Python 数据分析与应用

第六章：Pandas 统计分析及数据预处理

丁烨
dingye@dgut.edu.cn
计算机科学与技术学院

2023-11-08

東莞理工學院
DONGGUAN UNIVERSITY OF TECHNOLOGY

目录

Pandas 简介

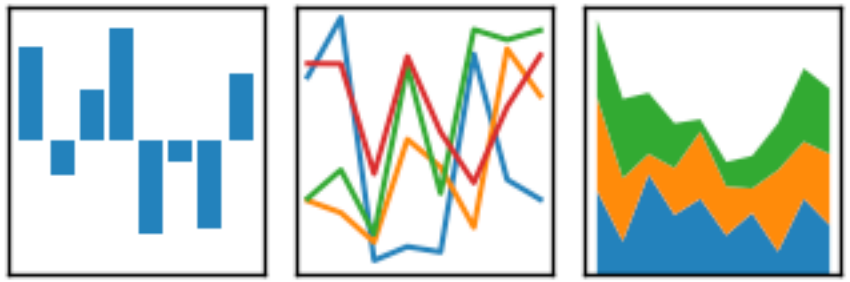
概述

- Pandas 是 Python 的一个扩展程序库
- 官方网站：https://pandas.pydata.org/
- 源代码：https://github.com/pandas-dev/pandas

- Pandas 是一个开源免费的 Python 扩展程序库
- 主要提供高性能且易用的数据结构及函数集用于数据分析方面的应用
- Pandas 是基于 NumPy 实现的，可以视为 NumPy 的高级版

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Pandas 简介

安装

- 使用 pip 安装 Pandas：
- `pip3 install --user -U pandas pytest hypothesis`

- 如果安装不成功，可尝试使用 apt 安装：
- `sudo apt install python3-pandas`

测试

- 测试 Matplotlib 是否安装成功：
- python3
- import pandas as pd

```
>>> pd.test()
running: pytest --skip-slow --skip-network C:\Users\TP\Anaconda3\envs\py36\lib\site-packages\pandas
============================= test session starts =============================
platform win32 -- Python 3.6.2, pytest-3.6.0, py-1.4.34, pluggy-0.4.0
rootdir: C:\Users\TP\Documents\Python\pandasdev\pandas, inifile: setup.cfg
collected 12145 items / 3 skipped

..........................................................................S.....
.......S................................................................................
................................................................................

==================== 12130 passed, 12 skipped in 368.339 seconds ====================
```

目录

Pandas 基本用法

基本概念

- Pandas 常被用于处理以下类型的数据：
- 表格数据，尤其是数据类型不同的表格数据
- 数据库数据，常与 SQLAlchemy / Django 等 ORM 模组混用
- 有序或无序的时间数据，且对数据没有采样率要求
- 矩阵数据（无论数据类型是否相同）
- 统计类数据，且可对数据加标签

Pandas 基本用法

基本数据结构

- Pandas 有两种基本的数据结构：
- Series：一维、可带标签的同构数组
- DataFrame：
- 二维、可带标签、可变大小的矩阵（表格）型数据
- 每一列可以异构，但同一列的数据必须同构
- Pandas 不宜处理高维数据

基本数据结构

❖ 导入 Pandas：

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

In [2]: import pandas as pd
```

❖ 通过传递一个列表来创建一个 Series，并由 Pandas 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
```

基本数据结构

- 通过传递一个 NumPy 数组来创建一个 DataFrame，采用 datetime 类型的索引，并对每一列设定标签

```
In [5]: dates = pd.date_range('20130101', periods=6)
In [6]: dates
Out[6]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              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
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

基本数据结构

❖ 通过传递字典（dict）来创建一个 DataFrame，列标签将被自动设定为字典的键

```
In [9]: df2 = pd.DataFrame({'A': 1.,
   ...:                     'B': pd.Timestamp('20130102'),
   ...:                     '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 2013-01-02  1.0  3   test  foo
1  1.0 2013-01-02  1.0  3  train  foo
2  1.0 2013-01-02  1.0  3   test  foo
3  1.0 2013-01-02  1.0  3  train  foo
```

基本数据结构

❖ 创建好的 DataFrame 每一列可以被设定为不同的数据结构和标签

```
In [11]: df2.dtypes
Out[11]:
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

❖ PyCharm 和 IPython 中，可以（通过按下 Tab 键）自动补全后续代码

```
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
```

数据访问

❖ Pandas 可以很方便的查看 DataFrame 的前几行和后几行

```
In [13]: df.head()
Out[13]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401


In [14]: df.tail(3)
Out[14]:
                   A         B         C         D
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
```

数据访问

❖ 查看 DataFrame 的索引和标签

```
In [15]: df.index
Out[15]:
DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
               '2013-01-05', '2013-01-06'],
              dtype='datetime64[ns]', freq='D')


In [16]: df.columns
Out[16]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

数据访问

- DataFrame.to_numpy() 提供了一个将 DataFrame 转换为 NumPy 的函数

- 注意：
- Pandas 和 NumPy 有结构上的本质区别：Pandas 允许 DataFrame 中的每一列数据设置为不同的数据结构、而 NumPy 要求整个数组同构
- 当使用 DataFrame.to_numpy() 时，Pandas 将寻找 NumPy 中最近的能表达原始数据的 dtype
- 如果 dtype 不同，Pandas 将对每一个数据元素调用格式转换函数
- 因此，当 DataFrame 元素较多的时候，此函数将消耗大量时间

数据访问

❖ 在上面的例子里，df 的所有元素都是 float 类型，因此无需进行类型转换

```
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 ]])
```

数据访问

❖ 在上面的例子里，df2 的元素具有不同的数据类型，因此下面的代码将消耗更多时间

```
In [18]: df2.to_numpy()
Out[18]:
array([[1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo']], dtype=object)
```

数据访问

❖ 使用 describe() 函数可以把数据的统计信息展示出来：

```
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
```

数据访问

❖ 使用 T 成员变量可以转置（Transpose）一个矩阵

```
In [20]: df.T
Out[20]:
   2013-01-01  2013-01-02  2013-01-03  2013-01-04  2013-01-05  2013-01-06
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
```

❖ 使用 sort_index() 函数可以对列标签进行排序：

```
In [21]: df.sort_index(axis=1, ascending=False)
Out[21]:
                   D         C         B         A
2013-01-01 -1.135632 -1.509059 -0.282863  0.469112
2013-01-02 -1.044236  0.119209 -0.173215  1.212112
2013-01-03  1.071804 -0.494929 -2.104569 -0.861849
2013-01-04  0.271860 -1.039575 -0.706771  0.721555
2013-01-05 -1.087401  0.276232  0.567020 -0.424972
2013-01-06  0.524988 -1.478427  0.113648 -0.673690
```

数据访问

❖ 使用 sort_index() 函数可以对指定列的数据进行排序：

```
In [22]: df.sort_values(by='B')
Out[22]:
                   A         B         C         D
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
```

选取数据

- 类似 NumPy，Pandas 亦提供了多种多样的选择数据的函数和成员变量
- 选择一列中的数据：

```
In [23]: df['A']
Out[23]:
2013-01-01    0.469112
2013-01-02    1.212112
2013-01-03   -0.861849
2013-01-04    0.721555
2013-01-05   -0.424972
2013-01-06   -0.673690
Freq: D, Name: A, dtype: float64
```

- 返回的数据是 Series 类型
- 也可以通过 df.A 完成类似的操作

选取数据

❖ 类似 NumPy，Pandas 也可以使用中括号（[]）来选取及切割数据

```
In [24]: df[0:3]
Out[24]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804


In [25]: df['20130102':'20130104']
Out[25]:
                   A         B         C         D
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
```

选取数据

❖ 使用行索引截取一行数据：

```
In [26]: df.loc[dates[0]]
Out[26]:
A    0.469112
B   -0.282863
C   -1.509059
D   -1.135632
Name: 2013-01-01 00:00:00, dtype: float64
```

选取数据

❖ 使用列索引截取多列的数据：

```
In [27]: df.loc[:, ['A', 'B']]
Out[27]:
                   A         B
2013-01-01  0.469112 -0.282863
2013-01-02  1.212112 -0.173215
2013-01-03 -0.861849 -2.104569
2013-01-04  0.721555 -0.706771
2013-01-05 -0.424972  0.567020
2013-01-06 -0.673690  0.113648
```

选取数据

❖ 同时使用行索引和列索引截取数据：

```
In [28]: df.loc['20130102':'20130104', ['A', 'B']]
Out[28]:
                   A         B
2013-01-02  1.212112 -0.173215
2013-01-03 -0.861849 -2.104569
2013-01-04  0.721555 -0.706771
```

选取数据

❖ 同时使用行索引和列索引截取一行数据：

```
In [29]: df.loc['20130102', ['A', 'B']]
Out[29]:
A    1.212112
B   -0.173215
Name: 2013-01-02 00:00:00, dtype: float64
```

选取数据

❖ 同时使用行索引和列索引截取一个元素：

```
In [30]: df.loc[dates[0], 'A']
Out[30]: 0.46911229990718628
```

```
In [31]: df.at[dates[0], 'A']
Out[31]: 0.46911229990718628
```

选取数据

❖ 使用数值索引截取一行数据：

```
In [32]: df.iloc[3]
Out[32]:
A    0.721555
B   -0.706771
C   -1.039575
D    0.271860
Name: 2013-01-04 00:00:00, dtype: float64
```

选取数据

❖ 使用数值索引截取行列数据：

```
In [33]: df.iloc[3:5, 0:2]
Out[33]:
                   A         B
2013-01-04  0.721555 -0.706771
2013-01-05 -0.424972  0.567020
```

选取数据

❖ 使用 NumPy 的数组索引机制截取数据：

```
In [34]: df.iloc[[1, 2, 4], [0, 2]]
Out[34]:
                   A         C
2013-01-02  1.212112  0.119209
2013-01-03 -0.861849 -0.494929
2013-01-05 -0.424972  0.276232
```

选取数据

❖ 仅截取行数据：

```
In [35]: df.iloc[1:3, :]
Out[35]:
                   A         B         C         D
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
```

选取数据

❖ 仅截取列数据：

```
In [36]: df.iloc[:, 1:3]
Out[36]:
                   B         C
2013-01-01 -0.282863 -1.509059
2013-01-02 -0.173215  0.119209
2013-01-03 -2.104569 -0.494929
2013-01-04 -0.706771 -1.039575
2013-01-05  0.567020  0.276232
2013-01-06  0.113648 -1.478427
```

选取数据

❖ 仅截取列数据：

```
In [36]: df.iloc[:, 1:3]
Out[36]:
                   B         C
2013-01-01 -0.282863 -1.509059
2013-01-02 -0.173215  0.119209
2013-01-03 -2.104569 -0.494929
2013-01-04 -0.706771 -1.039575
2013-01-05  0.567020  0.276232
2013-01-06  0.113648 -1.478427
```

选取数据

❖ 使用数值索引截取一个元素：

```
In [37]: df.iloc[1, 1]
Out[37]: -0.17321464905330858
```

```
In [38]: df.iat[1, 1]
Out[38]: -0.17321464905330858
```

布尔值索引

❖ 使用一列数据的布尔值索引截取整个数据集

```
In [39]: df[df.A > 0]
Out[39]:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
```

布尔值索引

❖ 通过布尔值索引选取整个数据集的数据

```
In [40]: df[df > 0]
Out[40]:
                   A         B         C         D
2013-01-01  0.469112       NaN       NaN       NaN
2013-01-02  1.212112       NaN  0.119209       NaN
2013-01-03       NaN       NaN       NaN  1.071804
2013-01-04  0.721555       NaN       NaN  0.271860
2013-01-05       NaN  0.567020  0.276232       NaN
2013-01-06       NaN  0.113648       NaN  0.524988
```

布尔值索引

❖ 使用 isin() 函数过滤数据：

```
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
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632    one
2013-01-02  1.212112 -0.173215  0.119209 -1.044236    one
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804    two
2013-01-04  0.721555 -0.706771 -1.039575  0.271860  three
2013-01-05 -0.424972  0.567020  0.276232 -1.087401   four
2013-01-06 -0.673690  0.113648 -1.478427  0.524988  three
In [44]: df2[df2['E'].isin(['two', 'four'])]
Out[44]:
                   A         B         C         D     E
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804   two
2013-01-05 -0.424972  0.567020  0.276232 -1.087401  four
```

赋值

❖ 通过自动赋值机制将两条数据连接到一起：

```
In [45]: s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range('20130102', periods=6))

In [46]: s1
Out[46]:
2013-01-02    1
2013-01-03    2
2013-01-04    3
2013-01-05    4
2013-01-06    5
2013-01-07    6
Freq: D, dtype: int64

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

赋值

❖ 通过列标签给元素赋值：

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

赋值

❖ 通过数值索引给元素赋值：

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

赋值

❖ 通过 NumPy 数组给一行或一列赋值：

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

赋值

❖ 通过简单的元素运算赋值：

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

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

In [54]: df2
Out[54]:
                   A         B         C  D    F
2013-01-01  0.000000  0.000000 -1.509059 -5  NaN
2013-01-02 -1.212112 -0.173215 -0.119209 -5 -1.0
2013-01-03 -0.861849 -2.104569 -0.494929 -5 -2.0
2013-01-04 -0.721555 -0.706771 -1.039575 -5 -3.0
2013-01-05 -0.424972 -0.567020 -0.276232 -5 -4.0
2013-01-06 -0.673690 -0.113648 -1.478427 -5 -5.0
```

处理缺失值

- Pandas 会将缺失值（或错误的值）设定为 Numpy 的 np.nan 类型
- 使用 reindex 函数可以方便的新增、修改、删除索引

```
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
2013-01-01  0.000000  0.000000 -1.509059  5  NaN  1.0
2013-01-02  1.212112 -0.173215  0.119209  5  1.0  1.0
2013-01-03 -0.861849 -2.104569 -0.494929  5  2.0  NaN
2013-01-04  0.721555 -0.706771 -1.039575  5  3.0  NaN
```

处理缺失值

❖ dropna 函数可以迅速抛弃含有 NaN 的行

```
In [58]: df1.dropna(how='any')
Out[58]:
                   A         B         C  D    F    E
2013-01-02  1.212112 -0.173215  0.119209  5  1.0  1.0
```

处理缺失值

❖ 也可以使用 fillna 来迅速替换所有含有缺失数据的行：

```
In [59]: df1.fillna(value=5)
Out[59]:
                   A         B         C  D    F    E
2013-01-01  0.000000  0.000000 -1.509059  5  5.0  1.0
2013-01-02  1.212112 -0.173215  0.119209  5  1.0  1.0
2013-01-03 -0.861849 -2.104569 -0.494929  5  2.0  5.0
2013-01-04  0.721555 -0.706771 -1.039575  5  3.0  5.0
```

处理缺失值

❖ Pandas 也允许迅速获取所有 NaN 的布尔值索引：

```
In [60]: pd.isna(df1)
Out[60]:
                A      B      C      D      F      E
2013-01-01  False  False  False  False   True  False
2013-01-02  False  False  False  False  False  False
2013-01-03  False  False  False  False  False   True
2013-01-04  False  False  False  False  False   True
```

数据操作

❖ Pandas 提供多种数据操作的函数，例如获取列平均值：

```
In [61]: df.mean()
Out[61]:
A   -0.004474
B   -0.383981
C   -0.687758
D    5.000000
F    3.000000
dtype: float64
```

❖ 获取行平均值：

```
In [62]: df.mean(1)
Out[62]:
2013-01-01    0.872735
2013-01-02    1.431621
2013-01-03    0.707731
2013-01-04    1.395042
2013-01-05    1.883656
2013-01-06    1.592306
Freq: D, dtype: float64
```

❖ Pandas 允许使用 shift 来平移数据集，平移后的默认值为 NaN：

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

In [64]: s
Out[64]:
2013-01-01    NaN
2013-01-02    NaN
2013-01-03    1.0
2013-01-04    3.0
2013-01-05    5.0
2013-01-06    NaN
Freq: D, dtype: float64
```

❖ 通过 sub 函数可以将两个数据集相减，没有数据的地方会产生 NaN：

```
In [65]: df.sub(s, axis='index')
Out[65]:
                   A         B         C    D    F
2013-01-01       NaN       NaN       NaN  NaN  NaN
2013-01-02       NaN       NaN       NaN  NaN  NaN
2013-01-03 -1.861849 -3.104569 -1.494929  4.0  1.0
2013-01-04 -2.278445 -3.706771 -4.039575  2.0  0.0
2013-01-05 -5.424972 -4.432980 -4.723768  0.0 -1.0
2013-01-06       NaN       NaN       NaN  NaN  NaN
```

❖ 使用 apply 函数可以执行 NumPy 的数值操作函数：

```
In [66]: df.apply(np.cumsum)
Out[66]:
                   A         B         C   D     F
2013-01-01  0.000000  0.000000 -1.509059   5   NaN
2013-01-02  1.212112 -0.173215 -1.389850  10   1.0
2013-01-03  0.350263 -2.277784 -1.884779  15   3.0
2013-01-04  1.071818 -2.984555 -2.924354  20   6.0
2013-01-05  0.646846 -2.417535 -2.648122  25  10.0
2013-01-06 -0.026844 -2.303886 -4.126549  30  15.0
```

❖ 使用 apply 函数可以执行 NumPy 的数值操作函数：

```
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
```

❖ 离散化及直方图：

```
In [68]: s = pd.Series(np.random.randint(0, 7, size=10))

In [69]: s
Out[69]:
0    4
1    2
2    1
3    2
4    6
5    4
6    4
7    6
8    4
9    4
dtype: int64
```

❖ 离散化及直方图：

```
In [70]: s.value_counts()
Out[70]:
4    5
6    2
2    2
1    1
dtype: int64
```

❖ 对于字符串数据，Pandas 可以直接在字符串上做各种操作：

```
In [71]: s = pd.Series(['A', 'B', 'C', 'Aaba', 'Baca', np.nan, 'CABA', 'dog', 'cat'])
In [72]: s.str.lower()
Out[72]:
0       a
1       b
2       c
3    aaba
4    baca
5     NaN
6    caba
7     dog
8     cat
dtype: object
```

数据整合

❖ Pandas 提供了多种分割数据、整合数据的函数：

```
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
```

数据整合

❖ Pandas 提供了多种分割数据、整合数据的函数：

```
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
```

❖ SQL 风格的数据整合（Join）方法的例子：

```
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
```

❖ SQL 风格的数据整合（Join）方法的例子：

```
In [86]: pd.merge(left, right, on='key')
Out[86]:
   key  lval  rval
0  foo     1     4
1  bar     2     5
```

数据整合

❖ 另一个 SQL 风格的数据整合（Join）方法，在这个例子中存在重名的键：

```
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
```

❖ 由于含有重名的键，使用 merge 将产生笛卡尔积：

```
In [81]: pd.merge(left, right, on='key')
Out[81]:
   key  lval  rval
0  foo     1     4
1  foo     1     5
2  foo     2     4
3  foo     2     5
```

❖ 向已存在的 DataFrame 实例添加数据：

```
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
```

❖ 向已存在的 DataFrame 实例添加数据：

```
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
```

数据组合

❖ 类似于 SQL，Pandas 的分组（Group By）操作通常包括以下几个步骤：

1. 将数据根据特定的条件分为几个不同的组
2. 对每组数据做数学操作（例如 Count：清点数量）
3. 将结果输出成一个完整的数据结构

```
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)})
    ....:
```

数据组合

```
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
```

数据组合

❖ 例如，分组后进行求和操作：

```
In [93]: df.groupby('A').sum()
Out[93]:
            C        D
A
bar -2.802588  2.42611
foo  3.146492 -0.63958
```

数据组合

❖ 将多列组合后将形成多级索引，多级索引下的数据仍然可以使用求和函数：

```
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
```

转换数据形式

❖ 下面是一个转换数据格式的例子：

```
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
```

转换数据形式

❖ 用 stack() 函数可以对数据进行列的堆叠：

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

In [101]: stacked
Out[101]:
first  second
bar    one     A    0.029399
               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
```

转换数据形式

❖ 类似的，我们可以使用 unstack() 函数来解开数据的堆叠：

```
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
```

转换数据形式

```
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
```

Pandas 基本用法

透视表

- Pandas 允许快速根据某些规则生成表格
- 这种表格被称为透视表（Pivot Tables）

```
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
```

透视表

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

时序数据

- Pandas 提供了非常强大的时序数据处理函数
- 时序数据的处理在 NumPy 中较为不方便，Pandas 弥补了这方面的问题

```
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
```

时序数据

❖ 含有时区信息的时序数据：

```
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
```

时序数据

```
In [114]: ts_utc = ts.tz_localize('UTC')

In [115]: ts_utc
Out[115]:
2012-03-06 00:00:00+00:00    0.464000
2012-03-07 00:00:00+00:00    0.227371
2012-03-08 00:00:00+00:00   -0.496922
2012-03-09 00:00:00+00:00    0.306389
2012-03-10 00:00:00+00:00   -2.290613
Freq: D, dtype: float64
```

❖ 转换成为另一时区：

```
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()
```

时序数据

```
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
```

时序数据

- 时间段和时间点做转换时可以使用一些转换函数
- 下面的例子中我们生成了一系列季度数据，并将这些数据像每月 1 号的九点钟对齐：

```
In [123]: prng = pd.period_range('1990Q1', '2000Q4', freq='Q-NOV')

In [124]: ts = pd.Series(np.random.randn(len(prng)), prng)

In [125]: ts.index = (prng.asfreq('M', 'e') + 1).asfreq('H', 's') + 9

In [126]: ts.head()
Out[126]:
1990-03-01 09:00   -0.902937
1990-06-01 09:00    0.068159
1990-09-01 09:00   -0.057873
1990-12-01 09:00   -0.368204
1991-03-01 09:00   -1.144073
Freq: H, dtype: float64
```

类别数据

❖ Pandas 的 DataFrame 允许引入类别数据：

```
In [127]: df = pd.DataFrame({"id": [1, 2, 3, 4, 5, 6],
   .....:                    "raw_grade": ['a', 'b', 'b', 'a', 'a', 'e']})
   .....:
```

类别数据

❖ 将原始数值分类数据转为类别数据：

```
In [128]: df["grade"] = df["raw_grade"].astype("category")

In [129]: df["grade"]
Out[129]:
0    a
1    b
2    b
3    a
4    a
5    e
Name: grade, dtype: category
Categories (3, object): [a, b, e]
```

类别数据

❖ 使用更加容易理解的分类名称：

```
In [130]: df["grade"].cat.categories = ["very good", "good", "very bad"]
```

类别数据

❖ Pandas的排序是按文字倒序排序的，不是按语义排序：

```
In [133]: df.sort_values(by="grade")
Out[133]:
   id raw_grade      grade
5   6         e   very bad
1   2         b       good
2   3         b       good
0   1         a  very good
3   4         a  very good
4   5         a  very good
```

类别数据

❖ 组合类别数据的时候，即使此类别数据量为 0，结果仍会返回此类别数据：

```
In [134]: df.groupby("grade").size()
Out[134]:
grade
very bad     1
bad          0
medium       0
good         2
very good    3
dtype: int64
```

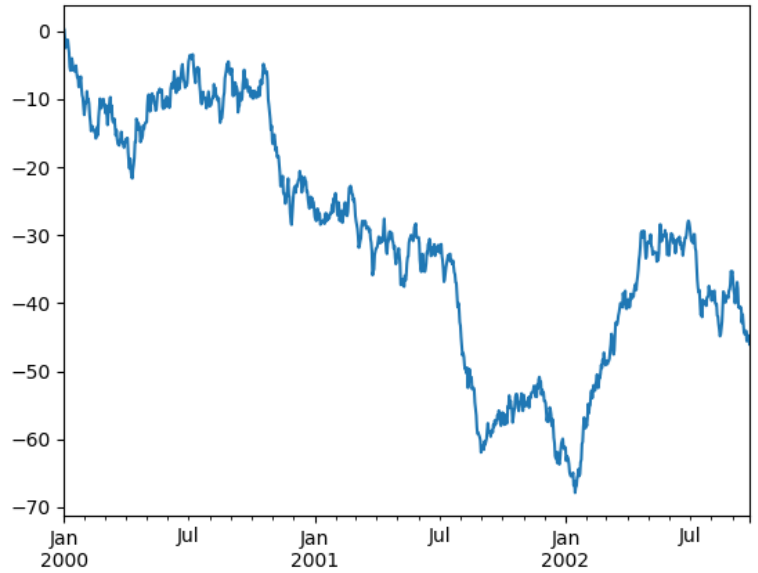
绘图

❖ Pandas 支持通过 Matplotlib 来绘制图表：

```
In [135]: ts = pd.Series(np.random.randn(1000),
   .....:                index=pd.date_range('1/1/2000', periods=1000))
   .....:

In [136]: ts = ts.cumsum()

In [137]: ts.plot()
Out[137]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2b5771ac88>
```



❖ 在 DataFrame 中，我们可以使用 plot() 函数来一次性绘制全部的列数据

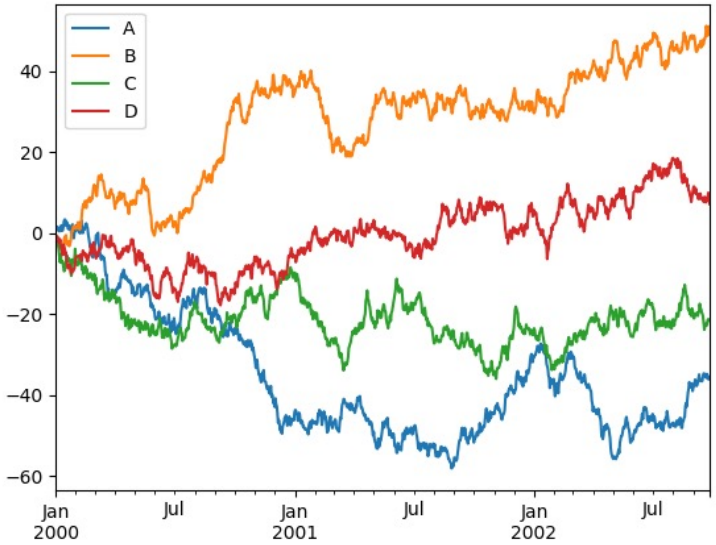
```
In [138]: df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index,
   .....:                   columns=['A', 'B', 'C', 'D'])
   .....:

In [139]: df = df.cumsum()

In [140]: plt.figure()
Out[140]: <Figure size 640x480 with 0 Axes>

In [141]: df.plot()
Out[141]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2b53a2d7f0>

In [142]: plt.legend(loc='best')
Out[142]: <matplotlib.legend.Legend at 0x7f2b539728d0>
```



导入、导出数据

❖ 导入、导出 CSV 数据：

```
In [143]: df.to_csv('foo.csv')
```

导入、导出数据

```
In [144]: pd.read_csv('foo.csv')
Out[144]:
     Unnamed: 0           A           B           C           D
0    2000-01-01    0.266457   -0.399641   -0.219582    1.186860
1    2000-01-02   -1.170732   -0.345873    1.653061   -0.282953
2    2000-01-03   -1.734933    0.530468    2.060811   -0.515536
3    2000-01-04   -1.555121    1.452620    0.239859   -1.156896
4    2000-01-05    0.578117    0.511371    0.103552   -2.428202
5    2000-01-06    0.478344    0.449933   -0.741620   -1.962409
6    2000-01-07    1.235339   -0.091757   -1.543861   -1.084753
..          ...         ...         ...         ...         ...
993  2002-09-20  -10.628548   -9.153563   -7.883146   28.313940
994  2002-09-21  -10.390377   -8.727491   -6.399645   30.914107
995  2002-09-22   -8.985362   -8.485624   -4.669462   31.367740
996  2002-09-23   -9.558560   -8.781216   -4.499815   30.518439
997  2002-09-24   -9.902058   -9.340490   -4.386639   30.105593
998  2002-09-25  -10.216020   -9.480682   -3.933802   29.758560
999  2002-09-26  -11.856774  -10.671012   -3.216025   29.369368
[1000 rows x 5 columns]
```

❖ 导入、导出 HDF 数据：

```
In [145]: df.to_hdf('foo.h5', 'df')
```

导入、导出数据

```
In [146]: pd.read_hdf('foo.h5', 'df')
Out[146]:
                   A          B         C          D
2000-01-01   0.266457  -0.399641 -0.219582   1.186860
2000-01-02  -1.170732  -0.345873  1.653061  -0.282953
2000-01-03  -1.734933   0.530468  2.060811  -0.515536
2000-01-04  -1.555121   1.452620  0.239859  -1.156896
2000-01-05   0.578117   0.511371  0.103552  -2.428202
2000-01-06   0.478344   0.449933 -0.741620  -1.962409
2000-01-07   1.235339  -0.091757 -1.543861  -1.084753
...               ...        ...       ...        ...
2002-09-20 -10.628548  -9.153563 -7.883146  28.313940
2002-09-21 -10.390377  -8.727491 -6.399645  30.914107
2002-09-22  -8.985362  -8.485624 -4.669462  31.367740
2002-09-23  -9.558560  -8.781216 -4.499815  30.518439
2002-09-24  -9.902058  -9.340490 -4.386639  30.105593
2002-09-25 -10.216020  -9.480682 -3.933802  29.758560
2002-09-26 -11.856774 -10.671012 -3.216025  29.369368
[1000 rows x 4 columns]
```

❖ 导入、导出 Excel 数据：

```
In [147]: df.to_excel('foo.xlsx', sheet_name='Sheet1')
```

导入、导出数据

```
In [148]: pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
Out[148]:
    Unnamed: 0          A          B         C          D
0   2000-01-01   0.266457  -0.399641 -0.219582   1.186860
1   2000-01-02  -1.170732  -0.345873  1.653061  -0.282953
2   2000-01-03  -1.734933   0.530468  2.060811  -0.515536
3   2000-01-04  -1.555121   1.452620  0.239859  -1.156896
4   2000-01-05   0.578117   0.511371  0.103552  -2.428202
5   2000-01-06   0.478344   0.449933 -0.741620  -1.962409
6   2000-01-07   1.235339  -0.091757 -1.543861  -1.084753
..         ...        ...        ...       ...        ...
993 2002-09-20 -10.628548  -9.153563 -7.883146  28.313940
994 2002-09-21 -10.390377  -8.727491 -6.399645  30.914107
995 2002-09-22  -8.985362  -8.485624 -4.669462  31.367740
996 2002-09-23  -9.558560  -8.781216 -4.499815  30.518439
997 2002-09-24  -9.902058  -9.340490 -4.386639  30.105593
998 2002-09-25 -10.216020  -9.480682 -3.933802  29.758560
999 2002-09-26 -11.856774 -10.671012 -3.216025  29.369368
[1000 rows x 5 columns]
```

错误处理

❖ 如果遇到 Pandas 的错误：

```
>>> if pd.Series([False, True, False]):
...     print("I was true")
Traceback
    ...
ValueError: The truth value of an array is ambiguous. Use a.empty, a.any() or a.all().
```

❖ 可以参考常见问题指南：

❖ http://pandas.pydata.org/pandas-docs/stable/user_guide/gotchas.html

目录

测试数据

❖ 各种创建测试数据的进阶方法：

```
In [1]: index = pd.date_range('1/1/2000', periods=8)

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

In [3]: df = pd.DataFrame(np.random.randn(8, 3), index=index,
   ...:                   columns=['A', 'B', 'C'])
   ...:

In [4]: wp = pd.Panel(np.random.randn(2, 5, 4), items=['Item1', 'Item2'],
   ...:               major_axis=pd.date_range('1/1/2000', periods=5),
   ...:               minor_axis=['A', 'B', 'C', 'D'])
   ...:
```

获取头部和尾部数据

❖ 如果希望查看 Series 或 DataFrame 对象的一小部分数据，我们可以使用 head() 及 tail() 函数，默认情况下会展示 5 个元素，也可以自定义需要展示的元素数量：

```
In [5]: long_series = pd.Series(np.random.randn(1000))
In [6]: long_series.head()
Out[6]:
0   -2.211372
1    0.974466
2   -2.006747
3   -0.410001
4   -0.078638
dtype: float64
In [7]: long_series.tail(3)
Out[7]:
997   -0.196166
998    0.380733
999   -0.275874
dtype: float64
```

参数及底层数据

❖ Pandas 对象拥有很多参数来方便用户访问元数据（Meta Data）：

```
In [8]: df[:2]
Out[8]:
                   A         B         C
2000-01-01 -0.173215  0.119209 -1.044236
2000-01-02 -0.861849 -2.104569 -0.494929

In [9]: df.columns = [x.lower() for x in df.columns]

In [10]: df
Out[10]:
                   a         b         c
2000-01-01 -0.173215  0.119209 -1.044236
2000-01-02 -0.861849 -2.104569 -0.494929
2000-01-03  1.071804  0.721555 -0.706771
2000-01-04 -1.039575  0.271860 -0.424972
2000-01-05  0.567020  0.276232 -1.087401
```

参数及底层数据

❖ 使用 .array 属性可以访问 Index 或 Series 的实际底层数据：

```
In [11]: s.array
Out[11]:
<PandasArray>
[ 0.4691122999071862, -0.2828633443286633,  -1.5090585031735124,
  -1.1356323710171934,   1.2121120250208506]
Length: 5, dtype: float64

In [12]: s.index.array
Out[12]:
<PandasArray>
['a', 'b', 'c', 'd', 'e']
Length: 5, dtype: object
```

参数及底层数据

❖ 如果需要将 Pandas 的数组转为 NumPy 的数组，可以使用：

```
In [13]: s.to_numpy()
Out[13]: array([ 0.4691, -0.2829, -1.5091, -1.1356,  1.2121])

In [14]: np.asarray(s)
Out[14]: array([ 0.4691, -0.2829, -1.5091, -1.1356,  1.2121])
```

参数及底层数据

- to_numpy() 在转换格式时，会对数据的 dtype 做一定修改
- 例如，处理时间，尤其是带有时区的时间数据时，Pandas 支持 datetime 类型，然而 NumPy 却没有对应的 dtype，因此在转换此类数据时，通常采用以下两种方案：

1. 将时间数据存储为 object 类型
2. 将时间数据存储为 datetime64[ns] 类型
3. 将时间数据转换为 UTC 的时间戳（Timestamp），然后按照 int64 类型存储

❖ 将时间数据存储为 object 类型：

```
In [15]: ser = pd.Series(pd.date_range('2000', periods=2, tz="CET"))

In [16]: ser.to_numpy(dtype=object)
Out[16]:
array([Timestamp('2000-01-01 00:00:00+0100', tz='CET', freq='D'),
       Timestamp('2000-01-02 00:00:00+0100', tz='CET', freq='D')], dtype=object)
```

❖ 将时间数据存储为 datetime64[ns] 类型：

```
In [17]: ser.to_numpy(dtype="datetime64[ns]")
Out[17]: array(['1999-12-31T23:00:00.000000000', '2000-01-01T23:00:00.000000000'],
dtype='datetime64[ns]')
```

参数及底层数据

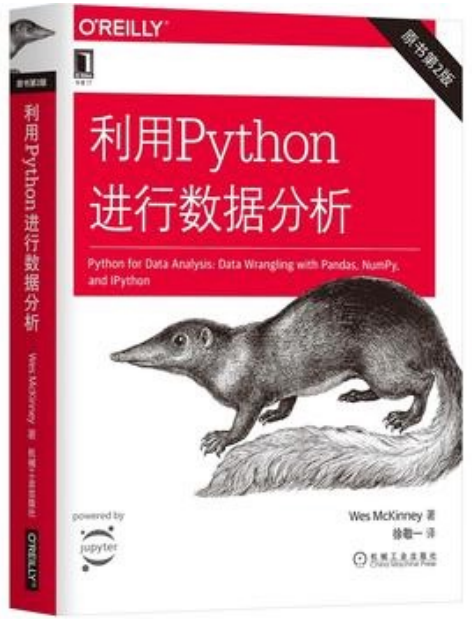
❖ 将时间数据转换为 UTC 的时间戳（Timestamp），然后按照 int64 类型存储：

```
In [17]: ser.to_numpy(dtype="int64")
Out[17]: array([946681200000000000, 946767600000000000])
```

❖ 数据是微秒（Nano Second），1 微秒等于 10^{-9} 秒

进阶功能

- 更多的进阶功能可以参考官方教程：
- http://pandas.pydata.org/pandas-docs/stable/getting_started/basics.html
- 函数详细说明文档及手册：
- http://pandas.pydata.org/pandas-docs/stable/reference/index.html
- Wes McKinney《利用 Python 进行数据分析》：
- http://product.dangdang.com/25312917.html



Thanks!