Learn ai

Introduction to Numpy

Lesson 2: Data Manipulation & Processing

An AI Commons initiative

AI

What is Numpy?

1. Open-source library in Python
2. Tools for data manipulation
3. Known for its multidimensional array and matrix data structures
4. Efficient and fast mathematical operations on arrays

Installing Numpy

1. Open up Terminal or Command Line
2. To install numpy, type:

```
$ pip3 install numpy
```

3. To import numpy in Python, type:

```
$ import numpy as np
```


Numpy comes pre-installed in Colab and Jupyter notebooks

Why use Numpy arrays over Python lists?

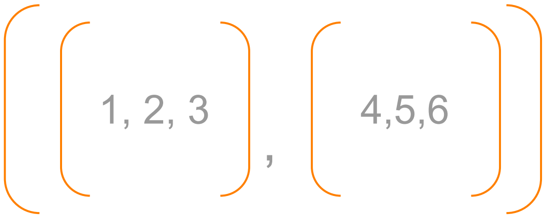
- Python lists can represent arrays, so why use Numpy arrays?

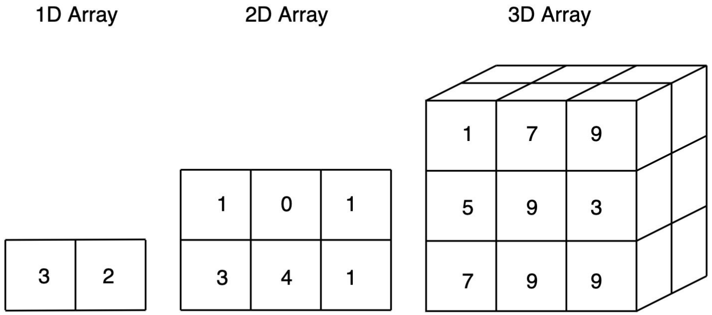
Numpy arrays properties:

- Faster methods
- Are more memory efficient
- Can only have homogenous elements (every element has the same type! Ex. only integers)

What is a Numpy Array?

A Numpy array can be visualized as a grid of values (even looks like one!) where each item is the same type.





Rank and Shape of Array

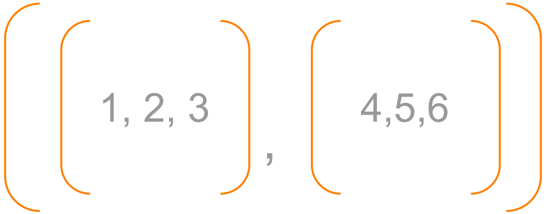
The Numpy array can also be understood by its ***rank*** and ***shape***:

Rank:

how many dimensions (or levels of nesting) it contains (default is rank 1, meaning one list!)

Shape:

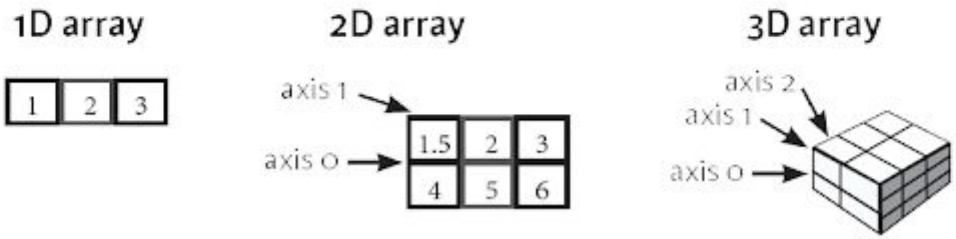
tuple of integers that give “shape” (grid-like, Cartesian) to the rank/dimension



Types of arrays

Arrays sometimes called “ndarray”, or “n-dimensional array”

- 1-D array is a Vector
- 2-D array is a Matrix



Creating arrays: Part 1

- np.array() creates an array
- Passing a list into np.array() creates an array from that list

Creating arrays: Part 2

Here, we are nesting two lists into one list to form a matrix (multi-dimensional array)

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

1	2
3	4

Creating arrays: Part 3

Create an array of zeros

- **Input:** np.zeros(2)
- **Output:** array([0., 0.])

Create an array of ones

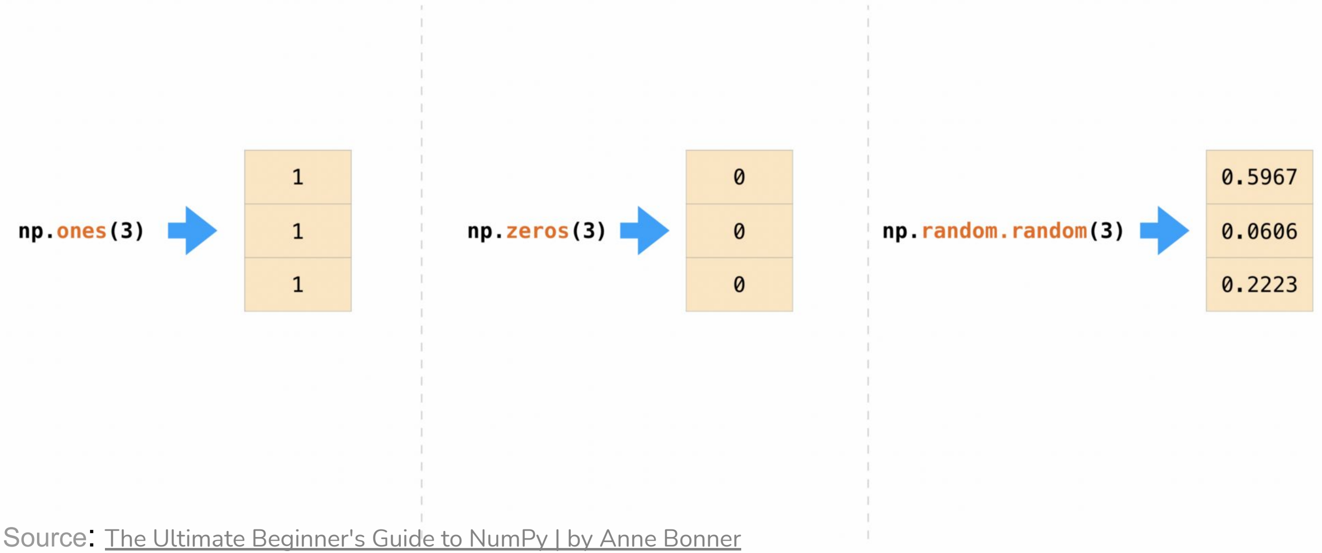
- **Input:** np.ones((2,2))
- **Output:** array([[1., 1.], [1., 1.]])

Create an array in a range

- **Input:** np.arange(4)
- **Output:** array([0, 1, 2, 3])

Creating arrays: Part 4

As you can see, you can explicitly specify the type of values we want within an array:



Source: The Ultimate Beginner's Guide to NumPy | by Anne Bonner

Manipulating arrays

Adding, removing and sorting

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

Adding to array:

np.append(arr, [4,5])

array([1,2,3,4,5])

Deleting item in array by index:

np.delete(arr, 1)

array([1, 3])

Sorting array:

np.sort(arr)

array([1,2,3])

Note: np.append() is actually quite slow, since it has to create and return a whole new array

Shape and size of an array

- **Ndarray.ndim:** Gives the # of axes in an array
- **Ndarray.size:** Gives the # of elements in an array
- **Ndarray.shape:** Gives a tuple indicating # of elements per dimension

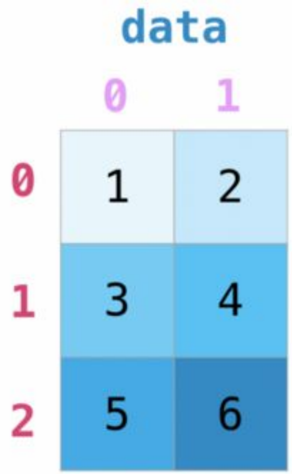
Ex.

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

data.**ndim** = 2

data.**size** = 6

data.**shape** = (2, 3)

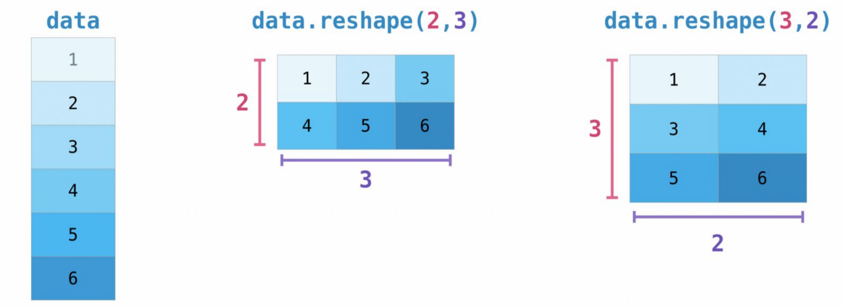


Reshaping arrays

We can “reshape” the structure/dimensions of an array as long as the elements (items) inside carry over to the new list:

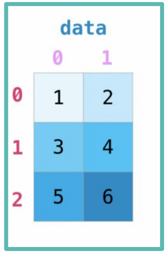
Reshaping arrays:

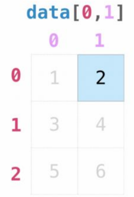
```
a = np.arange(6)
[0 1 2 3 4 5]
b = a.reshape(3,2)
[[0 1]
[2 3]
[4 5]]
```



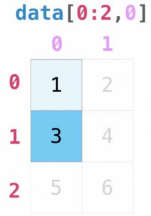
Indexing and Slicing arrays

- **Index into array**
  - Used to get value in row **a**, column **b**
  - **array[a, b]**
  - *Ex. data[0, 1]*



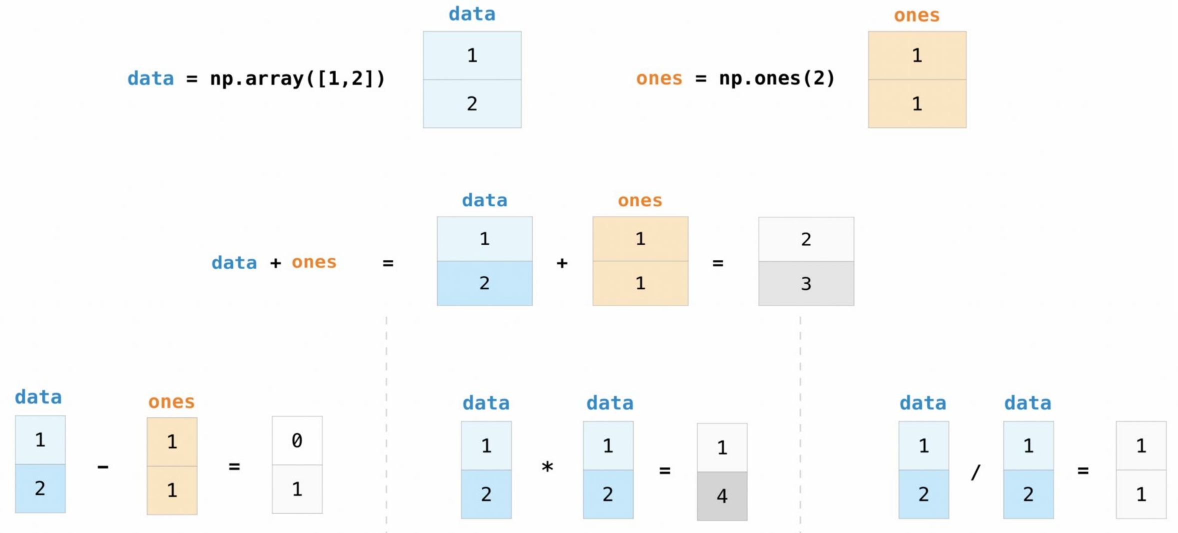


- **Slice rows and columns:**
  - Slice any subset of an array with Python slicing syntax
  - array[**row_a** : **row_b**, **col_a** : **col_b**]
  - Used to get subset of array from **row_a** to **row_b**, and **col_a** to **col_b**
  - *Ex. data[0:2, 0]   takes rows 0 & 1 in column 0*



Basic array operations

- Addition, subtraction, multiplication, & division
- Save time using Numpy Operations as they are fast and fast to implement!
- *E.g. you wouldn't loop through and add one to each value, you would do 'data + ones'!*



Broadcasting

- Applying a ***scalar value*** on an array (vector) is called ***broadcasting***
- Applies operation to ***every cell*** in array

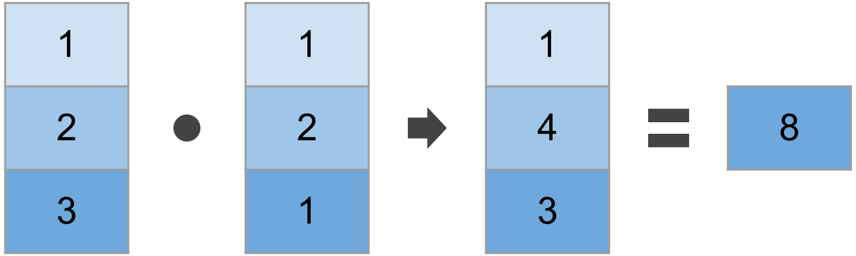
Ex. data * 1.6

Linear Algebra

Numpy also allows to operate its arrays as vectors and matrices.

Dot product:

```
v1 = np.array([1,2,3])
v2 = np.array([1,2,1])
np.dot(v1,v2)
```



Linear Algebra: Part 2

Transpose

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

```
m.T
```

1	3	1
4	2	2

→

1	4
3	2
1	2

Inverse

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

```
np.linalg.inv(m)
```

1	3
4	2

→

-0.2	0.3
0.4	-0.1

Linear Algebra: Part 3

Matrix multiplication:

```
m1 = np.array([[1, 3,
1],
                   [4, 2, 2]])
```

$$\begin{bmatrix} 1 & 3 & 1 \\ 4 & 2 & 2 \end{bmatrix} @ \begin{bmatrix} 1 & 4 \\ 3 & 2 \\ 1 & 2 \end{bmatrix}$$

$$\Downarrow$$

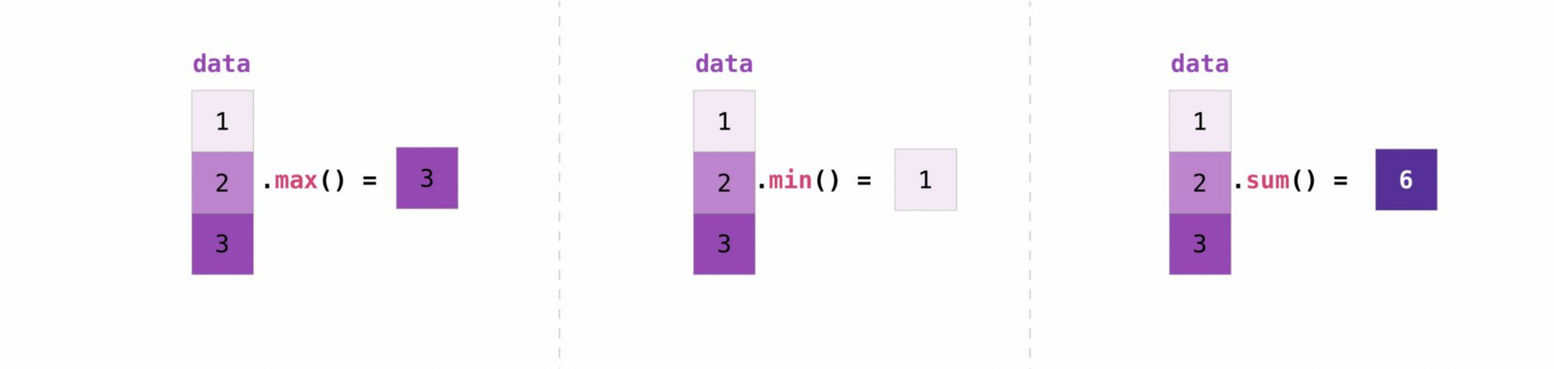
$$\begin{bmatrix} 11 & 12 \\ 12 & 24 \end{bmatrix}$$

```
m1 @ m1.T
```

```
np.matmul(m1, m1.T)
```

More useful operations...

Maximum, minimum, sum, mean (average), product, standard deviation (SD), and more:



Others: .prod(), .average(), .std(), etc.

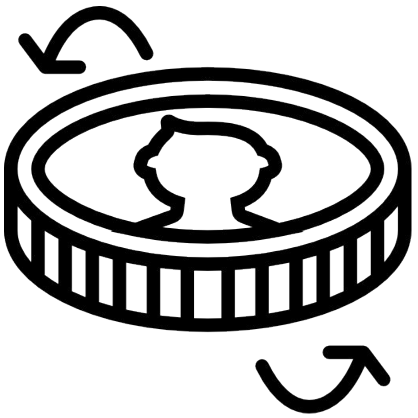
Probabilities

Numpy also offers some tools for sampling distributions and statistical analysis.

Flipping a coin 5 times:

```
np.random.randint(2, size=5)

> [0, 0, 1, 0, 1]
```



Probabilities: Part 2

Mean:

$$\mu = (1/N)\sum a_i$$

```
x = np.array([1, 3, 5, 8])
x.mean()

> 4.25
```

Standard deviation:

$$\sigma = sqrt((1/N)\sum(x - \mu))$$

```
x = np.array([1, 3, 5, 8])
x.std()

> 2.586
```

AI

Data Standardization

- Usually features are represented in columns.

- We do standardization to bring all features in the same range to improve prediction.

```
In [193]: A = np.array([[1,1,1], [4,5,6], [7,8,9]])

          def normalize(features):
              mean = np.mean(features, axis=0)
              print("Feature wise mean: ", mean)
              deviation = np.std(features, axis=0)
              print("Feature wise deviation: ", deviation)
              # to avoid division by 0
              std_feat = (features - mean)/(deviation+1e-8)
              return std_feat

          normalize(A)

          Feature wise mean:  [4.         4.66666667 5.33333333]
          Feature wise deviation:  [2.44948974 2.86744176 3.29983165]

Out[193]: array([[-1.22474487, -1.27872402, -1.3131983 ],
                 [ 0.        ,  0.11624764,  0.20203051],
                 [ 1.22474487,  1.16247638,  1.1111678 ]])
```

Linear Regression

Learn ai

- Data points (x_i,y_i) seem to follow approximately a linear distribution
- If we can find the corresponding line $y=f(x)=b_0x+b_1$ then we can *predict* a new point j value y_j for its corresponding value x_j: $y_j=f(x_j)$
- Data points will most always not fall exactly on the line. The difference between $f(x_i)$ and their true y_i will be an error term $e_i = y_i - f(x_i)$
- Linear regression *learns* b_0 and b_1 by minimizing the error term e_i in a function $J(b_0,b_1)=1/2n\sum e_i^2$

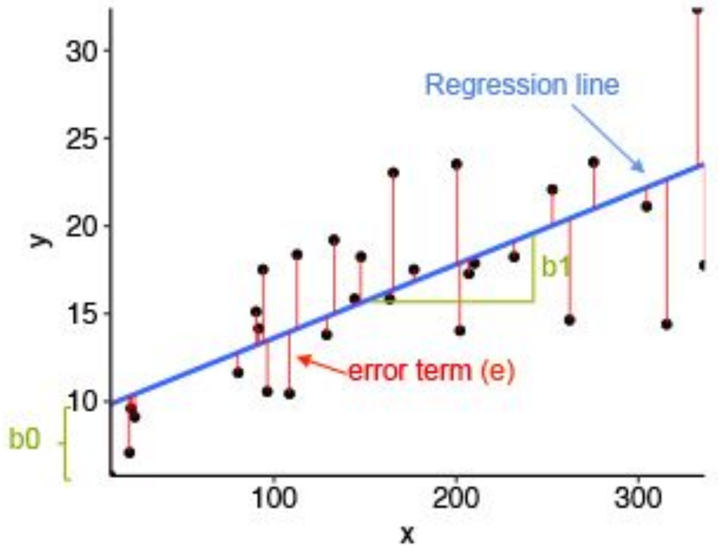


Image source: NextJournal

> Most supervised learning algorithms predict values by learning parameters, which they learn by minimizing error functions for the existing data.

AI

Linear Regression in numpy

For a data set with n values (x_i,y_i), $b_1=\sum_n x_i y_i - n x_m y_m \,/\, \sum_n (x_i-x_m)^2$ and $b0=y_m-b_1x_m$

where x_m is the mean of all x_i values and y_m for all y_i values

```
import numpy as np

# capture the data samples in an array
sample=np.array([[1,3],[2,4],[3,5.5],[4,8.2],[5,10],[6,11],[7,13],[8,14.2],[9,19],[10,20.3]])

# get the x and y values separately (for clarity -- not efficient coding)
x = sample[:,0]
y = sample[:,1]

# find the number of samples in the data
n = np.size(x)

# calculate the mean values for x and y
xm,ym = np.mean(x), np.mean(y)

# calculate the coefficients
b1 = (np.sum(y*x) - n*ym*xm ) / (np.sum(x**2) - n*xm**2)
b0 = ym - b1*xm

# print the results
print("The coefficients are b0 %2.2f and b1 %2.2f " % (b0,b1))
```

```
The coefficients are b0 0.17 and b1 1.94
```

Cheat Sheet

Here’s a cool cheatsheet for quick access to numpy syntax and functions! 😄

- Learn Statistics with NumPy: Introduction to NumPy Cheatsheet

Thank you! That's it!

Question time!

- Ask away! There are no dumb questions 🤓
- For practice, check our resource in the first slide and the cheat sheet at the end!

References

- - Python Numpy Tutorial (with Jupyter & Colab):
  - https://cs231n.github.io/python-numpy-tutorial/#numpy
- - NumPy Tutorial: A Simple Example-Based Guide:
  - https://stackabuse.com/numpy-tutorial-a-simple-example-based-guide/
- PluralSight:
  - https://www.pluralsight.com/guides/different-ways-create-numpy-arrays
- - Numpy | Python:
  - https://campus.datacamp.com/courses/intro-to-python-for-data-science/chapter-4-numpy?ex=1) (videos
- BEST SOURCE: (Images taken from here)
  https://towardsdatascience.com/the-ultimate-beginners-guide-to-numpy-f5a2f99aef54

Contributors

Learn ai

UofT AI

Elias Williams

UNIVERSIDAD DE CHILE

Jose Ignacio Diaz

Université de Montréal

Pavithra Rajasekar

AI Commons

Gilles Fayad

AI