Ciência de Dados com Python (2)

“Eu não posso ensinar nada a ninguém, eu só posso fazê-lo pensar.”

(Sócrates)

Numpy

NumPy

Let´s Code!

1

Introdução

Biblioteca Numpy

Numpy

- É uma biblioteca de álgebra linear para Python
- É o bloco de construção de todas as outras bibliotecas de análise de dados
- Extremamente rápida: principais métodos compilados em C

Arrays Numpy

- Principal ferramenta
- Essencialmente de duas formas: vetores e matrizes
- Vetores: arrays de uma dimensão
- Matrizes: arrays de duas dimensões (porém pode ter apenas uma linha ou coluna)

2

Criação de Vetores e Arrays

```
import numpy as np
```

```
lista = [1,2,3]
lista
```

```
[1, 2, 3]
```

```
np.array(lista)
```

```
array([1, 2, 3])
```

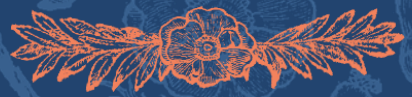
```
matriz = [[1,2,3],[4,5,6],[7,8,9]]
matriz
```

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

```
np.array(matriz)
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

Numpy Arrays



Métodos Incorporados

```
np.arange(0, 10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.arange(1,10,2)
```

```
array([1, 3, 5, 7, 9])
```

```
np.zeros(5)
```

```
array([0., 0., 0., 0., 0.])
```

```
np.zeros((2,3))
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

```
np.ones((3,2))
```

```
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
```

```
np.eye(4)
```

```
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

```
#Retorna números uniformemente espaçados ao longo de um intervalo especificado
np.linspace(0,10,3)
```

```
array([ 0.,  5., 10.])
```

```
np.linspace(0,10,50)
```

```
array([ 0.        ,  0.20408163,  0.40816327,  0.6122449 ,  0.81632653,
        1.02040816,  1.2244898 ,  1.42857143,  1.63265306,  1.83673469,
        2.04081633,  2.24489796,  2.44897959,  2.65306122,  2.85714286,
        3.06122449,  3.26530612,  3.46938776,  3.67346939,  3.87755102,
        4.08163265,  4.28571429,  4.48979592,  4.69387755,  4.89795918,
        5.10204082,  5.30612245,  5.51020408,  5.71428571,  5.91836735,
        6.12244898,  6.32653061,  6.53061224,  6.73469388,  6.93877551,
        7.14285714,  7.34693878,  7.55102041,  7.75510204,  7.95918367,
        8.16326531,  8.36734694,  8.57142857,  8.7755102 ,  8.97959184,
        9.18367347,  9.3877551 ,  9.59183673,  9.79591837, 10.        ])
```

Random

```
#Cria uma matriz da forma dada e preencha com amostras aleatórias
#de uma distribuição uniforme sobre (0, 1)
np.random.rand(2)
```

```
array([0.6293246 , 0.99008379])
```

```
np.random.rand(5,5)
```

```
array([[0.13349781, 0.38342681, 0.39401226, 0.72829789, 0.92875679],
       [0.77694924, 0.69653792, 0.09785123, 0.77076846, 0.47967558],
       [0.93598181, 0.59254196, 0.62571854, 0.64824259, 0.82429438],
       [0.70184597, 0.82003437, 0.4596253 , 0.34477016, 0.70470798],
       [0.47871555, 0.06902837, 0.61010705, 0.08856913, 0.67324948]])
```

```
#Retorna amostras da distribuição "normal".
#Ao contrário de rand, que é uniforme
np.random.randn(2)
```

```
array([-0.8434158,  0.9666766])
```

```
np.random.rand(5,5) * 100
```

```
array([[57.96775897, 19.50310736, 45.39069792, 34.4883529 , 98.13914559],
       [38.92606561, 97.3304982 , 56.13751788, 46.40366104, 94.10705292],
       [24.88900249, 72.35941519, 25.29976465, 24.49228929,  8.89548234],
       [64.60638946, 44.17266534, 55.87280889, 49.44425891,  3.78166248],
       [81.07637985, 34.12450189, 18.58308053, 80.17810599, 25.60731534]])
```

```
#Retorna inteiros aleatórios de
#"low" (inclusive) para "high" (exclusivo).
np.random.randint(1,100)
```

```
85
```

```
np.random.randint(1,100,10)
```

```
array([61,  3, 56, 21, 34, 73, 85, 68, 36, 74])
```

Atributos e Métodos

```
arr = np.arange(25)
ranarr = np.random.randint(0,50,10)
```

```
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24])
```

```
arr.reshape(5,5)
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

```
ranarr
```

```
array([ 0, 43, 41, 23, 44, 42, 27, 37, 44, 10])
```

```
ranarr.min()
```

```
0
```

```
ranarr.max()
```

```
44
```

```
ranarr.argmin()
```

```
0
```

```
ranarr.argmax()
```

```
4
```

```
arr.shape
```

```
(25,)
```

```
arr = arr.reshape(5,5)
```

```
arr.shape
```

```
(5, 5)
```

3

Indexação e Fatiamento

Indexação e Seleção

```
arr = np.arange(0,10)
arr
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
arr[7]
```

```
7
```

```
arr[3:8]
```

```
array([3, 4, 5, 6, 7])
```

```
arr[:5]
```

```
array([0, 1, 2, 3, 4])
```

```
arr[5:]
```

```
array([5, 6, 7, 8, 9])
```

```
arr[3:6] = 10
arr
```

```
array([ 0,  1,  2, 10, 10, 10,  6,  7,  8,  9])
```

Slice e Cópia

```
arr
```

```
array([ 0,  1,  2, 10, 10, 10,  6,  7,  8,  9])
```

```
fatia = arr[5:]
fatia
```

```
array([10,  6,  7,  8,  9])
```

```
fatia[:] = 100
fatia
```

```
array([100, 100, 100, 100, 100])
```

```
arr
```

```
array([  0,   1,   2,  10,  10, 100, 100, 100, 100, 100])
```

```
arr2 = arr.copy()
arr2
```

```
array([  0,   1,   2,  10,  10, 100, 100, 100, 100, 100])
```

Matriz 2D

```
arr2d = np.array(([5,10,15],[20,25,30],[35,40,45]))
arr2d
```

```
array([[ 5, 10, 15],
       [20, 25, 30],
       [35, 40, 45]])
```

```
arr2d[1]
```

```
array([20, 25, 30])
```

```
arr2d[0][2]
```

```
15
```

```
arr2d[1,0]
```

```
20
```

```
arr2d[:2,1:]
```

```
array([[10, 15],
       [25, 30]])
```

```
arr2d[:,1]
```

```
array([10, 25, 40])
```

Indexação “Fancy”

```
arr2d = np.zeros((10,10))
tam = arr2d.shape[1]
for i in range(tam):
    arr2d[i] = i

arr2d
```

```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [3., 3., 3., 3., 3., 3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.],
       [9., 9., 9., 9., 9., 9., 9., 9., 9., 9.]])
```

```
arr2d[[2,4,6,8]]
```

```
array([[2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [4., 4., 4., 4., 4., 4., 4., 4., 4., 4.],
       [6., 6., 6., 6., 6., 6., 6., 6., 6., 6.],
       [8., 8., 8., 8., 8., 8., 8., 8., 8., 8.]])
```

```
arr2d[[7,0,2,5]]
```

```
array([[7., 7., 7., 7., 7., 7., 7., 7., 7., 7.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [2., 2., 2., 2., 2., 2., 2., 2., 2., 2.],
       [5., 5., 5., 5., 5., 5., 5., 5., 5., 5.]])
```

Seleção

```
arr = np.arange(1,10)
arr
```

```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
arr > 4
```

```
array([False, False, False, False,  True,  True,  True,  True,  True])
```

```
bool = arr > 4
bool
```

```
array([False, False, False, False,  True,  True,  True,  True,  True])
```

```
arr[bool]
```

```
array([5, 6, 7, 8, 9])
```

```
arr[arr < 7]
```

```
array([1, 2, 3, 4, 5, 6])
```

```
x = 5
arr[arr >= x]
```

```
array([5, 6, 7, 8, 9])
```

4

Operações

Operações Básicas

```
arr = np.arange(0,16)
arr
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15])
```

```
arr + arr
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30])
```

```
arr - arr
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
arr * arr
```

```
array([  0,   1,   4,   9,  16,  25,  36,  49,  64,  81, 100, 121, 144,
       169, 196, 225])
```

```
arr / arr
```

```
<ipython-input-6-7f952cd3e0ce>:1: RuntimeWarning: invalid value encountered in
true_divide
  arr / arr
```

```
array([nan,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,
        1.,  1.,  1.])
```

Operações Escalares

```
arr + 10
```

```
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25])
```

```
arr - 7
```

```
array([-7, -6, -5, -4, -3, -2, -1,  0,  1,  2,  3,  4,  5,  6,  7,  8])
```

```
arr * 3
```

```
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45])
```

```
arr / 10
```

```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5])
```

```
arr ** 3
```

```
array([   0,    1,    8,   27,   64,  125,  216,  343,  512,  729, 1000,
       1331, 1728, 2197, 2744, 3375], dtype=int32)
```

```
np.sqrt(arr)
```

```
array([0.        , 1.        , 1.41421356, 1.73205081, 2.        ,
       2.23606798, 2.44948974, 2.64575131, 2.82842712, 3.        ,
       3.16227766, 3.31662479, 3.46410162, 3.60555128, 3.74165739,
       3.87298335])
```

```
np.exp(arr)
```

```
array([1.00000000e+00, 2.71828183e+00, 7.38905610e+00, 2.00855369e+01,
       5.45981500e+01, 1.48413159e+02, 4.03428793e+02, 1.09663316e+03,
       2.98095799e+03, 8.10308393e+03, 2.20264658e+04, 5.98741417e+04,
       1.62754791e+05, 4.42413392e+05, 1.20260428e+06, 3.26901737e+06])
```

```
np.sin(arr)
```

```
array([ 0.        ,  0.84147098,  0.90929743,  0.14112001, -0.7568025 ,
       -0.95892427, -0.2794155 ,  0.6569866 ,  0.98935825,  0.41211849,
       -0.54402111, -0.99999021, -0.53657292,  0.42016704,  0.99060736,
        0.65028784])
```

```
np.mean(arr)
```

```
7.5
```

```
np.std(arr)
```

```
4.6097722286464435
```

Funções Universais

Somas

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

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

```
x.sum()
```

```
10
```

```
x.sum(axis=0)   # colunas
```

```
array([4, 6])
```

```
x[:, 0].sum(), x[:, 1].sum()
```

```
(4, 6)
```

```
x.sum(axis=1)   # linhas
```

```
array([3, 7])
```

```
x[0, :].sum(), x[1, :].sum()
```

```
(3, 7)
```

Obrigada!

Alguma pergunta?

professora@lucilia.com.br

Créditos

- Template: SlidesCarnival
- Ilustrações: Icones stirpium by Matthias de L'Obel
- Python Brasil
- Scipy-lectures.org