Semana 4

Objetivo: Linguagem Python, aprofundando nas bibliotecas Numpy e conhecendo Pandas

Conteúdo: Bibliotecas Python:

1. NumPy – Números Randômicos
2. Numpy Fancy Indexing
3. Numpy Broadcasting
4. Pandas
5. Pandas – Ordenação, Agregação, Junção e Split

Desafio: Realizar o curso completo e responder todos os exercícios propostos neste material

Material adaptado de Prof. Felipe Meneguzzi e Henry Cagnini

NumPy – Números Randômicos

Números Randômicos

NumPy oferece uma biblioteca, numpy.random, para geração de números randômicos

- random
- randint
- choice

random amostra valores no intervalo [0, 1)

- aceita como parâmetro o número de dimensões do array

```
In [1]: import numpy as np
        np.random.random(3)

Out[1]: array([0.13226305, 0.162169  , 0.13844914])

In [2]: np.random.random((2, 2))

Out[2]: array([[0.36616014, 0.83278459],
               [0.37835511, 0.74274754]])
```

randint amostra valores inteiros

- Aceita como parâmetros:
    - limite inferior
    - limite superior
    - dimensões do array

```
In [3]: import numpy as np
```

Amostra apenas um valor, no intervalo [0, 5)

```
In [5]: np.random.randint(5)

Out[5]: 4
```

Amostra apenas um valor, no intervalo [6, 10)

```
In [6]: np.random.randint(6, 10)

Out[6]: 8
```

Amostra 3 valores no intervalo [6, 10)

```
In [7]: np.random.randint(low=6, high=10, size=3)

Out[7]: array([6, 7, 9])
```

Amostra uma matrix (2,2) com valores no intervalo [6, 10)

```
In [8]: np.random.randint(low=6, high=10, size=(2,2))

Out[8]: array([[7, 6],
               [9, 6]])
```

choice amostra valores dada uma distribuição

- Aceita como parâmetros:
    - a - Distribuição de valores
    - size - Número de amostras
    - replace - Se a amostragem é com reposição
    - p - Distribuição de probabilidades

```
In [9]: import numpy as np

In [10]: a = ['a', 'b', 'c', 'd', 'e']
         p = [0.2, 0.2, 0.4, 0.1, 0.1]
```

Amostra apenas um valor, com distribuição uniforme

```
In [17]: np.random.choice(a)

Out[17]: 'b'
```

Amostra 10 exemplos, com reposição

```
In [18]: np.random.choice(a, size=10)

Out[18]: array(['e', 'e', 'd', 'a', 'e', 'd', 'b', 'd', 'e', 'c'], dtype='<U1')
```

Amostra 10 exemplos, sem reposição

- Deve retornar um erro, já que existem mais amostras do que valores

```
In [19]: try:
             np.random.choice(a, size=10, replace=False)
         except Exception as e:
             print(e.args)

         ("Cannot take a larger sample than population when 'replace=False'",)
```

Amostra 5 exemplos, sem reposição

```
In [20]: np.random.choice(a, size=5, replace=False)

Out[20]: array(['e', 'c', 'a', 'd', 'b'], dtype='<U1')
```

Amostra 100 exemplos, com reposição, usando p como distribuição de probabilidades

- Vamos contar a ocorrência dos exemplos na amostra, para ver se obedecem a distribuição de probabilidades fornecida a função

```
In [21]: from collections import Counter

In [22]: sample = np.random.choice(a, size=100, p=p)

In [23]: counted = Counter(sample)
         for val, prob in zip(a, p):
             print('Valor: %s Esperado: %f Amostrado: %f' % (val, prob, counted[val]/100.))

         Valor: a Esperado: 0.200000 Amostrado: 0.220000
         Valor: b Esperado: 0.200000 Amostrado: 0.180000
         Valor: c Esperado: 0.400000 Amostrado: 0.310000
         Valor: d Esperado: 0.100000 Amostrado: 0.120000
         Valor: e Esperado: 0.100000 Amostrado: 0.170000
```

As frequências dos valores amostrados tendem a p a medida que o tamanho da amostra for maior

Estatísticas

Existem diversas funções para extrair estatísticas de distribuições:

- mean
- median
- percentile
- histogram

```
In [28]: import numpy as np
         np.random.seed(0)

In [29]: a = np.random.randint(0, 10, size=10)
         a.sort()
         print(a)

         [0 2 3 3 3 4 5 5 7 9]

In [30]: np.mean(a)

Out[30]: 4.1

In [31]: np.median(a)

Out[31]: 3.5
```

percentile extrai o *q*-ésimo percentil da distribuição de valores

- O valor deve ser entre [0, 100]
- O 50-ésimo percentil é a mediana

```
In [32]: a = [43, 54, 56, 61, 62, 66, 68, 69, 69, 70, 71, 72, 77, 78, 79, 85, 87, 88, 89, 93, 95, 96, 98, 99, 99]
         print(a)

         [43, 54, 56, 61, 62, 66, 68, 69, 69, 70, 71, 72, 77, 78, 79, 85, 87, 88, 89, 93, 95, 96, 98, 99, 99]

In [33]: np.percentile(a, 90, interpolation='nearest')

Out[33]: 98

In [34]: np.percentile(a, 50, interpolation='nearest')

Out[34]: 77
```

histogram agrupa os valores e mostra a quantidade de ocorrências para cada um deles

- Um parâmetro importante é o número de bins (i.e. intervalos a serem considerados)

```
In [35]: import numpy as np

In [36]: a = [43, 54, 56, 61, 62, 66, 68, 69, 69, 70, 71, 72, 77, 78, 79, 85, 87, 88, 89, 93, 95, 96, 98, 99, 99]
         print(a)

         [43, 54, 56, 61, 62, 66, 68, 69, 69, 70, 71, 72, 77, 78, 79, 85, 87, 88, 89, 93, 95, 96, 98, 99, 99]

In [37]: np.histogram(a, bins=10)

Out[37]: (array([1, 1, 1, 2, 5, 2, 3, 2, 3, 5], dtype=int64),
          array([43. , 48.6, 54.2, 59.8, 65.4, 71. , 76.6, 82.2, 87.8, 93.4, 99. ]))
```

Numpy Fancy Indexing

E se eu quiser selecionar linhas específicas de um array 2D (e.g. linha 3, 7 8)? É possível?

Resposta: sim!!

Numpy permite a indexação de arrays utilizando outros arrays (que contém índices!)

In [2]:
```
import numpy as np
a = np.arange(9).reshape(3, 3)
print('Array original\n', a)

print('Selecionando as linhas 0 e 2:\n', a[[0, 2], :])

print('Selecionando as linhas 0 e 2, colunas 0 e 0', a[[0, 2], [0, 0]])
```

```
Array original
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Selecionando as linhas 0 e 2:
 [[0 1 2]
 [6 7 8]]
Selecionando as linhas 0 e 2, colunas 0 e 0 [0 6]
```

In [3]:
```
a = np.arange(9)
print('Array original\n', a)
print('Indexando várias vezes o mesmo elemento:\n', a[[0,0,0, 1, 3, 3]])
print('Indexando várias vezes o mesmo elemento:\n', a[[7,7,7, 0,0, 4, 1]])
```

```
Array original
 [0 1 2 3 4 5 6 7 8]
Indexando várias vezes o mesmo elemento:
 [0 0 0 1 3 3]
Indexando várias vezes o mesmo elemento:
 [7 7 7 0 0 4 1]
```

In [4]:
```
a = np.arange(9).reshape(3, 3)
print('Array original\n', a)

print('Selecionando as colunas 0 e 2:\n', a[:, [0, -1]])

print('Selecionando as colunas 0 e 2, linhas 0 e 2', a[[0, -1], [0, 2]])

print('Selecionando as colunas 2 e 0, linhas 2 e 0', a[[-1, 0], [2, 0]])
```

```
Array original
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Selecionando as colunas 0 e 2:
 [[0 2]
 [3 5]
 [6 8]]
Selecionando as colunas 0 e 2, linhas 0 e 2 [0 8]
Selecionando as colunas 2 e 0, linhas 2 e 0 [8 0]
```

In [7]:
```
a = np.arange(25).reshape(5, 5)
print('Array original\n', a)

print('Selecionando as linhas 1 a 4 e colunas 0, 2 e 3:\n', a[1:4, [0, 2, 3]])
indice_diagonal  = np.arange(5)
print('Selecionando a diagional principal:\n', a[indice_diagonal, indice_diagonal])
```

```
Array original
 [[ 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]]
Selecionando as linhas 1 a 4 e colunas 0, 2 e 3:
 [[ 5  7  8]
 [10 12 13]
 [15 17 18]]
Selecionando a diagional principal:
 [ 0  6 12 18 24]
```

In [9]:
```
a = np.array([[ 0,  1,  2],
              [ 3,  4,  5],
              [ 6,  7,  8],
              [ 9, 10, 11]])

rows = np.array([[0, 0],
                 [3, 3]], dtype=np.int32)

columns = np.array([[0, 2],
                    [0, 2]], dtype=np.int32)

print('Selecionando por combinação de linha X coluna\n', a[rows, columns])
```

```
Selecionando por combinação de linha X coluna
 [[ 0  2]
 [ 9 11]]
```

In [10]:
```
a = np.arange(9).reshape(3, 3)
print('Array original\n', a)

print('Selecionando as colunas 0 e 2:\n', a[:, [0, -1]])

a[:, [0, -1]] = [[-1, -1], [-1, -1], [-1, -1]]

print('Substituindo o conteúdo das colunas 0 e 2:\n', a)
```

```
Array original
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Selecionando as colunas 0 e 2:
 [[0 2]
 [3 5]
 [6 8]]
Substituindo o conteúdo das colunas 0 e 2:
 [[-1  1 -1]
 [-1  4 -1]
 [-1  7 -1]]
```

Argsorting

Funções que começam com arg geralmente estão se referindo à localização do elemento (índice) no array.

Dúvida: Como pegar os k maiores elementos de um array? Resposta: argsort!!

A função argsort retorna os **índices** que ordenariam o array. Ou seja, se indexarmos o array com o resultado do argsort, temos o array ordenado.

```
In [11]: a = np.random.randint(0, 10, 9)
         print('Conteúdo do array:\n', a)
         indices = np.argsort(a)
         print('Índices retornados pelo argsort:\n', indices)
         print('Indexando com os índices retornados pelo argsort:\n', a[indices])
```

```
Conteúdo do array:
 [3 2 5 2 9 8 9 3 1]
Índices retornados pelo argsort:
 [8 1 3 0 7 2 5 4 6]
Indexando com os índices retornados pelo argsort:
 [1 2 2 3 3 5 8 9 9]
```

Unique

Como descobrimos quais os valores únicos em um array? Como descobrimos a frequência de cada valor único do array? Reposta: np.unique(...).

```
In [12]: a = np.random.randint(0, 5, 9)
         print('Conteúdo do array:\n', a)
         unicos, frequencia = np.unique(a, return_counts=True)
         print('Valores únicos do array:\n', unicos)
         print('Frequência de cada valor único do array:\n', frequencia)
```

```
Conteúdo do array:
 [2 0 1 4 2 2 0 4 0]
Valores únicos do array:
 [0 1 2 4]
Frequência de cada valor único do array:
 [3 1 3 2]
```

Boolean Indexing

E se eu quiser selecionar todos os elementos de um array que satisfaçam uma condição? Todos os elementos iguais a zero, por exemplo?

Solução: boolean indexing!

- A indexação por booleans permite indexar um array com uma máscara de valores booleanos.
- A Máscara deve ter exatamente o mesmo tamanho da dimensão que está sendo indexada.

In [13]:
```
a = np.arange(9)
print('Conteúdo do array:\n', a)
mascara = np.zeros_like(a, dtype=np.bool)
print('Conteúdo da máscara:\n', mascara)
mascara[5:] = True
print('Conteúdo da máscara:\n', mascara)
print('Resultado da indexação do array com a máscara:\n', a[mascara])
```

```
Conteúdo do array:
 [0 1 2 3 4 5 6 7 8]
Conteúdo da máscara:
 [False False False False False False False False False]
Conteúdo da máscara:
 [False False False False False  True  True  True  True]
Resultado da indexação do array com a máscara:
 [5 6 7 8]
```

In [14]:
```
a = np.arange(9)
print('Conteúdo do array:\n', a)
print('Conteúdo da máscara da condição a < 5:\n', a < 5)
print('Selecionando todos os valores menores que 5:\n', a[a < 5])
```

```
Conteúdo do array:
 [0 1 2 3 4 5 6 7 8]
Conteúdo da máscara da condição a < 5:
 [ True  True  True  True  True False False False False]
Selecionando todos os valores menores que 5:
 [0 1 2 3 4]
```

In [15]:
```
a = np.arange(9)
print('Conteúdo do array:\n', a)
print('Conteúdo da máscara da condição a == 2:\n', a == 2)
print('Selecionando todos os valores iguais a 2:\n', a[a == 2])
```

```
Conteúdo do array:
 [0 1 2 3 4 5 6 7 8]
Conteúdo da máscara da condição a == 2:
 [False False  True False False False False False False]
Selecionando todos os valores iguais a 2:
 [2]
```

In [16]:
```
a = np.arange(9)
print('Conteúdo do array:\n', a)
print('Conteúdo da máscara da condição a == 2 ou a == 3:\n', (a == 2) | (a == 3) )
print('Selecionando todos os valores iguais a 2 ou 3:\n', a[(a == 2) | (a == 3)])
```

```
Conteúdo do array:
 [0 1 2 3 4 5 6 7 8]
Conteúdo da máscara da condição a == 2 ou a == 3:
 [False False  True  True False False False False False]
Selecionando todos os valores iguais a 2 ou 3:
 [2 3]
```

In [17]:
```
a = np.random.randint(0, 50, 15)
print('Conteúdo do array:\n', a)
print('Selecionando o maior valor do array:\n', a[a == a.max()])
a[a == a.max()] = -1
print('Substituindo o maior valor por -1:\n', a)
```

```
Conteúdo do array:
 [46 26  7  2  1 19 34 25 38  3 44 35 20 45 18]
Selecionando o maior valor do array:
 [46]
Substituindo o maior valor por -1:
 [-1 26  7  2  1 19 34 25 38  3 44 35 20 45 18]
```

Como descobrimos quantos elementos satisfazem uma condição?

Dica: podemos fazer operações matemáticas com booleanos!!

In [18]:
```
a = np.random.randint(0, 50, 15)
print('Conteúdo do array:\n', a)
print('Máscara que seleciona os valores maiores que 25:\n', a > 25)
print('Número total de elementos maiores que 25:\n', np.sum(a > 25))
```

```
Conteúdo do array:
 [ 6 26 25 38 47 15 43 18 16 43 42 46 43 35 41]
Máscara que seleciona os valores maiores que 25:
 [False  True False  True  True False  True False False  True  True  True
  True  True  True]
Número total de elementos maiores que 25:
 10
```

E no caso de várias dimensões? Podemos usar indexação booleana? Reposta: Claro!

In [19]:
```
a = np.arange(9).reshape(3, 3)
print('Array original\n', a)
mascara1d = np.zeros(3, dtype=np.bool)
mascara1d[[0, 2]] = True
print('Conteúdo da máscara 1d:\n', mascara1d)
print('Resultado da indexação com a máscara 1d na primeira dimensão:\n', a[mascara1d, :])
print('Resultado da indexação com a máscara 1d na segunda dimensão:\n', a[:, mascara1d])
print('Resultado da indexação com a máscara 1d nas duas dimensões:\n', a[mascara1d, mascara1d])
```

```
Array original
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Conteúdo da máscara 1d:
 [ True False  True]
Resultado da indexação com a máscara 1d na primeira dimensão:
 [[0 1 2]
 [6 7 8]]
Resultado da indexação com a máscara 1d na segunda dimensão:
 [[0 2]
 [3 5]
 [6 8]]
Resultado da indexação com a máscara 1d nas duas dimensões:
 [0 8]
```

Consigo descobrir os índices que satisfazem uma condição? Sim!

Nonzero

In [20]:
```
a = np.arange(9).reshape(3, 3)
print('Array original\n', a)
mascara = a > 3
print('Conteúdo da máscara:\n', mascara)
print('Índices retornados pelo nonzero:\n', np.nonzero(mascara))
indice_x, indice_y = np.nonzero(mascara)
print('Indexando com o resultado do nonzero:\n', a[indice_x, indice_y])
```

```
Array original
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Conteúdo da máscara:
 [[False False False]
 [False  True  True]
 [ True  True  True]]
Índices retornados pelo nonzero:
 (array([1, 1, 2, 2, 2], dtype=int64), array([1, 2, 0, 1, 2], dtype=int64))
Indexando com o resultado do nonzero:
 [4 5 6 7 8]
```

Where(cond, [if true, if false])

```
In [21]: a = np.arange(9).reshape(3, 3)
         print('Array original\n', a)
         mascara = a > 3
         print('Conteúdo da máscara:\n', mascara)
         print('Conteúdo retornado pelo where:\n', np.where(mascara, a, a + 10))
```

```
Array original
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Conteúdo da máscara:
 [[False False False]
 [False  True  True]
 [ True  True  True]]
Conteúdo retornado pelo where:
 [[10 11 12]
 [13  4  5]
 [ 6  7  8]]
```

Exercícios - Parte 1

```
import numpy as np

import matplotlib.pyplot as plt

from PIL import Image

import csv
```

Exercício 1

Dado o array 2D abaixo, realize as seguintes operações:

1. A média das linhas 0, 2 e 3.
2. A média das colunas 0, 1 e 4.
3. A soma dos elementos das duas diagonais (utilizando indexação).

O seu código não deve conter nenhum tipo de loop (for/while)

```
a = np.arange(25).reshape(5, 5)
```

Exercício 2

Dado o array 2D abaixo, selecione os valores pares e todos os valores maiores que 10.

O seu código não deve conter nenhum tipo de loop (for/while)

```
a = np.arange(25).reshape(5, 5)
```

Exercício 3

Dado o array 2D abaixo, substitua os 5 menores valores por -1.

O seu código não deve conter nenhum tipo de loop (for/while)

```
a = np.random.randint(0, 50, (5, 5))
```

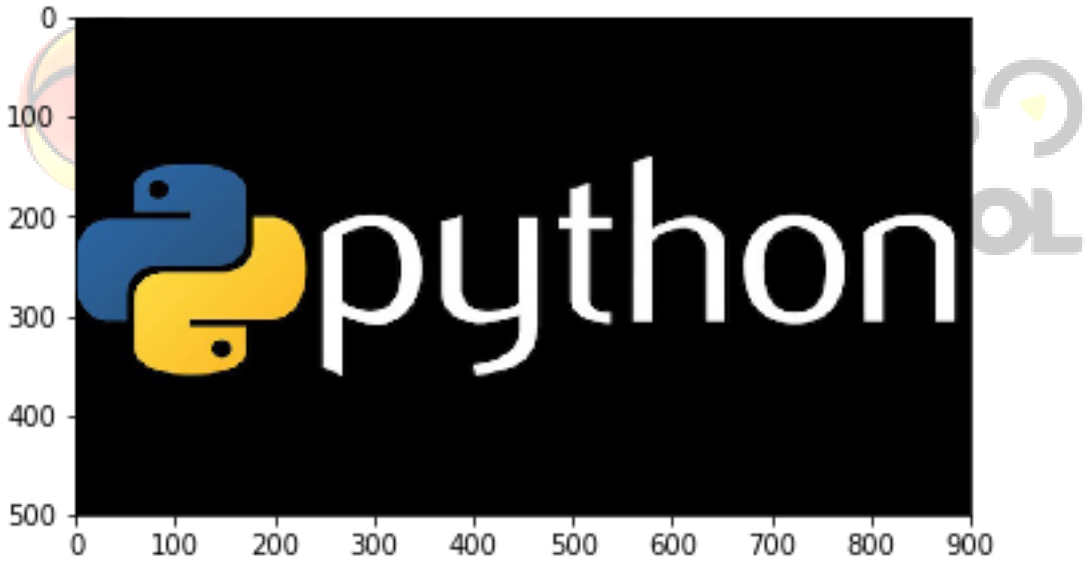
Exercício 4

Calcule o número de pixels pretos contidos na imagem abaixo. Lembre-se, um pixel é considerado "preto" quando seu valor é zero nos 3 canais de cores ao mesmo tempo (e.g. image[0, 0, :] == 0 é verdadeiro se o pixel na posição [0, 0] for preto).

O seu código não deve conter nenhum tipo de loop (for/while)

```
image = np.array(Image.open('python_image.jpg'))
print('Shape da imagem: {}, dtype: {}'.format(image.shape, image.dtype))
plt.imshow(image)
plt.show()



print(np.sum(np.sum(image, axis=2) == 0))
```

Shape da imagem: (500, 900, 3), dtype: uint8



355848

Exercício 5

Escreva uma função que retorne os índices dos k maiores valores de um array. Teste sua implementação.

O seu código não deve conter nenhum tipo de loop (for/while)

Exercício 6

Leia o arquivo csv **googleplaystore.csv** e realize a seguinte atividade sobre o dataset:

1. Faça um gráfico de barras contendo os top 5 apps por número de instalação.

2. Faça um gráfico de pizza (pie chart) mostrando as categorias de apps existentes no dataset de acordo com a frequência em que elas aparecem.
3. Mostre qual o app mais caro existente no dataset.
4. Mostre quantos apps são classificados como Mature 17+.
5. Mostre o top 10 apps por número de reviews bem como o respectivo número de reviews. Ordene a lista de forma decrescente por número de reviews.

Dica: faça a leitura do arquivo csv utilizando módulo csv do python. Fazendo a leitura desta maneira evita-se problemas com valores entre aspas que possuem vírgula. Utilize o código abaixo para fazer a leitura do arquivo:

```
with open('googleplaystore.csv', 'r') as f:
    arquivo_csv = csv.reader(f, delimiter=',', quotechar='"')
    for i, row in enumerate(arquivo_csv):
        print('linha: {}, conteúdo: {}'.format(i, row)
```

Numpy Broadcasting

"The term broadcasting describes how numpy treats arrays with different shapes during arithmetic operations."

Referência: https://docs.scipy.org/doc/numpy/user/theory.broadcasting.html#array-broadcasting-in-numpy

Para fazer operações matemáticas *elementwise* arrays os arrays devem ter o **mesmo shape**. Sendo assim, o código abaixo funciona?

```
a = np.arange(1, 4)

b = 2

print(a)

print(a * b)
```

```
In [1]: import numpy as np

In [2]: a = np.arange(1, 4)
        b = 2
        print(a)
        print(a * b)

        [1 2 3]
        [2 4 6]
```

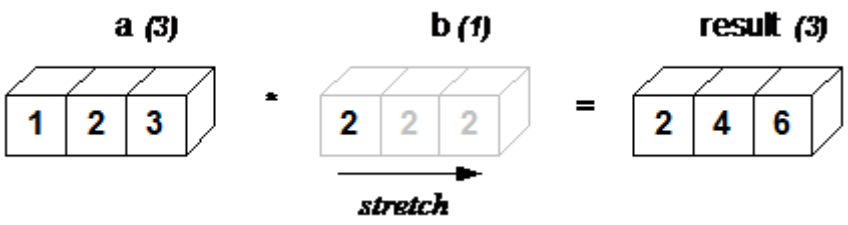
Para fazer operações matemáticas *elementwise* os arrays envolvidos devem ter o **mesmo shape**. Sendo assim, o código abaixo funciona?

```
a = np.arange(1, 4)

b = 2

print(a)

print(a * b)
```

O que faz com que o código acima funcione? Resposta: Broadcasting.

"Subject to **certain constraints**, the smaller array is “broadcast” across the larger array so that they have compatible shapes. Broadcasting provides a means of vectorizing array operations so that looping occurs in C instead of Python."

Referência: https://docs.scipy.org/doc/numpy/user/theory.broadcasting.html#array-broadcasting-in-numpy

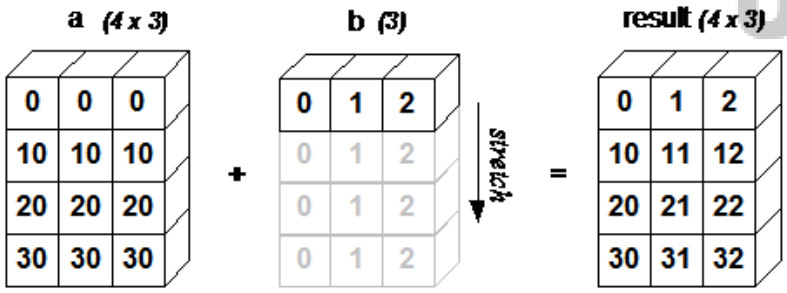


Para fazer operações matemáticas *elementwise* arrays os arrays devem ter o **mesmo shape**. Sendo assim, o código abaixo funciona?

```
a = np.array([[ 0.0,  0.0,  0.0],
        [10.0, 10.0, 10.0],
        [20.0, 20.0, 20.0],
        [30.0, 30.0, 30.0]]) # 4x3


b = np.array([[0.0, 1.0, 2.0]]) # 1x3


print(a + b)
```

```
In [3]: a = np.array([[ 0.0,  0.0,  0.0],
                      [10.0, 10.0, 10.0],
                      [20.0, 20.0, 20.0],
                      [30.0, 30.0, 30.0]]) # 4x3

        b = np.array([[0.0, 1.0, 2.0]]) # 1x3

        print(a + b)
```

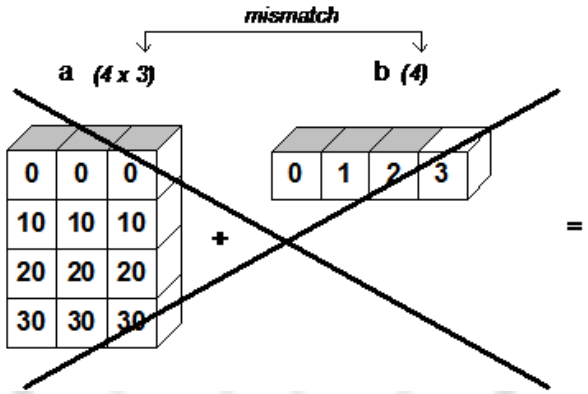
```
[[ 0.  1.  2.]
 [10. 11. 12.]
 [20. 21. 22.]
 [30. 31. 32.]]
```




```
a = np.array([[ 0.0,  0.0,  0.0],
        [10.0, 10.0, 10.0],
        [20.0, 20.0, 20.0],
        [30.0, 30.0, 30.0]]) # 4x3


b = np.array([0.0, 1.0, 2.0, 3.0]).reshape((4, 1)) # 4x1


print(a + b)
```

```
In [4]: a = np.array([[ 0.0,  0.0,  0.0],
                      [10.0, 10.0, 10.0],
                      [20.0, 20.0, 20.0],
                      [30.0, 30.0, 30.0]]) # 4x3

        b = np.array([0.0, 1.0, 2.0, 3.0]).reshape((4, 1)) # 4x1

        print(a + b)
```

```
[[ 0.  0.  0.]
 [11. 11. 11.]
 [22. 22. 22.]
 [33. 33. 33.]]
```



Adicionando dimensões no array

- Utilizando np.newaxis ou None na indexação
- Utilizando np.expand_dims(...)

```
In [5]: a = np.array([1,2,3])
        print('Shape de a: {}'.format(a.shape))
        b = a[np.newaxis, :] # adicionando uma dimensão no início
        print('Shape de b: {}'.format(b.shape))
        c = a[:, np.newaxis] # adicionando uma dimensão no fim
        print('Shape de c: {}'.format(c.shape))
```

```
Shape de a: (3,)
Shape de b: (1, 3)
Shape de c: (3, 1)
```

```
In [6]: print('np.newaxis é None? {}'.format(np.newaxis is None))
        a = np.array([1,2,3])
        print('Shape de a: {}'.format(a.shape))
        a = a[None, :]
        print('Shape de a: {}'.format(a.shape))
```

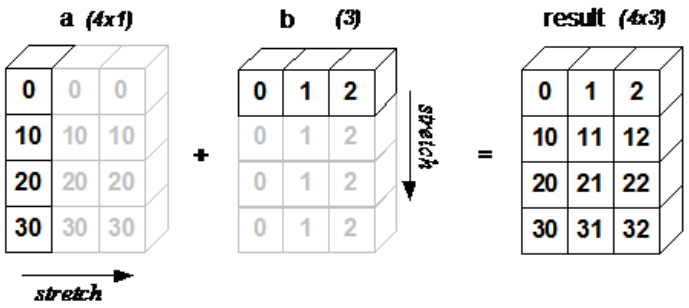
```
np.newaxis é None? True
Shape de a: (3,)
Shape de a: (1, 3)
```

```
In [7]: a = np.array([1,2,3])
        print('Shape de a: {}'.format(a.shape))
        a = np.expand_dims(a, 1)
        print('Shape de a: {}'.format(a.shape))
```

```
Shape de a: (3,)
Shape de a: (3, 1)
```

Para fazer operações matemáticas *elementwise* arrays os arrays devem ter o **mesmo shape**. Sendo assim, o código abaixo funciona?

```
a = np.array([0.0, 10.0, 20.0, 30.0])

b = np.array([0.0, 1.0, 2.0])

c = a[:, np.newaxis] + b[np.newaxis, :] # (4x1) + (1x3)


print(c)
```




```
In [8]: a = np.array([0.0, 10.0, 20.0, 30.0])
        b = np.array([0.0, 1.0, 2.0])
        c = a[:, np.newaxis] + b[np.newaxis, :]

        print(c)
```

```
[[ 0.  1.  2.]
 [10. 11. 12.]
 [20. 21. 22.]
 [30. 31. 32.]]
```

Combinando operações de redução e broadcasting.

```
In [9]: a = np.arange(9).reshape(3, 3)
        print('Conteúdo de a:\n', a)
        media_linhas = np.mean(a, axis=0)
        print('Media das linhas: {}, shape da média das linhas: {}'.format(media_linhas, media_linhas.shape))
        media_linhas = np.mean(a, axis=0, keepdims=True)
        print('Media das linhas: {}, shape da média das linhas: {}'.format(media_linhas, media_linhas.shape))
        a2 = a - media_linhas
        print('Conteúdo de a com média das linhas subtraídas:\n', a2)
```

```
Conteúdo de a:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Media das linhas: [3. 4. 5.], shape da média das linhas: (3,)
Media das linhas: [[3. 4. 5.]], shape da média das linhas: (1, 3)
Conteúdo de a com média das linhas subtraídas:
 [[-3. -3. -3.]
 [ 0.  0.  0.]
 [ 3.  3.  3.]]
```

In [10]:
```
a = np.arange(9).reshape(3, 3)
print('Conteúdo de a:\n', a)
media_colunas = np.mean(a, axis=1)
print('Media das colunas: {}, shape da média das colunas: {}'.format(media_colunas, media_colunas.shape))
media_colunas = np.mean(a, axis=1, keepdims=True)
print('Media das colunas: {}, shape da média das colunas: {}'.format(media_colunas, media_colunas.shape))
a2 = a - media_colunas
print('Conteúdo de a com média das colunas subtraídas:\n', a2)
```

```
Conteúdo de a:
 [[0 1 2]
 [3 4 5]
 [6 7 8]]
Media das colunas: [1. 4. 7.], shape da média das colunas: (3,)
Media das colunas: [[1.]
 [4.]
 [7.]], shape da média das colunas: (3, 1)
Conteúdo de a com média das colunas subtraídas:
 [[-1.  0.  1.]
 [-1.  0.  1.]
 [-1.  0.  1.]]
```

Combinando operações de redução e broadcasting em múltiplas dimensões.

In [11]:
```
a = np.random.randint(0, 300, (10, 10, 3))
print('Shape de a: {}'.format(a.shape))
print('Média dos canais: {}'.format(np.mean(a, axis=(0,1))))
media = np.mean(a, axis=(0, 1), keepdims=True)
print('Shape da média: {}'.format(media.shape))
a = a - media
print('Média do canal 0: {}'.format(np.mean(a[:, :, 0])))
print('Média do canal 1: {}'.format(np.mean(a[:, :, 1])))
print('Média do canal 2: {}'.format(np.mean(a[:, :, 2])))
```

```
Shape de a: (10, 10, 3)
Média dos canais: [146.12 157.98 156.72]
Shape da média: (1, 1, 3)
Média do canal 0: -4.547473508864641e-15
Média do canal 1: 7.958078640513123e-15
Média do canal 2: -2.2737367544323206e-15
```

In [12]:
```
a = np.arange(27).reshape(3, 3, 3)
scale = np.array([1.2, 1.5, 0.8]).reshape(1, 1, 3)
# print(a[:, :, 0])
# print(a[:, :, 1])
# print(a[:, :, 2])
print('Shape de a: {}'.format(a.shape))
print('Shape da escala (RGB): {}'.format(scale.shape))
a = a + scale
print('Conteúdo de a:\n', a[:, :, 0], '\n\n', a[:, :, 1], '\n\n', a[:, :, 2])
```

```
Shape de a: (3, 3, 3)
Shape da escala (RGB): (1, 1, 3)
Conteúdo de a:
 [[ 1.2  4.2  7.2]
 [10.2 13.2 16.2]
 [19.2 22.2 25.2]]

 [[ 2.5  5.5  8.5]
 [11.5 14.5 17.5]
 [20.5 23.5 26.5]]

 [[ 2.8  5.8  8.8]
 [11.8 14.8 17.8]
 [20.8 23.8 26.8]]
```

Leitura recomendada:

Documentação oficial sobre broadcasting:
https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html

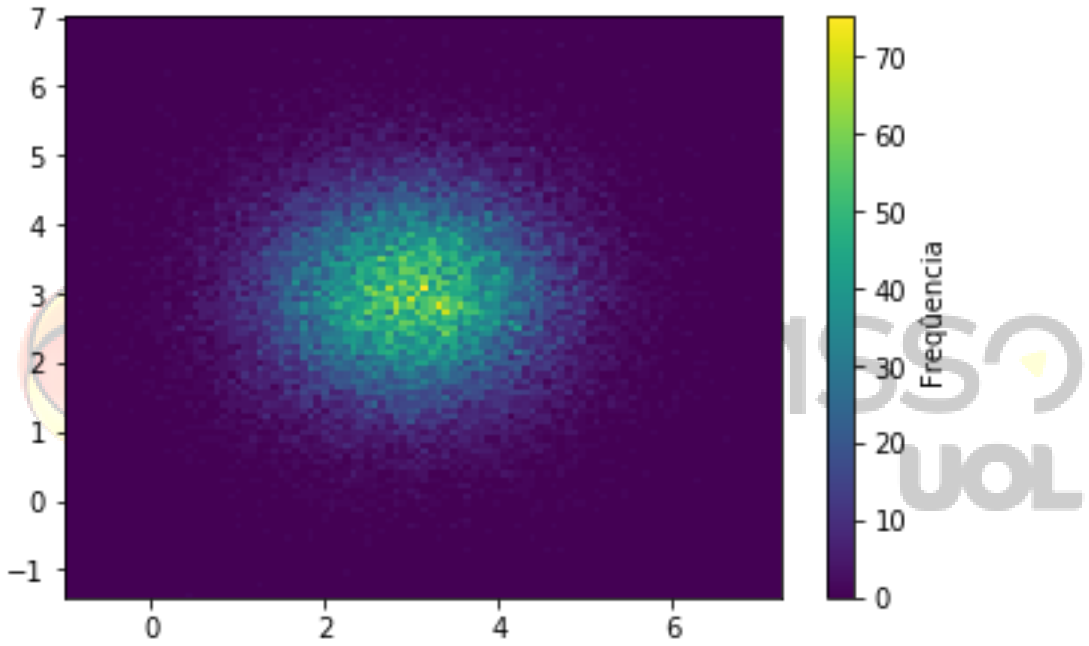
Exercícios - Parte 2

Exercício 1

Dada a distribuição normal 2D centralizada em 3 mostrada abaixo. Remova a média do das colunas (x, y) para centralizar a distribuição em zero.

O seu código não deve conter nenhum tipo de loop (for/while)

```
a = np.random.normal(loc=3, size=(50000, 2))
plt.hist2d(a[:, 0], a[:, 1], bins=100)
cbar = plt.colorbar()
cbar.ax.set_ylabel('Frequência')
plt.show()
```



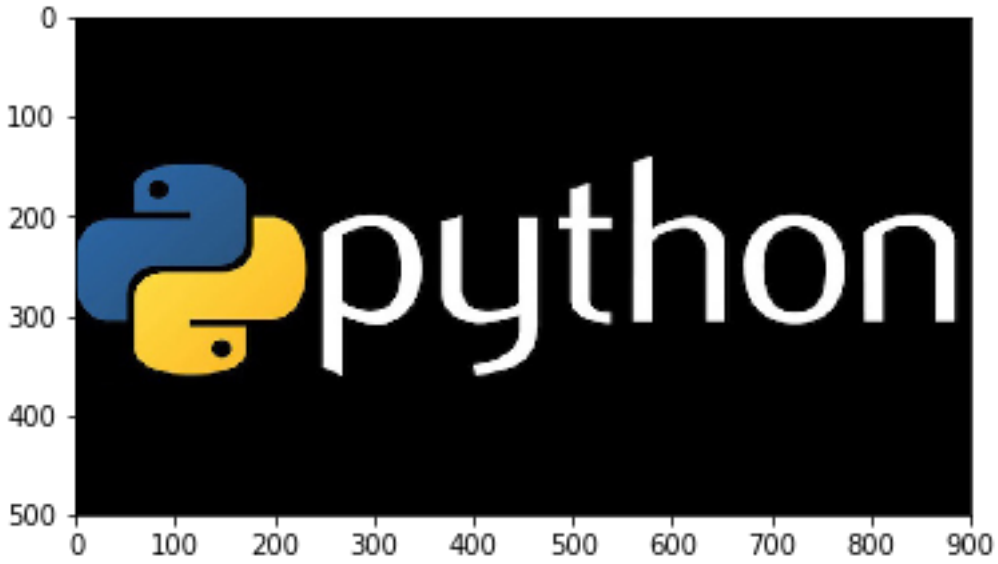
Exercício 2

Calcule a média de cada um dos canais da imagem abaixo (RGB), após calcular a média, para cada pixel da imagem, remova a média do seu respectivo canal.

O seu código não deve conter nenhum tipo de loop (for/while)

```
image = np.array(Image.open('python_image.jpg'))
print('Shape da imagem: {}, dtype: {}'.format(image.shape, image.dtype))
plt.imshow(image)
plt.show()
```

Shape da imagem: (500, 900, 3), dtype: uint8



Exercício 3

Faça a normalização dos dados a seguir utilizando a abordagem Z-score. A abordagem Z-score consiste em centralizar os dados em 0 (remover a média) e corrigir deformidades. A fórmula da normalização é a seguir:

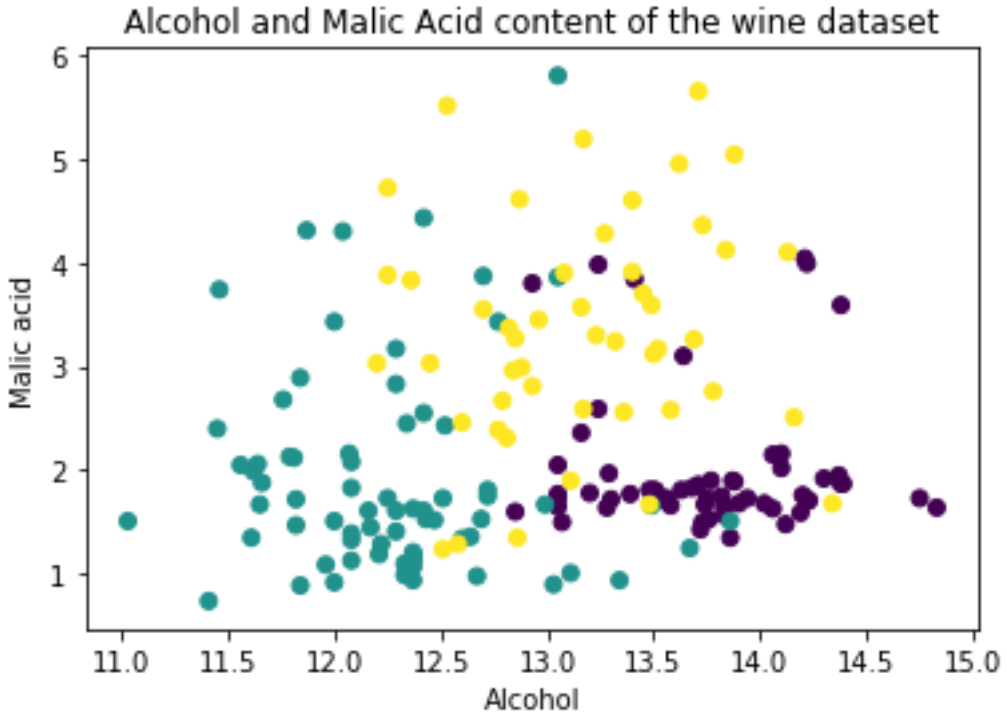
x_{i} = \frac{x_{i} - \mu_{i}}{\sigma_{i}}

Onde x_{i}*xi* é um atributo do dataset.

O seu código não deve conter nenhum tipo de loop (for/while)

```
dataset = np.loadtxt('wine_data.csv', delimiter=',')
dataset = dataset[:, :4] # ['Class label', 'Alcohol', 'Malic acid']
print(dataset.shape)
plt.scatter(dataset[:, 1], dataset[:, 2], c=dataset[:, 0])
plt.title('Alcohol and Malic Acid content of the wine dataset')
plt.xlabel('Alcohol')
plt.ylabel('Malic acid')
plt.show()
```

(178, 4)



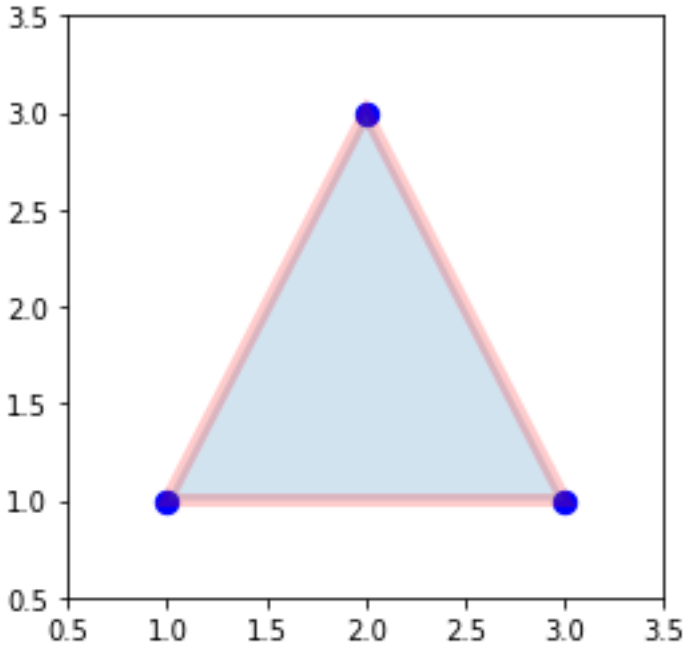
Exercício 4

Dada a função abaixo que recebe 3 vértices 2D (3x2) e plota o triângulo correspondente, calcule o centro do triângulo e utilize a função plot_triangle para plotar o triângulo juntamente com o seu centro.

O seu código não deve conter nenhum tipo de loop (for/while)

```
def plot_triangle(vertices, center=None):
    trishape = plt.Polygon(vertices, edgecolor='r', alpha=0.2, lw=5)
    _, ax = plt.subplots(figsize=(4, 4))
    ax.add_patch(trishape)
    ax.set_ylim([.5, 3.5])
    ax.set_xlim([.5, 3.5])
    if center is not None:
        ax.scatter(*center, color='g', marker='D', s=70)
    ax.scatter(vertices[:, 0], vertices[:, 1], color='b',  s=70)
    plt.show()


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

```
plot_triangle(coordenadas)
```



Exercício 5

Dado o array 2D abaixo, normalize cada linha pela sua própria norma L2. A fórmula da normalização L2 é a seguinte:

$$\frac{x_i}{\sqrt{x_1^2 + x_2^2 + ...x_n^2}}$$

para cada item

$$i \in 0, 1, ..., n$$

do array.

Você deve realizar todos os cálculos. Para este exercício você não deve usar a função np.linalg.norm(...).

O seu código não deve conter nenhum tipo de loop (for/while)

```
a = np.random.normal(size=(20, 100))
```

Pandas

- https://pandas.pydata.org/pandas-docs/stable/dsintro.html
- Pandas possui duas estruturas básicas: Series e DataFrame
- Series é utilizado para representar séries unidimensionais de informações, como um atributo (i.e. coluna) ou uma instância (i.e. linha)
- DataFrame é utilizado para representar informações bidimensionais, como datasets
- Instalar o Pandas

```
C:\WINDOWS\system32>pip install pandas
Collecting pandas
  Downloading https://files.pythonhosted.org/packages/b1/69/fcc29820befae2b96fd0b01225577af653e87cd0914634bb2d372a457bd7/pandas-0.25.1-cp37-cp37m-win_amd64.whl (9.2MB)
     |████████████████████████████████| 9.2MB 1.3MB/s
Requirement already satisfied: numpy>=1.13.3 in c:\program files\python37\lib\site-packages (from pandas) (1.17.2)
Requirement already satisfied: pytz>=2017.2 in c:\users\mateus.balen\appdata\roaming\python\python37\site-packages (from pandas) (2019.1)
Requirement already satisfied: python-dateutil>=2.6.1 in c:\users\mateus.balen\appdata\roaming\python\python37\site-packages (from pandas) (2.8.0)
Requirement already satisfied: six>=1.5 in c:\users\mateus.balen\appdata\roaming\python\python37\site-packages (from python-dateutil>=2.6.1->pandas) (1.11.0)
Installing collected packages: pandas
Successfully installed pandas-0.25.1
WARNING: You are using pip version 19.2.1, however version 19.2.3 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

Series

- Um conjunto deda dados unidimensionais (array, atributo)
- Pode ser resultado de uma seleção sobre um DataFrame ou poder ser criada diretamente

Documentação oficial:

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display as print_dataframe
```

In [3]:
```
s = pd.Series(['João', 'Roberto', 'Júlia'])
print(s)
print('Valor para o índice 0', s[0])
```

```
0       João
1    Roberto
2      Júlia
dtype: object
Valor para o índice 0 João
```

In [4]:
```
s = dict(roberto=72, camila=52, pedro=65)
s = pd.Series(s)
print(s)
print('Valor para o índice roberto', s['roberto'])
print('Índices da Serie:', s.index)
```

```
roberto    72
camila     52
pedro      65
dtype: int64
Valor para o índice roberto 72
Índices da Serie: Index(['roberto', 'camila', 'pedro'], dtype='object')
```

DataFrame

- É a estrutura de dados principal do Pandas
- Representa uma **tabela** similar à tabelas de bancos de dados
- Permite dados heterogêneos (uma coluna do tipo float, outra do tipo str, por exemplo)
- Permite operações semelhantes a operações de banco de dados (Seleção, agregação, etc...)

```
In [5]: df = pd.DataFrame(
            [['a', 'b', 'c'],
             ['d', 4, 5],
             [6, 7, 8]],
            index=['linha 0', 'linha 1', 'linha 2'],
            columns=['coluna 0', 'coluna 1', 'coluna 2'],
            copy=True
        )
        print_dataframe(df)
```

	coluna 0	coluna 1	coluna 2
linha 0	a	b	c
linha 1	d	4	5
linha 2	6	7	8

- Existem diversas formas de iniciar dataframes, uma das mais comuns sendo carregar a partir de um arquivo. O mais comum é a partir de um csv utilizando pandas.read_csv(...): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

```
In [6]: df = pd.read_csv('heart.csv', header=0)
        print_dataframe(df.head(5))
        print_dataframe(df[:5])
        print('Shape do dataframe:', df.shape)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

```
Shape do dataframe: (303, 14)
```

- Da mesma forma que é possível ler arquivos csv, é possível escrevê-los usando o método pandas.dataframe.to_csv(..): http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.to_csv.html

```
In [7]: df = pd.DataFrame([[0, 1, 2], [3, 4, 5]], index=['a', 'b'], columns=['x', 'y', 'z'])
        df.to_csv('exemplo_escrita.csv')

        # lê do arquivo recém criado
        df2= pd.read_csv('exemplo_escrita.csv', index_col=0)
        print_dataframe(df2)
```

	x	y	z
a	0	1	2
b	3	4	5

Indexação simples

- https://pandas.pydata.org/pandas-docs/stable/indexing.html
- DataFrames são indexados de duas formas: por linhas ou colunas
- As linhas podem ser indexadas de acordo com o índice do DataFrame
- O mesmo raciocínio é aplicado às colunas

Indexando colunas

- Podemos indexar colunas diretamente: df['NomeColuna'], ou múltiplas colunas df[['Col1', 'Col2']]

```
In [8]: df = pd.read_csv('heart.csv')
        print(df.index)  # imprime o índice das linhas
```

```
RangeIndex(start=0, stop=303, step=1)
```

```
In [9]: df = pd.read_csv('heart.csv')
        print(df.columns)  # imprime o índice das colunas
```

```
Index(['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach',
       'exang', 'oldpeak', 'slope', 'ca', 'thal', 'target'],
      dtype='object')
```

```
In [10]: print(df['age'][:5])
```

```
0    63
1    37
2    41
3    56
4    57
Name: age, dtype: int64
```

```
In [11]: print_dataframe(df[['age', 'sex']][:5])
```

	age	sex
0	63	1
1	37	1
2	41	0
3	56	1
4	57	0

Indexando linhas

Podemos indexar linhas de duas maneiras:

- utilizando df.loc[indice] caso o índice do dataframe não seja composto por inteiros
- utilizando df.iloc[indice] caso o índice do dataframe seja composto por inteiros

```
In [12]: print_dataframe(df.loc[1])
```

```
age          37.0
sex           1.0
cp            2.0
trestbps    130.0
chol        250.0
fbs           0.0
restecg       1.0
thalach     187.0
exang         0.0
oldpeak       3.5
slope         0.0
ca            0.0
thal          2.0
target        1.0
Name: 1, dtype: float64
```

```
In [13]: print_dataframe(df.iloc[[0, 3, 5]])
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
5	57	1	0	140	192	0	1	148	0	0.4	1	0	1	1

Indexação composta

- https://pandas.pydata.org/pandas-docs/stable/advanced.html
- A indexação padrão dos dataframes assume apenas um valor por índice
- Todavia, é possível utilizar índices compostos
- Útil quando estamos visualizando informações agregadas

```
In [14]: index = pd.MultiIndex.from_tuples([('2018', 'Ka'), ('1980', 'Fusca'), ('2014', 'Ka')], names=['ano', 'modelo'])

df = pd.DataFrame(
    data=[[4, 50000], [2, 5000], [2, 15000]],
    index=index,
    columns=['Número de portas', 'Valor']
)
print_dataframe(df)
```

		Número de portas	Valor
ano	modelo		
2018	Ka	4	50000
1980	Fusca	2	5000
2014	Ka	2	15000

```
In [15]: index = pd.MultiIndex.from_tuples([('2018', 'Ka'), ('1980', 'Fusca'), ('2014', 'Ka')], names=['ano', 'modelo'])

df = pd.DataFrame(
    data=[[4, 50000], [2, 5000], [2, 15000]],
    index=index,
    columns=['Número de portas', 'Valor']
)
print_dataframe(df.loc[[('1980', 'Fusca'), ('2014', 'Ka')]])
```

		Número de portas	Valor
ano	modelo		
1980	Fusca	2	5000
2014	Ka	2	15000

Seleção

- A seleção em pandas é similar ao que ocorre em bancos de dados relacionais
- É possível selecionar exemplos com base no valor de diversas colunas, simultaneamente
- https://pandas.pydata.org/pandas-docs/stable/indexing.html

```
In [16]: np.random.seed(0)
df = pd.DataFrame(data=np.random.random((3, 3)), index=['a', 'b', 'c'], columns=['x', 'y', 'z'])
print_dataframe(df)
```

	x	y	z
a	0.548814	0.715189	0.602763
b	0.544883	0.423655	0.645894
c	0.437587	0.891773	0.963663

Linhas em que o valor da coluna x é maior que 0.5

```
In [17]: df[df['x'] > 0.5]
```

Out[17]:

	x	y	z
a	0.548814	0.715189	0.602763
b	0.544883	0.423655	0.645894

```
In [18]: mascara = df['x'] > 0.5
         print(mascara)
         df[mascara]
```

```
a     True
b     True
c    False
Name: x, dtype: bool
```

Out[18]:

	x	y	z
a	0.548814	0.715189	0.602763
b	0.544883	0.423655	0.645894

Linhas em que o valor da coluna x é maior que 0.5 **E** o valor da coluna y é menor que 0.7

```
In [19]: df[(df['x'] > 0.5) & (df['y'] < 0.7)]
```

Out[19]:

	x	y	z
b	0.544883	0.423655	0.645894

Linhas em que o valor da coluna x é maior que 0.5 **OU** o valor da coluna y é menor que 0.7

```
In [20]: df[(df['x'] > 0.5) | (df['y'] < 0.7)]
```

Out[20]:

	x	y	z
a	0.548814	0.715189	0.602763
b	0.544883	0.423655	0.645894

In [21]:
```
df = pd.read_csv('heart.csv')
print_dataframe(df[df['sex'] == 0][:5])
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1
6	56	0	1	140	294	0	0	153	0	1.3	1	0	2	1
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
14	58	0	3	150	283	1	0	162	0	1.0	2	0	2	1

In [22]:
```
df = pd.read_csv('heart.csv')
print_dataframe(df[(df['sex'] == 0) &
    ((df['cp'] == 2) | (df['cp'] == 3))][:5])
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
11	48	0	2	130	275	0	1	139	0	0.2	2	0	2	1
14	58	0	3	150	283	1	0	162	0	1.0	2	0	2	1
15	50	0	2	120	219	0	1	158	0	1.6	1	0	2	1
16	58	0	2	120	340	0	1	172	0	0.0	2	0	2	1
17	66	0	3	150	226	0	1	114	0	2.6	0	0	2	1

In [23]:
```
df = pd.read_csv('heart.csv')
cp_mask = df['cp'][:10] == 3
print(cp_mask)
print('Soma da máscara:', np.sum(cp_mask))
```

```
0     True
1    False
2    False
3    False
4    False
5    False
6    False
7    False
8    False
9    False
Name: cp, dtype: bool
Soma da máscara: 1
```

Seleção

- A estrutura loc também aceita indexação por linhas e colunas
- Primeiro se passa a linha desejada, e depois o nome da coluna

```
In [24]: np.random.seed(0)
         df = pd.DataFrame(data=np.random.random((3, 3)),
                 index=['a', 'b', 'c'],
                 columns=['x', 'y', 'z'])
         display(df)
```

	x	y	z
a	0.548814	0.715189	0.602763
b	0.544883	0.423655	0.645894
c	0.437587	0.891773	0.963663

Seleciona a linha **a**

```
In [25]: df.loc['a']

Out[25]: x    0.548814
         y    0.715189
         z    0.602763
         Name: a, dtype: float64
```

Seleciona a linha **a** e a coluna **x**

```
In [26]: print_dataframe(df)
         df.loc[['a', 'b'], ['x', 'y']]
```

	x	y	z
a	0.548814	0.715189	0.602763
b	0.544883	0.423655	0.645894
c	0.437587	0.891773	0.963663

Out[26]:

	x	y
a	0.548814	0.715189
b	0.544883	0.423655

Seleciona a coluna **x**

```
In [27]: df['x']

Out[27]: a    0.548814
         b    0.544883
         c    0.437587
         Name: x, dtype: float64
```

Pandas e Numpy

A qualquer momento podemos converter os dados para Numpy utilizando a propriedade de DataFrames e Series: .values.

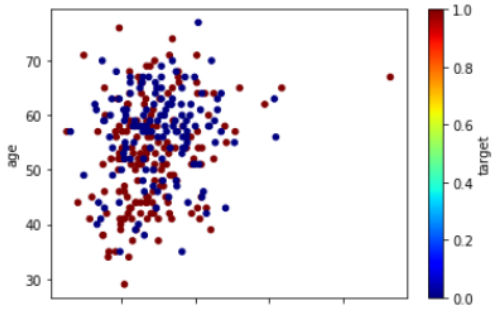
```
In [28]: df = pd.read_csv('heart.csv', header=0)
         np_array = df.values
         print(type(np_array), np_array.shape, np_array.dtype)

         <class 'numpy.ndarray'> (303, 14) float64
```

Visualização

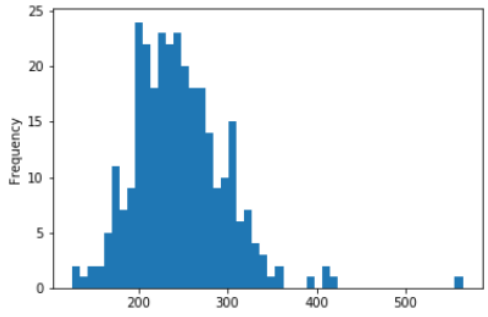
- Pandas fornece *atalhos* para visualizar dados, para que não precisamos chamar o matplotplib.

Documentação oficial: https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html

```
In [29]: df = pd.read_csv('heart.csv', header=0)
         df.plot.scatter(x='chol', y='age',
                         c='target', cmap='jet')
         plt.show()
```




```
In [30]: df = pd.read_csv('heart.csv', header=0)
         df['chol'].plot.hist(bins=50)
         plt.show()
```



```
In [31]: df = pd.read_csv('heart.csv', header=0)
men_cp = df[df['sex'] == 0]['cp']
men_cp.value_counts().plot.pie(
    title='Tipos de dores no peito (Mulheres)')
plt.show()
```

Exercícios – Parte 3

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display as print_dataframe
```

Exercício 1

Leia o arquivo actors.csv e faça os seguintes cálculos sobre o conjunto de dados utilizando **Pandas**:

1. O ator/atriz com maior número de filmes e o respectivo número de filmes.
2. A média do número de filmes.
3. O ator/atriz com a maior média por filme.
4. O nome do(s) filme(s) mais frequente(s) e sua respectiva frequência.

Exercício 2

Leia o arquivo csv googleplaystore.csv e realize as seguintes atividades sobre o dataset utilizando Pandas:

1. Faça um gráfico de barras contendo os top 5 apps por número de instalação.
2. Faça um gráfico de pizza (pie chart) mostrando as categorias de apps existentes no dataset de acordo com a frequência em que elas aparecem.
3. Mostre qual o app mais caro existente no dataset.
4. Mostre quantos apps são classificados como Mature 17+.
5. Mostre o top 10 apps por número de reviews bem como o respectivo número de reviews. Ordene a lista de forma decrescente por número de reviews.

Pandas – Ordenação, Agregação, Junção e Split

Ordenação

- https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.sort_values.html
- Assim como numpy, pandas também suporta ordenação de linhas/colunas
- A ordenação mantém a integridade das linhas/colunas

```
In [1]: import pandas as pd
        import numpy as np
        from IPython.display import display
```

```
In [2]: np.random.seed(0)
        df = pd.DataFrame(
            data=np.random.random((3, 3)),
            index=['a', 'b', 'c'],
            columns=['x', 'y', 'z']
        )
        display(df)
```

	x	y	z
a	0.548814	0.715189	0.602763
b	0.544883	0.423655	0.645894
c	0.437587	0.891773	0.963663

Ordena as linhas

```
In [3]: df.sort_values(by='x', axis=0)
```

Out[3]:

	x	y	z
c	0.437587	0.891773	0.963663
b	0.544883	0.423655	0.645894
a	0.548814	0.715189	0.602763

Ordena as colunas

```
In [4]: df.sort_values(by='a', axis=1)
```

Out[4]:

	x	z	y
a	0.548814	0.602763	0.715189
b	0.544883	0.645894	0.423655
c	0.437587	0.963663	0.891773

Função map

- pandas.DataFrame.apply

- Semelhante a função nativa de Python
- Aplica uma função a cada uma das linhas ou colunas de um DataFrame

In [5]:
```
dfa = pd.DataFrame(
    data=[
        [35, 'M', 170, 70],
        [32, 'F', 165, 55],
        [28, 'F', 160, 65],
        [30, 'M', 168, 68],
    ],
    index=pd.MultiIndex.from_tuples(
        [('cleiton', 'SP'), ('paola', 'SP'), ('laís', 'SP'), ('rafael', 'RJ')],
        names=['nome', 'cidade']
    ),
    columns=['idade', 'sexo', 'altura', 'peso'],
)

dfb = pd.DataFrame(
    data=[
        [31, 'F', 172, 51],
        [32, 'F', 165, 55],
    ],
    index=pd.MultiIndex.from_tuples([
        ('paola', 'SM'),('paola', 'SP')],
        names=['nome', 'cidade']),
    columns=['idade', 'sexo', 'altura', 'peso'],
)

display(dfa)
display(dfb)
```

		idade	sexo	altura	peso
nome	**cidade**				
cleiton	SP	35	M	170	70
paola	SP	32	F	165	55
laís	SP	28	F	160	65
rafael	RJ	30	M	168	68

		idade	sexo	altura	peso
nome	**cidade**				
paola	SM	31	F	172	51
	SP	32	F	165	55

In [6]:
```
def soma_idade(row):
    """
    Adiciona mais 2 anos a idade.
    """
    row['idade'] += 2
    return row

print('antes:')
display(dfa)
```

```
antes:
```

		idade	sexo	altura	peso
nome	cidade				
cleiton	SP	35	M	170	70
paola	SP	32	F	165	55
laís	SP	28	F	160	65
rafael	RJ	30	M	168	68

In [7]:
```
print('depois:')
dfa.apply(soma_idade, axis=1)  # aplica a cada uma das linhas. use axis=0 para aplicar a cada uma das colunas
```

```
depois:
```

Out[7]:

		idade	sexo	altura	peso
nome	cidade				
cleiton	SP	37	M	170	70
paola	SP	34	F	165	55
laís	SP	30	F	160	65
rafael	RJ	32	M	168	68

Remoção de duplicatas

- pandas.DataFrame.drop_duplicates
- Remove as duplicadas nas colunas especificadas
- Pode remover todas as linhas duplicadas, se nenhum conjunto de colunas for dado

In [8]:
```
df = pd.DataFrame(
    [['a', 'a', 'b'],
     ['a', 'a', 'c'],
     ['a', 'a', 'b']],
    columns=['C1', 'C2', 'C3'],
    index=['I1', 'I2', 'I3']
)

df
```

Out[8]:

	C1	C2	C3
I1	a	a	b
I2	a	a	c
I3	a	a	b

```
In [9]: df.drop_duplicates()
```

Out[9]:

	C1	C2	C3
l1	a	a	b
l2	a	a	c

```
In [10]: df.drop_duplicates(subset=['C1', 'C2'])
```

Out[10]:

	C1	C2	C3
l1	a	a	b

Medidas populacionais

Assim como o numpy, em pandas é possível extrair medidas populacionais dos atributos do DataFrame:

- Média
- Mediana

```
In [11]: dfa = pd.DataFrame(
             data=[
                 [35, 'M', 170, 70],
                 [32, 'F', 165, 55],
                 [28, 'F', 160, 65],
                 [30, 'M', 168, 68],
             ],
             index=pd.MultiIndex.from_tuples(
                 [('cleiton', 'SP'), ('paola', 'SP'), ('laís', 'SP'), ('rafael', 'RJ')],
                 names=['nome', 'cidade']
             ),
             columns=['idade', 'sexo', 'altura', 'peso'],
         )
```

```
In [12]: dfa.mean()
```

```
Out[12]: idade      31.25
         altura    165.75
         peso       64.50
         dtype: float64
```

```
In [13]: dfa.median()
```

```
Out[13]: idade      31.0
         altura    166.5
         peso       66.5
         dtype: float64
```

```
In [14]: dfa['idade'].mean()
```

```
Out[14]: 31.25
```

Imputação de dados faltantes

- Quando trabalhamos dados tabelados, é comum encontrar registros que possuem valores faltantes para alguns atributos

- Alguns algoritmos de aprendizado de máquina não conseguem tratar estes valores faltantes em tempo de execução, sendo necessário tratá-los em tempo de pré-processamento
- Existem duas opções para resolver isso:
    - Excluir o registro que possui valores faltantes
    - Imputar um valor para aquele atributo (Substituir com a média, mediana, moda, etc)

In [15]:
```
dfa = pd.DataFrame(
    data=[
        [31, 'F', 172, np.nan],
        [32, np.nan, 165, 55],
        [np.nan, 'F', 160, 65],
    ],
    index=pd.MultiIndex.from_tuples(
        [('paola', 'SM'), ('paola', 'SP'), ('laís', 'SP')],
        names=['nome', 'cidade']
    ),
    columns=['idade', 'sexo', 'altura', 'peso'],
)
```

In [16]:
```
display(dfa)
dfa['idade'].fillna(dfa['idade'].mean(), inplace=True)  # preenche com a média
dfa['peso'].fillna(dfa['peso'].median(), inplace=True)  # preenche com a mediana
mode = dfa['sexo'].mode()
dfa['sexo'].fillna(mode[0], inplace=True)  # preenche com a moda

display(dfa)
```

		idade	sexo	altura	peso
nome	cidade				
paola	SM	31.0	F	172	NaN
	SP	32.0	NaN	165	55.0
laís	SP	NaN	F	160	65.0

		idade	sexo	altura	peso
nome	cidade				
paola	SM	31.0	F	172	60.0
	SP	32.0	F	165	55.0
laís	SP	31.5	F	160	65.0

In [17]:
```
dfa = pd.DataFrame(
    data=[
        [31, 'F', 172, np.nan],
        [32, np.nan, 165, 55],
        [np.nan, 'F', 160, 65],
        [38, 'M', 172, 70],
    ],
    index=pd.MultiIndex.from_tuples(
        [('paola', 'SM'), ('paola', 'SP'), ('laís', 'SP'),('felipe', 'RS')],
        names=['nome', 'cidade']
    ),
    columns=['idade', 'sexo', 'altura', 'peso'],
)
display(dfa)
display(dfa.dropna())
```

nome	cidade	idade	sexo	altura	peso
paola	SM	31.0	F	172	NaN
	SP	32.0	NaN	165	55.0
laís	SP	NaN	F	160	65.0
felipe	RS	38.0	M	172	70.0

nome	cidade	idade	sexo	altura	peso
felipe	RS	38.0	M	172	70.0

Operações de banco de dados relacional

- União
- Junção
- Agrupamento

União

- https://pandas.pydata.org/pandas-docs/stable/merging.html
- pandas.DataFrame.merge
- Permite unir os registros de duas tabelas sem que seja necessário fazer um join
- Insere NaN nas colunas que não são compartilhadas entre as duas tabelas

In [18]:
```
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1'],
                        'A': ['A0', 'A1', 'A2', 'A3'],
                        'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                      'key2': ['K0', 'K0', 'K0', 'K0'],
                         'C': ['C0', 'C1', 'C2', 'C3'],
                         'D': ['D0', 'D1', 'D2', 'D3']}, index=[0,2,3,4])
```

In [19]:
```
display(left)
display(right)
result = pd.merge(left, right, how='inner')
display(result)
```

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

	key1	key2	C	D
0	K0	K0	C0	D0
2	K1	K0	C1	D1
3	K1	K0	C2	D2
4	K2	K0	C3	D3

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Junção/Intersecção

- O método pandas.DataFrame.merge também permite fazer junção, desde que se passe uma metodologia de junção e a chave

In [20]:
```
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1'],
                        'A': ['A0', 'A1', 'A2', 'A3'],
                        'B': ['B0', 'B1', 'B2', 'B3']})

right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                      'key2': ['K0', 'K0', 'K0', 'K0'],
                         'C': ['C0', 'C1', 'C2', 'C3'],
                         'D': ['D0', 'D1', 'D2', 'D3']})
```

In [21]:
```
display(left)
display(right)
result = pd.merge(left, right, how='inner', on=['key1', 'key2'])
display(result)
```

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

In [22]:
```
result = pd.merge(left, right,
         how='inner', on=['key1', 'key2'])
display(result)
```

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

Agrupamento

- Documentação oficial: https://pandas.pydata.org/pandas-docs/stable/groupby.html
- pandas.DataFrame.groupby
- Opera sob o mesmo princípio de um group by de banco de dados relacional: agrupa dados utilizando um atributo como pivô

In [23]:
```
dfa = pd.DataFrame(
    data=[
        [35, 'M', 170, 70],
        [32, 'F', 165, 55],
        [28, 'F', 160, 65],
        [30, 'M', 168, 68],
    ],
    index=pd.MultiIndex.from_tuples(
        [('cleiton', 'SP'), ('paola', 'SP'), ('laís', 'SP'), ('rafael', 'RJ')],
        names=['nome', 'cidade']
    ),
    columns=['idade', 'sexo', 'altura', 'peso'],
)

# dfb = pd.DataFrame(
#     data=[
#         [31, 'F', 172, 51],
#         [32, 'F', 165, 55],
#     ],
#     index=pd.MultiIndex.from_tuples([
#         ('paola', 'SM'),('paola', 'SP')],
#         names=['nome', 'cidade']),
#     columns=['idade', 'sexo', 'altura', 'peso'],
# )
```

In [24]:
```
display(dfa)
```

		idade	sexo	altura	peso
nome	cidade				
cleiton	SP	35	M	170	70
paola	SP	32	F	165	55
laís	SP	28	F	160	65
rafael	RJ	30	M	168	68

In [25]:
```
gb = dfa.groupby(by='sexo').mean()
display(gb)# gera um objeto do tipo pandas.core.groupby.groupby.DataFrameGroupBy
# gb.agg([np.mean, np.std])  # mostra os valores médios e de desvio padrão de cada um dos atributos restantes
```

	idade	altura	peso
sexo			
F	30.0	162.5	60.0
M	32.5	169.0	69.0

Método agg

Permite agregar dados usando uma ou mais operações sobre algum eixo

In [26]:
```
dfa.agg({'idade': [np.mean,'std']})
```

Out[26]:

	idade
mean	31.250000
std	2.986079

Leitura recomendada

- Documentação pandas: https://pandas.pydata.org/

Exercícios – Parte 4

Exercício 1

Leia os arquivos fifa19_info.csv e o arquivo fifa19_stats.csv e realize os seguintes cálculos sobre o conjunto de dados:

1. Calcule a média de agilidade dos jogadores do Barcelona e do Real Madrid.
2. Faça um gráfico de barras com os top 10 clubes de acordo com a soma de valor seus jogadores.
3. Mostre os top 5 países que mais possuem jogadores no conjunto de dados.
4. Mostre a foto (virtual e real) dos top 5 jogadores de acordo com a habilidade absoluta (Overall).
5. Mostre qual é o jogador mais velho e mais novo.
6. Faça um gráfico de dispersão que contém o top 10 jogadores de acordo com a habilidade absoluta (Overall) no eixo X e sua habilidade em bater pênaltis (Penalties) no eixo y.
7. Mostre qual o clube que contém, em média, os melhores batedores de pênalti.
8. Mostre qual o clube que contém, em média, os jogadores mais caros.
9. Faça um gráfico de radar dos Jogadores L. Messi, Sergio Ramos e De Gea. Utilize os atributos Balance, Stamina, ShotPower, Marking, Dribling, SprintSpeed, Strength, SlidingTackle e GKReflexes. Verifique o intervalo dos dados, talvez seja necessário normalizar.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display as print_dataframe
##########################
# Inicio do seu codigo #
##########################




##########################
#   Fim do seu codigo  #
##########################
from urllib.request import Request, urlopen
import io
from PIL import Image
import matplotlib.pyplot as plt
```

```
req = Request('https://cdn.sofifa.org/players/4/19/158023.png',
            headers={'User-Agent': 'Mozilla/5.0'})
webpage = urlopen(req).read()
webpage = Image.open(io.BytesIO(webpage))
plt.imshow(webpage)
plt.show()
```