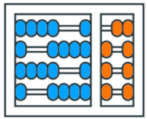


Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 - Aula 28 (Complementar)

A biblioteca `pandas` de Python

Algoritmos e Programação de Computadores

Turmas

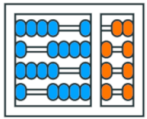
OVXZ

Prof. Lise R. R. Navarrete

lrommel@ic.unicamp.br

Quinta-feira, 07 de julho de 2022

19:00h - 21:00h (CB06)



Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 – Algoritmos e Programação de Computadores

Turmas

OVXZ

https://ic.unicamp.br/~mc102/

Site da Coordenação de MC102

Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06)

Quinta-feira, 19:00h - 21:00h (CB06)

Conteúdo

- Pandas
  - O que é pandas?
- Pandas: Series
  - Series desde um ndarray
  - Series desde um dict
  - Series desde um valor escalar
  - Series é ndarray-like
  - Series é dict-like
  - Series: Operações vetorizadas e alinhamento
  - Series: name
- Pandas: DataFrame
  - Desde um dict de Series
  - Desde um dict de ndarrays/lists
  - Desde um ndarray estruturado
  - Desde uma lista de dicts
  - Desde uma dict de tuplas
  - DataFrame usando from_dict()
  - DataFrame usando from_record()
  - Seleção de coluna, adição, exclusão
  - Trabalhando com DataFrame
  - Leitura de um arquivo csv

Pandas

Pandas

O que é pandas?

https://pandas.pydata.org/docs/getting_started/overview.html

O que é pandas?

"pandas" é uma biblioteca Python que fornece estruturas de dados rápidas, flexíveis e expressivas, projetadas para tornar o trabalho com dados "relacionais" ou "rotulados" fácil e intuitivo.

"pandas" é uma ferramenta de alto nível e de código aberto para fazer análises/manipulação de dados.

https://pandas.pydata.org/docs/getting_started/overview.html

"pandas" é adequado para muitos tipos diferentes de dados:

- Dados tabulares com colunas de tipo heterogêneo, como em uma tabela SQL ou planilha do Excel
- Dados de séries temporais ordenados e não ordenados (não necessariamente de frequência fixa).
- Dados de matriz arbitrária (tipo homogêneo ou heterogêneo) com rótulos de linha e coluna
- Qualquer outra forma de conjuntos de dados observacionais/estatísticos. Os dados não precisam ser rotulados para serem colocados em uma estrutura de dados pandas

https://pandas.pydata.org/docs/getting_started/overview.html

As duas estruturas de dados primárias de "pandas" são:

- **Series** (1-dimensional), e
- **DataFrame** (2-dimensional),

Ambas, lidam com a grande maioria dos casos de uso típicos em finanças, estatísticas, ciências sociais e muitas áreas de engenharia.

https://pandas.pydata.org/docs/getting_started/overview.html

O "pandas" é construído sobre NumPy e foi projetado para se integrar bem em um ambiente de computação científica com muitas outras bibliotecas de terceiros.

Para usuários de `R`, o `DataFrame` oferece tudo o que o `data.frame` do `R` oferece e muito mais.

https://pandas.pydata.org/docs/getting_started/overview.html

Entre as facilidades que "pandas" permite estão:

- Fácil manuseio de dados ausentes (representados como NaN).
- Mutabilidade de tamanho: as colunas podem ser inseridas e excluídas.
- Alinhamento de dados automático e explícito: os objetos podem ser alinhados explicitamente a um conjunto de rótulos, ou o usuário pode simplesmente ignorar os rótulos e deixar que Series, DataFrame, etc. façam o alinhamento automaticamente.
- Agrupamento versatil e flexível para realizar operações split-apply-combine em conjuntos de dados.

https://pandas.pydata.org/docs/getting_started/overview.html

Entre as facilidades que "pandas" permite estão:

- Facilita a conversão de dados irregulares e/ou indexados de forma diferente, desde outras estruturas de dados Python e/ou NumPy para objetos DataFrame.
- Fatiamento baseado em rótulos, indexação e tratamente de subconjuntos para trabalhar com grandes conjuntos de dados.
- Mescla e união intuitiva de conjuntos de dados.
- Remodelação flexível e pivotagem de conjuntos de dados.
- Rotulagem hierárquica de eixos (possível ter vários rótulos por tick).

https://pandas.pydata.org/docs/getting_started/overview.html

Entre as facilidades que "pandas" permite estão:

- Ferramentas de E/S robustas para carregar dados de arquivos simples (CSV e delimitados), arquivos Excel, bancos de dados e salvar/carregar dados do formato HDF5 ultrarrápido
- Funcionalidade específica de série temporal: geração de intervalo de datas e conversão de frequência, estatísticas de janela móvel, mudança de data e atraso.

https://pandas.pydata.org/docs/getting_started/overview.html

Data structures

Dimensions	Name	Description
1	Series	1D labeled homogeneously-typed array
2	DataFrame	General 2D labeled, size-mutable tabular structure with potentially heterogeneously-typed column

Pandas: Series

https://pandas.pydata.org/docs/user_guide/dsintro.html

Series

`Series` é um `array` rotulado unidimensional capaz de conter qualquer tipo de dados (inteiros, strings, números de ponto flutuante, objetos Python, etc.). Os rótulos de eixo são chamados coletivamente de índice.

A forma básica de criar um objeto *Series* é:

```
>>> s = pd.Series(data, index=index)
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

Em:

```
>>> s = pd.Series(data, index=index)
```

`data` pode ser:

- um dicionário de Python (`dict`)
- um `ndarray`
- um valor escalar (Ex: 7)

O índice passado `index` é uma lista de rótulos para o eixo.

Pandas: Series

Series desde um ndarray

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[1] import numpy as np
    import pandas as pd
```

```
[2] s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
    s
```

```
a   -0.256734
b    1.365663
c   -0.374949
d    0.184826
e   -0.471736
dtype: float64
```

```
[3] s.index
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

```
[4] s.values
```

```
array([-0.25673433,  1.36566314, -0.37494889,  0.18482568, -0.47173554])
```

```
[5] s.loc["d"] ,  s.iloc[3],  s.iat[3]
```

```
(0.1848256768347403, 0.1848256768347403, 0.1848256768347403)
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[6] d = pd.Series(np.random.randn(5), index=["a", "b", "c", "a", "e"])
    d
```

```
a   -0.903074
b   -0.157047
c   -0.077679
a    0.151888
e   -0.290763
dtype: float64
```

```
[7] d.loc["a"]
```

```
a   -0.903074
a    0.151888
dtype: float64
```

```
[8] type(d.loc["a"])
```

```
pandas.core.series.Series
```

```
[9] d.iloc[3],  d.iat[3]
```

```
(0.15188843357142645, 0.15188843357142645)
```

Pandas: Series

Series **desde um** dict

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[10] d = {"b": 1, "a": 0, "c": 2}
     d
```

```
     {'a': 0, 'b': 1, 'c': 2}
```

```
[11] pd.Series(d)
```

```
     b    1
     a    0
     c    2
     dtype: int64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[12] d = {"a": 0.0, "b": 1.0, "c": 2.0}
     pd.Series(d)
```

```
a    0.0
b    1.0
c    2.0
dtype: float64
```

```
[13] pd.Series(d, index=["b", "c", "d", "a"])
```

```
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
```

Pandas: Series

Series desde um valor escalar

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[14] pd.Series(5.0, index=["a", "b", "c", "d", "e"])
```

```
a    5.0
b    5.0
c    5.0
d    5.0
e    5.0
dtype: float64
```

```
[15] pd.Series(5.0, index=range(10))
```

```
0    5.0
1    5.0
2    5.0
3    5.0
4    5.0
5    5.0
6    5.0
7    5.0
8    5.0
9    5.0
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[16] [chr(x) for x in range(65,80)]
```

```
['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O']
```

```
[17] pd.Series(5.0, index=[chr(x) for x in range(65,80)] )
```

```
A    5.0
B    5.0
C    5.0
D    5.0
E    5.0
F    5.0
G    5.0
H    5.0
I    5.0
J    5.0
K    5.0
L    5.0
M    5.0
N    5.0
O    5.0
dtype: float64
```

Pandas: Series

Series é ndarray-like

https://pandas.pydata.org/docs/user_guide/dsintro.html

Series é ndarray-like:

`Series` atua de maneira muito semelhante a um `ndarray` e é um argumento válido para a maioria das funções **NumPy**. No entanto, operações como fatiar também fatiarão o índice.

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[18] s

     a   -0.719012
     b    0.030963
     c    0.694125
     d    0.431540
     e   -1.427871
     dtype: float64

[19] s[0]

     -0.7190118893698891

[20] s[:3]

     a   -0.719012
     b    0.030963
     c    0.694125
     dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[21] s
```

```
a   -0.719012
b    0.030963
c    0.694125
d    0.431540
e   -1.427871
dtype: float64
```

```
[22] s[s > s.median()]
```

```
c    0.694125
d    0.431540
dtype: float64
```

```
[23] s[[4, 3, 1]]
```

```
e   -1.427871
d    0.431540
b    0.030963
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[24] s
```

```
a   -0.719012
b    0.030963
c    0.694125
d    0.431540
e   -1.427871
dtype: float64
```

```
[25] np.exp(s)
```

```
a    0.487233
b    1.031447
c    2.001956
d    1.539627
e    0.239819
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[26] t = s[s > s.median()]
     t
```

```
b    1.533914
d    1.312888
dtype: float64
```

```
[27] t.loc["d"] = 1000
     t
```

```
b       1.533914
d    1000.000000
dtype: float64
```

```
[28] s
```

```
a    0.667637
b    1.533914
c    1.143517
d    1.312888
e    0.489935
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[29] r = s[:3]
     r
```

```
a    0.667637
b    1.533914
c    1.143517
dtype: float64
```

```
[30] r.loc["b"] = 1000
     r
```

```
a       0.667637
b    1000.000000
c       1.143517
dtype: float64
```

```
[31] s
```

```
a       0.667637
b    1000.000000
c       1.143517
d       1.312888
e       0.489935
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[32] s.array
```

```
<PandasArray>
[  -1.336675918551167,               1000.0, -0.38105897221873514,
  -0.7225368391408177,    2.033643072233945]
Length: 5, dtype: float64
```

```
[33] s.array[1] = 0
     s
```

```
a   -1.336676
b    0.000000
c   -0.381059
d   -0.722537
e    2.033643
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[34] u = s.to_numpy()
     u

     array([-1.33667592,  0.        , -0.38105897, -0.72253684,  2.03364307])

[35] u[1] = 3.141592

[36] s

     a   -1.336676
     b    3.141592
     c   -0.381059
     d   -0.722537
     e    2.033643
     dtype: float64
```

Pandas: Series

Series é dict-like

https://pandas.pydata.org/docs/user_guide/dsintro.html

Series é dict-like:

`Series` é como um `dict` de tamanho fixo em que você pode obter e definir valores pelo rótulo do índice.

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[37] s
```

```
a   -1.206114
b    3.141592
c   -0.255072
d    1.198554
e   -0.773335
dtype: float64
```

```
[38] s["a"]
```

```
-1.2061144801146053
```

```
[39] s["b"] = 555
     s
```

```
a     -1.206114
b    555.000000
c     -0.255072
d      1.198554
e     -0.773335
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[40] s
```

```
a     -1.206114
b    555.000000
c     -0.255072
d      1.198554
e     -0.773335
dtype: float64
```

```
[41] "e" in s
```

```
True
```

```
[42] "f" in s
```

```
False
```

```
s["f"]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[40] s
```

```
a     -1.206114
b    555.000000
c     -0.255072
d      1.198554
e     -0.773335
dtype: float64
```

```
[41] "e" in s
```

```
True
```

```
[42] "f" in s
```

```
False
```

```
if "f" in s: s["f"]
```

Pandas: `Series`

`Series`: Operações vetorizadas e alinhamento

https://pandas.pydata.org/docs/user_guide/dsintro.html

Operações vetorizadas e alinhamento de etiquetas

Ao trabalhar com matrizes NumPy, geralmente não é necessário fazer um loop de valor por valor. O mesmo acontece ao trabalhar com Um objeto Series em pandas. Um objeto Series também pode ser passado para a maioria dos métodos NumPy esperando um ndarray.

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[44] s
```

```
a      0.649910
b    555.000000
c      0.193911
d      0.541398
e      0.118713
dtype: float64
```

```
[45] s + s
```

```
a       1.299820
b    1110.000000
c       0.387821
d       1.082796
e       0.237427
dtype: float64
```

```
[46] s * 10
```

```
a       6.499099
b    5550.000000
c       1.939107
d       5.413978
e       1.187133
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[47] s
```

```
a      0.649910
b    555.000000
c      0.193911
d      0.541398
e      0.118713
dtype: float64
```

```
[48] s[1:] + s[:-1]
```

```
a           NaN
b    1110.000000
c       0.387821
d       1.082796
e           NaN
dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[49] a = pd.Series(5.0, index=[chr(x) for x in range(65,70)] )
```

```
A    5.0
B    5.0
C    5.0
D    5.0
E    5.0
dtype: float64
```

```
[50] b = pd.Series(range(4), index=["D","A","C","B"] )
     b
```

```
D    0
A    1
C    2
B    3
dtype: int64
```

```
[51] a + b
```

```
A    6.0
B    8.0
C    7.0
D    5.0
E    NaN
dtype: float64
```

Pandas: Series

Series: name

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[52] s = pd.Series(np.random.randn(3), name="nome")
```

```
0   -0.505768
1   -1.801945
2    0.453046
Name: nome, dtype: float64
```

```
[53] s.name
```

```
'nome'
```

```
[54] s.name = "outro nome"
     s
```

```
0   -0.505768
1   -1.801945
2    0.453046
Name: outro nome, dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[55] s2 = s.rename("apelido")
     s2
```

```
0   -0.505768
1   -1.801945
2    0.453046
Name: apelido, dtype: float64
```

```
[56] s2[1] = 1000
```

```
[57] s2
```

```
0      -0.505768
1    1000.000000
2       0.453046
Name: apelido, dtype: float64
```

```
[58] s
```

```
0   -0.505768
1   -1.801945
2    0.453046
Name: outro nome, dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[59] s3 = s.rename(index={0:"a",1:"b",2:"c"})
     s3
```

```
     a   -0.505768
     b   -1.801945
     c    0.453046
     Name: outro nome, dtype: float64
```

```
[60] s3["b"] = 1000
```

```
[61] s3
```

```
     a      -0.505768
     b    1000.000000
     c       0.453046
     Name: outro nome, dtype: float64
```

```
[62] s
```

```
     0   -0.505768
     1   -1.801945
     2    0.453046
     Name: outro nome, dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[63] s3
```

```
a      -0.505768
b    1000.000000
c       0.453046
Name: outro nome, dtype: float64
```

```
[64] s4 = s3.rename(str.upper)
     s4
```

```
A      -0.505768
B    1000.000000
C       0.453046
Name: outro nome, dtype: float64
```

```
[65] s4.to_numpy().flags.owndata
```

```
True
```

```
[66] s4["B"] = 55
     s3
```

```
a      -0.505768
b    1000.000000
c       0.453046
Name: outro nome, dtype: float64
```

Pandas: DataFrame

https://pandas.pydata.org/docs/user_guide/dsintro.html

O que é DataFrame?

DataFrame é uma estrutura de dados rotulada bidimensional com colunas que podem ser de tipos diferentes. Pode pensar nisso como uma planilha ou tabela SQL, ou um dict de objetos Series.

DataFrame geralmente é o objeto de pandas mais usado.

Assim como o Series, o DataFrame aceita muitos tipos diferentes de entrada.

https://pandas.pydata.org/docs/user_guide/dsintro.html

DataFrame pode ser construído desde:

- Dicionário de ndarrays de 1D, listas, dicts ou Series
- 2-D numpy.ndarray
- ndarray estruturado ou partes deste
- um objeto Series
- outro DataFrame

https://pandas.pydata.org/docs/user_guide/dsintro.html

Ao construir um DataFrame:

Junto com os dados, você pode opcionalmente passar argumentos de índice (rótulos de linha) e colunas (rótulos de coluna).

Se você passar um índice e/ou colunas, está garantindo o índice e/ou colunas do DataFrame resultante.

Assim, um dict de Series mais um índice específico descartará todos os dados que não correspondem ao índice passado.

Se os rótulos dos eixos não forem passados, eles serão construídos a partir dos dados de entrada com base em regras de bom senso.

Pandas: DataFrame

Desde um dict de Series

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[1] import numpy as np
    import pandas as pd
```

```
[2] d = {
        "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
        "two": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
    }
```

```
[3] df1 = pd.DataFrame(d)
    df1
```

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0
d	NaN	4.0

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[4] d = {
        "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
        "two": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
    }
```

```
[5] df2 = pd.DataFrame(d, index=["d", "b", "a"])
    df2
```

	one	two
d	NaN	4.0
b	2.0	2.0
a	1.0	1.0

```
[6] df3 = pd.DataFrame(d, index=["d", "b", "a"], columns=["two", "three"])
    df3
```

	two	three
d	4.0	NaN
b	2.0	NaN
a	1.0	NaN

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[7] d = {
        "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
        "two": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
    }
```

```
[8] df = pd.DataFrame(d)
    df
```

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0
d	NaN	4.0

```
[9] df.index
```

```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
[10] df.columns
```

```
Index(['one', 'two'], dtype='object')
```

Pandas: DataFrame

Desde um dict de ndarrays/lists

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[11] d = {"one": [1.0, 2.0, 3.0, 4.0], "two": [4.0, 3.0, 2.0, 1.0]}
```

```
[12] pd.DataFrame(d)
```

	one	two
0	1.0	4.0
1	2.0	3.0
2	3.0	2.0
3	4.0	1.0

```
[13] pd.DataFrame(d, index=["a", "b", "c", "d"])
```

	one	two
a	1.0	4.0
b	2.0	3.0
c	3.0	2.0
d	4.0	1.0

Pandas: DataFrame

Desde um ndarray estruturado

https://numpy.org/doc/stable/reference/arrays.dtypes.html

dtype:

'?'	boolean
'b'	(signed) byte
'B'	unsigned byte
'i'	(signed) integer
'u'	unsigned integer
'f'	floating-point
'c'	complex-floating point
'm'	timedelta
'M'	datetime
'O'	(Python) objects
'S', 'a'	zero-terminated bytes (not recommended)
'U'	Unicode string
'V'	raw data (void)

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[14] x = np.array([('Rex', 9, 81.0), ('Fido', 3, 27.0)],
                  dtype=[('name', 'U10'), ('age', 'i4'), ('weight', 'f4')])
     x

     array([('Rex', 9, 81.), ('Fido', 3, 27.)],
           dtype=[('name', '<U10'), ('age', '<i4'), ('weight', '<f4')])

[15] x[1]

     ('Fido', 3, 27.)

[16] x['age']

     array([9, 3], dtype=int32)

[17] x['age'] = 5
     x

     array([('Rex', 5, 81.), ('Fido', 5, 27.)],
           dtype=[('name', '<U10'), ('age', '<i4'), ('weight', '<f4')])
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[18] np.zeros((2,2))
```

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

```
[19] data = np.zeros((3,), dtype=[("A", "i4"), ("B", "f4"), ("C", "a10")])
     data
```

```
array([(0, 0., b''), (0, 0., b''), (0, 0., b'')],
      dtype=[('A', '<i4'), ('B', '<f4'), ('C', 'S10')])
```

```
[20] data[:] = [(1, 2.0, "Hello"), (2, 3.0, "World"), (0,0,0)]
     data
```

```
array([(1, 2., b'Hello'), (2, 3., b'World'), (0, 0., b'0')],
      dtype=[('A', '<i4'), ('B', '<f4'), ('C', 'S10')])
```

```
[21] pd.DataFrame(data)
```

	A	B	C
0	1	2.0	b'Hello'
1	2	3.0	b'World'
2	0	0.0	b'0'

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[22] data
```

```
array([(1, 2., b'Hello'), (2, 3., b'World'), (0, 0., b'0')],
      dtype=[('A', '<i4'), ('B', '<f4'), ('C', 'S10')])
```

```
[23] pd.DataFrame(data, index=["first", "second", "third"])
```

	A	B	C
first	1	2.0	b'Hello'
second	2	3.0	b'World'
third	0	0.0	b'0'

```
[24] pd.DataFrame(data, columns=["C", "A", "B"])
```

	C	A	B
0	b'Hello'	1	2.0
1	b'World'	2	3.0
2	b'0'	0	0.0

Pandas: `DataFrame`

Desde uma lista de `dicts`

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[25] data2 = [{"a": 1, "b": 2}, {"a": 5, "b": 10, "c": 20}]
```

```
[26] pd.DataFrame(data2)
```

	a	b	c
0	1	2	NaN
1	5	10	20.0

```
[27] pd.DataFrame(data2, index=["first", "second"])
```

	a	b	c
first	1	2	NaN
second	5	10	20.0

```
[28] pd.DataFrame(data2, columns=["a", "b"])
```

	a	b
0	1	2
1	5	10

Pandas: DataFrame

Desde uma dict de tuplas

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[29] pd.DataFrame(
        {
            ("a", "b"): {("A", "B"): 1, ("A", "C"): 2},
            ("a", "a"): {("A", "C"): 3, ("A", "D"): 4},
            ("a", "c"): {("A", "B"): 5, ("A", "C"): 6},
            ("b", "a"): {("A", "C"): 7, ("A", "B"): 8},
            ("b", "b"): {("A", "D"): 9, ("A", "B"): 10},
        }
    )
```

		a			b	
		b	a	c	a	b
A	B	1.0	NaN	5.0	8.0	10.0
	C	2.0	3.0	6.0	7.0	NaN
	D	NaN	4.0	NaN	NaN	9.0

Pandas: DataFrame

DataFrame usando from_dict()

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[30] k = dict([("A", [1, 2, 3]), ("B", [4, 5, 6])])
```

```
{'A': [1, 2, 3], 'B': [4, 5, 6]}
```

```
[31] pd.DataFrame(k)
```

	A	B
0	1	4
1	2	5
2	3	6

```
[32] pd.DataFrame.from_dict(k, orient="index")
```

	0	1	2
A	1	2	3
B	4	5	6

```
[33] pd.DataFrame.from_dict(k, orient="index", columns=["one","two","three"])
```

	one	two	three
A	1	2	3
B	4	5	6

Pandas: DataFrame

DataFrame usando from_record()

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[34] data
```

```
array([(1, 2., b'Hello'), (2, 3., b'World'), (0, 0., b'0')],
      dtype=[('A', '<i4'), ('B', '<f4'), ('C', 'S10')])
```

```
[35] pd.DataFrame.from_records(data, index="B")
```

B	A	C
2.0	1	b'Hello'
3.0	2	b'World'
0.0	0	b'0'

Pandas: `DataFrame`

Seleção de coluna, adição, exclusão

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[36] df
```

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0
d	NaN	4.0

```
[37] df["one"]
```

```
a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype: float64
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[38] df["three"] = df["one"] * df["two"]
     df
```

	one	two	three
a	1.0	1.0	1.0
b	2.0	2.0	4.0
c	3.0	3.0	9.0
d	NaN	4.0	NaN

```
[39] df["flag"] = df["one"] > 2
     df
```

	one	two	three	flag
a	1.0	1.0	1.0	False
b	2.0	2.0	4.0	False
c	3.0	3.0	9.0	True
d	NaN	4.0	NaN	False

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[40] df
```

	one	two	three	flag
a	1.0	1.0	1.0	False
b	2.0	2.0	4.0	False
c	3.0	3.0	9.0	True
d	NaN	4.0	NaN	False

```
[41] del df["two"]
     df
```

	one	three	flag
a	1.0	1.0	False
b	2.0	4.0	False
c	3.0	9.0	True
d	NaN	NaN	False

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[42] three = df.pop("three")
     three
```

```
a    1.0
b    4.0
c    9.0
d    NaN
Name: three, dtype: float64
```

```
[43] df
```

	one	flag
a	1.0	False
b	2.0	False
c	3.0	True
d	NaN	False

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[44] df
```

	one	flag
a	1.0	False
b	2.0	False
c	3.0	True
d	NaN	False

```
[45] df["foo"] = "bar"
     df
```

	one	flag	foo
a	1.0	False	bar
b	2.0	False	bar
c	3.0	True	bar
d	NaN	False	bar

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[46] df
```

	one	flag	foo
a	1.0	False	bar
b	2.0	False	bar
c	3.0	True	bar
d	NaN	False	bar

```
[47] df["one_trunc"] = df["one"][:2]
     df
```

	one	flag	foo	one_trunc
a	1.0	False	bar	1.0
b	2.0	False	bar	2.0
c	3.0	True	bar	NaN
d	NaN	False	bar	NaN

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[48] df
```

	one	flag	foo	one_trunc
a	1.0	False	bar	1.0
b	2.0	False	bar	2.0
c	3.0	True	bar	NaN
d	NaN	False	bar	NaN

```
[49] df.insert(1, "bar", df["one"])
     df
```

	one	bar	flag	foo	one_trunc
a	1.0	1.0	False	bar	1.0
b	2.0	2.0	False	bar	2.0
c	3.0	3.0	True	bar	NaN
d	NaN	NaN	False	bar	NaN

Pandas: DataFrame

Trabalhando com DataFrame

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[50] dfa = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
     dfa
```

	A	B
0	1	4
1	2	5
2	3	6

```
[51] dfa.assign(C=lambda x: x["A"] + x["B"], D=lambda x: x["A"] + x["C"])
```

	A	B	C	D
0	1	4	5	6
1	2	5	7	9
2	3	6	9	12

https://pandas.pydata.org/docs/user_guide/dsintro.html

	one	bar	flag	foo	one_trunc
a	1.0	1.0	False	bar	1.0
b	2.0	2.0	False	bar	2.0
c	3.0	3.0	True	bar	NaN
d	NaN	NaN	False	bar	NaN

```
[53] df.loc["b"]
```

```
one            2.0
bar            2.0
flag         False
foo            bar
one_trunc      2.0
Name: b, dtype: object
```

```
[54] df.iloc[2]
```

```
one            3.0
bar            3.0
flag          True
foo            bar
one_trunc      NaN
Name: c, dtype: object
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[55] X = pd.DataFrame(np.ones((10, 4)), columns=["A", "B", "C", "D"])
     X
```

	A	B	C	D
0	1.0	1.0	1.0	1.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0
5	1.0	1.0	1.0	1.0
6	1.0	1.0	1.0	1.0
7	1.0	1.0	1.0	1.0
8	1.0	1.0	1.0	1.0
9	1.0	1.0	1.0	1.0

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[56] X = pd.DataFrame(np.ones((10, 4)), columns=["A", "B", "C", "D"])
     Y = pd.DataFrame(np.ones((7, 3))*1000, columns=["A", "B", "C"])
```

```
[57] X + Y
```

	A	B	C	D
0	1001.0	1001.0	1001.0	NaN
1	1001.0	1001.0	1001.0	NaN
2	1001.0	1001.0	1001.0	NaN
3	1001.0	1001.0	1001.0	NaN
4	1001.0	1001.0	1001.0	NaN
5	1001.0	1001.0	1001.0	NaN
6	1001.0	1001.0	1001.0	NaN
7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[58] X = pd.DataFrame(np.ones((5, 4)), columns=["A", "B", "C", "D"])
```

```
[59] X.iloc[0] = 100
```

	A	B	C	D
0	100.0	100.0	100.0	100.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

```
[60] X - X.iloc[0]
```

	A	B	C	D
0	0.0	0.0	0.0	0.0
1	-99.0	-99.0	-99.0	-99.0
2	-99.0	-99.0	-99.0	-99.0
3	-99.0	-99.0	-99.0	-99.0

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[61] X
```

	A	B	C	D
0	100.0	100.0	100.0	100.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

```
[62] X * 5 + 7
```

	A	B	C	D
0	507.0	507.0	507.0	507.0
1	12.0	12.0	12.0	12.0
2	12.0	12.0	12.0	12.0
3	12.0	12.0	12.0	12.0
4	12.0	12.0	12.0	12.0

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[63] X
```

	A	B	C	D
0	100.0	100.0	100.0	100.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

```
[64] 1/X
```

	A	B	C	D
0	0.01	0.01	0.01	0.01
1	1.00	1.00	1.00	1.00
2	1.00	1.00	1.00	1.00
3	1.00	1.00	1.00	1.00
4	1.00	1.00	1.00	1.00

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[65] X
```

	A	B	C	D
0	100.0	100.0	100.0	100.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

```
[66] X**4
```

	A	B	C	D
0	100000000.0	100000000.0	100000000.0	100000000.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[67] df1 = pd.DataFrame({"a": [1, 0, 1], "b": [0, 1, 1]}, dtype=bool)
     df1
```

	a	b
0	True	False
1	False	True
2	True	True

```
[68] df2 = pd.DataFrame({"a": [0, 1, 1], "b": [1, 1, 0]}, dtype=bool)
     df2
```

	a	b
0	False	True
1	True	True
2	True	False

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[69] df1 = pd.DataFrame({"a": [1, 0, 1], "b": [0, 1, 1]}, dtype=bool)
     df2 = pd.DataFrame({"a": [0, 1, 1], "b": [1, 1, 0]}, dtype=bool)
```

```
[70] df1 & df2
```

	a	b
0	False	False
1	False	True
2	True	False

```
[71] df1 | df2
```

	a	b
0	True	True
1	True	True
2	True	True

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[72] df
```

	one	bar	flag	foo	one_trunc
a	1.0	1.0	False	bar	1.0
b	2.0	2.0	False	bar	2.0
c	3.0	3.0	True	bar	NaN
d	NaN	NaN	False	bar	NaN

```
[73] df[:5].T
```

	a	b	c	d
one	1.0	2.0	3.0	NaN
bar	1.0	2.0	3.0	NaN
flag	False	False	True	False
foo	bar	bar	bar	bar
one_trunc	1.0	2.0	NaN	NaN

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[74] X
```

	A	B	C	D
0	100.0	100.0	100.0	100.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

```
[75] np.exp(X)
```

	A	B	C	D
0	2.688117e+43	2.688117e+43	2.688117e+43	2.688117e+43
1	2.718282e+00	2.718282e+00	2.718282e+00	2.718282e+00
2	2.718282e+00	2.718282e+00	2.718282e+00	2.718282e+00
3	2.718282e+00	2.718282e+00	2.718282e+00	2.718282e+00
4	2.718282e+00	2.718282e+00	2.718282e+00	2.718282e+00

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[76] X
```

	A	B	C	D
0	100.0	100.0	100.0	100.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

```
[77] np.asarray(X)
```

```
array([[100., 100., 100., 100.],
       [  1.,   1.,   1.,   1.],
       [  1.,   1.,   1.,   1.],
       [  1.,   1.,   1.,   1.],
       [  1.,   1.,   1.,   1.]])
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[78] X
```

	A	B	C	D
0	100.0	100.0	100.0	100.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

```
[79] X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   A       5 non-null      float64
 1   B       5 non-null      float64
 2   C       5 non-null      float64
 3   D       5 non-null      float64
dtypes: float64(4)
memory usage: 288.0 bytes
```

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
[80] X
```

	A	B	C	D
0	100.0	100.0	100.0	100.0
1	1.0	1.0	1.0	1.0
2	1.0	1.0	1.0	1.0
3	1.0	1.0	1.0	1.0
4	1.0	1.0	1.0	1.0

```
[81] print(X.iloc[2:, :3].to_string())
```

```
     A    B    C
2  1.0  1.0  1.0
3  1.0  1.0  1.0
4  1.0  1.0  1.0
```

Pandas: DataFrame

Leitura de um arquivo csv

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
mydf = pd.read_csv("sample_data/mnist_test.csv")
mydf
```

	7	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	...	0.658	0.659	0.660	0.661	0.662	0.663	0.664	0.665	0.666	0.667
0	2	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
3	4	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
9994	2	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
9995	3	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
9996	4	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
9997	5	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0
9998	6	0	0	0	0	0	0	0	0	0	…	0	0	0	0	0	0	0	0	0	0

9999 rows × 785 columns

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
mydf2 = pd.read_csv("sample_data/california_housing_test.csv")
mydf2
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_h
0	-122.05	37.37	27.0	3885.0	661.0	1537.0	606.0	6.6085	
1	-118.30	34.26	43.0	1510.0	310.0	809.0	277.0	3.5990	
2	-117.81	33.78	27.0	3589.0	507.0	1484.0	495.0	5.7934	
3	-118.36	33.82	28.0	67.0	15.0	49.0	11.0	6.1359	
4	-119.67	36.33	19.0	1241.0	244.0	850.0	237.0	2.9375	
...	...	...	...	...	...	...	...	...	
2995	-119.86	34.42	23.0	1450.0	642.0	1258.0	607.0	1.1790	
2996	-118.14	34.06	27.0	5257.0	1082.0	3496.0	1036.0	3.3906	
2997	-119.70	36.30	10.0	956.0	201.0	693.0	220.0	2.2895	
2998	-117.12	34.10	40.0	96.0	14.0	46.0	14.0	3.2708	
2999	-119.63	34.42	42.0	1765.0	263.0	753.0	260.0	8.5608	

3000 rows × 9 columns

https://pandas.pydata.org/docs/user_guide/dsintro.html

```
mydf2.columns
```

```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value'],
      dtype='object')
```

Perguntas ....

Referências

- Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. https://ic.unicamp.br/~mc102/

  Aula Introdutória [slides] [vídeo]
  Primeira Aula de Laboratório [slides] [vídeo]
  Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [slides] [vídeo]
  Comandos Condicionais [slides] [vídeo]
  Comandos de Repetição [slides] [vídeo]
  Listas e Tuplas [slides] [vídeo]
  Strings [slides] [vídeo]
  Dicionários [slides] [vídeo]
  Funções [slides] [vídeo]
  Objetos Multidimensionais [slides] [vídeo]
  Algoritmos de Ordenação [slides] [vídeo]
  Algoritmos de Busca [slides] [vídeo]
  Recursão [slides] [vídeo]
  Algoritmos de Ordenação Recursivos [slides] [vídeo]
  Arquivos [slides] [vídeo]
  Expressões Regulares [slides] [vídeo]
  Execução de Testes no Google Cloud Shell [slides] [vídeo]
  Numpy [slides] [vídeo]
  Pandas [slides] [vídeo]

- Panda - Cursos de Computação em Python (IME -USP) https://panda.ime.usp.br/

  Como Pensar Como um Cientista da Computação https://panda.ime.usp.br/pensepy/static/pensepy/
  Aulas de Introdução à Computação em Python https://panda.ime.usp.br/aulasPython/static/aulasPython/

- Fabio Kon, Introdução à Ciência da Computação com Python http://bit.ly/FabioKon/
- Socratica, Python Programming Tutorials http://bit.ly/SocraticaPython/
- Google - online editor for cloud-native applications (Python programming) https://shell.cloud.google.com/
- w3schools - Python Tutorial https://www.w3schools.com/python/
- Outros, citados nos Slides.