💜

Data Manipulation: NumPy

About NumPy (Numerical Python Library):

→ Numpy is the fundamental package for numeric computing with Python. It provides powerful ways to create, store, and/or manipulate data, which makes it able to seamlessly and speedily integrate with a wide variety of databases.

```
#Importando bibliotecas
import numpy as np
import math
```

About Arrays:

→ Criação de arrays unidimensionais ou multidimensionais (matrizes)

```
# Arrays are displayed as a l
# array, we pass in a list as
a = np.array([1, 2, 3])
print(a)
# We can print the number of
print(a.ndim)

#Output: [1, 2, 3] 1
```

```
b = np.arange(1,16,1).reshape
#[[ 1  2  3  4  5], [ 6  7  8
```

→ Soma, máximo, mínimo e média

- `array.sum( )`
- `array.max( )`
- `array.min( )`
- `array.mean( )`

Indexing, Slicing and Iterating:

Indexing:

→ One -dimensional array: `array[x]`

→ Multidimensional: `array[x, y]`

Boolean Indexing:

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

#Cria uma lista de booleanos
print(a>5) #[[False False], [

#Retorna a quantidade de elem
print(a[a>5]) #[6]
```

→ Descrição de dataframe

- `df.head(x)` → Verifica primeiras x linhas
- `df.tail(x)` → Verifica últimas x linhas
- `df.shape` → Verifica o tamanho de cada dimensão de listas
- `df.dtype` → Verifica os tipos de dados contidas na lista

→ Geração de arrays

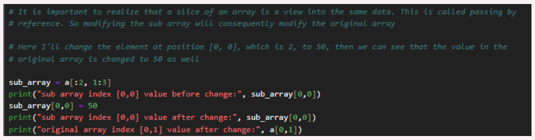
- `np.zeros( )`
- `np.ones( )`
- `np.random.rand( )`
- `np.arange( )`
- `np.linspace( )`

```
#Adicionar listas com 0s e 1s
d = np.zeros((2,3))
print(d) #[[0. 0. 0.], [0. 0.

e = np.ones((2,3))
print(e) #[[1. 1. 1.], [1. 1.

np.random.rand(2,3) #array([[
```

```
# We can also create a sequen
# starting bound and the seco
# each consecutive numbers

f = np.arange(10, 50, 2)
```

Slicing:

```
a = np.array([0,1,2,3,4,5])
print(a[:3]) #[0 1 2]
print(a[2:4]) #[2 3]

a = np.array([[1,2,3,4], [5,6
a[:2] #array([[1, 2, 3, 4], [
a[:2, 1:3] #array([[2, 3], [6

# So, in multidimensional arr
# selecting columns
```



```
# It is important to realize that a slice of an array is a view into the same data. This is called passing by
# reference. So modifying the sub array will consequently modify the original array

# Here I'll change the element at position [0, 0], which is 2, to 50, then we can see that the value in the
# original array is changed to 50 as well

sub_array = a[:2, 1:3]
print("sub array index [0,0] value before change:", sub_array[0,0])
sub_array[0,0] = 50
print("sub array index [0,0] value after change:", sub_array[0,0])
print("original array index [0,1] value after change:", a[0,1])
```

```
#array([10, 12, 14, 16, 18, 2
#       44, 46, 48])
```

```
# if we want to generate a se
# argument isn't the differen
np.linspace( 0, 2, 15 ) # 15

#array([0.        , 0.1428571
#       0.71428571, 0.8571428
#      1.42857143, 1.57142857
```

Array Operations:

→ Mathematical manipulation with arrays (addition, subtraction, square, exponents)

→ Matrix manipulation such as product, transpose, inverse, and so forth

→ Operações com matrizes:

- `A*B` X `A@B`
- `array.reshape( )`

```
# look at matrix product. if 
A = np.array([[1,1],[0,1]])
B = np.array([[2,0],[3,4]])
print(A*B) #[[2 0], [0 4]]

# if we want to do matrix pro
print(A@B) # [[5 4], [3 4]]
```