# 十分钟入门Pandas

这是对Pandas的简短介绍，主要面向新手。你可以在[Cookbook](http://pandas.pydata.org/pandas-docs/stable/cookbook.html" \l "cookbook)中看到更复杂的用法教程。

通常，我们按如下方式导入 Pandas：

In [1]: import numpy as np

In [2]: import pandas as pd

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#对象创建)对象创建

请参阅 [数据结构简介的部分](https://www.pypandas.cn/docs/getting_started/dsintro.html#dsintro)。

通过传入一些值的列表来创建一个[Series](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.html" \l "pandas.Series" \t "https://www.pypandas.cn/docs/getting_started/_blank)，Pandas会自动创建一个默认的整数索引：

In [3]: s = pd.Series([1, 3, 5, np.nan, 6, 8])

In [4]: s

Out[4]:

0 1.0

1 3.0

2 5.0

3 NaN

4 6.0

5 8.0

dtype: float64

通过传递带有日期时间索引和带标签列的NumPy数组来创建[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \t "https://www.pypandas.cn/docs/getting_started/_blank)：

In [5]: dates = pd.date\_range('20190901', periods=6)

In [6]: dates

Out[6]:

DatetimeIndex(['2019-09-01', '2019-09-02', '2019-09-03', '2019-09-04',

'2019-09-05', '2019-09-06'],

dtype='datetime64[ns]', freq='D')

In [7]: df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list('ABCD'))

In [8]: df

Out[8]:

A B C D

2019-09-01 0.469112 -0.282863 -1.509059 -1.135632

2019-09-02 1.212112 -0.173215 0.119209 -1.044236

2019-09-03 -0.861849 -2.104569 -0.494929 1.071804

2019-09-04 0.721555 -0.706771 -1.039575 0.271860

2019-09-05 -0.424972 0.567020 0.276232 -1.087401

2019-09-06 -0.673690 0.113648 -1.478427 0.524988

通过传递可以转化为类似Series的dict对象来创建DataFrame:

In [9]: df2 = pd.DataFrame({'A': 1.,

'B': pd.Timestamp('20190902'),

'C': pd.Series(1, index=list(range(4)), dtype='float32'),

'D': np.array([3] \* 4, dtype='int32'),

'E': pd.Categorical(["test", "train", "test", "train"]),

'F': 'foo'})

In [10]: df2

Out[10]:

A B C D E F

0 1.0 2019-09-02 1.0 3 test foo

1 1.0 2019-09-02 1.0 3 train foo

2 1.0 2019-09-02 1.0 3 test foo

3 1.0 2019-09-02 1.0 3 train foo

DataFrame的列具有不同的[数据类型](https://www.pypandas.cn/docs/getting_started/basics.html" \l "dtypes数据类型)。

In [11]: df2.dtypes

Out[11]:

A float64

B datetime64[ns]

C float32

D int32

E category

F object

dtype: object

如果你正在使用 IPython, 按下tab键会自动补全所有的列名以及公共属性。下面是可以补全的属性中的一部分：

In [12]: df2.<TAB> # noqa: E225, E999

df2.A df2.bool

df2.abs df2.boxplot

df2.add df2.C

df2.add\_prefix df2.clip

df2.add\_suffix df2.clip\_lower

df2.align df2.clip\_upper

df2.all df2.columns

df2.any df2.combine

df2.append df2.combine\_first

df2.apply df2.compound

df2.applymap df2.consolidate

df2.D

如你所见，列A、B、C和D将自动补全，E也存在；为了简洁起见，只显示了一部分属性。

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#查看数据)查看数据

请查看[基础部分](https://pandas.pydata.org/pandas-docs/stable/getting_started/basics.html#basics)

这里展示的是如何查看DataFrame顶部和尾部的数据：

In [13]: df.head()

Out[13]:

A B C D

2019-09-01 0.469112 -0.282863 -1.509059 -1.135632

2019-09-02 1.212112 -0.173215 0.119209 -1.044236

2019-09-03 -0.861849 -2.104569 -0.494929 1.071804

2019-09-04 0.721555 -0.706771 -1.039575 0.271860

2019-09-05 -0.424972 0.567020 0.276232 -1.087401

In [14]: df.tail(3)

Out[14]:

A B C D

2019-09-04 0.721555 -0.706771 -1.039575 0.271860

2019-09-05 -0.424972 0.567020 0.276232 -1.087401

2019-09-06 -0.673690 0.113648 -1.478427 0.524988

显示索引、列和底层NumPy数据：

In [15]: df.index

Out[15]:

DatetimeIndex(['2019-09-01', '2019-09-02', '2019-09-03', '2019-09-04',

'2019-09-05', '2019-09-06'],

dtype='datetime64[ns]', freq='D')

In [16]: df.columns

Out[16]: Index(['A', 'B', 'C', 'D'], dtype='object')

[DataFrame.to\_numpy()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html#pandas.DataFrame.to_numpy) 会给出一个比较底层的NumPy对象。注意，当你的 [DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \t "https://www.pypandas.cn/docs/getting_started/_blank) 有多个列并且每列的数据类型不同时，这个操作是不可行的，这也可以说是Pandas和NumPy之间的根本区别：**NumPy的每一个array对象只有一种数据类型，但是Pandas的每一列的数据类型都是相同的（译者注：Pandas不需要像Numpy那样所有元素的类型都相同）**. 当你调用 [DataFrame.to\_numpy()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html" \l "pandas.DataFrame.to_numpy" \t "https://www.pypandas.cn/docs/getting_started/_blank)时, Pandas会寻找可以涵盖DataFrame中所有元素类型的NumPy数据类型。 这可能最终成为对象，需要将每个值强制转换为Python对象。

对于 df, 对于所有值是浮点数的[DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \t "https://www.pypandas.cn/docs/getting_started/_blank) [DataFrame.to\_numpy()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html#pandas.DataFrame.to_numpy) 操作是高效的，并且不会复制数据。

In [17]: df.to\_numpy()

Out[17]:

array([[ 0.4691, -0.2829, -1.5091, -1.1356],

[ 1.2121, -0.1732, 0.1192, -1.0442],

[-0.8618, -2.1046, -0.4949, 1.0718],

[ 0.7216, -0.7068, -1.0396, 0.2719],

[-0.425 , 0.567 , 0.2762, -1.0874],

[-0.6737, 0.1136, -1.4784, 0.525 ]])

对于 df2, [DataFrame](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html" \l "pandas.DataFrame" \t "https://www.pypandas.cn/docs/getting_started/_blank) 对象有多种类型, [DataFrame.to\_numpy()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html" \l "pandas.DataFrame.to_numpy" \t "https://www.pypandas.cn/docs/getting_started/_blank) 操作是比较费事的。

In [18]: df2.to\_numpy()

Out[18]:

array([[1.0, Timestamp('2019-09-02 00:00:00'), 1.0, 3, 'test', 'foo'],

[1.0, Timestamp('2019-09-02 00:00:00'), 1.0, 3, 'train', 'foo'],

[1.0, Timestamp('2019-09-02 00:00:00'), 1.0, 3, 'test', 'foo'],

[1.0, Timestamp('2019-09-02 00:00:00'), 1.0, 3, 'train', 'foo']], dtype=object)

**提醒**

[DataFrame.to\_numpy()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.to_numpy.html#pandas.DataFrame.to_numpy) 的输出不包含行索引和列索引。

[describe()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.describe.html#pandas.DataFrame.describe) 方法显示数据的快速统计摘要：

In [19]: df.describe()

Out[19]:

A B C D

count 6.000000 6.000000 6.000000 6.000000

mean 0.073711 -0.431125 -0.687758 -0.233103

std 0.843157 0.922818 0.779887 0.973118

min -0.861849 -2.104569 -1.509059 -1.135632

25% -0.611510 -0.600794 -1.368714 -1.076610

50% 0.022070 -0.228039 -0.767252 -0.386188

75% 0.658444 0.041933 -0.034326 0.461706

max 1.212112 0.567020 0.276232 1.071804

转置数据：

In [20]: df.T

Out[20]:

2019-09-01 2019-09-02 2019-09-03 2019-09-04 2019-09-05 2019-09-06

A 0.469112 1.212112 -0.861849 0.721555 -0.424972 -0.673690

B -0.282863 -0.173215 -2.104569 -0.706771 0.567020 0.113648

C -1.509059 0.119209 -0.494929 -1.039575 0.276232 -1.478427

D -1.135632 -1.044236 1.071804 0.271860 -1.087401 0.524988

按轴排序:

In [21]: df.sort\_index(axis=1, ascending=False)

Out[21]:

D C B A

2019-09-01 -1.135632 -1.509059 -0.282863 0.469112

2019-09-02 -1.044236 0.119209 -0.173215 1.212112

2019-09-03 1.071804 -0.494929 -2.104569 -0.861849

2019-09-04 0.271860 -1.039575 -0.706771 0.721555

2019-09-05 -1.087401 0.276232 0.567020 -0.424972

2019-09-06 0.524988 -1.478427 0.113648 -0.673690

按值排序：

In [22]: df.sort\_values(by='B')

Out[22]:

A B C D

2019-09-03 -0.861849 -2.104569 -0.494929 1.071804

2019-09-04 0.721555 -0.706771 -1.039575 0.271860

2019-09-01 0.469112 -0.282863 -1.509059 -1.135632

2019-09-02 1.212112 -0.173215 0.119209 -1.044236

2019-09-06 -0.673690 0.113648 -1.478427 0.524988

2019-09-05 -0.424972 0.567020 0.276232 -1.087401

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#选择)选择

**提醒**

虽然用于选择和赋值的标准Python / Numpy表达式非常直观，并且便于交互工作，但是对于生产环境的代码，我们推荐优化的Pandas数据访问方法.at、.iat、.loc和.iloc。

参见索引[索引和选择数据](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#indexing)、[多索引/高级索引](https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#advanced)。

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#获取)获取

选择一个列，产生一个“Series”，相当于“df.A”：

In [23]: df['A']

Out[23]:

2019-09-01 0.469112

2019-09-02 1.212112

2019-09-03 -0.861849

2019-09-04 0.721555

2019-09-05 -0.424972

2019-09-06 -0.673690

Freq: D, Name: A, dtype: float64

通过[ ]选择，对行进行切片：

In [24]: df[0:3]

Out[24]:

A B C D

2019-09-01 0.469112 -0.282863 -1.509059 -1.135632

2019-09-02 1.212112 -0.173215 0.119209 -1.044236

2019-09-03 -0.861849 -2.104569 -0.494929 1.071804

In [25]: df['20190902':'20190904']

Out[25]:

A B C D

2019-09-02 1.212112 -0.173215 0.119209 -1.044236

2019-09-03 -0.861849 -2.104569 -0.494929 1.071804

2019-09-04 0.721555 -0.706771 -1.039575 0.271860

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#按标签选择)按标签选择

在 [Selection by Label](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#indexing-label)查看更多。

通过标签获取一行数据：

In [26]: df.loc[dates[0]]

Out[26]:

A 0.469112

B -0.282863

C -1.509059

D -1.135632

Name: 2019-09-01 00:00:00, dtype: float64

通过标签在多个轴上选择数据：

In [27]: df.loc[:, ['A', 'B']]

Out[27]:

A B

2019-09-01 0.469112 -0.282863

2019-09-02 1.212112 -0.173215

2019-09-03 -0.861849 -2.104569

2019-09-04 0.721555 -0.706771

2019-09-05 -0.424972 0.567020

2019-09-06 -0.673690 0.113648

通过标签同时在两个轴上切片：

In [28]: df.loc['20190902':'20190904', ['A', 'B']]

Out[28]:

A B

2019-09-02 1.212112 -0.173215

2019-09-03 -0.861849 -2.104569

2019-09-04 0.721555 -0.706771

减小返回对象的大小：

In [29]: df.loc['20190902', ['A', 'B']]

Out[29]:

A 1.212112

B -0.173215

Name: 2019-09-02 00:00:00, dtype: float64

获取标量值：

In [30]: df.loc[dates[0], 'A']

Out[30]: 0.46911229990718628

快速访问标量(和上面的方法效果相同)：

In [31]: df.at[dates[0], 'A']

Out[31]: 0.46911229990718628

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#按位置选择)按位置选择

在 [Selection by Position](http://pandas.pydata.org/Pandas-docs/stable/indexing.html#indexing-integer)查看更多

通过传递的整数的位置选择：

In [32]: df.iloc[3]

Out[32]:

A 0.721555

B -0.706771

C -1.039575

D 0.271860

Name: 2019-09-04 00:00:00, dtype: float64

通过整数切片，类似于numpy/Python：

In [33]: df.iloc[3:5, 0:2]

Out[33]:

A B2019-09-04 0.721555 -0.7067712019-09-05 -0.424972 0.567020

通过传递整数的列表按位置切片，类似于numpy/Python：

In [34]: df.iloc[[1, 2, 4], [0, 2]]

Out[34]:

A C

2019-09-02 1.212112 0.119209

2019-09-03 -0.861849 -0.494929

2019-09-05 -0.424972 0.276232

整行切片：

In [35]: df.iloc[1:3, :]

Out[35]:

A B C D

2019-09-02 1.212112 -0.173215 0.119209 -1.044236

2019-09-03 -0.861849 -2.104569 -0.494929 1.071804

整列切片：

In [36]: df.iloc[:, 1:3]

Out[36]:

B C

2019-09-01 -0.282863 -1.509059

2019-09-02 -0.173215 0.119209

2019-09-03 -2.104569 -0.494929

2019-09-04 -0.706771 -1.039575

2019-09-05 0.567020 0.276232

2019-09-06 0.113648 -1.478427

获取具体值：

In [37]: df.iloc[1, 1]

Out[37]: -0.17321464905330858

快速访问标量(等价于之前的方法)：

In [38]: df.iat[1, 1]

Out[38]: -0.17321464905330858

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#布尔索引)布尔索引

使用单个列的值来选择数据：

In [39]: df[df.A > 0]

Out[39]:

A B C D

2019-09-01 0.469112 -0.282863 -1.509059 -1.135632

2019-09-02 1.212112 -0.173215 0.119209 -1.044236

2019-09-04 0.721555 -0.706771 -1.039575 0.271860

从满足布尔条件的DataFrame中选择值：

In [40]: df[df > 0]

Out[40]:

A B C D

2019-09-01 0.469112 NaN NaN NaN

2019-09-02 1.212112 NaN 0.119209 NaN

2019-09-03 NaN NaN NaN 1.071804

2019-09-04 0.721555 NaN NaN 0.271860

2019-09-05 NaN 0.567020 0.276232 NaN

2019-09-06 NaN 0.113648 NaN 0.524988

使用 [isin()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.Series.isin.html" \l "pandas.Series.isin" \t "https://www.pypandas.cn/docs/getting_started/_blank) 方法过滤：

In [41]: df2 = df.copy()

In [42]: df2['E'] = ['one', 'one', 'two', 'three', 'four', 'three']

In [43]: df2

Out[43]:

A B C D E

2019-09-01 0.469112 -0.282863 -1.509059 -1.135632 one

2019-09-02 1.212112 -0.173215 0.119209 -1.044236 one

2019-09-03 -0.861849 -2.104569 -0.494929 1.071804 two

2019-09-04 0.721555 -0.706771 -1.039575 0.271860 three

2019-09-05 -0.424972 0.567020 0.276232 -1.087401 four

2019-09-06 -0.673690 0.113648 -1.478427 0.524988 three

In [44]: df2[df2['E'].isin(['two', 'four'])]

Out[44]:

A B C D E

2019-09-03 -0.861849 -2.104569 -0.494929 1.071804 two

2019-09-05 -0.424972 0.567020 0.276232 -1.087401 four

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#赋值)赋值

添加新列将自动根据索引对齐数据：

In [45]: s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date\_range('20190902', periods=6))

In [46]: s1

Out[46]:

2019-09-02 1

2019-09-03 2

2019-09-04 3

2019-09-05 4

2019-09-06 5

2019-09-07 6

Freq: D, dtype: int64

In [47]: df['F'] = s1

通过标签赋值：

In [48]: df.at[dates[0], 'A'] = 0

通过位置赋值：

In [49]: df.iat[0, 1] = 0

使用NumPy数组赋值：

In [50]: df.loc[:, 'D'] = np.array([5] \* len(df))

前面一系列赋值操作的结果：

In [51]: df

Out[51]:

A B C D F

2019-09-01 0.000000 0.000000 -1.509059 5 NaN

2019-09-02 1.212112 -0.173215 0.119209 5 1.0

2019-09-03 -0.861849 -2.104569 -0.494929 5 2.0

2019-09-04 0.721555 -0.706771 -1.039575 5 3.0

2019-09-05 -0.424972 0.567020 0.276232 5 4.0

2019-09-06 -0.673690 0.113648 -1.478427 5 5.0

带有where条件的赋值操作：

In [52]: df2 = df.copy()

In [53]: df2[df2 > 0] = -df2

In [54]: df2

Out[54]:

A B C D F

2019-09-01 0.000000 0.000000 -1.509059 -5 NaN

2019-09-02 -1.212112 -0.173215 -0.119209 -5 -1.0

2019-09-03 -0.861849 -2.104569 -0.494929 -5 -2.0

2019-09-04 -0.721555 -0.706771 -1.039575 -5 -3.0

2019-09-05 -0.424972 -0.567020 -0.276232 -5 -4.0

2019-09-06 -0.673690 -0.113648 -1.478427 -5 -5.0

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#缺失值)缺失值

Pandas主要使用值np.nan来表示缺失的数据。 默认情况下，它不包含在计算中。 在 [Missing Data section](https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html#missing-data)中查看更多。

重建索引允许你更改/添加/删除指定轴上的索引。 这个操作会返回一个副本(不会更改原来的对象)。

In [55]: df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ['E'])

In [56]: df1.loc[dates[0]:dates[1], 'E'] = 1

In [57]: df1

Out[57]:

A B C D F E

2019-09-01 0.000000 0.000000 -1.509059 5 NaN 1.0

2019-09-02 1.212112 -0.173215 0.119209 5 1.0 1.0

2019-09-03 -0.861849 -2.104569 -0.494929 5 2.0 NaN

2019-09-04 0.721555 -0.706771 -1.039575 5 3.0 NaN

删除任何带有缺失值的行：

In [58]: df1.dropna(how='any')

Out[58]:

A B C D F E

2019-09-02 1.212112 -0.173215 0.119209 5 1.0 1.0

填充缺失值：

In [59]: df1.fillna(value=5)

Out[59]:

A B C D F E

2019-09-01 0.000000 0.000000 -1.509059 5 5.0 1.0

2019-09-02 1.212112 -0.173215 0.119209 5 1.0 1.0

2019-09-03 -0.861849 -2.104569 -0.494929 5 2.0 5.0

2019-09-04 0.721555 -0.706771 -1.039575 5 3.0 5.0

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#操作)操作

在 [Basic section on Binary Ops](https://pandas.pydata.org/pandas-docs/stable/getting_started/basics.html#basics-binop)查看更多。

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#统计)统计

在一些操作中经常会排除缺失值。

进行描述性统计：

In [61]: df.mean()

Out[61]:

A -0.004474

B -0.383981

C -0.687758

D 5.000000

F 3.000000

dtype: float64

在其它轴(行)上进行同样的操作：

In [62]: df.mean(1)

Out[62]:

2019-09-01 0.872735

2019-09-02 1.431621

2019-09-03 0.707731

2019-09-04 1.395042

2019-09-05 1.883656

2019-09-06 1.592306

Freq: D, dtype: float64

使用具有不同维度且需要对齐的对象进行操作。 此外，Pandas会自动沿指定维度进行广播。

In [63]: s = pd.Series([1, 3, 5, np.nan, 6, 8], index=dates).shift(2)

In [64]: s

Out[64]:

2019-09-01 NaN

2019-09-02 NaN

2019-09-03 1.0

2019-09-04 3.0

2019-09-05 5.0

2019-09-06 NaN

Freq: D, dtype: float64

对不同维度的 pandas 对象进行减法操作：

In [65]: df.sub(s, axis='index')

Out[65]:

A B C D F

2019-09-01 NaN NaN NaN NaN NaN

2019-09-02 NaN NaN NaN NaN NaN

2019-09-03 -1.861849 -3.104569 -1.494929 4.0 1.0

2019-09-04 -2.278445 -3.706771 -4.039575 2.0 0.0

2019-09-05 -5.424972 -4.432980 -4.723768 0.0 -1.0

2019-09-06 NaN NaN NaN NaN NaN

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#应用-apply)应用(Apply)

将函数应用于数据：

In [67]: df.apply(lambda x: x.max() - x.min())

Out[67]:

A 2.073961

B 2.671590

C 1.785291

D 0.000000

F 4.000000

dtype: float64

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#直方图化)直方图化

在 [Histogramming and Discretization](https://pandas.pydata.org/pandas-docs/stable/getting_started/basics.html" \l "basics-discretization" \t "https://www.pypandas.cn/docs/getting_started/_blank)查看更多。

In [68]: s = pd.Series(np.random.randint(0, 70, size=10))

In [69]: s

Out[69]:

0

41

22

13

24

65

46

47

68

49

4

dtype: int64

In [70]: s.value\_counts()

Out[70]:

4

56

22

21

1

dtype: int64

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#字符串方法)字符串方法

Series在str属性中配备了一组字符串处理方法，可以轻松地对数组的每个元素进行操作，如下面的代码片段所示。 请注意，str中的模式匹配中默认情况下通常使用[正则表达式](https://docs.python.org/3/library/re.html" \t "https://www.pypandas.cn/docs/getting_started/_blank)。 请参阅[Vectorized String Methods](https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html" \l "text-string-methods" \t "https://www.pypandas.cn/docs/getting_started/_blank)。

In [71]: s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])

In [72]: s.str.lower()

Out[72]: 0 a1 b2 c3 aaba4 baca5 NaN6 caba7 dog8 cat

dtype: object

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#合并-merge)合并(Merge)

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#连接-concat)连接(Concat)

Pandas提供了各种工具，可以轻松地将Series，DataFrame和Panel对象与各种赋值逻辑组合在一起，用于索引和连接/合并类型操作时的关系代数功能。

在 [Merging section](https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html#merging)查看更多。

使用 [concat()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html" \l "pandas.concat" \t "https://www.pypandas.cn/docs/getting_started/_blank)连接Pandas对象：

In [73]: df = pd.DataFrame(np.random.randn(10, 4))

In [74]: df

Out[74]:

0 1 2 3

0 -0.548702 1.467327 -1.015962 -0.483075

1 1.637550 -1.217659 -0.291519 -1.745505

2 -0.263952 0.991460 -0.919069 0.266046

3 -0.709661 1.669052 1.037882 -1.705775

4 -0.919854 -0.042379 1.247642 -0.009920

5 0.290213 0.495767 0.362949 1.548106

6 -1.131345 -0.089329 0.337863 -0.945867

7 -0.932132 1.956030 0.017587 -0.016692

8 -0.575247 0.254161 -1.143704 0.215897

9 1.193555 -0.077118 -0.408530 -0.862495

# break it into pieces

In [75]: pieces = [df[:3], df[3:7], df[7:]]

In [76]: pd.concat(pieces)

Out[76]:

0 1 2 3

0 -0.548702 1.467327 -1.015962 -0.483075

1 1.637550 -1.217659 -0.291519 -1.745505

2 -0.263952 0.991460 -0.919069 0.266046

3 -0.709661 1.669052 1.037882 -1.705775

4 -0.919854 -0.042379 1.247642 -0.009920

5 0.290213 0.495767 0.362949 1.548106

6 -1.131345 -0.089329 0.337863 -0.945867

7 -0.932132 1.956030 0.017587 -0.016692

8 -0.575247 0.254161 -1.143704 0.215897

9 1.193555 -0.077118 -0.408530 -0.862495

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#join)Join

SQL风格的合并。 请参阅[数据库风格连接](https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html#merging-join)部分。

In [77]: left = pd.DataFrame({'key': ['foo', 'foo'], 'lval': [1, 2]})

In [78]: right = pd.DataFrame({'key': ['foo', 'foo'], 'rval': [4, 5]})

In [79]: left

Out[79]:

key lval

0 foo 1

1 foo 2

In [80]: right

Out[80]:

key rval

0 foo 4

1 foo 5

In [81]: pd.merge(left, right, on='key')

Out[81]:

key lval rval0 foo 1 41 foo 1 52 foo 2 43 foo 2 5

另一个例子是：

In [82]: left = pd.DataFrame({'key': ['foo', 'bar'], 'lval': [1, 2]})

In [83]: right = pd.DataFrame({'key': ['foo', 'bar'], 'rval': [4, 5]})

In [84]: left

Out[84]:

key lval

0 foo 1

1 bar 2

In [85]: right

Out[85]:

key rval

0 foo 4

1 bar 5

In [86]: pd.merge(left, right, on='key')

Out[86]:

key lval rval

0 foo 1 4

1 bar 2 5

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#追加-append)追加(Append)

给dataframe追加一行。 请参阅 [Appending](https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html#merging-concatenation) 部分。

In [87]: df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])

In [88]: df

Out[88]:

A B C D

0 1.346061 1.511763 1.627081 -0.990582

1 -0.441652 1.211526 0.268520 0.024580

2 -1.577585 0.396823 -0.105381 -0.532532

3 1.453749 1.208843 -0.080952 -0.264610

4 -0.727965 -0.589346 0.339969 -0.693205

5 -0.339355 0.593616 0.884345 1.591431

6 0.141809 0.220390 0.435589 0.192451

7 -0.096701 0.803351 1.715071 -0.708758

In [89]: s = df.iloc[3]

In [90]: df.append(s, ignore\_index=True)

Out[90]:

A B C D

0 1.346061 1.511763 1.627081 -0.990582

1 -0.441652 1.211526 0.268520 0.024580

2 -1.577585 0.396823 -0.105381 -0.532532

3 1.453749 1.208843 -0.080952 -0.264610

4 -0.727965 -0.589346 0.339969 -0.693205

5 -0.339355 0.593616 0.884345 1.591431

6 0.141809 0.220390 0.435589 0.192451

7 -0.096701 0.803351 1.715071 -0.708758

8 1.453749 1.208843 -0.080952 -0.264610

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#分组-grouping)分组(Grouping)

我们所说的“group by“是指涉及下列一项或多项步骤的程序：

* **分割**：根据一些标准将数据分解成组
* **应用**：将函数独立地应用于每个组
* **组合**：将结果组合成数据结构

请参阅 [Grouping 部分](https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html#groupby)。

In [91]: df = pd.DataFrame({'A': ['foo', 'bar', 'foo', 'bar',

'foo', 'bar', 'foo', 'foo'],

'B': ['one', 'one', 'two', 'three',

'two', 'two', 'one', 'three'],

'C': np.random.randn(8),

'D': np.random.randn(8)})

In [92]: df

Out[92]:

A B C D

0 foo one -1.202872 -0.055224

1 bar one -1.814470 2.395985

2 foo two 1.018601 1.552825

3 bar three -0.595447 0.166599

4 foo two 1.395433 0.047609

5 bar two -0.392670 -0.136473

6 foo one 0.007207 -0.561757

7 foo three 1.928123 -1.623033

分组，然后将[sum()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sum.html" \l "pandas.DataFrame.sum" \t "https://www.pypandas.cn/docs/getting_started/_blank)函数应用于分组结果：

In [93]: df.groupby('A').sum()

Out[93]:

C D

A

bar -2.802588 2.42611

foo 3.146492 -0.63958

按多列分组形成层次索引，同样，我们可以应用sum函数：

In [94]: df.groupby(['A', 'B']).sum()

Out[94]:

C D

A B

bar one -1.814470 2.395985

three -0.595447 0.166599

two -0.392670 -0.136473

foo one -1.195665 -0.616981

three 1.928123 -1.623033

two 2.414034 1.600434

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#重塑-reshaping)重塑(Reshaping)

请参阅 [Hierarchical Indexing](https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#advanced-hierarchical) 和 [Reshaping](https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html#reshaping-stacking)部分。

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#堆叠-stack)堆叠(Stack)

In [95]: tuples = list(zip(\*[['bar', 'bar', 'baz', 'baz',

'foo', 'foo', 'qux', 'qux'],

['one', 'two', 'one', 'two',

'one', 'two', 'one', 'two']]))

In [96]: index = pd.MultiIndex.from\_tuples(tuples, names=['first', 'second'])

In [97]: df = pd.DataFrame(np.random.randn(8, 2), index=index, columns=['A', 'B'])

In [98]: df2 = df[:4]

In [99]: df2

Out[99]:

A B

first second

bar one 0.029399 -0.542108

two 0.282696 -0.087302

baz one -1.575170 1.771208

two 0.816482 1.100230

[stack()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.stack.html#pandas.DataFrame.stack)方法压缩DataFrame的列：

In [100]: stacked = df2.stack()

In [101]: stacked

Out[101]:

first second

B -0.542108

two A 0.282696

B -0.087302

baz one A -1.575170

B 1.771208

two A 0.816482

B 1.100230

dtype: float64

“压缩”后的DataFrame或Series(具有MultiIndex作为索引)， [stack()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.stack.html#pandas.DataFrame.stack) 的逆操作是[unstack()](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.unstack.html" \l "pandas.DataFrame.unstack" \t "https://www.pypandas.cn/docs/getting_started/_blank)，默认情况下取消最后压缩的那个级别：

In [102]: stacked.unstack()

Out[102]:

A B

first second

bar one 0.029399 -0.542108

two 0.282696 -0.087302

baz one -1.575170 1.771208

two 0.816482 1.100230

In [103]: stacked.unstack(1)

Out[103]:

second one two

first

bar A 0.029399 0.282696

B -0.542108 -0.087302

baz A -1.575170 0.816482

B 1.771208 1.100230

In [104]: stacked.unstack(0)

Out[104]:

first bar baz

second

one A 0.029399 -1.575170

B -0.542108 1.771208

two A 0.282696 0.816482

B -0.087302 1.100230

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#数据透视表-pivottables)数据透视表(PivotTables)

请参阅[Pivot Tables](https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html" \l "reshaping-pivot" \t "https://www.pypandas.cn/docs/getting_started/_blank)部分。

In [105]: df = pd.DataFrame({'A': ['one', 'one', 'two', 'three'] \* 3,

'B': ['A', 'B', 'C'] \* 4,

'C': ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] \* 2,

'D': np.random.randn(12),

'E': np.random.randn(12)})

In [106]: df

Out[106]:

A B C D E

0 one A foo 1.418757 -0.179666

1 one B foo -1.879024 1.291836

2 two C foo 0.536826 -0.009614

3 three A bar 1.006160 0.392149

4 one B bar -0.029716 0.264599

5 one C bar -1.146178 -0.057409

6 two A foo 0.100900 -1.425638

7 three B foo -1.035018 1.024098

8 one C foo 0.314665 -0.106062

9 one A bar -0.773723 1.824375

10 two B bar -1.170653 0.595974

11 three C bar 0.648740 1.167115

我们可以非常轻松地从这些数据生成数据透视表：

In [107]: pd.pivot\_table(df, values='D', index=['A', 'B'], columns=['C'])

Out[107]:

C bar foo

A B

one A -0.773723 1.418757

B -0.029716 -1.879024

C -1.146178 0.314665

three A 1.006160 NaN

B NaN -1.035018

C 0.648740 NaN

two A NaN 0.100900

B -1.170653 NaN

C NaN 0.536826

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#时间序列-timeseries)时间序列(TimeSeries)

Pandas具有简单，强大且高效的功能，用于在频率转换期间执行重采样操作(例如，将第二数据转换为5分钟数据)。 这在财务应用程序中非常常见，但不仅限于此。请参阅[时间序列部分](https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#timeseries)部分。

In [108]: rng = pd.date\_range('1/1/2012', periods=100, freq='S')

In [109]: ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)

In [110]: ts.resample('5Min').sum()

Out[110]: 2012-01-01 25083

Freq: 5T, dtype: int64

时区代表：

In [111]: rng = pd.date\_range('3/6/2012 00:00', periods=5, freq='D')

In [112]: ts = pd.Series(np.random.randn(len(rng)), rng)

In [113]: ts

Out[113]:

2012-03-06 0.464000

2012-03-07 0.227371

2012-03-08 -0.496922

2012-03-09 0.306389

2012-03-10 -2.290613

Freq: D, dtype: float64

In [114]: ts\_utc = ts.tz\_localize('UTC')

In [115]: ts\_utc

Out[115]: 2012-03-06 00:00:00+00:00 0.4640002012-03-07 00:00:00+00:00 0.2273712012-03-08 00:00:00+00:00 -0.4969222012-03-09 00:00:00+00:00 0.3063892012-03-10 00:00:00+00:00 -2.290613

Freq: D, dtype: float64

转换为另一个时区：

In [116]: ts\_utc.tz\_convert('US/Eastern')

Out[116]:

2012-03-05 19:00:00-05:00 0.464000

2012-03-06 19:00:00-05:00 0.227371

2012-03-07 19:00:00-05:00 -0.496922

2012-03-08 19:00:00-05:00 0.306389

2012-03-09 19:00:00-05:00 -2.290613

Freq: D, dtype: float64

在时间跨度表示之间转换：

In [117]: rng = pd.date\_range('1/1/2012', periods=5, freq='M')

In [118]: ts = pd.Series(np.random.randn(len(rng)), index=rng)

In [119]: ts

Out[119]:

2012-01-31 -1.134623

2012-02-29 -1.561819

2012-03-31 -0.260838

2012-04-30 0.281957

2012-05-31 1.523962

Freq: M, dtype: float64

In [120]: ps = ts.to\_period()

In [121]: ps

Out[121]:

2012-01 -1.134623

2012-02 -1.561819

2012-03 -0.260838

2012-04 0.281957

2012-05 1.523962

Freq: M, dtype: float64

In [122]: ps.to\_timestamp()

Out[122]:

2012-01-01 -1.134623

2012-02-01 -1.561819

2012-03-01 -0.260838

2012-04-01 0.281957

2012-05-01 1.523962

Freq: MS, dtype: float64

## [#](https://www.pypandas.cn/docs/getting_started/10min.html#数据输入-输出)数据输入/输出

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#csv)CSV

[写入CSV文件](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html#io-store-in-csv)。

In [143]: df.to\_csv('foo.csv')

从CSV文件读数据：

In [144]: pd.read\_csv('foo.csv')

Out[144]:

Unnamed: 0 A B C D0 2000-01-01 0.266457 -0.399641 -0.219582 1.1868601 2000-01-02 -1.170732 -0.345873 1.653061 -0.2829532 2000-01-03 -1.734933 0.530468 2.060811 -0.5155363 2000-01-04 -1.555121 1.452620 0.239859 -1.1568964 2000-01-05 0.578117 0.511371 0.103552 -2.4282025 2000-01-06 0.478344 0.449933 -0.741620 -1.9624096 2000-01-07 1.235339 -0.091757 -1.543861 -1.084753.. ... ... ... ... ...993 2002-09-20 -10.628548 -9.153563 -7.883146 28.313940994 2002-09-21 -10.390377 -8.727491 -6.399645 30.914107995 2002-09-22 -8.985362 -8.485624 -4.669462 31.367740996 2002-09-23 -9.558560 -8.781216 -4.499815 30.518439997 2002-09-24 -9.902058 -9.340490 -4.386639 30.105593998 2002-09-25 -10.216020 -9.480682 -3.933802 29.758560999 2002-09-26 -11.856774 -10.671012 -3.216025 29.369368

[1000 rows x 5 columns]

### [#](https://www.pypandas.cn/docs/getting_started/10min.html#excel)Excel

请参阅[MS Excel](https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html" \l "io-excel" \t "https://www.pypandas.cn/docs/getting_started/_blank)。

写入excel文件：

In [147]: df.to\_excel('foo.xlsx', sheet\_name='Sheet1')

从Excel文件读取数据：

In [148]: pd.read\_excel('foo.xlsx', 'Sheet1', index\_col=None, na\_values=['NA'])

Out[148]:

Unnamed: 0 A B C D0 2000-01-01 0.266457 -0.399641 -0.219582 1.1868601 2000-01-02 -1.170732 -0.345873 1.653061 -0.2829532 2000-01-03 -1.734933 0.530468 2.060811 -0.5155363 2000-01-04 -1.555121 1.452620 0.239859 -1.1568964 2000-01-05 0.578117 0.511371 0.103552 -2.4282025 2000-01-06 0.478344 0.449933 -0.741620 -1.9624096 2000-01-07 1.235339 -0.091757 -1.543861 -1.084753.. ... ... ... ... ...993 2002-09-20 -10.628548 -9.153563 -7.883146 28.313940994 2002-09-21 -10.390377 -8.727491 -6.399645 30.914107995 2002-09-22 -8.985362 -8.485624 -4.669462 31.367740996 2002-09-23 -9.558560 -8.781216 -4.499815 30.518439997 2002-09-24 -9.902058 -9.340490 -4.386639 30.105593998 2002-09-25 -10.216020 -9.480682 -3.933802 29.758560999 2002-09-26 -11.856774 -10.671012 -3.216025 29.369368

[1000 rows x 5 columns]