Use-pandas

2019 年 8 月 22 日

[ ]:
```
# 安装 pandas
# pip install Pandas

# 运行测试套件
# 运行前需要安装: hypothesis 和 pytest
import pandas as pd
# pd.test()
```

[5]:
```
# 对象创建
# 传入一些值的列表来创建一个 Series,pandas 会自动创建一个默认的整数索引.
import pandas as pd
import numpy as np
import pprint

s = pd.Series([1,3,5,np.nan,6,8])
print(s)
print('-'*30)
# 传递带有日期时间索引和带标签列的 NumPy 数组来创建 DataFrame
dates = pd.date_range('20190815',periods=6)
pprint.pprint(dates)
```

```
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
------------------------------
DatetimeIndex(['2019-08-15', '2019-08-16', '2019-08-17', '2019-08-18',
               '2019-08-19', '2019-08-20'],
```

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

[8]:
```
df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))
# df.to_excel('./output.xlsx')
pprint.pprint(df)
```

```
                   A         B         C         D
2019-08-15  0.390134  0.870386 -0.658386  0.574883
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278
2019-08-17 -0.280608 -0.103164 -0.271675 -0.521797
2019-08-18 -0.515949 -1.077575  0.836807 -0.203942
2019-08-19 -0.837345  0.010786 -0.157282 -0.822339
2019-08-20  0.820916  1.243794 -0.320740 -0.064784
```

[9]:
```
# 转化为类似 Series 的 dict 对象来创建 DataFrame
df2 = pd.DataFrame({'A': 1.,
                    'B': pd.Timestamp('20190820'),
                    'C': pd.Series(1,index=list(range(4)),dtype='float32'),
                    'D': np.array([3] * 4,dtype='int32'),
                    'E': pd.Categorical(["test","train","test1","train2"]),
                    'F': 'foo'})
print(df2)
# DataFrame 的列具有不同的数据类型
print(df2.dtypes)
```

```
     A          B    C  D       E    F
0  1.0 2019-08-20  1.0  3    test  foo
1  1.0 2019-08-20  1.0  3   train  foo
2  1.0 2019-08-20  1.0  3   test1  foo
3  1.0 2019-08-20  1.0  3  train2  foo
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

[10]:
```
### 查看数据
# 查看 DataFrame 顶部数据
```

```
print(df.head(3))
print('+='*30)
# 查看 DataFrame 尾部数据
print(df.tail(3))
print('--+'*30)
# 显示索引，列和底层 NumPy 数据.
print(df.index)
print('-='*30)
print(df.columns)
print('-|'*30)
# DataFrame.to_numpy() 会给出 Numpy 对象. 输出时不包含行索引和列索引.
print(df.to_numpy())
```

```
                   A         B         C         D
2019-08-15  0.390134  0.870386 -0.658386  0.574883
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278
2019-08-17 -0.280608 -0.103164 -0.271675 -0.521797
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=
                   A         B         C         D
2019-08-18 -0.515949 -1.077575  0.836807 -0.203942
2019-08-19 -0.837345  0.010786 -0.157282 -0.822339
2019-08-20  0.820916  1.243794 -0.320740 -0.064784
--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
+--+--+--+
DatetimeIndex(['2019-08-15', '2019-08-16', '2019-08-17', '2019-08-18',
               '2019-08-19', '2019-08-20'],
              dtype='datetime64[ns]', freq='D')
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
Index(['A', 'B', 'C', 'D'], dtype='object')
-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|
[[ 0.39013393  0.87038581 -0.65838619  0.57488332]
 [-0.94540789  1.62448344 -1.70829855 -1.58327774]
 [-0.28060831 -0.10316405 -0.27167469 -0.5217975 ]
 [-0.51594885 -1.07757455  0.83680735 -0.20394234]
 [-0.83734547  0.01078588 -0.1572819  -0.82233919]
 [ 0.82091588  1.24379353 -0.32073955 -0.06478436]]
```

[11]:

```
# describe() 方法显示数据的快速统计摘要
print(df.describe())
```

```
print('--'*30)
# 转置数据
print(df.T)
print('=='*30)
# 按轴排序
print(df.sort_index(axis=1,ascending=False))
print('-='*30)
# 按值排序
print(df.sort_values(by='B'))
```

```
              A         B         C         D
count  6.000000  6.000000  6.000000  6.000000
mean  -0.228043  0.428118 -0.379929 -0.436876
std    0.700567  1.001574  0.823085  0.732994
min   -0.945408 -1.077575 -1.708299 -1.583278
25%   -0.756996 -0.074677 -0.573975 -0.747204
50%   -0.398279  0.440586 -0.296207 -0.362870
75%    0.222448  1.150442 -0.185880 -0.099574
max    0.820916  1.624483  0.836807  0.574883
------------------------------------------------------------
   2019-08-15  2019-08-16  2019-08-17  2019-08-18  2019-08-19  2019-08-20
A    0.390134   -0.945408   -0.280608   -0.515949   -0.837345    0.820916
B    0.870386    1.624483   -0.103164   -1.077575    0.010786    1.243794
C   -0.658386   -1.708299   -0.271675    0.836807   -0.157282   -0.320740
D    0.574883   -1.583278   -0.521797   -0.203942   -0.822339   -0.064784
============================================================
                   D         C         B         A
2019-08-15  0.574883 -0.658386  0.870386  0.390134
2019-08-16 -1.583278 -1.708299  1.624483 -0.945408
2019-08-17 -0.521797 -0.271675 -0.103164 -0.280608
2019-08-18 -0.203942  0.836807 -1.077575 -0.515949
2019-08-19 -0.822339 -0.157282  0.010786 -0.837345
2019-08-20 -0.064784 -0.320740  1.243794  0.820916
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
                   A         B         C         D
2019-08-18 -0.515949 -1.077575  0.836807 -0.203942
2019-08-17 -0.280608 -0.103164 -0.271675 -0.521797
2019-08-19 -0.837345  0.010786 -0.157282 -0.822339
2019-08-15  0.390134  0.870386 -0.658386  0.574883
```

```
2019-08-20  0.820916  1.243794 -0.320740 -0.064784
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278
```

[12]:
```
### 获取
print(df['A'])
# 对行进行切片
print(df[0:])
print(df[0:2])
print('-=='*30)
print(df['20190816':'20190818'])
```

```
2019-08-15    0.390134
2019-08-16   -0.945408
2019-08-17   -0.280608
2019-08-18   -0.515949
2019-08-19   -0.837345
2019-08-20    0.820916
Freq: D, Name: A, dtype: float64
                   A         B         C         D
2019-08-15  0.390134  0.870386 -0.658386  0.574883
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278
2019-08-17 -0.280608 -0.103164 -0.271675 -0.521797
2019-08-18 -0.515949 -1.077575  0.836807 -0.203942
2019-08-19 -0.837345  0.010786 -0.157282 -0.822339
2019-08-20  0.820916  1.243794 -0.320740 -0.064784
                   A         B         C         D
2019-08-15  0.390134  0.870386 -0.658386  0.574883
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278
-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-==-=
=-==-==-==
                   A         B         C         D
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278
2019-08-17 -0.280608 -0.103164 -0.271675 -0.521797
2019-08-18 -0.515949 -1.077575  0.836807 -0.203942
```

[13]:
```
### 按标签选择
# 通过标签获取一行数据
print(df.loc[dates[0]])
print(df.loc[dates[1]])
print('=='*30)
```

```
# 通过标签在多个轴上选择数据
print('通过标签在多个轴上选择数据')
print(df.loc[:,['A','B']])
print('--'*30)
print(df.loc[:,['C']])
```

```
A    0.390134
B    0.870386
C   -0.658386
D    0.574883
Name: 2019-08-15 00:00:00, dtype: float64
A   -0.945408
B    1.624483
C   -1.708299
D   -1.583278
Name: 2019-08-16 00:00:00, dtype: float64
============================================================
通过标签在多个轴上选择数据
                   A         B
2019-08-15  0.390134  0.870386
2019-08-16 -0.945408  1.624483
2019-08-17 -0.280608 -0.103164
2019-08-18 -0.515949 -1.077575
2019-08-19 -0.837345  0.010786
2019-08-20  0.820916  1.243794
------------------------------------------------------------
                   C
2019-08-15 -0.658386
2019-08-16 -1.708299
2019-08-17 -0.271675
2019-08-18  0.836807
2019-08-19 -0.157282
2019-08-20 -0.320740
```

[14]:
```
# 通过标签同时在两个轴上切片
print('通过标签同时在两个轴上切片')
print(df.loc['20190817':'20190819',['A','B']])
```

```
通过标签同时在两个轴上切片
                   A         B
```

```
2019-08-17 -0.280608 -0.103164
2019-08-18 -0.515949 -1.077575
2019-08-19 -0.837345  0.010786
```

[15]:
```
# 减小返回对象的大小
print(df.loc['20190820',['A','B']])
```

```
A    0.820916
B    1.243794
Name: 2019-08-20 00:00:00, dtype: float64
```

[16]:
```
# 获取标量值
print(df.loc[dates[0],'A'])
```

```
0.3901339290890978
```

[17]:
```
# 快速访问标量
print(df.at[dates[0],'A'])
```

```
0.3901339290890978
```

[18]:
```
### 布尔索引
# 使用单个列的值来选择数据
print(df[df.A > 0])   # 会输出为 True 的内容.
print(df.A > 0) # 将 True 和 False 都打印出来.
```

```
                   A         B         C         D
2019-08-15  0.390134  0.870386 -0.658386  0.574883
2019-08-20  0.820916  1.243794 -0.320740 -0.064784
2019-08-15     True
2019-08-16    False
2019-08-17    False
2019-08-18    False
2019-08-19    False
2019-08-20     True
Freq: D, Name: A, dtype: bool
```

[19]:
```
# 从满足布尔条件的 DataFrame 中选择值:
print(df[df > 0])
```

```
                   A         B         C         D
2019-08-15  0.390134  0.870386       NaN  0.574883
2019-08-16       NaN  1.624483       NaN       NaN
2019-08-17       NaN       NaN       NaN       NaN
2019-08-18       NaN       NaN  0.836807       NaN
2019-08-19       NaN  0.010786       NaN       NaN
2019-08-20  0.820916  1.243794       NaN       NaN
```

[20]:

```
# 使用 isin() 方法过滤
df3 = df.copy()
# print(df3)
# df3['E'] = ['one','one','two','three','four','three']
df3['E'] = ['one','two','three','four','five','six']

# print(df3)
print('-='*30)
print(df3[df3['E'].isin(['two','four'])])
```

```
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
                   A         B         C         D     E
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278   two
2019-08-18 -0.515949 -1.077575  0.836807 -0.203942  four
```

[21]:

```
### 赋值
# 添加新列将自动根据索引对齐数据.
s1 = pd.Series([1,2,3,4,5,6],index=pd.date_range('20190818',periods=6))
print(s1)
df3['F'] = s1
print(df3['F'])
```

```
2019-08-18    1
2019-08-19    2
2019-08-20    3
2019-08-21    4
2019-08-22    5
2019-08-23    6
Freq: D, dtype: int64
2019-08-15    NaN
2019-08-16    NaN
2019-08-17    NaN
```

```
2019-08-18    1.0
2019-08-19    2.0
2019-08-20    3.0
Freq: D, Name: F, dtype: float64
```

[22]:
```
# 通过标签赋值
df3.at[dates[0],'A'] = 0
print(df3)
```

```
                   A         B         C         D      E    F
2019-08-15  0.000000  0.870386 -0.658386  0.574883    one  NaN
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278    two  NaN
2019-08-17 -0.280608 -0.103164 -0.271675 -0.521797  three  NaN
2019-08-18 -0.515949 -1.077575  0.836807 -0.203942   four  1.0
2019-08-19 -0.837345  0.010786 -0.157282 -0.822339   five  2.0
2019-08-20  0.820916  1.243794 -0.320740 -0.064784    six  3.0
```

[27]:
```
### 通过位置赋值
df.iat[0,1] = 0
print(df)
```

```
                   A         B         C         D
2019-08-15  0.390134  0.000000 -0.658386  0.574883
2019-08-16 -0.945408  1.624483 -1.708299 -1.583278
2019-08-17 -0.280608 -0.103164 -0.271675 -0.521797
2019-08-18 -0.515949 -1.077575  0.836807 -0.203942
2019-08-19 -0.837345  0.010786 -0.157282 -0.822339
2019-08-20  0.820916  1.243794 -0.320740 -0.064784
```

[29]:
```
# 使用 NumPy 数组赋值
df3.loc[:,'D'] = np.array([5] * len(df))
print(df3) # 前面一系列赋值操作的结果.
```

```
                   A         B         C  D      E    F
2019-08-15  0.000000  0.870386 -0.658386  5    one  NaN
2019-08-16 -0.945408  1.624483 -1.708299  5    two  NaN
2019-08-17 -0.280608 -0.103164 -0.271675  5  three  NaN
2019-08-18 -0.515949 -1.077575  0.836807  5   four  1.0
2019-08-19 -0.837345  0.010786 -0.157282  5   five  2.0
2019-08-20  0.820916  1.243794 -0.320740  5    six  3.0
```

[41]:
```
# 带有 where 条件的赋值操作.
df2 = df.copy()
df2[df2 > 0] = -df2
print(df2)
```

```
                   A         B         C  D
2019-08-15 -0.390134  0.000000 -0.658386 -5
2019-08-16 -0.945408 -1.624483 -1.708299 -5
2019-08-17 -0.280608 -0.103164 -0.271675 -5
2019-08-18 -0.515949 -1.077575 -0.836807 -5
2019-08-19 -0.837345 -0.010786 -0.157282 -5
2019-08-20 -0.820916 -1.243794 -0.320740 -5
```

[42]:
```
### 缺失值
# pandas 主要使用值 np.nan 来表示缺失的数据.
# 重建索引允许更改/添加/删除指定轴上的索引.这个操作会返回一个副本.
df5 = df.reindex(index=dates[0:4],columns=list(df.columns) + ['E'])
df5.loc[dates[0]:dates[1],'E'] = 1
print(df5)
```

```
                   A         B         C  D    E
2019-08-15  0.390134  0.000000 -0.658386  5  1.0
2019-08-16 -0.945408  1.624483 -1.708299  5  1.0
2019-08-17 -0.280608 -0.103164 -0.271675  5  NaN
2019-08-18 -0.515949 -1.077575  0.836807  5  NaN
```

[45]:
```
# 删除任何带有缺失值的行
print(df5.dropna(how='any'))
```

```
                   A         B         C  D    E
2019-08-15  0.390134  0.000000 -0.658386  5  1.0
2019-08-16 -0.945408  1.624483 -1.708299  5  1.0
```

[46]:
```
# 填充缺失值
print(df5.fillna(value=5))
```

```
                   A         B         C  D    E
2019-08-15  0.390134  0.000000 -0.658386  5  1.0
2019-08-16 -0.945408  1.624483 -1.708299  5  1.0
2019-08-17 -0.280608 -0.103164 -0.271675  5  5.0
2019-08-18 -0.515949 -1.077575  0.836807  5  5.0
```

[47]:
```python
# 获取值为 nan 的掩码
print(pd.isna(df5))
```

```
                A      B      C      D      E
2019-08-15  False  False  False  False  False
2019-08-16  False  False  False  False  False
2019-08-17  False  False  False  False   True
2019-08-18  False  False  False  False   True
```

[49]:
```python
### 统计
# 进行描述性统计
print(df5.mean())
print('-='*30)
# 在其它轴上进行同样的操作:
print(df5.mean(1))
```

```
A   -0.337958
B    0.110936
C   -0.450388
D    5.000000
E    1.000000
dtype: float64
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
2019-08-15    1.146350
2019-08-16    0.994155
2019-08-17    1.086138
2019-08-18    1.060821
Freq: D, dtype: float64
```

[51]:
```python
# 使用具有不同维度且需要对齐的对象进行操作. pandas 会自动沿指定维度进行广播.
s = pd.Series([1,3,5,np.nan,6,8],index=dates).shift(2)
print(s)

print(df5.sub(s,axis='index'))
```

```
2019-08-15    NaN
2019-08-16    NaN
2019-08-17    1.0
2019-08-18    3.0
2019-08-19    5.0
```

```
2019-08-20    NaN
Freq: D, dtype: float64
                   A         B         C    D   E
2019-08-15       NaN       NaN       NaN  NaN NaN
2019-08-16       NaN       NaN       NaN  NaN NaN
2019-08-17 -1.280608 -1.103164 -1.271675  4.0 NaN
2019-08-18 -3.515949 -4.077575 -2.163193  2.0 NaN
2019-08-19       NaN       NaN       NaN  NaN NaN
2019-08-20       NaN       NaN       NaN  NaN NaN
```

[53]:
```
### 应用
# 将函数应用于数据
print(df5.apply(np.cumsum))

print(df5.apply(lambda x: x.max() - x.min()))
```

```
                   A         B         C   D    E
2019-08-15  0.390134  0.000000 -0.658386   5  1.0
2019-08-16 -0.555274  1.624483 -2.366685  10  2.0
2019-08-17 -0.835882  1.521319 -2.638359  15  NaN
2019-08-18 -1.351831  0.443745 -1.801552  20  NaN
A    1.335542
B    2.702058
C    2.545106
D    0.000000
E    0.000000
dtype: float64
```

[56]:
```
### 直方图化
s1 = pd.Series(np.random.randint(0,7,size=10))
print(s1)
print('=+'*30)
print(s1.value_counts())
```

```
0    1
1    0
2    5
3    2
4    3
5    1
```

```
6    5
7    2
8    4
9    5
dtype: int32
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
5    3
2    2
1    2
4    1
3    1
0    1
dtype: int64
```

[57]:
```
### 字符串方法
# Series 在 str 属性中有一组字符串处理方法，可对数组的每个元素进行操作.
s2 = pd.Series(['A','B','C','Aaba','Baca',np.nan,'CABA','dog','cat'])
print(s2.str.lower())
```

```
0       a
1       b
2       c
3    aaba
4    baca
5     NaN
6    caba
7     dog
8     cat
dtype: object
```

[60]:
```
## 合并
### 连接
# 使用 concat() 连接 pandas 对象.
df6 = pd.DataFrame(np.random.randn(10,4))
print(df6)
print('---'*30)
pieces = [df6[:3],df6[3:7],df6[7:]]
print(pd.concat(pieces))
```

```
          0         1         2         3
0  0.460560  1.698442 -0.933407  1.217372
```

```
1  0.066645  2.134691  1.738260 -1.113047
2 -0.199609  0.614373 -0.348059  0.450941
3  1.018399  0.284015 -2.738610  0.751599
4 -0.306876 -0.570027  0.186237 -1.291311
5 -0.546909 -1.050943 -0.429117  0.990815
6 -0.123894  0.594589  1.730954 -1.057447
7 -1.002860 -0.298396 -1.059377 -0.262950
8  1.146888  1.127948  1.513706  0.424602
9 -0.916949 -0.661869 -0.900454 -0.805987
--------------------------------------------------------------------------------
----------
          0         1         2         3
0  0.460560  1.698442 -0.933407  1.217372
1  0.066645  2.134691  1.738260 -1.113047
2 -0.199609  0.614373 -0.348059  0.450941
3  1.018399  0.284015 -2.738610  0.751599
4 -0.306876 -0.570027  0.186237 -1.291311
5 -0.546909 -1.050943 -0.429117  0.990815
6 -0.123894  0.594589  1.730954 -1.057447
7 -1.002860 -0.298396 -1.059377 -0.262950
8  1.146888  1.127948  1.513706  0.424602
9 -0.916949 -0.661869 -0.900454 -0.805987
```

[62]:

```
### Join
# SQL 风格的合并
left = pd.DataFrame({'key': ['foo','foo'],'lval':[1,2]})
right = pd.DataFrame({'key': ['foo','foo'],'rval': [4,5]})
print(left)
print('='*30)
print(right)
print('='*30)
print(pd.merge(left,right,on='key'))
```

```
   key  lval
0  foo     1
1  foo     2
==============================
   key  rval
0  foo     4
1  foo     5
```

```
==============================
   key  lval  rval
0  foo     1     4
1  foo     1     5
2  foo     2     4
3  foo     2     5
```

[63]:
```
# 另一个例子
left = pd.DataFrame({'key': ['foo','bar'],'lval': [1,2]})
right = pd.DataFrame({'key': ['foo','bar'],'rval':[4,5]})
print(left)
print('-'*30)
print(right)
print(pd.merge(left,right,on='key'))
```

```
   key  lval
0  foo     1
1  bar     2
------------------------------
   key  rval
0  foo     4
1  bar     5
   key  lval  rval
0  foo     1     4
1  bar     2     5
```

[66]:
```
### 追加
df7 = pd.DataFrame(np.random.randn(8,4),columns=['A','B','C','D'])
print(df7)
print('=='*30)
s3 = df7.iloc[3]
print(df7.append(s3,ignore_index=True))
```

```
          A         B         C         D
0 -0.268989 -0.933812 -2.041705 -0.507051
1  1.193505 -0.472899 -1.474935  0.269695
2  0.094337  0.345821 -0.780803 -2.039000
3  0.406391  0.037250 -0.616318 -1.482976
4  0.183275 -2.901859 -2.117727  0.027000
5 -0.554172  0.323265  0.158906  2.060658
```

```
6  0.198859  0.336248  1.059244 -0.264119
7  0.102135 -0.010265  0.371754 -0.401392
============================================================
          A         B         C         D
0 -0.268989 -0.933812 -2.041705 -0.507051
1  1.193505 -0.472899 -1.474935  0.269695
2  0.094337  0.345821 -0.780803 -2.039000
3  0.406391  0.037250 -0.616318 -1.482976
4  0.183275 -2.901859 -2.117727  0.027000
5 -0.554172  0.323265  0.158906  2.060658
6  0.198859  0.336248  1.059244 -0.264119
7  0.102135 -0.010265  0.371754 -0.401392
8  0.406391  0.037250 -0.616318 -1.482976
```

[68]:

```
### 分组
"""
group by 包括:
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.
"""
df8 = 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)})

print(df8)
# 分组, 然后将 sum() 函数应用于分组结果.
print(df8.groupby('A').sum())
print('=-'*30)
# 按多列分组形成层次索引, 用 sum 函数
print(df8.groupby(['A','B']).sum())
```

```
     A      B         C         D
0  foo    one  0.696916  1.007796
1  bar    one  0.748873  0.031474
2  foo    two  0.118730 -0.976749
3  bar  three  1.742898 -1.185333
```

```
4  foo    two  0.477294  0.425628
5  bar    two  0.196629 -0.349258
6  foo    one  0.621472 -0.751191
7  foo  three  0.144760 -0.522888
            C         D
A
bar  2.688400 -1.503117
foo  2.059172 -0.817404
=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-
                  C         D
A   B
bar one    0.748873  0.031474
    three  1.742898 -1.185333
    two    0.196629 -0.349258
foo one    1.318388  0.256605
    three  0.144760 -0.522888
    two    0.596024 -0.551120
```

[69]:

```
### 堆叠 (Stack)
tuples = list(zip(*[['bar','bar','baz','baz',
                    'foo','foo','qux','qux'],
                   ['one','two','one','two',
                   'one','two','one','two']]))

index = pd.MultiIndex.from_tuples(tuples,names=['first','second'])
df = pd.DataFrame(np.random.randn(8,2),index=index,columns=['A','B'])
df9 = df[:4]
print(df9)
```

```
                     A         B
first second
bar   one    -0.954108  0.395601
      two     0.401696  1.084930
baz   one    -0.736508 -0.261757
      two    -1.127218  0.274139
```

[70]:

```
### stack() 方法压缩 DataFrame 的列
stacked = df9.stack()
print(stacked)
```

```
first  second
bar    one     A   -0.954108
               B    0.395601
       two     A    0.401696
               B    1.084930
baz    one     A   -0.736508
               B   -0.261757
       two     A   -1.127218
               B    0.274139
dtype: float64
```

[72]:
```
# stack() 的逆操作是 unstack(), 默认情况下取消最后压缩的哪个级别.
print(stacked.unstack())
print('=='*30)
print(stacked.unstack(1))
print('-='*30)
print(stacked.unstack(0))
```

```
                     A         B
first second
bar   one    -0.954108  0.395601
      two     0.401696  1.084930
baz   one    -0.736508 -0.261757
      two    -1.127218  0.274139
============================================================
second        one       two
first
bar   A -0.954108  0.401696
      B  0.395601  1.084930
baz   A -0.736508 -1.127218
      B -0.261757  0.274139
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
first          bar       baz
second
one    A -0.954108 -0.736508
       B  0.395601 -0.261757
two    A  0.401696 -1.127218
       B  1.084930  0.274139
```

[75]:
```
### 数据透视表
df10 = 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)})

print(df10)
print('-='*30)
# 从这些数据生成数据透视表
pd.pivot_table(df10,values='D',index=['A','B'],columns=['C'])
```

```
        A  B    C         D         E
0     one  A  foo -1.170330  0.218408
1     one  B  foo  0.094075 -0.177464
2     two  C  foo -2.562008  0.530883
3   three  A  bar -0.176252 -1.400335
4     one  B  bar  0.353643 -0.156274
5     one  C  bar -1.079407 -0.382338
6     two  A  foo  1.182400 -0.863431
7   three  B  foo  0.182097  0.476877
8     one  C  foo -1.788838 -0.134557
9     one  A  bar  0.717134 -0.561533
10    two  B  bar -0.441708 -1.276190
11  three  C  bar  0.235704 -1.109769
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

[75]:
```
C             bar       foo
A     B
one   A  0.717134 -1.170330
      B  0.353643  0.094075
      C -1.079407 -1.788838
three A -0.176252       NaN
      B       NaN  0.182097
      C  0.235704       NaN
two   A       NaN  1.182400
      B -0.441708       NaN
      C       NaN -2.562008
```

[76]:
```
### 时间序列 (TimeSeries)
# 用于在频率转换期间执行重采样操作.
rng = pd.date_range('22/08/2019',periods=100,freq='S')
ts = pd.Series(np.random.randint(0,500,len(rng)),index=rng)
print(ts.resample('5Min').sum())
```

```
2019-08-22    24145
Freq: 5T, dtype: int32
```

[79]:
```
# 时区代表
rng = pd.date_range('21/08/2019 21:29:30',periods=5,freq='D')
ts = pd.Series(np.random.randn(len(rng)),rng)
print(ts)

print('-='*30)
ts_utc = ts.tz_localize('UTC')
print(ts_utc)

print('-='*30)
# 转换为另一个时区
print(ts_utc.tz_convert('US/Eastern'))
```

```
2019-08-21 21:29:30   -0.881633
2019-08-22 21:29:30    0.306377
2019-08-23 21:29:30   -0.481741
2019-08-24 21:29:30   -0.046010
2019-08-25 21:29:30   -2.325477
Freq: D, dtype: float64
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
2019-08-21 21:29:30+00:00   -0.881633
2019-08-22 21:29:30+00:00    0.306377
2019-08-23 21:29:30+00:00   -0.481741
2019-08-24 21:29:30+00:00   -0.046010
2019-08-25 21:29:30+00:00   -2.325477
Freq: D, dtype: float64
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
2019-08-21 17:29:30-04:00   -0.881633
2019-08-22 17:29:30-04:00    0.306377
2019-08-23 17:29:30-04:00   -0.481741
2019-08-24 17:29:30-04:00   -0.046010
```

```
2019-08-25 17:29:30-04:00   -2.325477
Freq: D, dtype: float64
```

[82]:

```
# 在时间跨度表示之间转换
rng = pd.date_range('22/08/2019',periods=5,freq='M')
ts = pd.Series(np.random.randn(len(rng)),index=rng)
print(ts)
print('-='*30)
ps = ts.to_period()
print(ps)
print('-='*30)
print(ps.to_timestamp())
```

```
2019-08-31    0.618355
2019-09-30    0.725058
2019-10-31   -0.625299
2019-11-30    1.990605
2019-12-31   -0.436197
Freq: M, dtype: float64
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
2019-08    0.618355
2019-09    0.725058
2019-10   -0.625299
2019-11    1.990605
2019-12   -0.436197
Freq: M, dtype: float64
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
2019-08-01    0.618355
2019-09-01    0.725058
2019-10-01   -0.625299
2019-11-01    1.990605
2019-12-01   -0.436197
Freq: MS, dtype: float64
```

[83]:

```
# 周期和时间戳之间的转换可以用算术函数.
# 示例: 以 11 月为结束年份的季度频率转换为季度结束后一个月末的上午 9 点.
prng = pd.period_range('2010Q1','2019Q4',freq='Q-NOV')
ts = pd.Series(np.random.randn(len(prng)),prng)
ts.index = (prng.asfreq('M','e') + 1).asfreq('H','s') + 9
print(ts.head())
```

```
2010-03-01 09:00    0.291307
2010-06-01 09:00   -0.274416
2010-09-01 09:00    0.660247
2010-12-01 09:00    2.908591
2011-03-01 09:00    0.360034
Freq: H, dtype: float64
```

[86]:
```
### 分类 (Categoricals)
# pandas 可以在 DataFrame 中包含分类数据.
df11 = pd.DataFrame({"id": [1,2,3,4,5,6],
                     "raw_grade": ['a','b','b','a','a','e']})
# 将原始成绩转换为 category 数据类型
df11["grade"] = df11["raw_grade"].astype("category")
print(df11["grade"])
print('-='*30)
```

```
0    a
1    b
2    b
3    a
4    a
5    e
Name: grade, dtype: category
Categories (3, object): [a, b, e]
-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

[87]:
```
# 将类别重命名为更有意义的名称 (通过调用 Series.cat.categories 来替换)
df11["grade"].cat.categories = ["very good","good","very bad"]
print(df11["grade"].cat.categories)
```

```
Index(['very good', 'good', 'very bad'], dtype='object')
```

[88]:
```
# 对 categories 重新排序并同时添加缺少的 category(Series.cat 下的方法默认返回一个新的 Series)
df11["grade"] = df11["grade"].cat.set_categories(["very bad","bad","medium",
                                                  "good","very good"])
print(df11["grade"])
```

```
0    very good
1         good
```

```
2         good
3    very good
4    very good
5     very bad
Name: grade, dtype: category
Categories (5, object): [very bad, bad, medium, good, very good]
```

[90]:
```
# 排序时按 categories 中的顺序排序，不是按照词汇顺序排序.
print(df11.sort_values(by="grade"))
```

```
   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
```

[91]:
```
# 按分好类的列分组 (groupby) 可以显示空 categories。
print(df11.groupby("grade").size())
```

```
grade
very bad     1
bad          0
medium       0
good         2
very good    3
dtype: int64
```

[92]:
```
### 绘图
ts = pd.Series(np.random.randn(1000),
               index=pd.date_range('22/08/2019',periods=1000))
ts = ts.cumsum()
ts.plot()
```

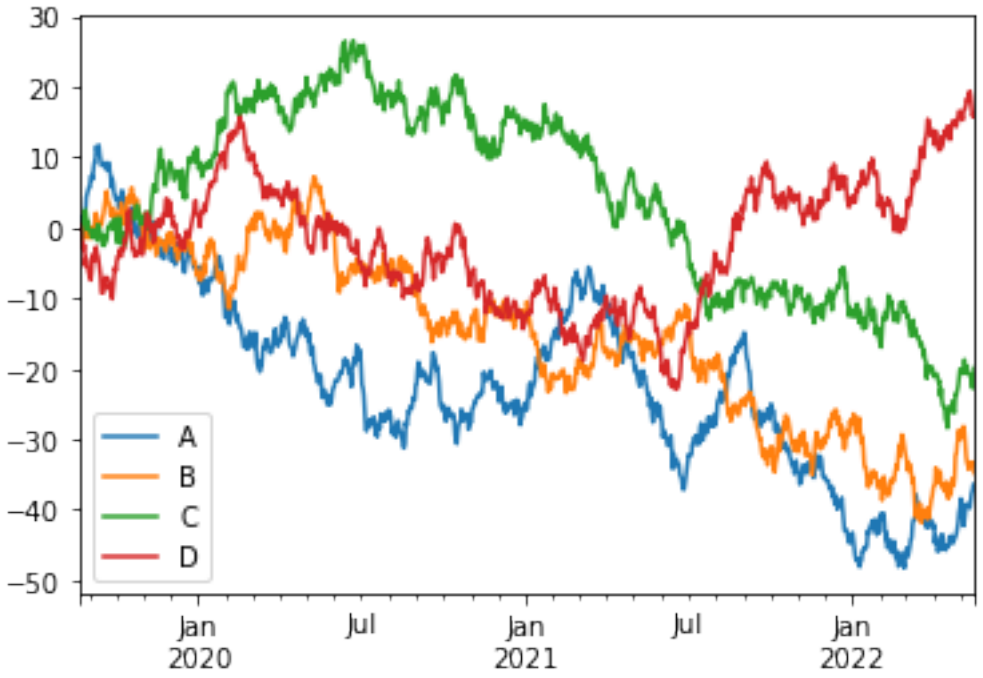
[92]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x18e40420be0>
```

[94]:
```
import matplotlib.pyplot as plt
# 在一个 DataFrame 中,plot 方法绘制带有 label 的所有列.
df12 = pd.DataFrame(np.random.randn(1000,4),index=ts.index,
                    columns=['A','B','C','D'])
```

```
df13 = df12.cumsum()
plt.figure()
df13.plot()
plt.legend(loc='best')
```

[94]:
```
<matplotlib.legend.Legend at 0x18e44cc7da0>
```

```
<Figure size 432x288 with 0 Axes>
```



[96]:
```
### 数据输入/输出
# 写入 csv 文件
df13.to_csv('./best.csv')
# 从 csv 文件读数据
pd.read_csv('./best.csv')
```

[96]:

	Unnamed: 0	A	B	C	D
0	2019-08-22	0.086278	-0.285369	-0.012890	0.452216
1	2019-08-23	1.412777	0.238720	-1.740038	-0.123636
2	2019-08-24	0.799518	-0.888554	-0.924861	-1.706895
3	2019-08-25	1.564615	-1.646619	0.184296	-2.976331

4	2019-08-26	1.903628	-1.025881	1.386930	-2.719044
5	2019-08-27	1.164920	-0.724013	1.236690	-2.495277
6	2019-08-28	2.003877	0.348358	2.549467	-5.438851
7	2019-08-29	4.304125	-0.459755	1.763327	-4.409135
8	2019-08-30	4.726267	-1.517907	-0.212574	-5.101913
9	2019-08-31	5.277158	-1.264182	0.906538	-5.894793
10	2019-09-01	5.747264	0.225777	-0.257817	-6.596933
11	2019-09-02	5.728967	-0.359076	-0.272538	-6.594936
12	2019-09-03	7.232054	-1.490280	-0.951729	-7.434688
13	2019-09-04	7.329095	-0.181231	-1.309576	-6.376228
14	2019-09-05	6.846514	-0.814378	-1.398260	-3.829608
15	2019-09-06	8.307232	-0.995645	-0.609413	-3.974617
16	2019-09-07	9.228199	0.449028	-0.260423	-4.760922
17	2019-09-08	9.240231	1.876106	-0.785967	-3.308362
18	2019-09-09	11.499487	2.060554	-0.621835	-2.734745
19	2019-09-10	11.136724	1.607395	-1.830549	-3.678861
20	2019-09-11	11.324174	0.234021	0.436278	-4.044490
21	2019-09-12	11.688123	0.480685	0.045486	-4.981361
22	2019-09-13	10.397040	0.789385	-0.474993	-6.011545
23	2019-09-14	9.622623	0.783358	-0.732386	-6.763026
24	2019-09-15	8.798661	0.350506	-1.422821	-9.280633
25	2019-09-16	8.830095	1.255681	-2.513342	-9.215975
26	2019-09-17	8.816481	1.816570	-1.076567	-9.238743
27	2019-09-18	8.880187	2.828825	-2.547987	-8.452422
28	2019-09-19	8.966015	3.723613	-2.234985	-7.050768
29	2019-09-20	9.232280	4.353496	-1.996265	-7.063971
..	...	...	...	...	...
970	2022-04-18	-45.275887	-38.456353	-26.531187	14.421749
971	2022-04-19	-44.002236	-37.696672	-25.418910	14.249406
972	2022-04-20	-44.642954	-37.883943	-25.814418	12.476328
973	2022-04-21	-43.705724	-37.782619	-26.105218	13.804820
974	2022-04-22	-43.948830	-35.624793	-25.417428	14.373226
975	2022-04-23	-44.775484	-36.112726	-24.593706	14.960976
976	2022-04-24	-44.390465	-36.597568	-25.960654	15.231098
977	2022-04-25	-44.124158	-35.449020	-26.256403	14.554672
978	2022-04-26	-43.787155	-32.231542	-23.635263	15.059662
979	2022-04-27	-43.446441	-32.079368	-23.743243	14.329340
980	2022-04-28	-42.585329	-30.558988	-23.094814	15.005021
981	2022-04-29	-41.927229	-29.851797	-22.484511	15.606392

```
982  2022-04-30 -40.432236 -28.679688 -21.350023  16.526730
983  2022-05-01 -39.727727 -29.900442 -20.075486  16.371484
984  2022-05-02 -38.433593 -29.665018 -20.016460  15.524179
985  2022-05-03 -40.108432 -29.816410 -20.053582  15.825901
986  2022-05-04 -41.125896 -29.777453 -20.701162  15.273460
987  2022-05-05 -42.305817 -28.186017 -20.416914  16.431363
988  2022-05-06 -41.530917 -28.890207 -18.720874  16.904327
989  2022-05-07 -40.495595 -31.051429 -19.713676  16.879255
990  2022-05-08 -39.210233 -32.562016 -20.680950  17.791731
991  2022-05-09 -39.441070 -33.387647 -19.635072  18.340815
992  2022-05-10 -39.571472 -33.718997 -20.707962  17.894646
993  2022-05-11 -39.164793 -34.306108 -20.772325  18.980810
994  2022-05-12 -39.785943 -33.419699 -20.702588  19.356781
995  2022-05-13 -38.782627 -33.519556 -21.117958  18.636167
996  2022-05-14 -37.672782 -33.244602 -22.166996  17.095003
997  2022-05-15 -37.262084 -33.519251 -22.783061  15.825166
998  2022-05-16 -36.399520 -33.362399 -21.874367  16.140229
999  2022-05-17 -36.634692 -34.768721 -19.929670  15.806925

[1000 rows x 5 columns]
```

[102]:
```
### HDF5
# pip install tables
# 写入 HDF5
df13.to_hdf('./best.h5','df')
# 从 HDF5 读数据
pd.read_hdf('./best.h5','df')
```

[102]:

	A	B	C	D
2019-08-22	0.086278	-0.285369	-0.012890	0.452216
2019-08-23	1.412777	0.238720	-1.740038	-0.123636
2019-08-24	0.799518	-0.888554	-0.924861	-1.706895
2019-08-25	1.564615	-1.646619	0.184296	-2.976331
2019-08-26	1.903628	-1.025881	1.386930	-2.719044
2019-08-27	1.164920	-0.724013	1.236690	-2.495277
2019-08-28	2.003877	0.348358	2.549467	-5.438851
2019-08-29	4.304125	-0.459755	1.763327	-4.409135
2019-08-30	4.726267	-1.517907	-0.212574	-5.101913
2019-08-31	5.277158	-1.264182	0.906538	-5.894793
2019-09-01	5.747264	0.225777	-0.257817	-6.596933

2019-09-02	5.728967	-0.359076	-0.272538	-6.594936
2019-09-03	7.232054	-1.490280	-0.951729	-7.434688
2019-09-04	7.329095	-0.181231	-1.309576	-6.376228
2019-09-05	6.846514	-0.814378	-1.398260	-3.829608
2019-09-06	8.307232	-0.995645	-0.609413	-3.974617
2019-09-07	9.228199	0.449028	-0.260423	-4.760922
2019-09-08	9.240231	1.876106	-0.785967	-3.308362
2019-09-09	11.499487	2.060554	-0.621835	-2.734745
2019-09-10	11.136724	1.607395	-1.830549	-3.678861
2019-09-11	11.324174	0.234021	0.436278	-4.044490
2019-09-12	11.688123	0.480685	0.045486	-4.981361
2019-09-13	10.397040	0.789385	-0.474993	-6.011545
2019-09-14	9.622623	0.783358	-0.732386	-6.763026
2019-09-15	8.798661	0.350506	-1.422821	-9.280633
2019-09-16	8.830095	1.255681	-2.513342	-9.215975
2019-09-17	8.816481	1.816570	-1.076567	-9.238743
2019-09-18	8.880187	2.828825	-2.547987	-8.452422
2019-09-19	8.966015	3.723613	-2.234985	-7.050768
2019-09-20	9.232280	4.353496	-1.996265	-7.063971
...	...	...	...	...
2022-04-18	-45.275887	-38.456353	-26.531187	14.421749
2022-04-19	-44.002236	-37.696672	-25.418910	14.249406
2022-04-20	-44.642954	-37.883943	-25.814418	12.476328
2022-04-21	-43.705724	-37.782619	-26.105218	13.804820
2022-04-22	-43.948830	-35.624793	-25.417428	14.373226
2022-04-23	-44.775484	-36.112726	-24.593706	14.960976
2022-04-24	-44.390465	-36.597568	-25.960654	15.231098
2022-04-25	-44.124158	-35.449020	-26.256403	14.554672
2022-04-26	-43.787155	-32.231542	-23.635263	15.059662
2022-04-27	-43.446441	-32.079368	-23.743243	14.329340
2022-04-28	-42.585329	-30.558988	-23.094814	15.005021
2022-04-29	-41.927229	-29.851797	-22.484511	15.606392
2022-04-30	-40.432236	-28.679688	-21.350023	16.526730
2022-05-01	-39.727727	-29.900442	-20.075486	16.371484
2022-05-02	-38.433593	-29.665018	-20.016460	15.524179
2022-05-03	-40.108432	-29.816410	-20.053582	15.825901
2022-05-04	-41.125896	-29.777453	-20.701162	15.273460
2022-05-05	-42.305817	-28.186017	-20.416914	16.431363
2022-05-06	-41.530917	-28.890207	-18.720874	16.904327

```
2022-05-07 -40.495595 -31.051429 -19.713676  16.879255
2022-05-08 -39.210233 -32.562016 -20.680950  17.791731
2022-05-09 -39.441070 -33.387647 -19.635072  18.340815
2022-05-10 -39.571472 -33.718997 -20.707962  17.894646
2022-05-11 -39.164793 -34.306108 -20.772325  18.980810
2022-05-12 -39.785943 -33.419699 -20.702588  19.356781
2022-05-13 -38.782627 -33.519556 -21.117958  18.636167
2022-05-14 -37.672782 -33.244602 -22.166996  17.095003
2022-05-15 -37.262084 -33.519251 -22.783061  15.825166
2022-05-16 -36.399520 -33.362399 -21.874367  16.140229
2022-05-17 -36.634692 -34.768721 -19.929670  15.806925

[1000 rows x 4 columns]
```

[103]:
```
### Excel
# 写入 excel 文件
df13.to_excel('./best.xlsx',sheet_name='best')
# 从 excel 文件读取数据
pd.read_excel('./best.xlsx','best',index_col=None,na_values=['NA'])
```

[103]:

	Unnamed: 0	A	B	C	D
0	2019-08-22	0.086278	-0.285369	-0.012890	0.452216
1	2019-08-23	1.412777	0.238720	-1.740038	-0.123636
2	2019-08-24	0.799518	-0.888554	-0.924861	-1.706895
3	2019-08-25	1.564615	-1.646619	0.184296	-2.976331
4	2019-08-26	1.903628	-1.025881	1.386930	-2.719044
5	2019-08-27	1.164920	-0.724013	1.236690	-2.495277
6	2019-08-28	2.003877	0.348358	2.549467	-5.438851
7	2019-08-29	4.304125	-0.459755	1.763327	-4.409135
8	2019-08-30	4.726267	-1.517907	-0.212574	-5.101913
9	2019-08-31	5.277158	-1.264182	0.906538	-5.894793
10	2019-09-01	5.747264	0.225777	-0.257817	-6.596933
11	2019-09-02	5.728967	-0.359076	-0.272538	-6.594936
12	2019-09-03	7.232054	-1.490280	-0.951729	-7.434688
13	2019-09-04	7.329095	-0.181231	-1.309576	-6.376228
14	2019-09-05	6.846514	-0.814378	-1.398260	-3.829608
15	2019-09-06	8.307232	-0.995645	-0.609413	-3.974617
16	2019-09-07	9.228199	0.449028	-0.260423	-4.760922
17	2019-09-08	9.240231	1.876106	-0.785967	-3.308362
18	2019-09-09	11.499487	2.060554	-0.621835	-2.734745

19	2019-09-10	11.136724	1.607395	-1.830549	-3.678861
20	2019-09-11	11.324174	0.234021	0.436278	-4.044490
21	2019-09-12	11.688123	0.480685	0.045486	-4.981361
22	2019-09-13	10.397040	0.789385	-0.474993	-6.011545
23	2019-09-14	9.622623	0.783358	-0.732386	-6.763026
24	2019-09-15	8.798661	0.350506	-1.422821	-9.280633
25	2019-09-16	8.830095	1.255681	-2.513342	-9.215975
26	2019-09-17	8.816481	1.816570	-1.076567	-9.238743
27	2019-09-18	8.880187	2.828825	-2.547987	-8.452422
28	2019-09-19	8.966015	3.723613	-2.234985	-7.050768
29	2019-09-20	9.232280	4.353496	-1.996265	-7.063971
..	...	...	...	...	...
970	2022-04-18	-45.275887	-38.456353	-26.531187	14.421749
971	2022-04-19	-44.002236	-37.696672	-25.418910	14.249406
972	2022-04-20	-44.642954	-37.883943	-25.814418	12.476328
973	2022-04-21	-43.705724	-37.782619	-26.105218	13.804820
974	2022-04-22	-43.948830	-35.624793	-25.417428	14.373226
975	2022-04-23	-44.775484	-36.112726	-24.593706	14.960976
976	2022-04-24	-44.390465	-36.597568	-25.960654	15.231098
977	2022-04-25	-44.124158	-35.449020	-26.256403	14.554672
978	2022-04-26	-43.787155	-32.231542	-23.635263	15.059662
979	2022-04-27	-43.446441	-32.079368	-23.743243	14.329340
980	2022-04-28	-42.585329	-30.558988	-23.094814	15.005021
981	2022-04-29	-41.927229	-29.851797	-22.484511	15.606392
982	2022-04-30	-40.432236	-28.679688	-21.350023	16.526730
983	2022-05-01	-39.727727	-29.900442	-20.075486	16.371484
984	2022-05-02	-38.433593	-29.665018	-20.016460	15.524179
985	2022-05-03	-40.108432	-29.816410	-20.053582	15.825901
986	2022-05-04	-41.125896	-29.777453	-20.701162	15.273460
987	2022-05-05	-42.305817	-28.186017	-20.416914	16.431363
988	2022-05-06	-41.530917	-28.890207	-18.720874	16.904327
989	2022-05-07	-40.495595	-31.051429	-19.713676	16.879255
990	2022-05-08	-39.210233	-32.562016	-20.680950	17.791731
991	2022-05-09	-39.441070	-33.387647	-19.635072	18.340815
992	2022-05-10	-39.571472	-33.718997	-20.707962	17.894646
993	2022-05-11	-39.164793	-34.306108	-20.772325	18.980810
994	2022-05-12	-39.785943	-33.419699	-20.702588	19.356781
995	2022-05-13	-38.782627	-33.519556	-21.117958	18.636167
996	2022-05-14	-37.672782	-33.244602	-22.166996	17.095003

```
997 2022-05-15 -37.262084 -33.519251 -22.783061  15.825166
998 2022-05-16 -36.399520 -33.362399 -21.874367  16.140229
999 2022-05-17 -36.634692 -34.768721 -19.929670  15.806925

[1000 rows x 5 columns]
```

[104]:
```
# Gotchas 坑
# 异常
if pd.Series([False,True,False]):
    print("I was true")
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-104-c92ce82998ee> in <module>
  1 # Gotchas 坑
  2 # 异常
----> 3 if pd.Series([False,True,False]):
  4     print("I was true")


f:\pythonproject\env\scriptenv\lib\site-packages\pandas\core\generic.py in␣
↪__nonzero__(self)
   1476         raise ValueError("The truth value of a {0} is ambiguous. "
   1477                          "Use a.empty, a.bool(), a.item(), a.any() or a.
↪all()."
-> 1478                          .format(self.__class__.__name__))
   1479
   1480     __bool__ = __nonzero__


ValueError: The truth value of a Series is ambiguous. Use a.empty, a.
↪bool(), a.item(), a.any() or a.all().
```