Aula 6A - Manipulação de Dados

Gustavo Oliveira e Andréa Rocha

Departamento de Computação Científica / UFPB

Julho de 2020

1 Manipulação de Dados em *Python*

1.1 A biblioteca *pandas*

- *Pandas* é uma biblioteca para leitura, tratamento e manipulação de dados em *Python*.
- Ela introduz duas novas estruturas de dados:
  - *Series*;
  - *DataFrame*.
- Um *DataFrame* é uma estrutura de dados tabular com linhas e colunas rotuladas.

	Peso	Altura	Idade	Gênero
Ana	55	162	20	`feminino`
João	80	178	19	`masculino`
Maria	62	164	21	`feminino`
Pedro	67	165	22	`masculino`
Túlio	73	171	20	`masculino`

- As colunas do *DataFrame* são vetores unidimensionais do tipo *Series*.
- As linhas são rotuladas por uma estrutura de dados especial chamada *index*.
- Os *index* no *Pandas* são listas personalizadas de rótulos que permitem pesquisa rápida e algumas operações importantes.
- Para podermos apresentar estas estruturas de dados vamos importar as bibliotecas:
  - *numpy*: utilizando a abreviação usual *np*;
  - *pandas*: utilizando sua abreviação usual *pd*:

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

2 *Series* do *pandas*

- As *Series*:
  - São vetores, ou seja, são *arrays* unidimensionais;
  - possuem *index* para cada entrada (e são muito eficientes em operar com base nos *index*);

– podem possuir qualquer tipo de dado (inteiro, *string*, *float*, etc.).

2.1 Criando um objeto do tipo *Series*

O método padrão é utilizar a função *Series* da biblioteca pandas:

```
serie_exemplo = pd.Series(dados_de_interesse, index=indice_de_interesse)
```

No exemplo acima, *dados_de_interesse* pode ser:

- um dicionário (objeto do tipo *dict*);
- uma lista;
- um *array* do *numpy*;
- um escalar, tal como o número 1.

2.2 Criando *Series* a partir de dicionários:

[2]:
```
dicionario_exemplo = {'Ana':20, 'João': 19, 'Maria': 21, 'Pedro': 22, 'Túlio':␣
↪20}
```

[3]:
```
pd.Series(dicionario_exemplo)
```

[3]:
```
Ana      20
João     19
Maria    21
Pedro    22
Túlio    20
dtype: int64
```

- O *index* foi obtido a partir das "chaves" dos dicionários.
- Assim, no caso do exemplo, o *index* foi dado por "Ana", "João", "Maria", "Pedro" e "Túlio".
- A ordem do *index* foi dada pela ordem de entrada no dicionário.

Podemos fornecer um novo *index* ao dicionário já criado

[4]:
```
pd.Series(dicionario_exemplo, index=['Maria', 'Maria', 'ana', 'Paula', 'Túlio',␣
↪'Pedro'])
```

[4]:
```
Maria    21.0
Maria    21.0
ana       NaN
Paula     NaN
Túlio    20.0
Pedro    22.0
dtype: float64
```

- O marcador padrão do *pandas* para dados faltantes é o *NaN* (*not a number*).

2.3 Criando *Series* a partir de listas

[5]:
```
lista_exemplo = [1,2,3,4,5]
```

[6]:
```
pd.Series(lista_exemplo)
```

[6]:
```
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

- Se os *index* não forem fornecidos, o *pandas* atribuirá automaticamente os valores *0, 1, ..., N-1*, onde *N* é o número de elementos da lista.

2.4 Criando *Series* a partir de *arrays* do *numpy*

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

[8]:
```
pd.Series(array_exemplo)
```

[8]:
```
0    1
1    2
2    3
3    4
4    5
dtype: int32
```

2.4.1 Fornecendo um *index* na criação da *Series*

O total de elementos do *index* deve ser igual ao tamanho do *array*.

[9]:
```
pd.Series(array_exemplo, index=['a','b','c','d','e','f'])
```

```
␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-9-f8e840b4247a> in <module>
----> 1 pd.Series(array_exemplo, index=['a','b','c','d','e','f'])

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py in␣
↪__init__(self, data, index, dtype, name, copy, fastpath)
```

```
    290                     if len(index) != len(data):
    291                         raise ValueError(
--> 292                             f"Length of passed values is␣
↪{len(data)}, "
    293                             f"index implies {len(index)}."
    294                         )


ValueError: Length of passed values is 5, index implies 6.
```

O *pandas* aceita que os elementos no *index* não sejam único.

[10]:
```
pd.Series(array_exemplo, index=['a','b','c','d','e'])
```

[10]:
```
a    1
b    2
c    3
d    4
e    5
dtype: int32
```

[11]:
```
pd.Series(array_exemplo, index=['a','a','b','b','c'])
```

[11]:
```
a    1
a    2
b    3
b    4
c    5
dtype: int32
```

Um erro ocorrerá se uma operação que dependa da unicidade dos elementos no *index* for realizada. A exemplo do método *reindex*.

[12]:
```
series_exemplo = pd.Series(array_exemplo, index=['a','a','b','b','c'])
```

[13]:
```
series_exemplo.reindex(['b','a','c','d','e'])
```

```
      ␣
↪---------------------------------------------------------------------------

ValueError                                Traceback (most recent call␣
↪last)

<ipython-input-13-0cada38c890b> in <module>
----> 1 series_exemplo.reindex(['b','a','c','d','e'])
```

```
      C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py in␣
↪reindex(self, index, **kwargs)
     4028     @Appender(generic.NDFrame.reindex.__doc__)
     4029     def reindex(self, index=None, **kwargs):
  -> 4030         return super().reindex(index=index, **kwargs)
     4031
     4032     def drop(


      C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in␣
↪reindex(self, *args, **kwargs)
     4542         # perform the reindex on the axes
     4543         return self._reindex_axes(
  -> 4544             axes, level, limit, tolerance, method, fill_value, copy
     4545         ).__finalize__(self)
     4546


      C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in␣
↪_reindex_axes(self, axes, level, limit, tolerance, method, fill_value, copy)
     4565                 fill_value=fill_value,
     4566                 copy=copy,
  -> 4567                 allow_dups=False,
     4568             )
     4569


      C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py in␣
↪_reindex_with_indexers(self, reindexers, fill_value, copy, allow_dups)
     4611                 fill_value=fill_value,
     4612                 allow_dups=allow_dups,
  -> 4613                 copy=copy,
     4614             )
     4615


      C:
↪\ProgramData\Anaconda3\lib\site-packages\pandas\core\internals\managers.py in␣
↪reindex_indexer(self, new_axis, indexer, axis, fill_value, allow_dups, copy)
     1249         # some axes don't allow reindexing with dups
     1250         if not allow_dups:
  -> 1251             self.axes[axis]._can_reindex(indexer)
     1252
     1253         if axis >= self.ndim:
```

```
       C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py␣
↪in _can_reindex(self, indexer)
      3097         # trying to reindex on an axis with duplicates
      3098         if not self.is_unique and len(indexer):
   -> 3099             raise ValueError("cannot reindex from a duplicate axis")
      3100
      3101     def reindex(self, target, method=None, level=None, limit=None,␣
↪tolerance=None):


       ValueError: cannot reindex from a duplicate axis
```

2.5 Criando *Series* a partir de escalares

[14]:
```
pd.Series(1, index=['a', 'b', 'c', 'd'])
```

[14]:
```
a    1
b    1
c    1
d    1
dtype: int64
```

- Neste caso, um índice **deve** ser fornecido!

2.6 *Series* se comportam como *arrays* do *numpy*

- Uma *Serie* do *pandas* se comporta como um *array* unidimensional do *numpy*.
- Pode ser utilizada como argumento para a maioria das funções do *numpy*.
- A diferença é que o *index* aparece.

Exemplo:

[15]:
```
series_exemplo = pd.Series(array_exemplo, index=['a','b','c','d','e'])
```

[16]:
```
series_exemplo[2]
```

[16]:
```
3
```

[17]:
```
series_exemplo[:2]
```

[17]:
```
a    1
b    2
dtype: int32
```

[18]:
```
np.log(series_exemplo)
```

[18]:
```
a    0.000000
b    0.693147
c    1.098612
d    1.386294
e    1.609438
dtype: float64
```

Mais exemplos:

[19]:
```
serie_1 = pd.Series([1,2,3,4,5])
```

[20]:
```
serie_2 = pd.Series([4,5,6,7,8])
```

[21]:
```
serie_1 + serie_2
```

[21]:
```
0     5
1     7
2     9
3    11
4    13
dtype: int64
```

[22]:
```
serie_1 * 2 - serie_2 * 3
```

[22]:
```
0   -10
1   -11
2   -12
3   -13
4   -14
dtype: int64
```

- Assim como *arrays* do *numpy*, as *Series* do *pandas* também possuem atributos *dtype* (data type).

[23]:
```
series_exemplo.dtype
```

[23]:
```
dtype('int32')
```

- Se o interesse for em utilizar os dados de uma *Serie* do *pandas* como um *array* do *numpy*, basta utilizar o método *to_numpy*.

[24]:
```
series_exemplo.to_numpy()
```

[24]:
```
array([1, 2, 3, 4, 5])
```

2.7 *Series* se comportam como dicionários

- Podemos acessar os elementos da *Series* através das chaves fornecidas no *index*.

[25]:
```
series_exemplo
```

[25]:
```
a    1
b    2
c    3
d    4
e    5
dtype: int32
```

[26]:
```
series_exemplo['a']
```

[26]:
```
1
```

- Podemos adicionar novos elementos associados a chaves novas.

[27]:
```
series_exemplo['f'] = 6
```

[28]:
```
series_exemplo
```

[28]:
```
a    1
b    2
c    3
d    4
e    5
f    6
dtype: int64
```

[29]:
```
'f' in series_exemplo
```

[29]:
```
True
```

[30]:
```
'g' in series_exemplo
```

[30]:
```
False
```

Mais exemplos:

[31]:
```
series_exemplo['g']
```

```
␣
↪---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call␣
↪last)

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py␣
↪in get_value(self, series, key)
```

```
   4410             try:
-> 4411                 return libindex.get_value_at(s, key)
   4412             except IndexError:


    pandas\_libs\index.pyx in pandas._libs.index.get_value_at()


    pandas\_libs\index.pyx in pandas._libs.index.get_value_at()


    pandas\_libs\util.pxd in pandas._libs.util.get_value_at()


    pandas\_libs\util.pxd in pandas._libs.util.validate_indexer()


    TypeError: 'str' object cannot be interpreted as an integer


During handling of the above exception, another exception occurred:


    KeyError                                  Traceback (most recent call␣
↪last)

    <ipython-input-31-12f1880611a9> in <module>
----> 1 series_exemplo['g']


    C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py in␣
↪__getitem__(self, key)
    869         key = com.apply_if_callable(key, self)
    870         try:
--> 871             result = self.index.get_value(self, key)
    872
    873             if not is_scalar(result):


    C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py␣
↪in get_value(self, series, key)
   4417                     raise InvalidIndexError(key)
   4418                 else:
-> 4419                     raise e1
   4420             except Exception:
   4421                 raise e1
```

```
      C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py␣
↪in get_value(self, series, key)
     4403         k = self._convert_scalar_indexer(k, kind="getitem")
     4404         try:
  -> 4405             return self._engine.get_value(s, k, tz=getattr(series.
↪dtype, "tz", None))
     4406         except KeyError as e1:
     4407             if len(self) > 0 and (self.holds_integer() or self.
↪is_boolean()):


      pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_value()


      pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_value()


      pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()


      pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()


      pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.
↪PyObjectHashTable.get_item()


      KeyError: 'g'
```

[32]:
```
series_exemplo.get('g')
```

Podemos utilizar o método *get* para lidar com chaves que possivelmente podem não existir, colocando o *NaN* do *numpy* como alternativa caso não exista nenhum valor atribuído:

[33]:
```
series_exemplo.get('g',np.nan)
```

[33]:
```
nan
```

2.7.1 Atributo *name* de uma *Serie* do *pandas*

- Uma *Serie* do *pandas* possui um atributo opcional *name*.
- O atributo *name* é muito útil nos *DataFrames* do *pandas*.

[34]:
```
serie_com_nome = pd.Series(dicionario_exemplo, name = "Idade")
```

[35]:
```
serie_com_nome
```

[35]:
```
Ana      20
João     19
Maria    21
Pedro    22
Túlio    20
Name: Idade, dtype: int64
```

2.8 Exemplo de função útil: *pd.date_range*

Em vários casos os índices são dados por datas. Uma função muito útil para criar índices a partir de datas é a função *pd.date_range*.

Alguns argumentos desta função: * start: *string* contendo a data que serve como limite à esquerda das datas. Padrão: *None* * end: *string* contendo a data que serve como limite à direita das datas. Padrão: *None* * freq: frequência a ser considerada. Por exemplo, dias (*D*), horas (*H*), semanas (*W*), fins de meses (*M*), inícios de meses (*MS*), fins de anos (*Y*), inícios de anos (*YS*), etc. Pode-se também utilizar múltiplos, como por exemplo *5H*, *2Y*, etc. Padrão: *None*. * periods: número de períodos a serem considerados (o período é determinado pelo argumento *freq*).

Exemplos de uso da função *pd.date_range*:

[36]:
```
pd.date_range(start='1/1/2020', freq='W', periods=10)
```

[36]:
```
DatetimeIndex(['2020-01-05', '2020-01-12', '2020-01-19', '2020-01-26',
               '2020-02-02', '2020-02-09', '2020-02-16', '2020-02-23',
               '2020-03-01', '2020-03-08'],
              dtype='datetime64[ns]', freq='W-SUN')
```

[37]:
```
pd.date_range(start='2010-01-01', freq='2Y', periods=10)
```

[37]:
```
DatetimeIndex(['2010-12-31', '2012-12-31', '2014-12-31', '2016-12-31',
               '2018-12-31', '2020-12-31', '2022-12-31', '2024-12-31',
               '2026-12-31', '2028-12-31'],
              dtype='datetime64[ns]', freq='2A-DEC')
```

[38]:
```
pd.date_range('1/1/2020', freq='5H', periods=10)
```

[38]:
```
DatetimeIndex(['2020-01-01 00:00:00', '2020-01-01 05:00:00',
               '2020-01-01 10:00:00', '2020-01-01 15:00:00',
               '2020-01-01 20:00:00', '2020-01-02 01:00:00',
               '2020-01-02 06:00:00', '2020-01-02 11:00:00',
               '2020-01-02 16:00:00', '2020-01-02 21:00:00'],
              dtype='datetime64[ns]', freq='5H')
```

[39]:
```
pd.date_range(start='2010-01-01', freq='3YS', periods=3)
```

[39]:
```
DatetimeIndex(['2010-01-01', '2013-01-01', '2016-01-01'],
dtype='datetime64[ns]', freq='3AS-JAN')
```

Vamos então criar um *DataFrame* para exemplos.

[40]:
```
indice_exemplo = pd.date_range('2020-01-01', periods=10, freq='D')
```

[41]:
```
serie_1 = pd.Series(np.random.randn(10),index=indice_exemplo)
serie_2 = pd.Series(np.random.randn(10),index=indice_exemplo)
```

3 *DataFrames* do *pandas*

- Os *DataFrames*:
  - São tabelas, ou seja, são bidimensionais;
  - Cada coluna pode ser de um tipo diferente de dado (inteiro, *string*, *float*, etc.);
  - Cada coluna é uma *Serie* do *pandas*.

3.1 Criando um *DataFrame*

O método padrão é através da função *DataFrame* do *pandas*:

```
df_exemplo = pd.DataFrame(dados_de_interesse, index =indice_de_interesse,
                          columns = colunas_de_interesse)
```

Na hora de criar um *DataFrame* podemos informar: * *index*: rótulos para as linhas (atributos *index* das *Series*). * *columns*: rótulos para as colunas (atributos *name* das *Series*).

No exemplo, *dados_de_interesse* pode ser: * Um dicionário de: * *arrays* unidimensionais do *numpy*; * ou listas; * ou dicionários; * ou *Series* do *pandas*. * Um *array* bidimensional do *numpy*; * Uma *Serie* do *Pandas*; * Outro *DataFrame*.

3.1.1 Criando *DataFrames* a partir de dicionários de *Series*

- As *Series* do dicionário não precisam possuir o mesmo número de elementos.
- O *index* do *DataFrame* será dado pela **união** dos *index* de todas as *Series* contidas no dicionário.

Exemplo:

[42]:
```
serie_Idade = pd.Series({'Ana':20, 'João': 19, 'Maria': 21, 'Pedro': 22},␣
↪name="Idade")
```

[43]:
```
serie_Peso = pd.Series({'Ana':55, 'João': 80, 'Maria': 62, 'Pedro': 67, 'Túlio':
↪ 73}, name="Peso")
```

[44]:
```
serie_Altura = pd.Series({'Ana':162, 'João': 178, 'Maria': 162, 'Pedro': 165,␣
↪'Túlio': 171}, name="Altura")
```

[45]:
```
dicionario_series_exemplo = {'Idade': serie_Idade, 'Peso': serie_Peso, 'Altura':
↪ serie_Altura}
```

[46]:
```
df_dict_series = pd.DataFrame(dicionario_series_exemplo)
```

[47]:
```
df_dict_series
```

[47]:

	Idade	Peso	Altura
Ana	20.0	55	162
João	19.0	80	178
Maria	21.0	62	162
Pedro	22.0	67	165
Túlio	NaN	73	171

Mais exemplos:

[48]:
```
pd.DataFrame(dicionario_series_exemplo, index=['Ana','Maria'])
```

[48]:

	Idade	Peso	Altura
Ana	20	55	162
Maria	21	62	162

[49]:
```
pd.DataFrame(dicionario_series_exemplo, index=['Ana','Maria'],␣
↪columns=['Peso','Altura'])
```

[49]:

	Peso	Altura
Ana	55	162
Maria	62	162

Mais exemplos:

[50]:
```
pd.DataFrame(dicionario_series_exemplo, index=['Ana','Maria','Paula'],
             columns=['Peso','Altura','IMC'])
```

[50]:

	Peso	Altura	IMC
Ana	55.0	162.0	NaN
Maria	62.0	162.0	NaN
Paula	NaN	NaN	NaN

[51]:
```
df_exemplo_IMC = pd.DataFrame(dicionario_series_exemplo,
             columns=['Peso','Altura','IMC'])
```

[52]:
```
df_exemplo_IMC['IMC']=round(df_exemplo_IMC['Peso']/(df_exemplo_IMC['Altura']/
↪100)**2,2)
```

[53]:
```
df_exemplo_IMC
```

[53]:
```
        Peso  Altura    IMC
Ana       55     162  20.96
João      80     178  25.25
Maria     62     162  23.62
Pedro     67     165  24.61
Túlio     73     171  24.96
```

3.1.2 Criando *DataFrames* a partir de dicionários de listas ou *arrays* do *numpy*:

- Os *arrays* ou as listas **devem** possuir o mesmo comprimento.
- Se o *index* não for informado, o *index* será dado de forma similar ao do objeto tipo *Series*.

Exemplo com dicionário de listas:

[54]:
```
dicionario_lista_exemplo = {'Idade': [20,19,21,22,20],
                            'Peso': [55,80,62,67,73],
                            'Altura': [162,178,162,165,171]}
```

[55]:
```
pd.DataFrame(dicionario_lista_exemplo)
```

[55]:
```
   Idade  Peso  Altura
0     20    55     162
1     19    80     178
2     21    62     162
3     22    67     165
4     20    73     171
```

Mais exemplos:

[56]:
```
pd.DataFrame(dicionario_lista_exemplo,␣
↪index=['Ana','João','Maria','Pedro','Túlio'])
```

[56]:
```
        Idade  Peso  Altura
Ana        20    55     162
João       19    80     178
Maria      21    62     162
Pedro      22    67     165
Túlio      20    73     171
```

Exemplos com dicionário de *arrays* do *numpy*:

[57]:
```
dicionario_array_exemplo = {'Idade': np.array([20,19,21,22,20]),
                            'Peso': np.array([55,80,62,67,73]),
                            'Altura': np.array([162,178,162,165,171])}
```

[58]:
```
pd.DataFrame(dicionario_array_exemplo)
```

[58]:
	Idade	Peso	Altura
0	20	55	162
1	19	80	178
2	21	62	162
3	22	67	165
4	20	73	171

Mais exemplos:

[59]:
```
pd.DataFrame(dicionario_array_exemplo,␣
↪index=['Ana','João','Maria','Pedro','Túlio'])
```

[59]:
	Idade	Peso	Altura
Ana	20	55	162
João	19	80	178
Maria	21	62	162
Pedro	22	67	165
Túlio	20	73	171

3.1.3 Criando *DataFrames* a partir de uma *Serie* do *pandas*

- Neste caso o *DataFrame* terá o mesmo *index* que a *Serie* do *pandas* e apenas uma coluna.

[60]:
```
series_exemplo = pd.Series({'Ana':20, 'João': 19, 'Maria': 21, 'Pedro': 22,␣
↪'Túlio': 20})
```

[61]:
```
pd.DataFrame(series_exemplo)
```

[61]:
	0
Ana	20
João	19
Maria	21
Pedro	22
Túlio	20

- Caso a *Serie* possua um atributo *name*, este será o nome da coluna do *DataFrame*.

[62]:
```
series_exemplo_Idade = pd.Series({'Ana':20, 'João': 19, 'Maria': 21, 'Pedro':␣
↪22, 'Túlio': 20}, name="Idade")
```

[63]:
```
pd.DataFrame(series_exemplo_Idade)
```

[63]:
	Idade
Ana	20
João	19
Maria	21
Pedro	22
Túlio	20

3.1.4 Criando *DataFrames* a partir de lista de *Series* do *pandas*

- Neste caso a entrada dos dados da lista no *DataFrame* será feita por linha.

[64]:
```
pd.DataFrame([serie_Peso, serie_Altura, serie_Idade])
```

[64]:

	Ana	João	Maria	Pedro	Túlio
Peso	55.0	80.0	62.0	67.0	73.0
Altura	162.0	178.0	162.0	165.0	171.0
Idade	20.0	19.0	21.0	22.0	NaN

- Podemos corrigir a orientação usando o método *transpose*.

[65]:
```
pd.DataFrame([serie_Peso, serie_Altura, serie_Idade]).transpose()
```

[65]:

	Peso	Altura	Idade
Ana	55.0	162.0	20.0
João	80.0	178.0	19.0
Maria	62.0	162.0	21.0
Pedro	67.0	165.0	22.0
Túlio	73.0	171.0	NaN

3.2 Criando *DataFrames* a partir de arquivos

Para criar *DataFrames* a partir de arquivos, precisamos de funções do tipo *pd.read_FORMATO*, onde *FORMATO* indica o formato a ser importado e supondo que a biblioteca *pandas* foi importada com o nome *pd*.

Os formatos mais comuns são:

- *csv* (comma-separated values),
- *excel*,
- *hdf5* (comumente utilizado em *big data*),
- *json* (comumente utilizado em páginas da internet).

As funções para leitura correspondentes são: * *pd.read_csv*, * *pd.read_excel*, * *pd.read_hdf*, * *pd.read_json*,

respectivamente.

De todas essas a função mais utilizada é a *pd.read_csv*.

Ela possui vários argumentos. Vejamos os mais utilizados:

- file_path_or_buffer: o endereço do arquivo a ser lido. Pode ser um endereço da internet.
- *sep*: o separador entre as entradas de dados. O separador padrão é ‘,’.
- index_col: qual a coluna que deve ser usada para formar o *index*. O padrão é *None*.

Porém pode ser alterado para outro. Um separador comumente encontrado é o (TAB). * names: nomes das colunas a serem usados. O padrão é *None*. * header: número da linha que servirá como nome para as colunas. O padrão é ‘infer’ (ou seja, tenta deduzir automaticamente). Se os nomes das colunas forem passados através do *names*, então *header* será automaticamente considerado como *None*.

Exemplo: Considere o arquivo exemplo_data.csv contendo:

```
,coluna_1,coluna_2
2020-01-01,-0.4160923582996922,1.8103644347460834
2020-01-02,-0.1379696602473578,2.5785204825192785
2020-01-03,0.5758273450544708,0.06086648807755068
2020-01-04,-0.017367186564883633,1.2995865328684455
2020-01-05,1.3842792448510655,-0.3817320973859929
2020-01-06,0.5497056238566345,-1.308789022968975
2020-01-07,-0.2822962331437976,-1.6889791765925102
2020-01-08,-0.9897300598660013,-0.028120707936426497
2020-01-09,0.27558240737928663,-0.1776585993494299
2020-01-10,0.6851316082235455,0.5025348904591399
```

Para ler o arquivo acima basta fazer:

[66]:
```
df_exemplo_0 = pd.read_csv('exemplo_data.csv')
```

[67]:
```
df_exemplo_0
```

[67]:

	Unnamed: 0	coluna_1	coluna_2
0	2020-01-01	-0.416092	1.810364
1	2020-01-02	-0.137970	2.578520
2	2020-01-03	0.575827	0.060866
3	2020-01-04	-0.017367	1.299587
4	2020-01-05	1.384279	-0.381732
5	2020-01-06	0.549706	-1.308789
6	2020-01-07	-0.282296	-1.688979
7	2020-01-08	-0.989730	-0.028121
8	2020-01-09	0.275582	-0.177659
9	2020-01-10	0.685132	0.502535

No exemplo anterior, as colunas receberam nomes corretamentes exceto pela primeira coluna que gostaríamos de considerar como *index*. Neste caso fazemos:

[68]:
```
df_exemplo = pd.read_csv('exemplo_data.csv', index_col=0)
```

[69]:
```
df_exemplo
```

[69]:

	coluna_1	coluna_2
2020-01-01	-0.416092	1.810364
2020-01-02	-0.137970	2.578520
2020-01-03	0.575827	0.060866
2020-01-04	-0.017367	1.299587
2020-01-05	1.384279	-0.381732
2020-01-06	0.549706	-1.308789
2020-01-07	-0.282296	-1.688979
2020-01-08	-0.989730	-0.028121
2020-01-09	0.275582	-0.177659

```
2020-01-10  0.685132  0.502535
```

3.3 O método *head* do *DataFrame*

- O método *head* retorna as primeiras 5 linhas se não houver argumento.

[70]:
```
df_exemplo.head()
```

[70]:
```
            coluna_1  coluna_2
2020-01-01 -0.416092  1.810364
2020-01-02 -0.137970  2.578520
2020-01-03  0.575827  0.060866
2020-01-04 -0.017367  1.299587
2020-01-05  1.384279 -0.381732
```

- Ou as primeiras n linhas, se for passado o argumento n.

[71]:
```
df_exemplo.head(2)
```

[71]:
```
            coluna_1  coluna_2
2020-01-01 -0.416092  1.810364
2020-01-02 -0.137970  2.578520
```

[72]:
```
df_exemplo.head(7)
```

[72]:
```
            coluna_1  coluna_2
2020-01-01 -0.416092  1.810364
2020-01-02 -0.137970  2.578520
2020-01-03  0.575827  0.060866
2020-01-04 -0.017367  1.299587
2020-01-05  1.384279 -0.381732
2020-01-06  0.549706 -1.308789
2020-01-07 -0.282296 -1.688979
```

3.4 O método *tail* do *DataFrame*

O método *tail* retorna as últimas 5 linhas se não houver argumento.

[73]:
```
df_exemplo.tail()
```

[73]:
```
            coluna_1  coluna_2
2020-01-06  0.549706 -1.308789
2020-01-07 -0.282296 -1.688979
2020-01-08 -0.989730 -0.028121
2020-01-09  0.275582 -0.177659
2020-01-10  0.685132  0.502535
```

- Ou as últimas n linhas, se for passado o argumento n.

[74]:
```
df_exemplo.tail(2)
```

[74]:
```
            coluna_1  coluna_2
2020-01-09  0.275582 -0.177659
2020-01-10  0.685132  0.502535
```

[75]:
```
df_exemplo.tail(7)
```

[75]:
```
            coluna_1  coluna_2
2020-01-04 -0.017367  1.299587
2020-01-05  1.384279 -0.381732
2020-01-06  0.549706 -1.308789
2020-01-07 -0.282296 -1.688979
2020-01-08 -0.989730 -0.028121
2020-01-09  0.275582 -0.177659
2020-01-10  0.685132  0.502535
```

3.5 Atributos de *Series* e *DataFrames*

- **shape** fornece as dimensões do objeto em questão (*Series* ou *DataFrame*) em formato consistente com o atributo *shape* de um *array* do *numpy*.
- **index** fornece o índice do objeto. No caso do *DataFrame* são os rótulos das linhas.
- **columns** (apenas disponível para *DataFrames*) fornece as colunas.

Exemplo:

[76]:
```
df_exemplo.shape
```

[76]:
```
(10, 2)
```

[77]:
```
serie_1.shape
```

[77]:
```
(10,)
```

[78]:
```
df_exemplo.index
```

[78]:
```
Index(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04', '2020-01-05',
       '2020-01-06', '2020-01-07', '2020-01-08', '2020-01-09', '2020-01-10'],
      dtype='object')
```

[79]:
```
serie_1.index
```

[79]:
```
DatetimeIndex(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
               '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
               '2020-01-09', '2020-01-10'],
              dtype='datetime64[ns]', freq='D')
```

[80]:
```
df_exemplo.columns
```

[80]:
```
Index(['coluna_1', 'coluna_2'], dtype='object')
```

Se quisermos obter os dados contidos nos *index* ou nas *Series* podemos utilizar a propriedade *.array*:

[81]:
```
serie_1.index.array
```

[81]:
```
<DatetimeArray>
['2020-01-01 00:00:00', '2020-01-02 00:00:00', '2020-01-03 00:00:00',
 '2020-01-04 00:00:00', '2020-01-05 00:00:00', '2020-01-06 00:00:00',
 '2020-01-07 00:00:00', '2020-01-08 00:00:00', '2020-01-09 00:00:00',
 '2020-01-10 00:00:00']
Length: 10, dtype: datetime64[ns]
```

[82]:
```
df_exemplo.columns.array
```

[82]:
```
<PandasArray>
['coluna_1', 'coluna_2']
Length: 2, dtype: object
```

Se o interesse for em obter os dados como um *array* do *numpy*, devemos utilizar o método *.to_numpy()*.

Exemplo:

[83]:
```
serie_1.index.to_numpy()
```

[83]:
```
array(['2020-01-01T00:00:00.000000000', '2020-01-02T00:00:00.000000000',
       '2020-01-03T00:00:00.000000000', '2020-01-04T00:00:00.000000000',
       '2020-01-05T00:00:00.000000000', '2020-01-06T00:00:00.000000000',
       '2020-01-07T00:00:00.000000000', '2020-01-08T00:00:00.000000000',
       '2020-01-09T00:00:00.000000000', '2020-01-10T00:00:00.000000000'],
      dtype='datetime64[ns]')
```

[84]:
```
df_exemplo.columns.to_numpy()
```

[84]:
```
array(['coluna_1', 'coluna_2'], dtype=object)
```

O método *.to_numpy()* também está disponível em *DataFrames*:

[85]:
```
df_exemplo.to_numpy()
```

[85]:
```
array([[-0.41609236,  1.81036443],
       [-0.13796966,  2.57852048],
       [ 0.57582735,  0.06086649],
       [-0.01736719,  1.29958653],
       [ 1.38427924, -0.3817321 ],
       [ 0.54970562, -1.30878902],
       [-0.28229623, -1.68897918],
       [-0.98973006, -0.02812071],
```

```
       [ 0.27558241, -0.1776586 ],
       [ 0.68513161,  0.50253489]])
```

- A função do *numpy np.asarray()* é compatível com *index*, *columns* e *DataFrames* do *pandas*:

[86]:
```
np.asarray(df_exemplo.index)
```

[86]:
```
array(['2020-01-01', '2020-01-02', '2020-01-03', '2020-01-04',
       '2020-01-05', '2020-01-06', '2020-01-07', '2020-01-08',
       '2020-01-09', '2020-01-10'], dtype=object)
```

[87]:
```
np.asarray(df_exemplo.columns)
```

[87]:
```
array(['coluna_1', 'coluna_2'], dtype=object)
```

[88]:
```
np.asarray(df_exemplo)
```

[88]:
```
array([[-0.41609236,  1.81036443],
       [-0.13796966,  2.57852048],
       [ 0.57582735,  0.06086649],
       [-0.01736719,  1.29958653],
       [ 1.38427924, -0.3817321 ],
       [ 0.54970562, -1.30878902],
       [-0.28229623, -1.68897918],
       [-0.98973006, -0.02812071],
       [ 0.27558241, -0.1776586 ],
       [ 0.68513161,  0.50253489]])
```

3.6 Informações sobre as colunas de um *DataFrame*

Para obtermos uma breve descrição sobre as colunas de um *DataFrame* utilizamos o método *info*.

Exemplo:

[89]:
```
df_exemplo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10 entries, 2020-01-01 to 2020-01-10
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   coluna_1  10 non-null     float64
 1   coluna_2  10 non-null     float64
dtypes: float64(2)
memory usage: 240.0+ bytes
```

3.7 Criando arquivos a partir de *DataFrames*

Para criar arquivos a partir de *DataFrames*, basta utilizar os métodos do tipo *.to_FORMATO*, onde *FORMATO* indica o formato a ser exportado e supondo que a biblioteca *pandas* foi importada com o nome *pd*.

Com relação aos tipos de arquivo anteriores, os métodos para exportação correspondentes são: * *.to_csv('endereço_do_arquivo')*, * *.to_excel('endereço_do_arquivo')*, * *.to_hdf('endereço_do_arquivo')*, * *.to_json('endereço_do_arquivo')*,

onde 'endereço_do_arquivo' é uma *string* que contém o endereço do arquivo a ser exportado.

Exemplo:

Para exportar para o arquivo exemplo_novo.csv, utilizaremos o método *.to_csv* ao *DataFrame df_exemplo*:

[90]:
```
df_exemplo.to_csv('exemplo_novo.csv')
```

3.8 Exemplo COVID-19 PB

Dados diários de COVID-19 do estado da Paraíba:

Fonte: https://superset.plataformatarget.com.br/superset/dashboard/microdados/

[91]:
```
dados_covid_PB = pd.read_csv('https://superset.plataformatarget.com.br/superset/
↪explore_json/?form_data=%7B%22slice_id%22%3A1550%7D&csv=true',
                             sep=';', index_col=0)
```

[92]:
```
dados_covid_PB.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 127 entries, 2020-07-20 to 2020-03-16
Data columns (total 7 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   casosAcumulados   127 non-null    int64
 1   casosNovos        127 non-null    int64
 2   descartados       127 non-null    int64
 3   recuperados       127 non-null    int64
 4   obitosAcumulados  127 non-null    int64
 5   obitosNovos       127 non-null    int64
 6   Letalidade        127 non-null    float64
dtypes: float64(1), int64(6)
memory usage: 7.9+ KB
```

[93]:
```
dados_covid_PB.head()
```

[93]:

data	casosAcumulados	casosNovos	descartados	recuperados	\
2020-07-20	67680	298	76190	24486	

```
2020-07-19            67382         411        76186        24439
2020-07-18            66971         624        76176        24437
2020-07-17            66347         924        76102        24390
2020-07-16            65423        1484        75757        24253

            obitosAcumulados  obitosNovos  Letalidade
data
2020-07-20              1517           31    0.022414
2020-07-19              1486            9    0.022053
2020-07-18              1477           31    0.022054
2020-07-17              1446           28    0.021795
2020-07-16              1418           35    0.021674
```

[94]:
```
dados_covid_PB.tail()
```

[94]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
data
2020-03-20                0           0            0            0
2020-03-19                0           0            0            0
2020-03-18                0           0            0            0
2020-03-17                0           0            0            0
2020-03-16                0           0            0            0

            obitosAcumulados  obitosNovos  Letalidade
data
2020-03-20                 0            0         0.0
2020-03-19                 0            0         0.0
2020-03-18                 0            0         0.0
2020-03-17                 0            0         0.0
2020-03-16                 0            0         0.0
```

[95]:
```
dados_covid_PB['estado'] = 'PB'
```

[96]:
```
dados_covid_PB.head()
```

[96]:
```
            casosAcumulados  casosNovos  descartados  recuperados  \
data
2020-07-20            67680         298        76190        24486
2020-07-19            67382         411        76186        24439
2020-07-18            66971         624        76176        24437
2020-07-17            66347         924        76102        24390
2020-07-16            65423        1484        75757        24253

            obitosAcumulados  obitosNovos  Letalidade estado
data
2020-07-20              1517           31    0.022414     PB
2020-07-19              1486            9    0.022053     PB
```

```
2020-07-18              1477          31    0.022054     PB
2020-07-17              1446          28    0.021795     PB
2020-07-16              1418          35    0.021674     PB
```

[97]:
```
dados_covid_PB.to_csv('dadoscovidpb.csv')
```