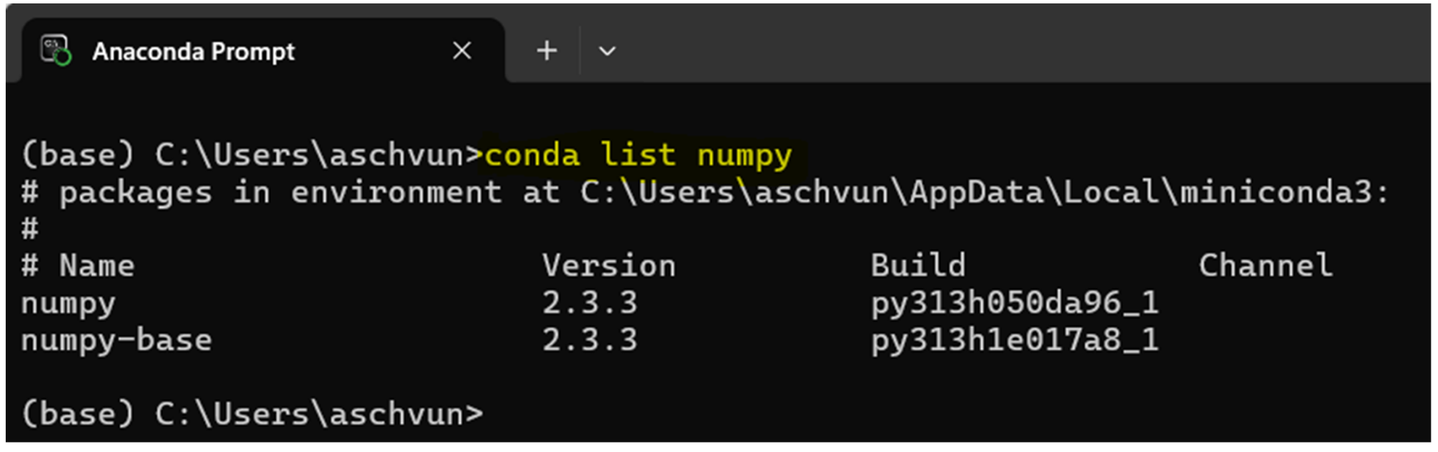
NumPy is not a built-in module in standard Python package

- it is a 3rd part library which normally need to be installed separately
  - e.g. using **`pip install numpy`**

```
(base) C:\Users\aschvun>pip show numpy
Name: numpy
Version: 2.3.3
Summary: Fundamental package for array computing in Python
```

However, other python distribution or package may include NumPy as a pre-installed package

- e.g. Jupyter Lab, Anaconda



NumPy Array

NumPy uses a data structure call ndarray (N-Dimensional array) to store data

To use the NumPy module: use **import** (and usually we alias it to np)

```
import numpy as np
```

We can then use its **np.array()** function to convert other data to NumPy array

Example: Converting a list to numpy array

```
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)
```

```
print(type(arr1))
```

Creating Array

`linspace()` creates an instance of array with evenly spaced values

```
new_array=np.linspace(0,11,12)
print(new_array)
```

`arange()` creates an instance of array (excluding the endpoint)

```
new_array=np.arange(start=1, stop=12, step=2)
print(new_array)
```

But Multi Dimension array (e.g. Matrix) are commonly used to store the datasets used in many ML applications.

Multi Dimension Array

To create a new array of specific shape (e.g. a 3x3 matrix)

```
new_arr = np.empty([3,3])
print(new_arr)
```

- the content of the created array will be random.

To create a new array that is filled with zeros or ones

```
new_arr = np.zeros([3,3])
print(new_arr)
```

```
new_arr = np.ones([3,5])
print(new_arr)
```

Array Applications

Use case of Matrix:

Find the Taxi fare charging model from Changi Airport based on the following data

- $25: 10km distance and 30 minutes duration
- $20: 8km distance and 20 minutes duration
- $37: 15km distance and 50 minutes duration

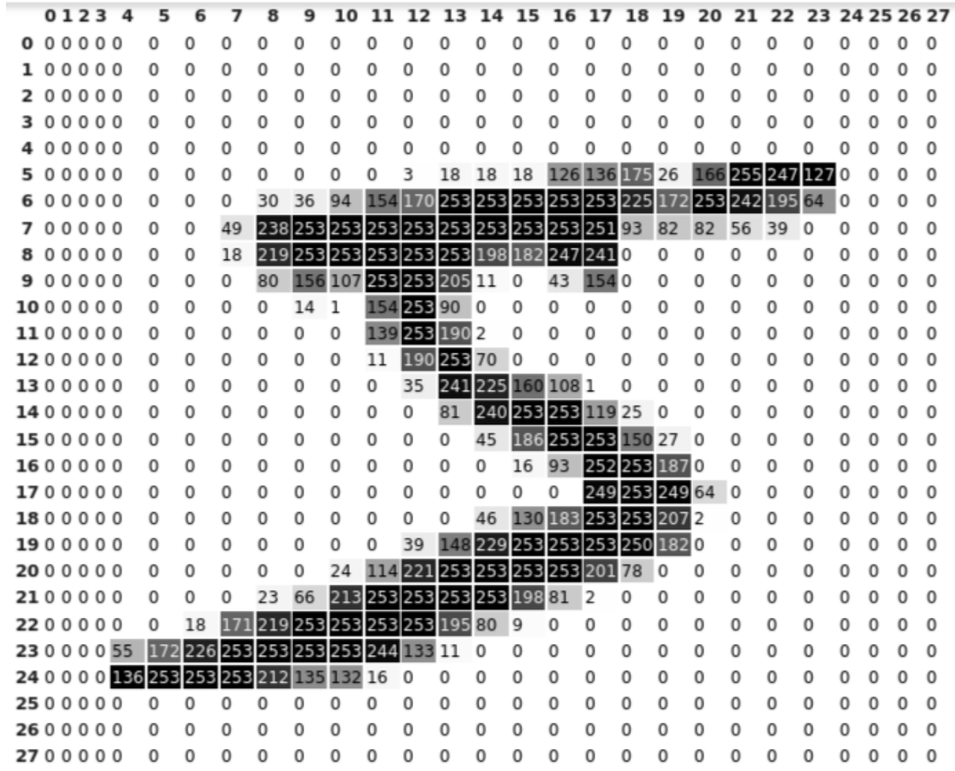
We can present the problem using array: $\begin{bmatrix} 25 \\ 20 \\ 37 \end{bmatrix} = \begin{bmatrix} 10 & 30 & 1 \\ 8 & 20 & 1 \\ 15 & 50 & 1 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$

In ML, we say that the 3x3 matrix contains the feature values of the model

ML Applications

In ML, dataset are stored using NumPy arrays.
Example: a grey scale digit 5 (from the MNIST dataset)

- 28x28 pixels image
  - i.e. can be stored as a 28x28 array
- each entry (pixel) is hence a feature of the digit 5



Finger Exercise: Create a zeros ndarray that can be used to store the features of a color photo such as the following.

```
color_arr = np.zeros([3,28,28])
color_arr
```

```
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., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]], shape=(3, 28, 28))
```

Creating array from List

If your dataset is already loaded as a list

- you can change it from list to array using **`np.array()`**

```
data = [11, 22, 33, 44, 55]
arr = np.array(data)
print(type(data), type(arr))
```

```
data = [[1, 2],
        [3, 4],
        [5, 6]]
arr = np.array(data)
print(type(data), type(arr), arr.shape)
```

Array Dimension

Array can have different level of depth: Dimension

- to find the dimension, use shape method

```
arr1 = np.array([1, 2, 3, 4, 5])
print(arr1)
print(arr1.shape)   #get the shape of array
```

A 2-D array is an array with 1-D arrays as its element:

```
arr2 = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print("arr2 -> ndim:", arr2.ndim)
print("arr2 -> shape:", arr2.shape)
print("arr2 -> size:", arr2.size)
print("arr2 -> dtype:", arr2.dtype)
```

A 3-D array is an array with2-D arrays as its elements

```
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr3)
print(arr3.shape)   #get the shape of array
```

Accessing array (by Index)

```
arr2 = np.array([[1, 2, 3], [7, 8, 9], [10, 11, 12]])
print(arr2)
print(arr2.shape)
print(arr2[0, 1])
```

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

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr)
print(arr[1, 1:4])
```

```
arr = np.array([1, 2, 3])
for x in arr:
  print(x)
```

Data type within the Array

To check the data type of the element in the array: dtype

```
arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])
print(arr.dtype)
```

```
arr = np.array(['apples', 'banana', 'cherry'])
print(arr)
print(arr.dtype)
```

- `i` - integer
- `b` - boolean
- `u` - unsigned integer
- `f` - float
- `c` - complex float
- `m` - timedelta
- `M` - datetime
- `O` - object
- `S` - string
- `U` - unicode string

Data Type and Copy

To specify a specific data type: dtype

```
arr = np.array([10, 2, 3, 4], dtype='S')
print(arr)
print(arr.dtype)
print(arr[2])
```

‘b’ for ByteString - 8-bit encode characters

To make a copy: copy

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
newarr = arr.copy()   # make a copy
arr[0] = 42   # change the value
print(arr)
print(newarr)
```

Data Type and Copy (cont.)

To make a copy and specify the data type: astype

```
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype(int)  #change to integer

print(newarr)
print(newarr.dtype)
```

```
arr = np.array([1, 0, 3])
newarr = arr.astype(bool)  #change to boolen True/False
print(newarr)
print(newarr.dtype)
```

Changing the Array Shape

To reshape the dimension of the array: reshape

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)  #reshape the dimension
print(newarr)
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
ewarr = arr.reshape(2, 3, 2)
print(newarr)
```

```
arr = np.array([[1, 2, 3], [4, 5, 6]])
newarr = arr.reshape(-1)   #flatten the array to 1D
print(newarr)
```

```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
newarr = arr.reshape(2, 2, -1)  #let numpy figure out the dimension
print(newarr)
```

NumPy Array Manipulation

Manipulate array methods

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
newarr = np.concatenate((arr1, arr2))  #join 2 arrays
print(newarr)
```

```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
newarr = np.concatenate((arr1, arr2), axis=1) # join based on row
print(newarr)
```

```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
newarr = np.stack((arr1, arr2), axis=1)  #stack on top of each other
print(newarr)
```

```
print(np.vstack((arr1,arr2)))
```

```
print(np.hstack((arr1,arr2)))
```

NumPy Array Manipulation

Separate array into different sets

```
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 3)   #slit into multiple array
print(newarr)
```

```
arr = np.array([1, 2, 3, 4, 5, 6])
newarr = np.array_split(arr, 4)
print(newarr)
```

```
data = np.array([[1, 2, 3],
                 [4, 5, 6],
                 [7, 8, 9],
                 [10, 11, 12]])
# separate data in to 3 groups
train, val, test = data[:2,:],data[2:3,:],data[3:,:]
print(train)
print(val)
print(test)
```

NumPy Arithmetic

Arithmetic operations

```
x = [1, 2, 3, 4]
y = [4, 5, 6, 7]
z = np.add(x, y)  # add the two array
print(z)
```

```
arr1 = np.array([10, 20, 30, 40, 50, 60])
arr2 = np.array([20, 21, 22, 23, 24, 25])

newarr = np.subtract(arr1, arr2)
print(newarr)
newarr = np.multiply(arr1, arr2)
print(newarr)
newarr = np.divide(arr1, arr2)
print(newarr)
```

```
arr = np.around(3.1666, 2)  #round to two decimal
print(arr)

arr = np.trunc([-3.1666, 3.6667])  #remove the decimal
print(arr)
```

Searching the Array Element

To search for an entry

```
arr = np.array([7,2,3,4,5,3,8,9,3,8])
x=np.where(arr==3)
print(x)
```

Result : a tuple (array([2, 5, 8],)
which means that the value 3 is present at index 2, 5, and 8.

NumPy Sorting

Sorting the elements in the array

```
arr = np.array([3, 2, 0, 1, 5])
print(np.sort(arr))
```

Sort and Insert an element

```
arr = np.array([3, 7, 1, 5, 6])
arr=np.sort(arr)
print(arr)
x = np.searchsorted(arr, 4)  # where should  number 4 be inserted?
print(x)
np.insert(arr, x, 4)  #insert the number 4
```

NumPy Filtering

Filtering

```
arr = np.array([7, 9, 12, 22, 26])
x = [True, False, True, False, True]   #filtering
newarr = arr[x]
print(newarr)
```

```
arr = np.array([7, 12, 9, 22, 26])
filter_arr = arr > 10
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

```
[False  True False  True  True]
[12 22 26]
```

```
arr = np.array([7, 9, 12, 21, 26, 47, 59])
filter_arr = arr %2 == 1    #odd number
newarr = arr[filter_arr]
print(filter_arr)
print(newarr)
```

NumPy Random

Run the following a few times

```
from numpy import random
x = random.randint(100)  #generate a random number from 0 to 100
print(x)
```

To generate array with random elements

```
x=random.randint(100, size=(5))   # generate a 1-D randowm array
print(x)
```

```
x = random.rand(5) # generate floating point array
print(x)
```

```
x = random.rand(3, 5)
print(x)
```

```
array([[0.1369241 , 0.63514555, 0.45892327, 0.86226227, 0.07055414],
       [0.14154741, 0.16044108, 0.6317394 , 0.52230413, 0.41127843],
       [0.91332844, 0.87400771, 0.18259781, 0.85969548, 0.24263689]])
```

NumPy Random

Select randomly: choice

```
x = random.choice([3, 5, 7, 9])  #select 1 from the array
print(x)
```

```
x = random.choice([3, 5, 7, 9], size=(3, 5))  #generate a 3-D random array
print(x)
```

Generate a normal distribution data with mean of 170 and SD = 10, 250 points

```
x = np.random.normal(170, 10, 250)
print(x)
```

NumPy Statistical Analysis Functions

NumPy provides the following functions for statistical analysis

```
data=np.arange(1,22,3)
print(data)
print(np.mean(data))
print(np.average(data))
print(np.median(data))
print(np.min(data))
print(np.max(data))
print(np.ptp(data))
print(np.var(data))  #peak-to-peak range
print(np.std(data))
print(np.percentile(data,[25,75]))  #25th and 75th percentile
```

Normalization

In ML, features values are usually scaled to a common range (e.g. 0 to 1) for more effective processing

```
data = np.random.normal(170, 10, 250)
normalized_data = (data - np.min(data)) / (np.max(data) - np.min(data))
print(normalized_data)
```

```
array([200.12360843, 180.74243719, 174.3672282 , 165.64259316,
       159.24032182, 179.17913948, 147.47114674, 172.93253567,
       168.15021992, 162.42760724, 186.32978738, 177.33334151,
       166.31626755, 178.6777297 , 162.60364618, 169.74016959,
       156.56756682, 169.28155224, 175.46283575, 178.05984737,
       163.01257759, 168.28686855, 172.63396313, 177.21865612,
       156.53509509, 176.91871939, 179.53058595, 166.95741269,
       169.18433671, 168.42594352, 143.9395602 , 164.39076968,
```

```
array([0.96588686, 0.63509706, 0.52628764, 0.37737918, 0.26810787,
       0.60841534, 0.06723646, 0.5018009 , 0.42017832, 0.32250713,
       0.73045964, 0.57691203, 0.38887718, 0.59985749, 0.32551169,
       0.44731492, 0.22249039, 0.43948743, 0.54498701, 0.58931173,
       0.33249116, 0.42251058, 0.49670499, 0.57495463, 0.22193618,
       0.56983543, 0.61441369, 0.39981998, 0.43782819, 0.42488425,
       0.00696081, 0.35601358, 0.21431645, 0.58560599, 0.57029432,
       0.29501768, 0.49290994, 0.14929607, 0.52399619, 0.48454804,
```

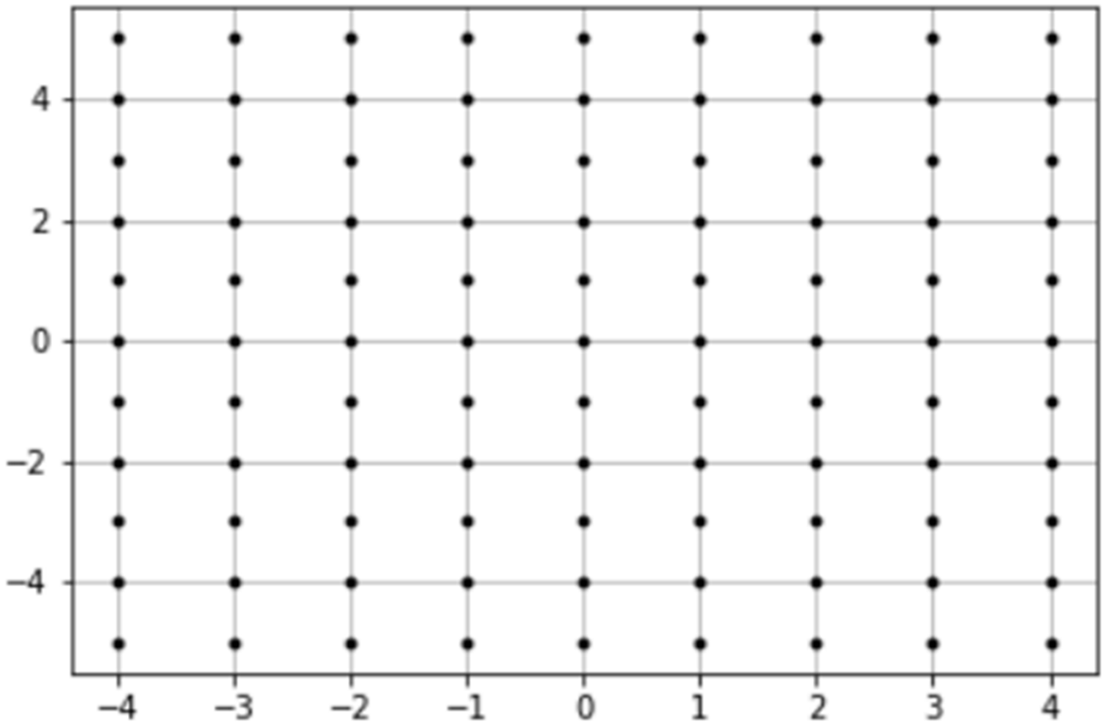
Coordinates Function

`meshgrid()` function

- generate two 2-Dimensional arrays representing the X and Y coordinates of all the points:

```
x = np.linspace(-4, 4, 9)
y = np.linspace(-5, 5, 11)

X, Y = np.meshgrid(x, y)
print(X)
print(Y)
```



Mathematical Functions

NumPy also provides many mathematics functions

Trigonometry functions:

```
np.sin(np.pi/2)
```

```
np.cos(np.pi/4)
```

```
np.rad2deg(np.pi)
```

```
np.deg2rad(180)
```

Euler function:

```
np.exp(1) #e^1
```

Logarithm function:

```
np.log10(100) #base 10 log
```

```
np.log(np.exp(1)) #natural log
```

NumPy Linear Algebra Functions

NumPy also provides many mathematics functions for linear algebra operations

Solving linear equations (finding w) : Aw = B

```python
# Define matrix A and vector B
A = np.array([[3, 1], [1, 2]])
B = np.array([9, 8])
# Solve the system of linear equations Ax = B
x = np.linalg.solve(A, B)
print("Solution to Ax = B:", x)
```

Finger Exercise

Find the formula used in the charging model of Taxi fare

Given: Taxi fare charging model from Changi Airport

$25: 10km distance and 30 minutes duration

$20: 8km distance and 20 minutes duration

$37: 15km distance and 50 minutes duration

- represented as $\begin{bmatrix}25\\20\\37\end{bmatrix} = \begin{bmatrix}10 & 30 & 1\\8 & 20 & 1\\15 & 50 & 1\end{bmatrix} \begin{bmatrix}w_0\\w_1\\w_2\end{bmatrix}$ (i.e. the w_x values)

Express it as B = Aw

```
import numpy as np
A = np.array([[10, 30, 1],
              [ 8, 20, 1],
              [15, 50, 1]])
```

```
B = np.array([25,
              20,
              37])
```

```
w = np.linalg.solve(A, B)
```

```
w
```

```
array([2. , 0.1, 2. ])
```