Untitled

January 22, 2022

[112]:
```
import numpy as np
import math
```

[ ]:
```

```

1 Criando Array

[6]:
```
#arrays podem ser mostrados como listas ou listas de listas e podem ser criados␣
 ↪como listas. Quando criando um array nós
#passamos uma lista como um argumento numpy array

a = np.array(['1','2','3'])
print(a)
#pode imprimir o número de dimensões de uma lista usando o atributo "ndim"
print(a.ndim)
```

```
['1' '2' '3']
1
```

[4]:
```
#se inserirmos uma lista de listas, criamos então um array multi dimensional,␣
 ↪por exemplo uma matrix
b = np.array([[1,2,3], [4,5,6]])
b
b.ndim
```

[4]:
```
2
```

[5]:
```
#podemos imprimir o comprimento do array usando o atributo "shape", que retorna␣
 ↪uma tupla, no caso = 2 linhas 3 colunas
b.shape
```

[5]:
```
(2, 3)
```

[11]:
```
#verificar o tipo de dados de um array "dtype"
print(a.dtype)
print(b.dtype)
```

```
<U1
int64
```

[17]:
```
#floats também são aceitos em arrays numpy
c = np.array([1.1, 5, 2.8])
c.dtype.name
c
#ele converte automaticamente o 5 que era inteiro para um numero float sem␣
 ↪perder precisão
#ele tenta manter o tipo do dado homogêneo
```

[17]:
```
array([1.1, 5. , 2.8])
```

[18]:
```
#as vezes sabemos o formato do array que queremos criar mas não sabemos o que␣
 ↪colocar nele, o numpy oferece
#funções para criar arrays com dados iniciais, tipo 0 e 1

#criando dois arrays com o mesmo formato potém com dados diferentes
d = np.zeros((2,3))
print(d)

e = np.ones((2,3))
print(e)
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1.]
 [1. 1. 1.]]
```

[21]:
```
#criando um array com números aleatórios
np.random.randint(2,3)
```

[21]:
```
2
```

[24]:
```
#ainda da pra criar uma sequência de números com a função arange().
#o primeiro argumento é o número de partida, o segundo argumento é o número de␣
 ↪parada e
#o terceiro argumento é a diferença entre os números, o "pulo"

#criando um array de números pares de 10 (incluso) até 50 (excluso)
f = np.arange(10, 50, 2) #começa no dez (incluso) e termina no 50 (sem contar o␣
 ↪50), pulando de 2 em 2
print(f)
```

```
[10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48]
```

[ ]:
```
#se quisermos criar uma sequencia de floats, podemos usar o linspace(). nessa␣
 ↪função,
```

```
#o terceiro argumento não é a diferença entre um número e o outro (o pulo) e␣
 ↪sim a quantidade de itens que queremos criar
np.linspace(0, 2, 15) #me gere 15 números entre 0 (incluso) e 2 (excluso)
```

2 Operações com arrays

[32]:
```
#da pra fazer manipulações matematicas (adição, subtração, divisão...) assim␣
 ↪como usar arrays booleanos (true, false)
#da pra manipular matrizes como produto, transpor, inverter e tal
a = np.array([10, 20, 30, 40])
b = np.array([1, 2, 3, 4])

#a menos b
c = a - b
print(c)

#a vezes b
d = a * b
print(d)
```

```
[ 9 18 27 36]
[ 10  40  90 160]
```

[11]:
```
#com operações aritiméticas podemos converter o valor dos dados, tipo␣
 ↪temperatura e distância

far = np.array([32, -16, 106, -30])

#formua de conversão
#((ºF - 32) / 1.8)

cel = ((far - 32) / 1.8)
cel
```

[11]:
```
array([  0.        , -26.66666667,  41.11111111, -34.44444444])
```

[13]:
```
#usar booleano para checar se uma temperatura de um array é maior que 20ºC
#vai retornar True se for verdadeiro ou False se for falso
cel > 20
```

[13]:
```
array([False, False,  True, False])
```

[14]:
```
#usar módulo para checar se um número do array é par
cel % 2 == 0
```

[14]:
```
array([ True, False, False, False])
```

[39]:
```
#numpy suporta manipulação de matrizes
#produto de matriz

A = np.array([[1,2], [3,4]])
B = np.array([[1,2], [3,4]])

#A = np.array([[a, b], [c, d]])
#B = np.array([[e, f], [g, h]])
# o processo é: a*e + b*g, a*f + b*h
#               c*e + d*g, c*f + d*h

soma1 = 1*1 + 2*3, 1*2 + 2*4

soma2 = 3*1 + 4*3, 3*2 + 4*4

print(A*B)
print('---')
print(A@B)
print('---')
print(soma1)
print('---')
print(soma2)
```

```
[[ 1  4]
 [ 9 16]]
---
[[ 7 10]
 [15 22]]
---
(7, 10)
---
(15, 22)
```

[40]:
```
#para ver o formato da matriz podemos fazer o .shape
A.shape
#(2, 2) --> duas linhas duas colunas
```

[40]:
```
(2, 2)
```

[45]:
```
#quando manipulando arrays de data types diferentes, ele vai colocar no final o␣
 ↪resultado data type mais geral que existe,
#isso é chamado upcasting

#array integers
ar1 = np.array([[2, 4], [6, 8]])
#array float
ar2 = np.array([[2.5, 4.5], [6.5, 8.5]])
#somando os arrays
```

```
ar3 = ar1 + ar2
print(ar3)
print(ar3.dtype)
```

```
[[ 4.5  8.5]
 [12.5 16.5]]
float64
```

[46]:
```
#numpy tem funções como max, min, sum, mean
print(ar3.mean())
print(ar3.max())
print(ar3.min())
print(ar3.sum())
```

```
10.5
16.5
4.5
42.0
```

[48]:
```
#com arrays multidimensionais, podemos fazer a mesma coisa com cada linha e/ou␣
 ↪coluna
#criando um array de 15 elementos de 1 a 15 com dimensão 3x5
quin = np.arange(1, 16, 1).reshape(3, 5)
quin
```

[48]:
```
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])
```

3 indexação, slicing e iteração

[49]:
```
#um array unidimensional funciona quase que como uma lista, para pegar um␣
 ↪elemento usamos o [x]
ar = np.array([13,2,56,6,8,1,2])
ar[5]
```

[49]:
```
1
```

[53]:
```
#já para um array multidimensional, usamos um index integer
ar2 = np.array([[2,3], [5,4], [1, 9], [2,5]])
ar2
```

[53]:
```
array([[2, 3],
       [5, 4],
       [1, 9],
       [2, 5]])
```

[58]:
```
#para pegar o elemento, precisamos inserir o número da linha e o segundo o da␣
 ↪coluna [x, y]
#lemrbando que em python começa no 0
ar2[3,0]
```

[58]:
```
2
```

[75]:
```
#para pegar mais de um elemento podemos colocá-los direto numa lista usando a␣
 ↪função doa rray
ar3 = np.array([ar2[0,0], ar2[1,1], ar2[2,0], ar2[3,1]])
print(ar3)
print('---')
#ou
print(ar2[[0,1,2,3], [0,1,0,1]]) #primeiro diz as linhas e depois quais colunas
```

```
[2 4 1 5]
---
[2 4 1 5]
```

3.1 Boolean indexing

[78]:
```
#para achar elementos maiores que 3
print(ar2 > 3)
#retorna um array com verdadeiro ou falso
```

```
[[False False]
 [ True  True]
 [False  True]
 [False  True]]
```

3.2 Slicing

[81]:
```
#slicing é uma forma de criar sub-arrays com base num array original
#para um array unidimensional, ele funciona quase como uma lista
#para fazer o slice, usa-se o sinal de pois pontos : . por exemplo, se␣
 ↪colocarmos o :5 no index, nós teriamos
# todos os elementos de 0 a 5 excluindo o 5
ar = np.array([1,2,3,4,5,6,7,8,9,10])
print(ar[:5])
```

```
[1 2 3 4 5]
```

[82]:
```
#daí se colocar dois números com os dois pontos, teríamos todos os elementos de␣
 ↪x a y
#[3:8] todos os elementos de 3(incluso) a 8 (excluso)
print(ar[3:8])
```

```
[4 5 6 7 8]
```

[92]:
```
#agora para arrays multimensionais
ar2 = np.array([[1,2,3,4], [5,6,7,8], [9, 10, 11,12]])
print(ar2)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

[87]:
```
#se adicionarmos [:2] teriamos todos os elemntos da coluna zero e da coluna um
print(ar2[:2])
#ou seja, ele tráz as linhas com todos os elementos e não os elementos␣
 ↪individuais
```

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

[88]:
```
#se adicionar dois argumentos ar2[:2, 1:3] retorna as duas primeiras linhas e␣
 ↪os elementos de 1 a 3 (excluso)
print(ar2[:2, 1:3])
```

```
[[2 3]
 [6 7]]
```

[94]:
```
#assim, em arrays multidimensionais, o primeiro argumento é para selecionar␣
 ↪colunas e o segundo argumento é para colunas
ar2
```

[94]:
```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

[99]:
```
#mudar um elemento num subarray muda ele também no array original, isso é␣
 ↪chamado passado por referência
#então, modoificando um sub array, vai modificar o original

sub_ar2 = ar2[:2, 1:3]

print('sub_ar2 index [0,0] antes de mudar: ', sub_ar2[0,0])
sub_ar2[0,0] = 50
print('sub_ar2 [0,0] depios de mudar: ', sub_ar2[0,0])
print('array original ar2 [0,1] depois de mudar: ', ar2[0,1])
print('---')
print(ar2)
```

```
sub_ar2 index [0,0] antes de mudar:  2
sub_ar2 [0,0] depios de mudar:  50
```

```
array original ar2 [0,1] depois de mudar:  50
---
[[ 1 50  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

[105]:
```
sub2_ar2 = sub_ar2
sub2_ar2[0,1] = 5
print(sub2_ar2)
print('---')
print(ar2)
```

```
[[50  5]
 [ 6  7]]
---
[[ 1 50  5  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

4 trabalhando com datasets e numpy

5 dataset de vinhos

[115]:
```
#carregabdo o arquivo csv, mas por algum acaso ele está com ponto e virgula ao␣
 ↪invés de só a virgula separando
wines = np.genfromtxt('winequality-red.csv', delimiter = ';', skip_header=1)
```

[116]:
```
wines
```

[116]:
```
array([[ 7.4  ,  0.7  ,  0.   , ...,  0.56 ,  9.4  ,  5.   ],
       [ 7.8  ,  0.88 ,  0.   , ...,  0.68 ,  9.8  ,  5.   ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  0.65 ,  9.8  ,  5.   ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  0.75 , 11.   ,  6.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  0.71 , 10.2  ,  5.   ],
       [ 6.   ,  0.31 ,  0.47 , ...,  0.66 , 11.   ,  6.   ]])
```

[118]:
```
#para selecionar a coluna da acidez, a primeira coluna, nós podemos buscar␣
 ↪inserindo o index da coluna no array
#lembrando que para arrays multidimensionais, o primeiro argumento é a linha e␣
 ↪o segundo a coluna
# se dermos só uma argumento, ele retorna uma lista simples

#todas as linhas da primeira colun
print('um número para cortar: ',wines[:, 0]) #retorna a primeira coluna em␣
 ↪forma de lista
#para retornarmos o valor da primeira coluna mas na forma de linhas, onde cada␣
 ↪valor está na sua linha:
```

```
print('0 a 1 para cortar: \n', wines[:, 0:1]) #aqui retorna os valores na forma␣
 ↪de uma única coluna
```

```
um número para cortar:  [7.4 7.8 7.8 ... 6.3 5.9 6. ]
0 a 1 para cortar:
 [[7.4]
 [7.8]
 [7.8]
 ...
 [6.3]
 [5.9]
 [6. ]]
```

[121]:
```
#se for pra pegar o intervalo entre a primeira e a terceira coluna:
wines[:, 0:3]
```

[121]:
```
array([[7.4  , 0.7  , 0.   ],
       [7.8  , 0.88 , 0.   ],
       [7.8  , 0.76 , 0.04 ],
       ...,
       [6.3  , 0.51 , 0.13 ],
       [5.9  , 0.645, 0.12 ],
       [6.   , 0.31 , 0.47 ]])
```

[122]:
```
#e se for pra buscar um número de colunas não consecutivas, tipo 0, 2, 4:␣
 ↪criamos um array e colocamos esse array como
#o segundo argumento da busca
wines[:,[0,2,4]]
```

[122]:
```
array([[7.4  , 0.   , 0.076],
       [7.8  , 0.   , 0.098],
       [7.8  , 0.04 , 0.092],
       ...,
       [6.3  , 0.13 , 0.076],
       [5.9  , 0.12 , 0.075],
       [6.   , 0.47 , 0.067]])
```

[124]:
```
#para fazer um resumo do dataset. se quisermos saber o valor médio da qualidade␣
 ↪do vinho vermelho,
#selecionamos a coluna de qualidade. o jeito mais apropriado de fazer isso é␣
 ↪usando o valor -1 para buscar a última coluna
#já que números negativos quer dizer que estamos buscando pelo final da lista
wines[:, -1].mean()
```

[124]:
```
5.6360225140712945
```

6 dataset de admissão escolar

[153]:
```
graduate_admission = np.genfromtxt('Admission_Predict.csv', dtype=None,␣
↪delimiter=",", skip_header=1,
                                   names=("Serial_No", 'GRE_Score',␣
↪'TOELF_Score', 'University_Rating', 'SOP',
                                          'LOR', 'CGPA', 'Research', 'Chance_of␣
↪Admit'))
graduate_admission.shape
```

[153]:
```
(400,)
```

[154]:
```
#podemos retornar uma coluna do array usando apenas o nome que demos aqui em␣
↪cima
graduate_admission['CGPA'][0:5]
```

[154]:
```
array([9.65, 8.87, 8.  , 8.67, 8.21])
```

[155]:
```
#para deixar o valor numa escala de 0 a 4, podemos dividir o valor por 10 e␣
↪multiplocar por 40
graduate_admission['CGPA'] = graduate_admission['CGPA'] / 10*4
```

[156]:
```
graduate_admission['CGPA'][0:20]
```

[156]:
```
array([3.86 , 3.548, 3.2  , 3.468, 3.284, 3.736, 3.28 , 3.16 , 3.2  ,
       3.44 , 3.36 , 3.6  , 3.64 , 3.2  , 3.28 , 3.32 , 3.48 , 3.2  ,
       3.52 , 3.4  ])
```

[157]:
```
#usando boolean mask para descobrir quantos alunos tiveram experiência com␣
↪pesquisa/research criando uma mascara booleana e
#passando ela para o operador index do array

len(graduate_admission[graduate_admission['Research'] == 1])
```

[157]:
```
219
```

[159]:
```
#identificar quandos alunos tem maiores chances de admissão >80% dos que tem␣
↪menos chance de admissão <40%
#primeiro usar a mascara booleana para pegar apenas aqueles alunos em que␣
↪estamos interessados
#baseado na sua chance de admissão, daí pegamos seu CGPA Score e imprimimos o␣
↪valor médio
print(graduate_admission[graduate_admission['Chance_of_Admit']> 0.
↪8]['GRE_Score'].mean())
print(graduate_admission[graduate_admission['Chance_of_Admit']< 0.
↪4]['GRE_Score'].mean())
```

```
328.7350427350427
302.2857142857143
```

[160]:
```
print(graduate_admission[graduate_admission['Chance_of_Admit']> 0.8]['CGPA'].
 ↪mean())
print(graduate_admission[graduate_admission['Chance_of_Admit']< 0.4]['CGPA'].
 ↪mean())
```

```
3.7106666666666666
3.0222857142857142
```