ARRAY FUNCTIONS

April 21, 2021

1 Main Functions (Arrays)

In this practice we are going to see the main functions with arrays. Arrays are fundamental for algebra, since they allow us to carry out a multitude of operations on our computer, spending less resources than if we used other storage structures. In addition, they are generally used to work with one type of data at a time, which greatly simplifies operations.

The basic package for creating vectors and matrices in Python is **Numpy**.

Advantages:

- Spend less memory.
- Faster to read.
- You don't have to check what datatype it is.

[2]:
```
# Importar paquetes
#==================================
import numpy as np
```

2 Mathematics

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

# Scalar operations
#==================================
# print(a+2, b-2)
# print(a*2, b/2)


# Arrays operations
#==================================
c = np.array([5,6,7,8])

# print(a+c)
print(a*c) # Multiply element by element NOT matrix operations
```

```
[ 5 12 21 32]
```

3 Linear Algebra

Multiplying one Matrix by Another Matrix

In linear algebra we are trying to multiple matrices by matrices properties and that's a different process.

[11]:
```
a = np.ones(shape=(2,3))  # 2x3
b = np.array([[1,2],[3,4],[5,6]]) #3x2


# To multiply matrices .dot() or .matmul()
#================================

result_dot = np.dot(a=a, b=b)  # 2x3 * 3x2 = 2x2
print("Dot_function: \n", result_dot)

result_matmul = np.matmul(a,b)
print("Matmul function: \n", result_matmul)


print(result_dot==result_matmul)
```

```
Dot_function:
 [[ 9. 12.]
 [ 9. 12.]]
Matmul function:
 [[ 9. 12.]
 [ 9. 12.]]
[[ True  True]
 [ True  True]]
```

3.1 Other functions

- Determinant of the matrix
- Inverse
- Transponse

[12]:
```
identity = np.identity(n=3)

# Calculate the determinant
#================================
print("Determinant: ", np.linalg.det(identity))



# Make the matrix transponse
#================================
```

```
a = np.random.randint(low=1, high=10, size=(3,3))
trans = a.T

print("Matrix: \n", a)
print("Matrix Tranponse: \n", trans)



# Make the matrix inverse
#================================

inv = np.linalg.inv(a)
print("Matrix inverse: \n", inv)
```

```
Determinant:  1.0
Matrix:
 [[6 9 7]
 [3 2 6]
 [6 7 4]]
Matrix Tranponse:
 [[6 3 6]
 [9 2 7]
 [7 6 4]]
Matrix inverse:
 [[-0.45333333  0.17333333  0.53333333]
 [ 0.32       -0.24       -0.2       ]
 [ 0.12        0.16       -0.2       ]]
```

4 Statistics

[26]:
```
stats = np.array([[1,2,3],[4,5,6]])

# Find the min and max into the array
#================================

print("Min: {}".format(np.min(stats)))
print("Max: {}".format(np.max(stats, axis=1)))
print("-"*20)


# Get rudimentary statistics
#================================

print("Mean: ", np.mean(stats))
print("Sumatory: ", np.sum(stats))
print("Non-null values: ", np.count_nonzero(stats)) # Similar to use isnull().
 ↪sum()
```

```
Min: 1
Max: [3 6]
--------------------
Mean:  3.5
Sumatory:  21
Non-null values:  6
```

5 Merge Arrays

- Horizontal
- Vertical

[36]:
```
a = np.array([[1,2], [3,4]])
b = np.array([[1,1], [2,2]])


c = np.vstack((a,b))
print("Vertical Stack: \n", c)


d = np.hstack((a,b))
print("Horizontal Stack: \n", d)
```

```
Vertical Stack:
 [[1 2]
 [3 4]
 [1 1]
 [2 2]]
Horizontal Stack:
 [[1 2 1 1]
 [3 4 2 2]]
```

6 It's all by now!

6.1 Session information

[3]:
```
from sinfo import sinfo

sinfo()
```

```
-----
numpy       1.19.2
sinfo       0.3.1
-----
IPython             7.19.0
jupyter_client      6.1.7
jupyter_core        4.7.0
jupyterlab          2.2.6
```

```
notebook            6.1.6
-----
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Windows-10-10.0.19041-SP0
8 logical CPU cores, Intel64 Family 6 Model 126 Stepping 5, GenuineIntel
-----
Session information updated at 2021-04-18 19:29
```