RA27246_aula6_ex6-5_cap7

April 26, 2024

0.1 IA376I – Tópicos em Engenharia de Computação VII

0.1.1 Tópico: Análise de Dados Visual (Visual Analytics)

Professora: Wu, Shin - Ting **Aluno:** Luiz Roberto Albano Junior **RA:** 272746

Aula 06 - 19/04/2024

0.1.2 Exercícios 6.5

Exercício 2: Reproduza os exemplos fornecidos nos Capítulos 6, 7 e 8 em [89] (Python) ou nos Capítulos 9 a 16 em [90] (R). Em ambas as referências, são abordadas diversas funções adicionais de manipulação dos dados, além das apresentadas neste capítulo, proporcionando uma visão mais abrangente das capacidades das respectivas linguagens de programação.

Reprodução dos passos:

Capítulo 7: limpeza e preparação de dados

Tratamento de dados ausentes

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

#Base da URL do repositório com os arquivos de exemplo utilizados no livro
base_url = "https://raw.githubusercontent.com/wesm/pydata-book/3rd-edition/"

float_data = pd.Series([1.2, -3.5, np.nan, 0])
float_data
```

[ ]:
```
0    1.2
1   -3.5
2    NaN
3    0.0
dtype: float64
```

[ ]:
```python
float_data.isna()
```

[ ]:

```
0    False
1    False
2     True
3    False
dtype: bool
```

[ ]:

```
string_data = pd.Series(["aardvark", np.nan, None, "avocado"])
string_data
```

[ ]:

```
0    aardvark
1         NaN
2        None
3     avocado
dtype: object
```

[ ]:

```
string_data.isna()
```

[ ]:

```
0    False
1     True
2     True
3    False
dtype: bool
```

[ ]:

```
float_data = pd.Series([1, 2, None], dtype='float64')
float_data
```

[ ]:

```
0    1.0
1    2.0
2    NaN
dtype: float64
```

[ ]:

```
float_data.isna()
```

[ ]:

```
0    False
1    False
2     True
dtype: bool
```

[ ]:

```
data = pd.Series([1, np.nan, 3.5, np.nan, 7])
data.dropna()
```

[ ]:

```
0    1.0
2    3.5
4    7.0
dtype: float64
```

[ ]:

```
data[data.notna()]
```

```
[ ]: 0    1.0
     2    3.5
     4    7.0
     dtype: float64
```

```
[ ]: data = pd.DataFrame([
         [1., 6.5, 3.],
         [1., np.nan, np.nan],
         [np.nan, np.nan, np.nan],
         [np.nan, 6.5, 3.]
     ])
     data
```

```
[ ]:      0    1    2
     0  1.0  6.5  3.0
     1  1.0  NaN  NaN
     2  NaN  NaN  NaN
     3  NaN  6.5  3.0
```

```
[ ]: data.dropna()
```

```
[ ]:      0    1    2
     0  1.0  6.5  3.0
```

```
[ ]: data.dropna(how="all")
```

```
[ ]:      0    1    2
     0  1.0  6.5  3.0
     1  1.0  NaN  NaN
     3  NaN  6.5  3.0
```

```
[ ]: data[4] = np.nan
     data
```

```
[ ]:      0    1    2   4
     0  1.0  6.5  3.0 NaN
     1  1.0  NaN  NaN NaN
     2  NaN  NaN  NaN NaN
     3  NaN  6.5  3.0 NaN
```

```
[ ]: data.dropna(axis="columns", how="all")
```

```
[ ]:      0    1    2
     0  1.0  6.5  3.0
     1  1.0  NaN  NaN
     2  NaN  NaN  NaN
     3  NaN  6.5  3.0
```

[ ]:

```python
df = pd.DataFrame(np.random.standard_normal((7, 3)))
df.iloc[:4, 1] = np.nan
df.iloc[:2, 2] = np.nan
df
```

[ ]:

```
          0         1         2
0 -1.000587       NaN       NaN
1  0.068673       NaN       NaN
2 -1.312555       NaN  1.303557
3 -0.243463       NaN -0.780911
4 -0.125387  0.708737 -0.966796
5  1.796565  2.950215 -0.947793
6 -0.360379 -1.212389  0.269852
```

[ ]:

```python
df.dropna()
```

[ ]:

```
          0         1         2
4 -0.125387  0.708737 -0.966796
5  1.796565  2.950215 -0.947793
6 -0.360379 -1.212389  0.269852
```

[ ]:

```python
df.dropna(thresh=2)
```

[ ]:

```
          0         1         2
2 -1.312555       NaN  1.303557
3 -0.243463       NaN -0.780911
4 -0.125387  0.708737 -0.966796
5  1.796565  2.950215 -0.947793
6 -0.360379 -1.212389  0.269852
```

[ ]:

```python
df.fillna(0)
```

[ ]:

```
          0         1         2
0 -1.000587  0.000000  0.000000
1  0.068673  0.000000  0.000000
2 -1.312555  0.000000  1.303557
3 -0.243463  0.000000 -0.780911
4 -0.125387  0.708737 -0.966796
5  1.796565  2.950215 -0.947793
6 -0.360379 -1.212389  0.269852
```

[ ]:

```python
df.fillna({1: 0.5, 2: 0})
```

[ ]:

```
          0         1         2
0 -1.000587  0.500000  0.000000
1  0.068673  0.500000  0.000000
2 -1.312555  0.500000  1.303557
```

```
3 -0.243463  0.500000 -0.780911
4 -0.125387  0.708737 -0.966796
5  1.796565  2.950215 -0.947793
6 -0.360379 -1.212389  0.269852
```

[ ]:

```
df = pd.DataFrame(np.random.standard_normal((6, 3)))
df.iloc[2:, 1] = np.nan
df.iloc[4:, 2] = np.nan
df
```

[ ]:

```
          0         1         2
0  0.354313  1.344383 -0.526733
1  1.090709 -1.020539 -0.914861
2 -0.738487       NaN  0.948925
3  0.453201       NaN -0.483352
4 -0.110790       NaN       NaN
5  0.029640       NaN       NaN
```

[ ]:

```
df.ffill()
```

[ ]:

```
          0         1         2
0  0.354313  1.344383 -0.526733
1  1.090709 -1.020539 -0.914861
2 -0.738487 -1.020539  0.948925
3  0.453201 -1.020539 -0.483352
4 -0.110790 -1.020539 -0.483352
5  0.029640 -1.020539 -0.483352
```

[ ]:

```
df.ffill(limit=2)
```

[ ]:

```
          0         1         2
0  0.354313  1.344383 -0.526733
1  1.090709 -1.020539 -0.914861
2 -0.738487 -1.020539  0.948925
3  0.453201 -1.020539 -0.483352
4 -0.110790       NaN -0.483352
5  0.029640       NaN -0.483352
```

[ ]:

```
data = pd.Series([1., np.nan, 3.5, np.nan, 7])
data.fillna(data.mean())
```

[ ]:

```
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
dtype: float64
```

Transformação de dados

[ ]:
```
data = pd.DataFrame({"k1": ["one", "two"] * 3 + ["two"], "k2": [1, 1, 2, 3, 3,␣
 ↪4, 4]})
data
```

[ ]:
```
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
6  two   4
```

[ ]:
```
data.duplicated()
```

[ ]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

[ ]:
```
data.drop_duplicates()
```

[ ]:
```
    k1  k2
0  one   1
1  two   1
2  one   2
3  two   3
4  one   3
5  two   4
```

[ ]:
```
data["v1"] = range(7)
data
```

[ ]:
```
    k1  k2  v1
0  one   1   0
1  two   1   1
2  one   2   2
3  two   3   3
4  one   3   4
5  two   4   5
6  two   4   6
```

[ ]:

```
data.drop_duplicates(subset=["k1"])
```

[ ]:

```
    k1  k2  v1
0  one   1   0
1  two   1   1
```

[ ]:

```
data.drop_duplicates(["k1", "k2"], keep="last")
```

[ ]:

```
    k1  k2  v1
0  one   1   0
1  two   1   1
2  one   2   2
3  two   3   3
4  one   3   4
6  two   4   6
```

[ ]:

```
data = pd.DataFrame({
    "food": ["bacon", "pulled pork", "bacon", "pastrami", "corned beef",␣
 ↪"bacon", "pastrami", "honey ham", "nova lox"],
    "ounces": [4, 3, 12, 6, 7.5, 8, 3, 5, 6]
})
data
```

[ ]:

```
          food  ounces
0        bacon     4.0
1  pulled pork     3.0
2        bacon    12.0
3     pastrami     6.0
4  corned beef     7.5
5        bacon     8.0
6     pastrami     3.0
7    honey ham     5.0
8     nova lox     6.0
```

[ ]:

```
meat_to_animal = {
  "bacon": "pig",
  "pulled pork": "pig",
  "pastrami": "cow",
  "corned beef": "cow",
  "honey ham": "pig",
  "nova lox": "salmon"
}

data["animal"] = data["food"].map(meat_to_animal)
data
```

[ ]:

```
          food  ounces  animal
0        bacon     4.0     pig
1  pulled pork     3.0     pig
2        bacon    12.0     pig
3     pastrami     6.0     cow
4  corned beef     7.5     cow
5        bacon     8.0     pig
6     pastrami     3.0     cow
7    honey ham     5.0     pig
8     nova lox     6.0  salmon
```

[ ]:

```
def get_animal(x):
    return meat_to_animal[x]

data["food"].map(get_animal)
```

[ ]:

```
0       pig
1       pig
2       pig
3       cow
4       cow
5       pig
6       cow
7       pig
8    salmon
Name: food, dtype: object
```

[ ]:

```
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data
```

[ ]:

```
0       1.0
1    -999.0
2       2.0
3    -999.0
4   -1000.0
5       3.0
dtype: float64
```

[ ]:

```
data.replace(-999, np.nan)
```

[ ]:

```
0       1.0
1       NaN
2       2.0
3       NaN
4   -1000.0
5       3.0
dtype: float64
```

[ ]:

```
data.replace([-999, -1000], np.nan)
```

[ ]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

[ ]:

```
data.replace([-999, -1000], [np.nan, 0])
```

[ ]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

[ ]:

```
data.replace({-999: np.nan, -1000: 0})
```

[ ]:

```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

[ ]:

```
data = pd.DataFrame(
    np.arange(12).reshape((3, 4)),
    index=["Ohio", "Colorado", "New York"],
    columns=["one", "two", "three", "four"]
)
data
```

[ ]:

```
          one  two  three  four
Ohio        0    1      2     3
Colorado    4    5      6     7
New York    8    9     10    11
```

[ ]:

```
def transform(x):
    return x[:4].upper()

data.index.map(transform)
```

```
[ ]: Index(['OHIO', 'COLO', 'NEW '], dtype='object')
```

```
[ ]: data.index = data.index.map(transform)
     data
```

```
[ ]:       one  two  three  four
     OHIO    0    1      2     3
     COLO    4    5      6     7
     NEW     8    9     10    11
```

```
[ ]: data.rename(index=str.title, columns=str.upper)
```

```
[ ]:       ONE  TWO  THREE  FOUR
     Ohio    0    1      2     3
     Colo    4    5      6     7
     New     8    9     10    11
```

```
[ ]: data.rename(index={"OHIO": "INDIANA"}, columns={"three": "peekaboo"})
```

```
[ ]:          one  two  peekaboo  four
     INDIANA    0    1         2     3
     COLO       4    5         6     7
     NEW        8    9        10    11
```

```
[ ]: ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

```
[ ]: bins = [18, 25, 35, 60, 100]
     age_categories = pd.cut(ages, bins)
     age_categories
```

```
[ ]: [(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], …, (25, 35], (60, 100],
     (35, 60], (35, 60], (25, 35]]
     Length: 12
     Categories (4, interval[int64, right]): [(18, 25] < (25, 35] < (35, 60] < (60,
     100]]
```

```
[ ]: age_categories.codes
```

```
[ ]: array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
```

```
[ ]: age_categories.categories
```

```
[ ]: IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]], dtype='interval[int64,
     right]')
```

```
[ ]: age_categories.categories[0]
[ ]: Interval(18, 25, closed='right')
```

```
[ ]: pd.value_counts(age_categories)
```

```
/tmp/ipykernel_29990/3010498523.py:1: FutureWarning: pandas.value_counts is
deprecated and will be removed in a future version. Use
pd.Series(obj).value_counts() instead.
  pd.value_counts(age_categories)
```

```
[ ]: (18, 25]     5
     (25, 35]     3
     (35, 60]     3
     (60, 100]    1
     Name: count, dtype: int64
```

```
[ ]: pd.cut(ages, bins, right=False)
```

```
[ ]: [[18, 25), [18, 25), [25, 35), [25, 35), [18, 25), …, [25, 35), [60, 100),
     [35, 60), [35, 60), [25, 35)]
     Length: 12
     Categories (4, interval[int64, left]): [[18, 25) < [25, 35) < [35, 60) < [60,
     100)]
```

```
[ ]: group_names = ["Youth", "YoungAdult", "MiddleAged", "Senior"]
     pd.cut(ages, bins, labels=group_names)
```

```
[ ]: ['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', …, 'YoungAdult', 'Senior',
     'MiddleAged', 'MiddleAged', 'YoungAdult']
     Length: 12
     Categories (4, object): ['Youth' < 'YoungAdult' < 'MiddleAged' < 'Senior']
```

```
[ ]: data = np.random.uniform(size=20)
     pd.cut(data, 4, precision=2)
```

```
[ ]: [(0.086, 0.31], (0.77, 1.0], (0.54, 0.77], (0.086, 0.31], (0.086, 0.31], …,
     (0.77, 1.0], (0.086, 0.31], (0.086, 0.31], (0.31, 0.54], (0.31, 0.54]]
     Length: 20
     Categories (4, interval[float64, right]): [(0.086, 0.31] < (0.31, 0.54] < (0.54,
     0.77] < (0.77, 1.0]]
```

```
[ ]: data = np.random.standard_normal(1000)
     quartiles = pd.qcut(data, 4, precision=2)
     quartiles
```

```
[ ]: [(0.63, 3.93], (-2.96, -0.7], (-0.014, 0.63], (-0.014, 0.63], (0.63, 3.93], …,
     (-0.014, 0.63], (-0.014, 0.63], (-0.7, -0.014], (0.63, 3.93], (-0.014, 0.63]]
     Length: 1000
```

```
Categories (4, interval[float64, right]): [(-2.96, -0.7] < (-0.7, -0.014] <
(-0.014, 0.63] < (0.63, 3.93]]
```

```
[ ]: pd.value_counts(quartiles)
```

```
/tmp/ipykernel_29990/3472704981.py:1: FutureWarning: pandas.value_counts is
deprecated and will be removed in a future version. Use
pd.Series(obj).value_counts() instead.
  pd.value_counts(quartiles)
```

```
[ ]: (-2.96, -0.7]     250
     (-0.7, -0.014]    250
     (-0.014, 0.63]    250
     (0.63, 3.93]      250
     Name: count, dtype: int64
```

```
[ ]: pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.]).value_counts()
```

```
[ ]: (-2.979, -1.342]    100
     (-1.342, 0.0119]    400
     (0.0119, 1.338]     400
     (1.338, 2.832]      100
     Name: count, dtype: int64
```

```
[ ]: data = pd.DataFrame(np.random.standard_normal((1000, 4)))
     data.describe()
```

```
[ ]:                  0            1            2            3
     count  1000.000000  1000.000000  1000.000000  1000.000000
     mean      0.022434    -0.004741    -0.058187     0.044673
     std       1.008762     0.995184     0.991329     0.997007
     min      -3.184377    -3.745356    -3.428254    -3.645860
     25%      -0.628122    -0.697084    -0.747478    -0.599807
     50%      -0.013609    -0.029924    -0.091364     0.047101
     75%       0.695298     0.694459     0.618965     0.740562
     max       3.525865     2.735527     3.366626     2.653656
```

```
[ ]: col = data[2]
     col[col.abs() > 3]
```

```
[ ]: 270   -3.428254
     647    3.366626
     Name: 2, dtype: float64
```

```
[ ]: data[(data.abs() > 3).any(axis="columns")]
```

[ ]:

```
            0         1         2         3
53  -0.025907 -3.399312 -0.974657 -0.685312
72   3.260383  0.963301  1.201206 -1.852001
148 -0.196713 -3.745356 -1.520113 -0.346839
247 -3.056990  1.918403 -0.578828  1.847446
270  0.326045  0.425384 -3.428254 -0.296336
334 -3.184377  1.369891 -1.074833 -0.089937
555  0.208011 -0.150923 -0.362528 -3.548824
647  0.193299  1.397822  3.366626 -2.372214
794  3.525865  0.283070  0.544635  0.462204
814 -0.450721 -0.080332  0.599947 -3.645860
```

[ ]:

```
data[data.abs() > 3] = np.sign(data) * 3
data.describe()
```

[ ]:

```
                 0            1            2            3
count  1000.000000  1000.000000  1000.000000  1000.000000
mean      0.021889    -0.003596    -0.058126     0.045868
std       1.005520     0.991368     0.988761     0.992986
min      -3.000000    -3.000000    -3.000000    -3.000000
25%      -0.628122    -0.697084    -0.747478    -0.599807
50%      -0.013609    -0.029924    -0.091364     0.047101
75%       0.695298     0.694459     0.618965     0.740562
max       3.000000     2.735527     3.000000     2.653656
```

[ ]:

```
np.sign(data).head()
```

[ ]:

```
     0    1    2    3
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
```

[ ]:

```
df = pd.DataFrame(np.arange(5 * 7).reshape((5, 7)))
df
```

[ ]:

```
    0   1   2   3   4   5   6
0   0   1   2   3   4   5   6
1   7   8   9  10  11  12  13
2  14  15  16  17  18  19  20
3  21  22  23  24  25  26  27
4  28  29  30  31  32  33  34
```

[ ]:

```
sampler = np.random.permutation(5)
sampler
```

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

```
[ ]: df.take(sampler)
```

```
[ ]:     0   1   2   3   4   5   6
     1   7   8   9  10  11  12  13
     3  21  22  23  24  25  26  27
     2  14  15  16  17  18  19  20
     4  28  29  30  31  32  33  34
     0   0   1   2   3   4   5   6
```

```
[ ]: df.iloc[sampler]
```

```
[ ]:     0   1   2   3   4   5   6
     1   7   8   9  10  11  12  13
     3  21  22  23  24  25  26  27
     2  14  15  16  17  18  19  20
     4  28  29  30  31  32  33  34
     0   0   1   2   3   4   5   6
```

```
[ ]: column_sampler = np.random.permutation(7)
     column_sampler
```

```
[ ]: array([6, 1, 2, 4, 5, 0, 3])
```

```
[ ]: df.take(column_sampler, axis="columns")
```

```
[ ]:     6   1   2   4   5   0   3
     0   6   1   2   4   5   0   3
     1  13   8   9  11  12   7  10
     2  20  15  16  18  19  14  17
     3  27  22  23  25  26  21  24
     4  34  29  30  32  33  28  31
```

```
[ ]: df.sample(n=3)
```

```
[ ]:     0   1   2   3   4   5   6
     4  28  29  30  31  32  33  34
     0   0   1   2   3   4   5   6
     1   7   8   9  10  11  12  13
```

```
[ ]: choices = pd.Series([5, 7, -1, 6, 4])
     choices.sample(n=10, replace=True)
```

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

```
4    4
3    6
0    5
3    6
1    7
4    4
0    5
dtype: int64
```

```
[ ]: df = pd.DataFrame({
         "key": ["b", "b", "a", "c", "a", "b"],
         "data1": range(6)
     })
     df
```

```
[ ]:   key  data1
     0   b      0
     1   b      1
     2   a      2
     3   c      3
     4   a      4
     5   b      5
```

```
[ ]: pd.get_dummies(df["key"], dtype=float)
```

```
[ ]:      a    b    c
     0  0.0  1.0  0.0
     1  0.0  1.0  0.0
     2  1.0  0.0  0.0
     3  0.0  0.0  1.0
     4  1.0  0.0  0.0
     5  0.0  1.0  0.0
```

```
[ ]: dummies = pd.get_dummies(df["key"], prefix="key", dtype=float)
     df_with_dummy = df[["data1"]].join(dummies)
     df_with_dummy
```

```
[ ]:    data1  key_a  key_b  key_c
     0      0    0.0    1.0    0.0
     1      1    0.0    1.0    0.0
     2      2    1.0    0.0    0.0
     3      3    0.0    0.0    1.0
     4      4    1.0    0.0    0.0
     5      5    0.0    1.0    0.0
```

```
[ ]: mnames = ["movie_id", "title", "genres"]
     movies = pd.read_table(base_url + "datasets/movielens/movies.dat", sep="::",
                            header=None, names=mnames, engine="python")
     movies[:10]
```

```
[ ]:   movie_id                               title                        genres
    0         1                    Toy Story (1995)   Animation|Children's|Comedy
    1         2                      Jumanji (1995)  Adventure|Children's|Fantasy
    2         3             Grumpier Old Men (1995)                Comedy|Romance
    3         4            Waiting to Exhale (1995)                  Comedy|Drama
    4         5  Father of the Bride Part II (1995)                        Comedy
    5         6                         Heat (1995)         Action|Crime|Thriller
    6         7                      Sabrina (1995)                Comedy|Romance
    7         8                 Tom and Huck (1995)          Adventure|Children's
    8         9                 Sudden Death (1995)                        Action
    9        10                    GoldenEye (1995)     Action|Adventure|Thriller
```

```
[ ]: dummies = movies["genres"].str.get_dummies("|")
     dummies.iloc[:10, :6]
```

```
[ ]:   Action  Adventure  Animation  Children's  Comedy  Crime
    0       0          0          1           1       1      0
    1       0          1          0           1       0      0
    2       0          0          0           0       1      0
    3       0          0          0           0       1      0
    4       0          0          0           0       1      0
    5       1          0          0           0       0      1
    6       0          0          0           0       1      0
    7       0          1          0           1       0      0
    8       1          0          0           0       0      0
    9       1          1          0           0       0      0
```

```
[ ]: movies_windic = movies.join(dummies.add_prefix("Genre_"))
     movies_windic.iloc[0]
```

```
[ ]: movie_id                                       1
     title                           Toy Story (1995)
     genres               Animation|Children's|Comedy
     Genre_Action                                   0
     Genre_Adventure                                0
     Genre_Animation                                1
     Genre_Children's                               1
     Genre_Comedy                                   1
     Genre_Crime                                    0
     Genre_Documentary                              0
     Genre_Drama                                    0
     Genre_Fantasy                                  0
     Genre_Film-Noir                                0
     Genre_Horror                                   0
```

```
Genre_Musical                                  0
Genre_Mystery                                  0
Genre_Romance                                  0
Genre_Sci-Fi                                   0
Genre_Thriller                                 0
Genre_War                                      0
Genre_Western                                  0
Name: 0, dtype: object
```

```
[ ]: np.random.seed(12345) # to make the example repeatable
     values = np.random.uniform(size=10)
     values
```

```
[ ]: array([0.92961609, 0.31637555, 0.18391881, 0.20456028, 0.56772503,
            0.5955447 , 0.96451452, 0.6531771 , 0.74890664, 0.65356987])
```

```
[ ]: bins = [0, 0.2, 0.4, 0.6, 0.8, 1]
     pd.get_dummies(pd.cut(values, bins))
```

```
[ ]:    (0.0, 0.2]  (0.2, 0.4]  (0.4, 0.6]  (0.6, 0.8]  (0.8, 1.0]
     0       False       False       False       False        True
     1       False        True       False       False       False
     2        True       False       False       False       False
     3       False        True       False       False       False
     4       False       False        True       False       False
     5       False       False        True       False       False
     6       False       False       False       False        True
     7       False       False       False        True       False
     8       False       False       False        True       False
     9       False       False       False        True       False
```

Tipos de dados de extensão

```
[ ]: s = pd.Series([1, 2, 3, None])
     s
```

```
[ ]: 0    1.0
     1    2.0
     2    3.0
     3    NaN
     dtype: float64
```

```
[ ]: s.dtype
```

```
[ ]: dtype('float64')
[ ]: s = pd.Series([1, 2, 3, None], dtype=pd.Int64Dtype())
     s
```

```
[ ]: 0       1
     1       2
     2       3
     3    <NA>
     dtype: Int64
```

```
[ ]: s.isna()
```

```
[ ]: 0    False
     1    False
     2    False
     3     True
     dtype: bool
```

```
[ ]: s.dtype
```

```
[ ]: Int64Dtype()
```

```
[ ]: s[3]
```

```
[ ]: <NA>
```

```
[ ]: s[3] is pd.NA
```

```
[ ]: True
```

```
[ ]: s = pd.Series([1, 2, 3, None], dtype="Int64")
```

```
[ ]: s = pd.Series(['one', 'two', None, 'three'], dtype=pd.StringDtype())
     s
```

```
[ ]: 0      one
     1      two
     2     <NA>
     3    three
     dtype: string
```

```
[ ]: df = pd.DataFrame({"A": [1, 2, None, 4],
                        "B": ["one", "two", "three", None],
                        "C": [False, None, False, True]})
     df
```

```
[ ]:      A      B      C
     0  1.0    one  False
```

```
1  2.0    two   None
2  NaN  three  False
3  4.0   None   True
```

[ ]:
```
df["A"] = df["A"].astype("Int64")
df["B"] = df["B"].astype("string")
df["C"] = df["C"].astype("boolean")
df
```

[ ]:
```
      A      B      C
0     1    one  False
1     2    two   <NA>
2  <NA>  three  False
3     4   <NA>   True
```

Manipulação de string

[ ]:
```
val = "a,b,  guido"
val.split(",")
```

[ ]:
```
['a', 'b', '  guido']
```

[ ]:
```
pieces = [x.strip() for x in val.split(",")]
pieces
```

[ ]:
```
['a', 'b', 'guido']
```

[ ]:
```
first, second, third = pieces
first + "::" + second + "::" + third
```

[ ]:
```
'a::b::guido'
```

[ ]:
```
"::".join(pieces)
```

[ ]:
```
'a::b::guido'
```

[ ]:
```
"guido" in val
```

[ ]:
```
True
```

[ ]:
```
val.index(",")
```

[ ]:
```
1
```

[ ]:
```
val.find(":")
```

[ ]:
```
-1
```

[ ]:
```
val.index(":")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
/tmp/ipykernel_29990/2601145560.py in <module>
----> 1 val.index(":")

ValueError: substring not found
```

[ ]:
```
val.count(",")
```

[ ]:
```
2
```

[ ]:
```
val.replace(",", "::")
```

[ ]:
```
'a::b::  guido'
```

[ ]:
```
val.replace(",", "")
```

[ ]:
```
'ab  guido'
```

Expressões regulares

[ ]:
```
import re
text = "foo    bar\t baz  \tqux"
re.split(r"\s+", text)
```

[ ]:
```
['foo', 'bar', 'baz', 'qux']
```

[ ]:
```
regex = re.compile(r"\s+")
regex.split(text)
```

[ ]:
```
['foo', 'bar', 'baz', 'qux']
```

[ ]:
```
regex.findall(text)
```

[ ]:
```
['    ', '\t ', '  \t']
```

[ ]:
```
text = """Dave dave@google.com
Steve steve@gmail.com
Rob rob@gmail.com
Ryan ryan@yahoo.com"""
pattern = r"[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}"

# re.IGNORECASE makes the regex case insensitive
regex = re.compile(pattern, flags=re.IGNORECASE)
```

```
[ ]: regex.findall(text)
```

```
[ ]: ['dave@google.com', 'steve@gmail.com', 'rob@gmail.com', 'ryan@yahoo.com']
```

```
[ ]: m = regex.search(text)
     m
```

```
[ ]: <re.Match object; span=(5, 20), match='dave@google.com'>
```

```
[ ]: text[m.start():m.end()]
```

```
[ ]: 'dave@google.com'
```

```
[ ]: print(regex.match(text))
```

```
None
```

```
[ ]: print(regex.sub("REDACTED", text))
```

```
Dave REDACTED
Steve REDACTED
Rob REDACTED
Ryan REDACTED
```

```
[ ]: pattern = r"([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})"
     regex = re.compile(pattern, flags=re.IGNORECASE)
```

```
[ ]: m = regex.match("wesm@bright.net")
     m.groups()
```

```
[ ]: ('wesm', 'bright', 'net')
```

```
[ ]: regex.findall(text)
```

```
[ ]: [('dave', 'google', 'com'),
      ('steve', 'gmail', 'com'),
      ('rob', 'gmail', 'com'),
      ('ryan', 'yahoo', 'com')]
```

```
[ ]: print(regex.sub(r"Username: \1, Domain: \2, Suffix: \3", text))
```

```
Dave Username: dave, Domain: google, Suffix: com
Steve Username: steve, Domain: gmail, Suffix: com
Rob Username: rob, Domain: gmail, Suffix: com
Ryan Username: ryan, Domain: yahoo, Suffix: com
```

Funções de strings em pandas

[ ]:

```
data = {"Dave": "dave@google.com", "Steve": "steve@gmail.com",
        "Rob": "rob@gmail.com", "Wes": np.nan}
data = pd.Series(data)
data
```

[ ]:

```
Dave     dave@google.com
Steve    steve@gmail.com
Rob        rob@gmail.com
Wes                  NaN
dtype: object
```

[ ]:

```
data.isna()
```

[ ]:

```
Dave     False
Steve    False
Rob      False
Wes       True
dtype: bool
```

[ ]:

```
data.str.contains("gmail")
```

[ ]:

```
Dave     False
Steve     True
Rob       True
Wes        NaN
dtype: object
```

[ ]:

```
data_as_string_ext = data.astype('string')
data_as_string_ext
```

[ ]:

```
Dave     dave@google.com
Steve    steve@gmail.com
Rob        rob@gmail.com
Wes                 <NA>
dtype: string
```

[ ]:

```
data_as_string_ext.str.contains("gmail")
```

[ ]:

```
Dave     False
Steve     True
Rob       True
Wes       <NA>
dtype: boolean
```

[ ]:

```
pattern = r"([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})"
data.str.findall(pattern, flags=re.IGNORECASE)
```

```
[ ]: Dave     [(dave, google, com)]
     Steve    [(steve, gmail, com)]
     Rob        [(rob, gmail, com)]
     Wes                        NaN
     dtype: object
```

```
[ ]: matches = data.str.findall(pattern, flags=re.IGNORECASE).str[0]
     matches
```

```
[ ]: Dave     dave@google.com
     Steve    steve@gmail.com
     Rob        rob@gmail.com
     Wes                  NaN
     dtype: object
```

```
[ ]: matches.str.get(1)
```

```
[ ]: Dave       a
     Steve      t
     Rob        o
     Wes      NaN
     dtype: object
```

```
[ ]: data.str[:5]
```

```
[ ]: Dave     dave@
     Steve    steve
     Rob      rob@g
     Wes        NaN
     dtype: object
```

```
[ ]: data.str.extract(pattern, flags=re.IGNORECASE)
```

```
[ ]:            0       1    2
     Dave    dave  google  com
     Steve  steve   gmail  com
     Rob      rob   gmail  com
     Wes      NaN     NaN  NaN
```

Dados categpricos

```
[ ]: values = pd.Series(['apple', 'orange', 'apple',
                         'apple'] * 2)
     values
```

```
[ ]: 0     apple
     1    orange
     2     apple
```

```
3     apple
4     apple
5    orange
6     apple
7     apple
dtype: object
```

```
[ ]: pd.unique(values)
```

```
[ ]: array(['apple', 'orange'], dtype=object)
```

```
[ ]: pd.value_counts(values)
```

```
/tmp/ipykernel_29990/3297668723.py:1: FutureWarning: pandas.value_counts is
deprecated and will be removed in a future version. Use
pd.Series(obj).value_counts() instead.
  pd.value_counts(values)
```

```
[ ]: apple     6
     orange    2
     Name: count, dtype: int64
```

```
[ ]: values = pd.Series([0, 1, 0, 0] * 2)
     dim = pd.Series(['apple', 'orange'])
     values
```

```
[ ]: 0    0
     1    1
     2    0
     3    0
     4    0
     5    1
     6    0
     7    0
     dtype: int64
```

```
[ ]: dim
```

```
[ ]: 0     apple
     1    orange
     dtype: object
```

```
[ ]: dim.take(values)
```

```
[ ]: 0     apple
     1    orange
     0     apple
```

```
0     apple
0     apple
1    orange
0     apple
0     apple
dtype: object
```

```
[ ]: fruits = ['apple', 'orange', 'apple', 'apple'] * 2
     N = len(fruits)
     rng = np.random.default_rng(seed=12345)
     df = pd.DataFrame({'fruit': fruits,
                        'basket_id': np.arange(N),
                        'count': rng.integers(3, 15, size=N),
                        'weight': rng.uniform(0, 4, size=N)},
                       columns=['basket_id', 'fruit', 'count', 'weight'])
     df
```

```
[ ]:    basket_id   fruit  count    weight
     0          0   apple     11  1.564438
     1          1  orange      5  1.331256
     2          2   apple     12  2.393235
     3          3   apple      6  0.746937
     4          4   apple      5  2.691024
     5          5  orange     12  3.767211
     6          6   apple     10  0.992983
     7          7   apple     11  3.795525
```

```
[ ]: fruit_cat = df['fruit'].astype('category')
     fruit_cat
```

```
[ ]: 0     apple
     1    orange
     2     apple
     3     apple
     4     apple
     5    orange
     6     apple
     7     apple
     Name: fruit, dtype: category
     Categories (2, object): ['apple', 'orange']
```

```
[ ]: c = fruit_cat.array
     type(c)
```

```
[ ]: pandas.core.arrays.categorical.Categorical
```

```
[ ]: c.categories
[ ]: Index(['apple', 'orange'], dtype='object')
```

```
[ ]: c.codes
```

```
[ ]: array([0, 1, 0, 0, 0, 1, 0, 0], dtype=int8)
```

```
[ ]: dict(enumerate(c.categories))
```

```
[ ]: {0: 'apple', 1: 'orange'}
```

```
[ ]: df['fruit'] = df['fruit'].astype('category')
     df["fruit"]
```

```
[ ]: 0     apple
     1    orange
     2     apple
     3     apple
     4     apple
     5    orange
     6     apple
     7     apple
     Name: fruit, dtype: category
     Categories (2, object): ['apple', 'orange']
```

```
[ ]: my_categories = pd.Categorical(['foo', 'bar', 'baz', 'foo', 'bar'])
     my_categories
```

```
[ ]: ['foo', 'bar', 'baz', 'foo', 'bar']
     Categories (3, object): ['bar', 'baz', 'foo']
```

```
[ ]: categories = ['foo', 'bar', 'baz']
     codes = [0, 1, 2, 0, 0, 1]
     my_cats_2 = pd.Categorical.from_codes(codes, categories)
     my_cats_2
```

```
[ ]: ['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
     Categories (3, object): ['foo', 'bar', 'baz']
```

```
[ ]: ordered_cat = pd.Categorical.from_codes(codes, categories, ordered=True)
     ordered_cat
```

```
[ ]: ['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
     Categories (3, object): ['foo' < 'bar' < 'baz']
```

```
[ ]: my_cats_2.as_ordered()
[ ]: ['foo', 'bar', 'baz', 'foo', 'foo', 'bar']
     Categories (3, object): ['foo' < 'bar' < 'baz']
```

```
[ ]: rng = np.random.default_rng(seed=12345)
     draws = rng.standard_normal(1000)
     draws[:5]
```

```
[ ]: array([-1.42382504,  1.26372846, -0.87066174, -0.25917323, -0.07534331])
```

```
[ ]: bins = pd.qcut(draws, 4)
     bins
```

```
[ ]: [(-3.121, -0.675], (0.687, 3.211], (-3.121, -0.675], (-0.675, 0.0134], (-0.675,
     0.0134], …, (0.0134, 0.687], (0.0134, 0.687], (-0.675, 0.0134], (0.0134,
     0.687], (-0.675, 0.0134]]
     Length: 1000
     Categories (4, interval[float64, right]): [(-3.121, -0.675] < (-0.675, 0.0134] <
     (0.0134, 0.687] < (0.687, 3.211]]
```

```
[ ]: bins = pd.qcut(draws, 4, labels=['Q1', 'Q2', 'Q3', 'Q4'])
     bins
```

```
[ ]: ['Q1', 'Q4', 'Q1', 'Q2', 'Q2', …, 'Q3', 'Q3', 'Q2', 'Q3', 'Q2']
     Length: 1000
     Categories (4, object): ['Q1' < 'Q2' < 'Q3' < 'Q4']
```

```
[ ]: bins.codes[:10]
```

```
[ ]: array([0, 3, 0, 1, 1, 0, 0, 2, 2, 0], dtype=int8)
```

```
[ ]: bins = pd.Series(bins, name='quartile')
     results = (pd.Series(draws)
                .groupby(bins)
                .agg(['count', 'min', 'max'])
                .reset_index())
     results
```

```
/tmp/ipykernel_29990/2483392743.py:3: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future version of
pandas. Pass observed=False to retain current behavior or observed=True to adopt
the future default and silence this warning.
  .groupby(bins)
```

```
[ ]:   quartile  count       min       max
     0       Q1    250 -3.119609 -0.678494
     1       Q2    250 -0.673305  0.008009
     2       Q3    250  0.018753  0.686183
```

```
3       Q4    250  0.688282  3.211418
```

```
[ ]: results['quartile']
```

```
[ ]: 0    Q1
     1    Q2
     2    Q3
     3    Q4
     Name: quartile, dtype: category
     Categories (4, object): ['Q1' < 'Q2' < 'Q3' < 'Q4']
```

```
[ ]: N = 10_000_000
     labels = pd.Series(['foo', 'bar', 'baz', 'qux'] * (N // 4))
     categories = labels.astype('category')
     labels.memory_usage(deep=True)
```

```
[ ]: 600000128
```

```
[ ]: categories.memory_usage(deep=True)
```

```
[ ]: 10000540
```

```
[ ]: %time _ = labels.astype('category')
```

```
CPU times: user 833 ms, sys: 364 ms, total: 1.2 s
Wall time: 1.26 s
```

```
[ ]: %timeit labels.value_counts()
```

```
808 ms ± 160 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
[ ]: %timeit categories.value_counts()
```

```
205 ms ± 39 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
[ ]: s = pd.Series(['a', 'b', 'c', 'd'] * 2)
     cat_s = s.astype('category')
     cat_s
```

```
[ ]: 0    a
     1    b
     2    c
     3    d
     4    a
     5    b
     6    c
     7    d
     dtype: category
     Categories (4, object): ['a', 'b', 'c', 'd']
```

```
[ ]: cat_s.cat.codes
```

```
[ ]: 0    0
     1    1
     2    2
     3    3
     4    0
     5    1
     6    2
     7    3
     dtype: int8
```

```
[ ]: cat_s.cat.categories
```

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

```
[ ]: actual_categories = ['a', 'b', 'c', 'd', 'e']
     cat_s2 = cat_s.cat.set_categories(actual_categories)
     cat_s2
```

```
[ ]: 0    a
     1    b
     2    c
     3    d
     4    a
     5    b
     6    c
     7    d
     dtype: category
     Categories (5, object): ['a', 'b', 'c', 'd', 'e']
```

```
[ ]: cat_s.value_counts()
```

```
[ ]: a    2
     b    2
     c    2
     d    2
     Name: count, dtype: int64
```

```
[ ]: cat_s2.value_counts()
```

```
[ ]: a    2
     b    2
     c    2
     d    2
```

```
e    0
Name: count, dtype: int64
```

```
[ ]: cat_s3 = cat_s[cat_s.isin(['a', 'b'])]
     cat_s3
```

```
[ ]: 0    a
     1    b
     4    a
     5    b
     dtype: category
     Categories (4, object): ['a', 'b', 'c', 'd']
```

```
[ ]: cat_s3.cat.remove_unused_categories()
```

```
[ ]: 0    a
     1    b
     4    a
     5    b
     dtype: category
     Categories (2, object): ['a', 'b']
```

```
[ ]: cat_s = pd.Series(['a', 'b', 'c', 'd'] * 2, dtype='category')
     cat_s
```

```
[ ]: 0    a
     1    b
     2    c
     3    d
     4    a
     5    b
     6    c
     7    d
     dtype: category
     Categories (4, object): ['a', 'b', 'c', 'd']
```

```
[ ]: pd.get_dummies(cat_s, dtype=float)
```

```
[ ]:      a    b    c    d
     0  1.0  0.0  0.0  0.0
     1  0.0  1.0  0.0  0.0
     2  0.0  0.0  1.0  0.0
     3  0.0  0.0  0.0  1.0
     4  1.0  0.0  0.0  0.0
     5  0.0  1.0  0.0  0.0
     6  0.0  0.0  1.0  0.0
     7  0.0  0.0  0.0  1.0
```