TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT THÀNH PHỐ HỒ CHÍ MINH
KHOA ĐIỆN - ĐIỆN TỬ

BÁO CÁO GIỮA KỲ

CHAPTER 5: GETTING STARTED WITH PANDAS

Môn học: Cơ sở dữ liệu khoa học
Giảng viên: TS.Nguyễn Mạnh Hùng

Danh sách sinh viên thực hiện

Mã số SV	Họ và tên
20139060	Phan Ngọc Anh
20139061	Trần Đình Hoàng Anh
20139069	Phan Tấn Đạt
20139072	Nguyễn Lương Phú Gia
20139084	Lê Đức Phong

TP. Hồ Chí Minh,18 tháng 11 năm 2022

NHẬN XÉT CỦA GIẢNG VIÊN

. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

Contents

Getting Started with pandas

Pandas là một thư viện chứa các cấu trúc dữ liệu cấp cao và các công cụ thao tác được thiết kế để giúp phân tích dữ liệu nhanh chóng và dễ dàng bằng Python. Pandas được xây dựng dựa trên Numpy và giúp nó dễ dàng sử dụng trong các ứng dụng chủ yếu là Numpy.
Pandas có thể làm tốt các công việc:
- Cấu trúc dữ liệu với các trục được gắn nhãn hỗ trợ căn chỉnh dữ liệu tự động và rõ ràng. Điều này ngăn ngừa các lỗi phổ biến – những kết quả từ dữ liệu bị lệch và hoạt động với dữ liệu được lập chỉ mục khác nhau đến từ các nguồn khác nhau
- Tích hợp các chức năng time series
- Các cấu trúc dữ liệu giống nhau xử lý cùng dữ liệu chuỗi thời gian(time series) và dữ liệu chuỗi không thời gian(non – time series)
- Các phép toán số học và rút gọn (như tính tổng trên một trục) sẽ chuyển sang dữ liệu thô (axis lable)
- Xử lý linh hoạt các dữ liệu bị thiếu
- Hợp nhất và các hoạt động liên quan khác được tìm thấy trong cơ sở dữ liệu, các cơ sở dữ liệu nổi tiếng (Ví dụ: dựa trên SQL)

```
In [1]: from pandas import Series, DataFrame
In [2]: import pandas as pd
```

Khi nào ta thấy pd. trong code, nó đề cập đến Pandas. Series và DataFrame

1 Introduction to pandas Data Structures

Để bắt đầu dữ liệu với pandas, ta sẽ cần biết về hai loại cấu trúc dữ liệu: Series và DataFrame.

1.1 Series

Series là một mảng một chiều chứa một mảng dữ liệu (của bất kỳ kiểu dữ liệu Numpy) và một mảng dữ liệu nhãn được liên kết -được gọi là index của nó. Series đơn giản nhất được hình thành từ một mảng dữ liệu:

```
In [4]: obj = Series([4, 7, -5, 3])
In [5]: obj
Out[5]:
0   4
1   7
2   -5
3   3
```

Khi sử dụng thì chuỗi của Series được hiển thị bao gồm chỉ mục ở bên trái và các giá trị ở bên phải. Vì ở đây ta không chỉ định chỉ mục cho dữ liệu, nên mặc định nó bao gồm các số nguyên từ 0 đến N-1 (N là độ dài của dữ liệu). Bạn có thể lấy mảng và chỉ mục của đối tượng thuộc Series thông qua thuộc tính values và index của đối tượng đó:

```
In [6]: obj.values
Out[6]: array([ 4, 7, -5, 3])

In [7]: obj.index
Out[7]: Int64Index([0, 1, 2, 3])
```

Thông thường, ta nên tạo một Series với một chỉ mục xác định cho mỗi điểm dữ liệu

```
In [8]: obj2 = Series([4, 7, -5, 3], index=['d', 'b', 'a', 'c'])

In [9]: obj2
Out[9]:

d   4
b   7
a   -5
c   3

In [10]: obj2.index
Out[10]: Index([d, b, a, c], dtype=object)
```

So với mảng Numpy thông thường, ta có thể sử dụng các giá trị trong chỉ mục khi chọn các giá trị đơn lẻ hoặc một tập hợp các giá trị:

```
In [11]: obj2['a']
Out[11]: -5

In [12]: obj2['d'] = 6

In [13]: obj2[['c', 'a', 'd']]
Out[13]:
c 3
a -5
d 6
```

Các phép toán mảng NumPy, chẳng hạn như bộ lọc với một mảng boolean, phép nhân vô hướng hay áp dụng các hàm toán học, sẽ bảo toàn liên kết chỉ mục - giá trị:

```
In [14]: obj2
Out[14]:
d 6
b 7
a -5
c 3
```

```
In [15]: obj2[obj2 > 0]
Out[15]:
d 6
b 7
c 3
```

```
In [16]: obj2 * 2
Out[16]:
d 12
b 14
a -10
c 6
```

```
In [17]: np.exp(obj2)
Out[17]:
d     403.428793
b     1096.633158
a     0.006738
c     20.085537
```

Có thể nói Series là một độ dài cố định, có thứ tự và nó là một ánh xạ của giá trị chỉ mục tới giá trị dữ liệu. Nó có thể được thay thế bằng nhiều chức năng ví dụ như dict:

```
In [18]: 'b' in obj2
Out[18]: True

In [19]: 'e' in obj2
Out[19]: False
```

Bạn có thể tạo một Series từ nó bằng cách gán vào dict:

```
In [20]: sdata = {'Ohio': 35000,
        'Texas': 71000, 'Oregon': 16000, 'Utah': 5000}

In [21]: obj3 = Series(sdata)

In [22]: obj3
Out[22]:
Ohio 35000
Oregon 16000
Texas 71000
Utah 5000
```

Khi chỉ truyền một dict, chỉ mục ở kết quả trả về của Series sẽ có các dict's keys theo thứ tự được sắp xếp.

```
In [23]: states = ['California', 'Ohio', 'Oregon', 'Texas']

In [24]: obj4 = Series(sdata, index=states)

In [25]: obj4
Out[25]:
California      NaN
Ohio            35000
Oregon          16000
Texas           71000
```

Trong trường hợp này, ba giá trị được tìm thấy trong sdata - đã được đặt ở các vị trí thích hợp ngoài trừ 'California', nó xuất hiện dưới dạng NaN(not a number) được coi là non-sidered trong pandas để đánh dấu các giá trị bị thiếu hoặc các giá trị NA. Ta sẽ sử dụng các thuật ngữ "missing" or "NA" để chỉ các dữ liệu bị thiếu. Các hàm isnull và notnull trong pandas nên được sử dụng để phát hiện dữ liệu bị thiếu:

```
In [26]: pd.isnull(obj4)    In [27]: pd.notnull(obj4)
Out[26]:                    Out[27]:
California      True        California      False
Ohio            False       Ohio            True
Oregon          False       Oregon          True
Texas           False       Texas           True
```

Series còn có dưới dạng phương thức instace

```
In [28]: obj4.isnull()
Out[28]:
California      True
Ohio            False
Oregon          False
Texas           False
```

Ta bàn luận về dữ liệu thị thiếu chi tiết hơn ở chương sau
Một tính năng quan trọng của Series đối với các ứng dụng là nó tự động căn chỉnh dữ liệu được lập chỉ mục khác nhau trong các phép toán số học:

```
In [29]: obj3
Out[29]:
Ohio 35000
Oregon 16000
Texas 71000
Utah 5000
```

```
In [30]: obj4
Out[30]:
California NaN
Ohio 35000
Oregon 16000
Texas 71000
```

```
In [31]: obj3 + obj4
Out[31]:
California      NaN
Ohio            70000
Oregon          32000
Texas           142000
Utah            NaN
```

Các tính năng căn chỉnh dữ liệu được đề cập tới như một chủ đề riêng biệt
Cả bản thân đối tượng Series và chỉ mục của nó đều có thuộc tính name, thuộc tính này được tích hợp với các lĩnh vực chính khác của chức năng Pandas:

```
In [32]: obj4.name = 'population'
In [33]: obj4.index.name = 'state'
In [34]: obj4
Out[34]:
state
California      NaN
Ohio            35000
Oregon          16000
Texas           71000
Name: population
```

Một chỉ mục của Series có thể được thay đổi tại chỗ bằng cách gán:

```
In [35]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
In [36]: obj
Out[36]:
Bob     4
Steve   7
Jeff    -5
Ryan    3
```

1.2 DataFrame

Một DataFrame đại diện cho một cấu trúc dữ liệu dạng bảng giống như bảng tính chứa một tập hợp các cột có thứ tự, mỗi cột có thể là một kiểu giá trị khác nhau (số, chuỗi, boolean, v.v). DataFrame có cả chỉ mục hàng và cột; nó có thể được coi như một dict của Series (một cho tất cả chia sẻ cùng một chỉ mục). So sánh với các cấu trúc giống DataFrame khác mà bạn có thể đã sử dụng trước đây (như R's data.frame), các hoạt động định hướng theo hàng và hướng cột trong DataFrame được xử lý gần như đối xứng.
Bên dưới, dữ liệu được lưu trữ dưới dạng một hoặc nhiều khối hai chiều chứ không phải list, dict hoặc một số tập hợp mảng một chiều khác.
Trong khi DataFrame lưu trữ dữ liệu một cashc nội bộ ở định dạng hai chiều, bạn có thể dễ dàng biểu diễn dữ liệu nhiều chiều hơn trong một bảng định dạng bằng cách sử dụng lập chỉ mục phân cấp, những phần tiếp theo sẽ xử lý dữ liệu nâng cao hơn ở Pandas.
Có nhiều cách để xây dựng DataFrame, mặc dù một trong những cách phổ biến nhất là từ một dict của các lists cùng chiều dài hoặc Numpy arrays

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
'year': [2000, 2001, 2002, 2001, 2002],
'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame(data)
```

Kết quả DataFrame sẽ được chỉ định tự động như với Series và các cột được sắp xếp theo thứ tự

```
In [38]: frame
Out[38]:
    pop     state   year
0   1.5     Ohio    2000
1   1.7     Ohio    2001
2   3.6     Ohio    2002
3   2.4     Nevada  2001
4   2.9     Nevada  2002
```

Nếu bạn chỉ định một chuỗi các cột, thì các cột của DataFrame sẽ chính xác là những gì bạn thêm vào

```
In [39]: DataFrame(data, columns=['year', 'state', 'pop'])
Out[39]:
    year    state   pop
0   2000    Ohio    1.5
1   2001    Ohio    1.7
2   2002    Ohio    3.6
3   2001    Nevada  2.4
4   2002    Nevada  2.9
```

Như với Series, nếu bạn chuyển một cột không có trong data, no sẽ xuất hiện giá trị NA trong kết quả trả về:

```
In [40]: frame2 = DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
....: index=['one', 'two', 'three', 'four', 'five'])

In [41]: frame2
Out[41]:
```

```
        year    state   pop     debt
one     2000    Ohio    1.5     NaN
two     2001    Ohio    1.7     NaN
three   2002    Ohio    3.6     NaN
four    2001    Nevada  2.4     NaN
five    2002    Nevada  2.9     NaN

In [42]: frame2.columns
Out[42]: Index([year, state, pop, debt], dtype=object)
```

Một cột trong DataFrame có thể được truy xuất dưới dạng Series hoặc bằng ký hiệu giống chính hoặc bằng thuộc tính:

```
In [43]: frame2['state']    In [44]: frame2.year
Out[43]:                    Out[44]:
one         Ohio            one 2000
two         Ohio            two 2001
three       Ohio            three 2002
four        Nevada          four 2001
five        Nevada          five 2002
Name: state                 Name: year
```

Lưu ý rằng Series được trả về có cùng chỉ mục với DataFrame và thuộc tính name đã được thiết lập thích hợp
Các hàng cũng có thể được truy xuất theo vị trí hoặc tên bằng một số phương pháp, chẳng hạn như trường lập chỉ mục ix(loc) (nhiều hơn nữa về điều này sau này):

```
In [45]: frame2.ix['three']
Out[45]:
year    2002
state   Ohio
pop     3.6
debt    NaN
Name:   three
```

Các cột có thể được sửa đổi bằng cách phân công. Ví dụ: cột trống 'debt' có thể được gán một giá trị vô hướng hoặc một mảng giá trị:

```
In [46]: frame2['debt'] = 16.5
In [47]: frame2
Out[47]:
        year    state   pop     debt
one     2000    Ohio    1.5     16.5
two     2001    Ohio    1.7     16.5
three   2002    Ohio    3.6     16.5
four    2001    Nevada  2.4     16.5
five    2002    Nevada  2.9     16.5
In [48]: frame2['debt'] = np.arange(5.)
In [49]: frame2
Out[49]:
        year    state   pop     debt
one     2000    Ohio    1.5     0
```

```
two     2001    Ohio    1.7     1
three   2002    Ohio    3.6     2
four    2001    Nevada  2.4     3
five     2002   Nevada  2.9     4
```

Khi gán danh sách hoặc mảng cho một cột, độ dài của giá trị phải khớp với độ dài của DataFrame. Nếu bạn chỉ định một Series, thay vào đó, chuỗi đó sẽ được tuân thủ chính xác với chỉ mục của DataFrame, chèn các giá trị bị thiếu vào bất kỳ lỗ nào:

```
In [50]: val = Series([-1.2, -1.5, -1.7],
            index=['two', 'four', 'five'])
In [51]: frame2['debt'] = val
In [52]: frame2
Out[52]:
        year    state   pop     debt
one     2000    Ohio    1.5     NaN
two     2001    Ohio    1.7     -1.2
three   2002    Ohio    3.6     NaN
four    2001    Nevada  2.4     -1.5
five    2002    Nevada  2.9     -1.7
```

Gán một cột không tồn tại sẽ tạo ra một cột mới. Từ khóa del sẽ xóa các cột như với một dict:

```
In [53]: frame2['eastern'] = frame2.state == 'Ohio'
In [54]: frame2
Out[54]:
        year    state   pop     debt    eastern
one     2000    Ohio    1.5     NaN     True
two     2001    Ohio    1.7     -1.2    True
three   2002    Ohio    3.6     NaN     True
four    2001    Nevada  2.4     -1.5    False
five    2002    Nevada  2.9     -1.7    False
In [55]: del frame2['eastern']
In [56]: frame2.columns
Out[56]: Index([year, state, pop, debt], dtype=object)
```

Cột được trả về khi lập chỉ mục DataFrame là một chế độ xem trên dữ liệu cơ bản, không phải là một bản sao. Do đó, bất kỳ sửa đổi tại chỗ nào đối với Series sẽ được phản ánh trong DataFrame. Có thể sao chép cột một cách rõ ràng bằng phương pháp sao chép của Series
Một dạng dữ liệu phổ biến khác là định dạng dict of dict lồng nhau:

```
In [57]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},
....: 'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

Nếu được chuyển đến DataFrame, nó sẽ diễn giải các khóa dict bên ngoài là các cột và bên trong các phím làm chỉ số hàng:

```
In [58]: frame3 = DataFrame(pop)
In [59]: frame3
Out[59]:
        Nevada  Ohio
2000    NaN     1.5
2001    2.4     1.7
2002    2.9     3.6
```

Tất nhiên, bạn luôn có thể chuyển đổi kết quả:

```
In [60]: frame3.T
Out[60]:
        2000    2001    2002
Nevada  NaN     2.4     2.9
Ohio    1.5     1.7     3.6
```

Các keys trong các ô bên trong được liên kết và sắp xếp để tạo thành chỉ mục trong kết quả:

```
In [61]: DataFrame(pop, index=[2001, 2002, 2003])
Out[61]:
        Nevada  Ohio
2001    2.4     1.7
2002    2.9     3.6
2003    NaN     NaN
```

Dicts of Series được đối xử trong cùng một cách:

```
In [62]: pdata = {'Ohio': frame3['Ohio'][:-1],
....: 'Nevada': frame3['Nevada'][:2]}
In [63]: DataFrame(pdata)
Out[63]:
        Nevada  Ohio
2000    NaN     1.5
2001    2.4     1.7
```

Để có danh sách đầy đủ những thứ bạn có thể chuyển hàm tạo DataFrame, hãy xem Bảng 5-1 Nếu chỉ mục và các cột của DataFrame có thuộc tính tên được đặt, thì những thuộc tính này cũng sẽ hiển thị:

```
In [64]: frame3.index.name = 'year'; frame3.columns.name = 'state'
In [65]: frame3
Out[65]:
state   Nevada  Ohio
year
2000    NaN     1.5
2001    2.4     1.7
    2002 2.9    3.6
```

Giống như Series, thuộc tính values trả về dữ liệu có trong DataFrame dưới dạng 2D

```
In [66]: frame3.values
Out[66]:
array([ [ nan, 1.5],
        [ 2.4, 1.7],
        [ 2.9, 3.6]] )
```

Nếu các cột của DataFrame là các kiểu khác nhau, thì kiểu của mảng giá trị sẽ là được chọn để chứa tất cả các cột:

```
In [67]: frame2.values
Out[67]:
array([ [2000, Ohio, 1.5, nan],
        [2001, Ohio, 1.7, -1.2],
        [2002, Ohio, 3.6, nan],
        [2001, Nevada, 2.4, -1.5],
        [2002, Nevada, 2.9, -1.7]], dtype=object )
```

Table 1: Đầu vào dữ liệu có thể có cho hàm khởi tạo DataFrame

Loại	Lưu ý
2D ndarray	Là một ma trận dữ liệu, truyền vào các nhãn hàng và cột tùy
Dict of arrays,list, hoặc tuples	Mỗi trình tự trở thành một cột trong DataFrame. Tất cả trình tự phải là cùng kích thước
Cấu trúc Numpy	Được coi là trường hợp 'dict of arrays'
Dict of Series	Mỗi giá trị trở thành một cột. Các chỉ mục từ mỗi Series được hợp nhất với nhau để tạo thành chỉ mục hàng của kết quả nếu không có chỉ mục rõ ràng nào được thêm vào
Dict of dicts	Mỗi mệnh đề bên trong trở thành một cột. Keys được hợp nhất để tạo thành chỉ mục hàng như trong trường hợp "dict of Series"
List of dicts or Series	Mỗi mục trở thành một hàng trong DataFrame. Liên hợp các keys dict hoặc chỉ mục Series trở thành nhãn cột của DataFrame
List of list or tuples	Được xử lý như trưởng hợp "2D ndarray"
DataFrame khác	Các chỉ mục của DataFrame được sử dụng trừ khi các chỉ mục khác được thông qua
Mảng mặt nạ Numpy	Giống với trường hợp "2D narray" ngoại trừ giá trị bị che trở thành NA/missing trong kết quả DataFrame

1.3 Index Objects

Các đối tượng chỉ mục của pandas chịu trách nhiệm giữ các axis lable và các dữ liệu lớn khác (như axis name or names). Bất kỳ mảng hoặc sequence of labels được sử dụng khi cấu trúc Series hoặc DataFrame được chuyển đổi thành index phù hợp:

```
In [68]: obj = Series(range(3), index=['a', 'b', 'c'])

In [69]: index = obj.index

In [70]: index
Out[70]: Index([a, b, c], dtype=object)

In [71]: index[1:]
Out[71]: Index([b, c], dtype=object)
```

Các đối tượng Index là cố định và do đó người dùng không thể sửa đổi:

```
In [72]: index[1] = 'd'
Exception                                 Traceback (most recent call last)
<ipython-input-72-676fdeb26a68> in <module>()
----> 1 index[1] = 'd'
/Users/wesm/code/pandas/pandas/core/index.pyc in __setitem__(self, key, value)
302 def __setitem__(self, key, value):
303 """Disable the setting of values."""
--> 304 raise Exception(str(self.__class__) + ' object is immutable')
305
306 def __getitem__(self, key):
Exception: <class 'pandas.core.index.Index'> object is immutable
```

Tính bất biến(Immutability) rất quan trọng để các đối tượng Index có thể được chia sẻ một cách an toàn giữa các dữ liệu.

```
In [73]: index = pd.Index(np.arange(3))

In [74]: obj2 = Series([1.5, -2.5, 0], index=index)

In [75]: obj2.index is index
Out[75]: True
```

Bảng 1-2 Có danh sách các lớp index xây dựng sẵn trong thư viện. Với một số nỗ lực phát triển, Index thậm chí có thể được phân lớp để thực hiện đồng nhất chức năng chuyên dụng của các index.

Nhiều người dùng sẽ không cần biết nhiều về các đối tượng Index Nhưng chúng dù sao cũng là một phần quan trọng trong mô hình dữ liệu của pandas.

Bảng 1-2. Các đối tượng index chính trong index

Class	Description
Index	Đối tượng chỉ mục chung nhất, đại diện cho các nhãn trục trong một mảng NumPy của các đối tượng Python
Int64index	Chỉ mục chuyên biệt cho các giá trị số nguyên.
MultiIndex	"Phân cấp" đại diện cho nhiều mức độ lập chỉ mục trên một trục duy nhất. Có thể nghĩ tương tự như một mảng tuple'
DatetimeIndex	Lưu trữ dấu thời gian nano giây.
PeriodIndex	Chỉ số chuyên biệt PeriodIndex cho dữ liệu khoảng thời gian

Ngoài việc giống như mảng, một index cũng có chức năng như một tập hợp kích thước cố định.

```
In [76]: frame3
Out[76]: state Nevada Ohio
         year
         2000   NaN    1.5
         2001   2.4    1.7
         2002   2.9    3.6

In [77]: 'Ohio' in frame3.columns
Out[77]: True

In [78]: 2003 in frame3.index
Out[78]: False
```

Mỗi index có một số phương thức và thuộc tính cho logic tập hợp và trả lời các câu hỏi phổ biến khác về dữ liệu mà nó chứa. Những điều này được tóm tắt trong Bảng 1-3

Bảng 1-3. Các phương thức chỉ mục và thuộc tính.

Method	Description
append	Concatenate với các đối tượng Index bổ sung, tạo ra một chỉ mục mới
diff	Tính toán sự khác biệt đặt dưới dạng chỉ mục.
union	Compute set union.
isin	Tính toán mảng boolean cho biết mỗi giá trị có được chứa trong bộ sưu tập được truyền hay không.
delete	Tính toán chỉ mục mới với phần tử tại chỉ mục đã xóa.
drop	Tính toán chỉ mục mới bằng cách xóa các giá trị đã truyền.
insert	Tính toán chỉ mục mới bằng cách chèn phần tử tại index i.
ismonotonic	Trả về True nếu mỗi phần tử lớn hơn hoặc bằng phần trước đó.
isunique	Trả về True nếu Chỉ mục không có giá trị trùng lặp.
unique	Tính toán mảng các giá trị duy nhất trong chỉ mục.
itersection	Tính toán tập giá trị giao nhau.

2 Essential Functionality

Trong phần này, tôi sẽ hướng dẫn bạn về cơ chế cơ bản của việc tương tác với dữ liệu có trong Chuỗi hoặc DataFrame. Các chương sắp tới sẽ đào sâu hơn vào các chủ đề phân tích và thao tác dữ liệu bằng cách sử dụng Pandas. Cuốn sách này không nhằm mục đích phục vụ cũng như tài liệu đầy đủ cho thư viện Pandas. Thay vào đó tôi tập trung vào nhiều nhất các tính năng quan trọng, để lại những thứ ít phổ biến hơn để bạn có thể tự khám phá.

2.1 Reindexing

Một phương thức quan trọng trên các đối tượng gấu trúc là reindex, có nghĩa là tạo một đối tượng mới với dữ liệu phù hợp với một chỉ mục mới. Hãy xem xét một ví dụ đơn giản ở trên:

```
In [79]: obj = Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a','c'])

In [80]: obj
Out[80]:
d     4.5
b     7.2
a    -5.3
c     3.6
```

Việc gọi reindex trên Series này sắp xếp lại dữ liệu theo index mới, giới thiệu các giá trị bị thiếu nếu chưa có bất kỳ giá trị chỉ mục nào:

```
In [81]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])

In [82]: obj2
Out[82]:
a   -5.3
b    7.2
c    3.6
d    4.5
e    NaN

In [83]: obj.reindex(['a', 'b', 'c', 'd', 'e'], fill_value=0)
Out[83]:
a    -5.3
b     7.2
c     3.6
d     4.5
e     0.0
```

Đối với dữ liệu có thứ tự như time Series, bạn có thể thực hiện một số nội suy hoặc điền các giá trị khi lập chỉ mục lại. Tùy chọn phương pháp cho phép chúng tôi thực hiện việc này, bằng cách sử dụng một phương pháp như ffill để thực hiện điền vào các giá trị:

```
In [84]: obj3 = Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])

In [85]: obj3.reindex(range(6), method='ffill')
Out[85]:
0    blue
1    blue
2    purple
3    purple
4    yellow
5    yellow
```

Bảng 1-4 Liệt kê các tùy chọn phương pháp có sẵn. Tại thời điểm này, cần phải áp dụng phép nội suy phức tạp hơn so với việc lấp đầy và chuyển tiếp sau khi thực hiện.

Bảng 1-4 . Reindex method (interpolation) options.

Argument	Description
ffill or pad	Điền (hoặc chuyển) các giá trị về phía trước
bfill or backfill	Điền (hoặc mang) các giá trị lùi lại.

Với DataFrame, reindex có thể thay đổi index (hàng), cột hoặc cả hai. Khi chỉ được truyền một chuỗi, các hàng được lập index lại trong kết quả:

```
In [86]: frame =DataFrame(np.arange(9).reshape((3,3)),index=['a','c','d'],
 ....: columns=['Ohio', 'Texas', 'California'])

In [87]: frame
Out[87]:
Ohio Texas California
a 0 1 2
c 3 4 5
d 6 7 8

In [88]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])

In [89]: frame2
Out[89]:
Ohio Texas California
a    0      1    2
b    NaN    NaN  NaN
c    3      4    5
d    6      7    8
```

Các cột có thể được lập index lại bằng cách sử dụng từ khóa cột:

```
In [90]: states = ['Texas', 'Utah', 'California']

In [91]: frame.reindex(columns=states)
```

```
Out[91]:
Texas Utah California
a   1   NaN  2
c   4   NaN  5
d   7   NaN  8
```

Cả hai đều có thể được lập chỉ mục lại trong một lần chụp, mặc dù phép nội suy sẽ chỉ áp dụng theo hàng (axis0):

```
In [92]: frame.reindex(index=['a', 'b', 'c', 'd'], method='ffill',
....: columns=states)
Out[92]:
    Texas Utah California
a   1     NaN     2
b   1     NaN     2
c   4     NaN     5
d   7     NaN     8
```

Như bạn sẽ thấy ngay sau đây, việc lập chỉ mục lại có thể được thực hiện ngắn gọn hơn bằng cách lập chỉ mục nhãn với ix:

```
In [93]: frame.ix[['a', 'b', 'c', 'd'], states]
Out[93]:
    Texas   Utah    California
a    1      NaN      2
b    NaN    NaN      NaN
c    4      NaN      5
d    7      NaN      8
```

Bảng 1-5 . reindex function arguments.

Argument	Description
index	Trình tự mới để sử dụng làm chỉ mục. Có thể là cá thể Index hoặc bất kỳ cấu trúc dữ liệu Python giống chuỗi nào khác. Một chỉ mục sẽ được sử dụng chính xác mà không có bất kỳ sự sao chép nào.
method	Phương pháp nội suy (điền), xem Bảng 1-5 để biết các tùy chọn.
fillvalue	Giá trị thay thế để sử dụng khi giới thiệu dữ liệu bị thiếu bằng cách lặp lại index.
limit	Khi điền trước hoặc lấp đầy, khoảng cách kích thước tối đa để lấp đầy.
level	Đối sánh Chỉ mục đơn giản với cấp độ MultiIndex, nếu không, hãy chọn tập hợp con của nó.
copy	Không sao chép dữ liệu cơ bản nếu chỉ mục mới tương đương với chỉ mục cũ. Đúng theo mặc định (tức là luôn sao chép dữ liệu).

2.2 Dropping entries from an axis

Dễ dàng loại bỏ một hoặc nhiều mục nhập khỏi một trục nếu bạn có mảng chỉ mục hoặc danh sách không có các mục nhập đó. Vì điều đó có thể yêu cầu một chút logic và thiết lập, phương thức drop sẽ trả về một đối tượng mới với giá trị được chỉ định hoặc các giá trị bị xóa khỏi trục:

```
In [94]: obj = Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])

In [95]: new_obj = obj.drop('c')

In [96]: new_obj
Out[96]:
a    0
b    1
d    3
e    4

In [97]: obj.drop(['d', 'c'])
Out[97]:
a    0
b    1
e    4
```

Với DataFrame, các giá trị chỉ mục có thể bị xóa khỏi một trong hai trục:

```
In [98]: data = DataFrame(np.arange(16).reshape((4, 4)),
....:                     index=['Ohio', 'Colorado', 'Utah', 'New York'],
....:                     columns=['one', 'two', 'three', 'four'])

In [99]: data.drop(['Colorado', 'Ohio'])
Out[99]:
          one     two     three   four
Utah      8       9       10      11
New York  12      13      14      15

In [100]: data.drop('two', axis=1)
Out[100]:
          one three four
Ohio      0   2     3
Colorado  4   6     7
Utah      8   10    11
New York  12  14    15

In [101]: data.drop(['two', 'four'], axis=1)
Out[101]:
          one     three
Ohio      0       2
Colorado  4       6
Utah      8       10
New York  12      14
```

2.3 Indexing, selection, and filtering

Lập chỉ mục chuỗi (obj [...]) hoạt động tương tự như lập chỉ mục mảng NumPy, ngoại trừ bạn có thể sử dụng các giá trị chỉ mục của Sê-ri thay vì chỉ sử dụng số nguyên. Dưới đây là một số ví dụ về điều này:

```
In [102]: obj = Series(np.arange(4.), index=['a', 'b', 'c', 'd'])

In [103]: obj['b']
Out[103]: 1.0 Out

In [104]: obj[1]
[104]: 1.0

In [105]: obj[2:4]
Out[105]: Out[106]:
c    2
d    3

In [106]: obj[['b', 'a', 'd']]

   b    1
   a    0
   d    3

In [107]: obj[[1, 3]]
Out[107]:
b    1
d    3

In [108]: obj[obj < 2]
Out[108]:
   a    0
   b    1
```

Cắt bằng nhãn hoạt động khác với cách cắt trong Python bình thường ở điểm cuối là bao gồm:

```
In [109]: obj['b':'c']
Out[109]:
b   1
c   2
```

Cài đặt bằng cách sử dụng các phương pháp này hoạt động giống như bạn mong đợi:

```
In [110]: obj['b':'c'] = 5

In [111]: obj
Out[111]:
a    0
b    5
c    5
d    3
```

Như bạn đã thấy ở trên, lập chỉ mục vào DataFrame là để truy xuất một hoặc nhiều cột với một giá trị hoặc một chuỗi:

```
In [112]: data = DataFrame(np.arange(16).reshape((4, 4)),
.....:                      index=['Ohio', 'Colorado', 'Utah', 'New York'],
.....:                     columns=['one', 'two', 'three', 'four'])

In [113]: data
Out[113]:
          one  two   three   four
Ohio       0   1     2       3
Colorado  4    5     6       7
Utah       8   9     10      11
New York  12   13    14      15

In [114]: data['two']                          In [115]: data[['three', 'one']]
```

```
Out[114]:
Ohio         1
Colorado     5
Utah         9
New York     13
Name:        two
```

```
Out[115]:
             three   one
Ohio     2       0
Colorado 6       4
Utah     10      8
New York 14      12
```

Việc lập chỉ mục như thế này có một vài trường hợp đặc biệt. Đầu tiên chọn hàng bằng cách cắt hoặc boolean mảng:

```
In [116]: data[:2]
Out[116]:
          one  two  three  four
Ohio      0    1    2      3
Colorado  4    5    6      7
```

```
In [117]: data[data['three'] > 5]
Out[117]:
                  one  two  three  four
Colorado      4     5   6      7
Utah          8     9   10     11
New York      12    13  14     15
```

Điều này có vẻ không phù hợp với một số độc giả, nhưng cú pháp này đã phát sinh tính thực tế và không có gì hơn. Một trường hợp sử dụng khác là lập chỉ mục với DataFrame boolean, chẳng hạn như một được tạo ra bởi một so sánh.

```
In [118]: data < 5
Out[118]: one  two  three  four
Ohio      True  True  True  True
Colorado  True  False False False
Utah      False False False False
New York  False False False False

In [119]: data[data < 5] = 0

In [120]: data
Out[120]:
         one  two  three  four
Ohio     0    0    0      0
Colorado 0    5    6      7
Utah     8    9    10    11
New York 12   13   14    15
```

Điều này nhằm mục đích làm cho DataFrame về mặt cú pháp giống một ndarray hơn trong trường hợp này. Để lập chỉ mục nhãn DataFrame trên các hàng, tôi giới thiệu trường lập chỉ mục đặc biệt ix. Nó cho phép bạn chọn một tập hợp con của các hàng và cột từ DataFrame với NumPy như ký hiệu cộng với nhãn trục. Như tôi đã đề cập trước đó, đây cũng là một cách ít dài dòng hơn để lập lại chỉ mục:

```
In [121]: data.ix['Colorado', ['two', 'three']]
Out[121]:
two     5
three   6
Name: Colorado
```

```
In [122]: data.ix[['Colorado', 'Utah'], [3, 0, 1]]
Out[122]:
          four one two
Colorado    7    0   5
Utah        11   8   9
```

```
In [123]: data.ix[2]
Out[123]:
one     8
two     9
three   10
four    11
Name: Utah
```

```
In [124]: data.ix[:'Utah', 'two']
Out[124]:
   Ohio     0
   Colorado 5
   Utah     9
   Name: two
```

```
In [125]: data.ix[data.three > 5, :3]
Out[125]:
          one two three
Colorado    0   5   6
Utah        8   9   10
New York    12  13  14
```

Vì vậy, có nhiều cách để chọn và sắp xếp lại dữ liệu có trong một đối tượng gấu trúc. Đối với DataFrame, có một bản tóm tắt ngắn về nhiều trong số chúng trong Bảng 5-6. Bạn có một số lượng các tùy chọn bổ sung khi làm việc với các chỉ mục phân cấp như bạn sẽ làm sau này hiểu.

Khi thiết kế gấu trúc, tôi cảm thấy rằng phải nhập frame [:, col] để chọn một cột quá dài dòng (và dễ xảy ra lỗi), vì lựa chọn cột là một trong những hoạt động phổ biến nhất. Vì vậy, tôi đã đánh đổi thiết kế để đẩy tất cả lập chỉ mục nhãn chi tiết vào ix.

Bảng 1-6. Indexing options with DataFrame

Type	Notes
obj[val]	Chọn một cột hoặc chuỗi cột từ DataFrame. Trường hợp đặc biệt con veniences: mảng boolean (hàng lọc), lát cắt (hàng lát cắt) hoặc boolean DataFrame (bộ giá trị dựa trên một số tiêu chí).
obj.ix[val]	Chọn một hàng của tập hợp con các hàng từ DataFrame.
obj.ix[:, val]	"Phân cấp" đại diện cho nhiều mức độ lập chỉ mục trên một trục duy nhất. Có thể nghĩ tương tự như một mảng tuple’
obj.ix[val1, val2]	Chọn một cột trong tập hợp con của các cột.
reindex method	Chuyển đổi một hoặc nhiều trục thành các chỉ mục mới.
xs method	Chọn hàng hoặc cột đơn làm chuỗi theo nhãn.
icol, irow methods	Chọn một cột hoặc hàng, tương ứng, làm chuỗi theo vị trí số nguyên.
Getvalue,setvalue method	Chọn giá trị đơn lẻ theo nhãn hàng và cột.

2.4 Arithmetic and data alignment

Một trong những tính năng quan trọng nhất của pandas là hành vi số học giữa các đối tượng có chỉ số khác nhau. Khi cộng các đối tượng lại với nhau, nếu bất kỳ cặp index nào không giống nhau, chỉ mục tương ứng trong kết quả sẽ là sự hợp nhất của các cặp chỉ mục. Hãy nhìn vào một ví dụ đơn giản:

```
In [126]: s1 = Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])

In [127]: s2 = Series([-2.1, 3.6, -1.5, 4, 3.1], index=['a', 'c', 'e', 'f','g'])

In [128]: s1          In [129]: s2
Out[128]:             Out[129]:
a   7.3               a   -2.1
c   -2.5              c   3.6
d   3.4               e   -1.5
e   1.5               f   4.0
                      g   3.1
```

Cộng lại với nhau ta có:

```
In [130]: s1 + s2
Out[130]:
a 5.2
c 1.1
d NaN
e 0.0
f NaN
g NaN
```

Sự liên kết dữ liệu nội bộ giới thiệu các giá trị NA (not available) trong các chỉ số không trùng lặp. Các giá trị bị thiếu có giá trị phổ biến trong các phép tính số học. Trong trường hợp có DataFrame, căn chỉnh được thực hiện trên cả hàng và cột:

```
In [131]: df1 = DataFrame(np.arange(9.).reshape((3,3)),columns=list('bcd'),
 .....:                   index=['Ohio', 'Texas', 'Colorado'])

In [132]: df2 = DataFrame(np.arange(12.).reshape((4,3)),columns=list('bde'),
.....:                   index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [133]: df1              In [134]: df2
Out[133]:                  Out[134]:
          b  c  d                    b  d  e
Ohio      0  1  2          Utah      0  1  2
Texas     3  4  5          Ohio      3  4  5
Colorado  6  7  8          Texas     6  7  8
                           Oregon    9  10 11
```

Thêm các dữ liệu này lại với nhau sẽ trả về một DataFrame có chỉ mục và các cột là sự kết hợp của các giá trị trong mỗi DataFrame:

```
In [135]: df1 + df2
Out[135]:
          b   c   d  e
Colorado  NaN NaN NaN NaN
```

```
Ohio      3   NaN 6   NaN
Oregon    NaN NaN NaN NaN
Texas     9   NaN 12  NaN
Utah      NaN NaN NaN NaN
```

Arithmetic methods with fill values

Trong các phép toán số học giữa các đối tượng được lập index khác nhau, bạn có thể muốn điền vào một giá trị đặc biệt, chẳng hạn như 0, khi nhãn trục được tìm thấy trong một đối tượng chứ không phải đối tượng khác:

```
In [136]: df1 = DataFrame(np.arange(12.).reshape((3,4)),columns=list('abcd')

In [137]: df2 = DataFrame(np.arange(20.).reshape((4,5)),columns=list('abcde)
```

```
In [138]: df1
Out[138]:
   a  b  c  d
0  0  1  2  3
1  4  5  6  7
2  8  9  10 11
```

```
In [139]: df2
Out[139]:
   a  b  c  d  e
0  0  1  2  3  4
1  5  6  7  8  9
2  10 11 12 13 14
3  15 16 17 18 19
```

Việc thêm các giá trị này lại với nhau sẽ tạo ra các giá trị NA ở các vị trí không trùng lặp:

```
In [140]: df1 + df2
Out[140]:

    a   b   c   d   e
0   0   2   4   6 NaN
1   9  11  13  15 NaN
2  18  20  22  24 NaN
3 NaN NaN NaN NaN NaN
```

Sử dụng phương thức add trên df1, chuyển df2 và một đối số tới fill_value:

```
In [141]: df1.add(df2, fill_value=0)
Out[141]:
   a   b   c   d   e
0  0   2   4   6   4
1  9  11  13  15   9
2 18  20  22  24  14
3 15  16  17  18  19
```

Việc thêm các giá trị này lại với nhau sẽ tạo ra các giá trị NA ở các vị trí không trùng lặp:

```
In [142]: df1.reindex(columns=df2.columns, fill_value=0)
Out[142]:
   a b  c  d e
0  0 1  2  3 0
1  4 5  6  7 0
2  8 9 10 11 0
```

Bang 5-7.Flexible arithmetic methods

Method	Description
add	Method for addition (+)
sub	Method for subtraction (-)
div	Method for division (/)
mul	Method for multiplication (*)

Operations between DataFrame and Series

Như với mảng NumPy, số học giữa DataFrame và Series được xác định rõ ràng. Đầu tiên, như một ví dụ thúc đẩy, hãy xem xét sự khác biệt giữa mảng 2 chiều và một trong các hàng của nó:

```
In [143]: arr = np.arange(12.).reshape((3, 4))

In [144]: arr
Out[144]:
array([[ 0.,  1.,  2.,  3.],
       [ 4.,  5.,  6.,  7.],
       [ 8.,  9., 10., 11.]])

In [145]: arr[0]
Out[145]: array([ 0.,  1.,  2.,  3.])

In [146]: arr - arr[0]
Out[146]:
array([[ 0.,  0.,  0.,  0.],
       [ 4.,  4.,  4.,  4.],
       [ 8.,  8.,  8.,  8.]])
```

Điều này được gọi là phát sóng và được giải thích chi tiết hơn trong Chương 12. Các hoạt động giữa DataFrame và Series tương tự:

```
In [147]: frame =DataFrame(np.arange(12.).reshape((4,3)),columns=list('bde'),
   .....:                  index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [148]: series = frame.ix[0]

In [149]: frame
Out[149]:
        b  d  e
Utah    0  1  2
Ohio    3  4  5
Texas   6  7  8
Oregon  9 10 11

In [150]: series
Out[150]:
b   0
d   1
e   2
Name: Utah
```

Theo mặc định, số học giữa DataFrame và Series khớp với chỉ số của Series trên các cột của DataFrame, truyền xuống các hàng:

```
In [151]: frame - series
Out[151]:
        b  d  e
```

```
Utah    0  0  0
Ohio    3  3  3
Texas   6  6  6
Oregon  9  9  9
```

Nếu không tìm thấy giá trị chỉ mục trong các cột của DataFrame hoặc index của Series, thì các đối tượng sẽ được lập index lại để tạo thành liên kết:

```
In [152]: series2 = Series(range(3), index=['b', 'e', 'f'])

In [153]: frame + series2
Out[153]:
        b   d  e    f
Utah    0 NaN  3  NaN
Ohio    3 NaN  6  NaN
Texas   6 NaN  9  NaN
Oregon  9 NaN  12 NaN
```

Thay vào đó, nếu bạn muốn truyền phát trên các cột, khớp trên các hàng, bạn phải sử dụng một trong các phương pháp số học. Ví dụ:

```
In [154]: series3 = frame['d']

In [155]: frame          In [156]: series3
Out[155]:                Out[156]:
        b   d   e        Utah        1
Utah    0   1   2        Ohio        4
Ohio    3   4   5        Texas       7
Texas   6   7   8        Oregon     10
Oregon  9  10  11        Name: d

In [157]: frame.sub(series3, axis=0)
Out[157]:
         b  d   e
Utah    -1  0   1
Ohio    -1  0   1
Texas   -1  0   1
Oregon  -1  0   1
```

Số axis mà bạn vượt qua là axis phù hợp. Trong trường hợp này, chúng tôi muốn đối sánh trên index hàng của DataFrame và phát trên toàn bộ.

2.4.1 Function application and mapping

NumPy ufuncs (element-wise array methods_phép toán trên các phần tử tương ứng giữa các mảng) hoạt động tốt với các đối tượng pandas:

```
In [158]: frame = DataFrame(np.random.randn(4, 3), columns=list('bde'),
 .....:                     index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [159]: frame                          In [160]: np.abs(frame)
Out[159]:                                Out[160]:
              b         d         e                    b         d         e
```

```
Utah   -0.204708  0.478943 -0.519439   Utah    0.204708  0.478943  0.519439
Ohio   -0.555730  1.965781  1.393406   Ohio    0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023   Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189 -1.296221   Oregon  1.246435  1.007189  1.296221
```

Một hoạt động thường xuyên khác là áp dụng một hàm trên mảng 1 chiều cho mỗi cột hoặc hàng. Phương pháp ứng dụng của DataFrame thực hiện chính xác điều này:

```
In [161]: f = lambda x: x.max() - x.min()

In [162]: frame.apply(f)
Out[162]:
b   1.802165
d   1.684034
e   2.689627

In [163]: frame.apply(f, axis=1)
Out[163]:
Utah     0.998382
Ohio     2.521511
Texas    0.676115
Oregon   2.542656
```

Nhiều thống kê mảng phổ biến nhất (như tổng và trung bình) là các phương thức DataFrame, vì vậy việc sử dụng ứng dụng là không cần thiết. Hàm được truyền để áp dụng không cần trả về giá trị vô hướng, nó cũng có thể trả về một chuỗi (Series) có nhiều giá trị:

```
In [164]: def f(x):
.....:        return Series([x.min(), x.max()], index=['min', 'max'])

In [165]: frame.apply(f)
Out[165]:
             b         d          e
min  -0.555730  0.281746  -1.296221
max   1.246435  1.965781   1.393406
```

Các hàm Element-wise Python cũng có thể được sử dụng. Giả sử bạn muốn tính một chuỗi (Series) được định dạng từ mỗi giá trị dấu phẩy động trong frame. Bạn có thể làm điều này với applymap:

```
In [166]: format = lambda x: '%.2f' % x

In [167]: frame.applymap(format)
Out[167]:
           b     d      e
Utah   -0.20  0.48  -0.52
Ohio   -0.56  1.97   1.39
Texas   0.09  0.28   0.77
Oregon  1.25  1.01  -1.30
```

Lý do cho tên applymap là Series có một phương pháp bản đồ để ứng dụng một hàm element-wise.

```
In [168]: frame['e'].map(format)
Out[168]:
Utah    -0.52
Ohio     1.39
Texas    0.77
Oregon  -1.30
Name: e
```

2.4.2 Sorting and ranking

Sắp xếp một tập dữ liệu theo một số tiêu chí là một thao tác quan trọng khác được tích hợp sẵn. Để sắp xếp theo từ điển theo index hàng hoặc cột, hãy sử dụng phương thức sort_index, phương thức này trả về một đối tượng mới, được sắp xếp:

```
In [169]: obj = Series(range(4), index=['d', 'a', 'b', 'c'])

In [170]: obj.sort_index()
Out[170]:
a   1
b   2
c   3
d   0
```

Với DataFrame, bạn có thể sắp xếp theo index trên một trong hai trục:

```
In [171]: frame = DataFrame(np.arange(8).reshape((2,4)),index=['three','one'],
   .....:                   columns=['d', 'a', 'b', 'c'])

In [172]: frame.sort_index()
Out[172]:
       d  a  b  c
one    4  5  6  7
three  0  1  2  3

In [173]: frame.sort_index(axis=1)
Out[173]:
       a  b  c  d
three  1  2  3  0
one    5  6  7  4
```

Dữ liệu được sắp xếp theo thứ tự tăng dần theo mặc định, nhưng cũng có thể được sắp xếp theo thứ tự giảm dần:

```
In [174]: frame.sort_index(axis=1, ascending=False)
Out[174]:
       d  c  b  a
three  0  3  2  1
one    4  7  6  5
```

Để sắp xếp một Series theo các giá trị của nó, hãy sử dụng phương pháp sắp xếp:

```
In [175]: obj = Series([4, 7, -3, 2])

In [176]: obj.order()
Out[176]:
2   -3
3    2
0    4
1    7
```

Mọi giá trị bị thiếu đều được sắp xếp vào cuối Chuỗi (Series) theo mặc định:

```
In [177]: obj = Series([4, np.nan, 7, np.nan, -3, 2])

In [178]: obj.order()
Out[178]:
4    -3
```

```
5     2
0     4
2     7
1   NaN
3   NaN
```

Trên DataFrame, bạn có thể muốn sắp xếp các giá trị theo một hoặc nhiều cột. Để làm như vậy, hãy chuyển một hoặc nhiều tên cột vào tùy chọn bằng:

```
In [179]: frame = DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]})

In [180]: frame
Out[180]:
   a  b
0  0  4
1  1  7
2  0 -3
3  1  2

In [181]: frame.sort_index(by='b')
Out[181]:
   a  b
2  0 -3
3  1  2
0  0  4
1  1  7
```

Để sắp xếp theo nhiều cột, hãy chuyển danh sách các tên:

```
In [182]: frame.sort_index(by=['a', 'b'])
Out[182]:
   a  b
2  0 -3
0  0  4
3  1  2
1  1  7
```

Ranking có liên quan chặt chẽ đến việc sắp xếp, chỉ định thứ hạng từ một thông qua số điểm dữ liệu hợp lệ trong một mảng. Nó tương tự như các chỉ số sắp xếp gián tiếp được tạo bởi numpy.argsort, ngoại trừ việc các mối quan hệ bị phá vỡ theo một quy tắc. Các phương pháp xếp hạng cho Series và DataFrame là nơi để xem xét; theo mặc định, xếp hạng phá vỡ mối quan hệ bằng cách gán cho mỗi nhóm thứ hạng trung bình:

```
In [183]: obj = Series([7, -5, 7, 4, 2, 0, 4])

In [184]: obj.rank()
Out[184]:
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
```

Thứ hạng cũng có thể được chỉ định theo thứ tự chúng ta quan sát trong dữ liệu:

```
In [185]: obj.rank(method='first')
Out[185]:
0    6
```

Bảng 5-8. Tie-breaking methods with rank

Method	Description
'average'	Default: assign the average rank to each entry in the equal group.
'min'	Use the minimum rank for the whole group.
'max'	Use the maximum rank for the whole group.
'first'	Assign ranks in the order the values appear in the data.

```
1    1
2    7
3    4
4    3
5    2
6    5
```

Đương nhiên, bạn cũng có thể xếp hạng theo thứ tự giảm dần:

```
In [186]: obj.rank(ascending=False, method='max')
Out[186]:
0    2
1    7
2    2
3    4
4    5
5    6
6    4
```

Xem Bảng 5-8 để biết danh sách các phương pháp tie-breaking có sẵn. DataFrame có thể tính toán thứ hạng qua các hàng hoặc cột:

```
In [187]: frame = DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
   .....:                    'c': [-2, 5, 8, -2.5]})

In [188]: frame
Out[188]:
   a     b     c
0  0   4.3  -2.0
1  1   7.0   5.0
2  0  -3.0   8.0
3  1   2.0  -2.5

In [189]: frame.rank(axis=1)
Out[189]:
   a  b  c
0  2  3  1
1  1  3  2
2  2  1  3
3  2  3  1
```

2.4.3 Axis indexes with duplicate values

Cho đến bây giờ, tất cả các ví dụ cho thấy đều có nhãn trục duy nhất (giá trị index). Mặc dù nhiều hàm pandas (như reindex) yêu cầu các nhãn phải là duy nhất, nhưng điều này không bắt buộc. Hãy xem xét một Series nhỏ với các chỉ số trùng lặp:

```
In [190]: obj = Series(range(5), index=['a', 'a', 'b', 'b', 'c'])

In [191]: obj
Out[191]:
```

```
a    0
a    1
b    2
b    3
c    4
```

Thuộc tính duy nhất của index có thể cho bạn biết liệu các giá trị của nó có phải là duy nhất hay không:

```
In [192]: obj.index.is_unique
Out[192]: False
```

Lựa chọn dữ liệu là một trong những điều chính hoạt động khác nhau với các bản sao. Việc lập index một giá trị với nhiều mục nhập trả về một Chuỗi trong khi các mục nhập đơn lẻ trả về giá trị vô hướng:

```
In [193]: obj['a']      In [194]: obj['c']
Out[193]:               Out[194]: 4
a    0
a    1
```

Logic tương tự mở rộng đến việc lập index các hàng trong DataFrame:

```
In [195]: df = DataFrame(np.random.randn(4, 3), index=['a', 'a', 'b', 'b'])

In [196]: df
Out[196]:
          0         1         2
a  0.274992  0.228913  1.352917
a  0.886429 -2.001637 -0.371843
b  1.669025 -0.438570 -0.539741
b  0.476985  3.248944 -1.021228

In [197]: df.ix['b']
Out[197]:
         0         1         2
b 1.669025 -0.438570 -0.539741
b 0.476985  3.248944 -1.021228
```

3 Summarizing and Computing Descriptive Statistics

Các đối tượng pandas được trang bị một tập hợp các phương pháp thống kê và toán học phổ biến. Hầu hết trong số này là các phương pháp như tóm tắt và trích xuất giá trị duy nhất (như tính tổng , tính trung bình) từ Series hoặc DataFrame. So với các phương pháp tương đương của vani mảng NumPy, tất cả chúng đều được xây dựng từ đầu để loại trừ dữ liệu bị thiếu. Xem xét một DataFrame nhỏ:

```
In [198]: df = DataFrame([[1.4, np.nan], [7.1, -4.5],
 .....:           [np.nan, np.nan], [0.75, -1.3]],
 .....:           index=['a', 'b', 'c', 'd'],
 .....:           columns=['one', 'two'])
In [199]: df
Out[199]:
   one   two
a  1.40  NaN
b  7.10  -4.5
c  NaN   NaN
d  0.75 -1.3
```

Gọi phương thức *sum* của DataFrame trả về một Series tính tổng giá trị của mỗi cột:

```
In [200]: df.sum()
Out[200]:
one   9.25
two  -5.80
```

Khi có tham số axis = 1, có nghĩa là hàm sum được áp dụng tính tổng giá trị trên các hàng:

```
In [201]: df.sum(axis=1)
Out[201]:
a   1.40
b   2.60
c   NaN
d  -0.55
```

Ta có thể tính trung bình các giá trị trong DataFrame (theo hàng hoặc cột) bằng hàm mean() , với tham số skipna = False để không bỏ qua các giá trị NaN thì nó sẽ kiểm tra và trả về NaN nếu có bất kỳ giá trị nào bị missing trong dữ liệu.

```
In [202]: df.mean(axis=1, skipna=False)
Out[202]:
a   NaN
b   1.300
c   NaN
d  -0.275
```

Xem Bảng 5-9 để biết danh sách các tùy chọn phổ biến cho từng tùy chọn phương pháp giảm

Bảng 5-9. Tùy chọn cho phương pháp giảm.

Phương thức	**Mô tả**
axis	Bằng 0 áp dụng cho hàng của Dataframe và 1 cho cột
skipna	Loại bỏ các giá trị bị thiếu.Mặc định bằng True
level	Giảm việc nhóm theo mức nếu axis được lập chỉ mục theo thứ tự bậc

Một số phương pháp, như idxmin và idxmax, trả về số liệu thống kê gián tiếp như giá trị chỉ số trong đó đạt được các giá trị lớn nhất hoặc nhỏ nhất:

```
In [203]: df.idxmax()
Out[203]:
one   b
two   d
```

Các phương pháp khác là cumsum:

```
In [204]: df.cumsum()
Out[204]:
   one   two
a  1.40  NaN
b  8.50 -4.5
c  NaN   NaN
d  9.25 -5.8
```

Một loại phương pháp khác không phải là giảm cũng không phải là tích lũy. describe là một. Ví dụ như vậy, tạo ra nhiều số liệu thống kê tóm tắt trong một lần:

```
In [205]: df.describe()
Out[205]:
            one        two
count  3.000000   2.000000
mean   3.083333  -2.900000
std    3.493685   2.262742
min    0.750000  -4.500000
25%    1.075000  -3.700000
50%    1.400000  -2.900000
75%    4.250000  -2.100000
max    7.100000  -1.300000
```

Trên dữ liệu không phải số, describe tạo ra một Series trả về số liệu thống kê tóm tắt như: count, unique ,top ,freq.

```
In [206]: obj = Series(['a', 'a', 'b', 'c'] * 4)
In [207]: obj.describe()
Out[207]:
count    16
unique   3
top      a
freq     8
```

Xem Bảng 5-10 để biết danh sách đầy đủ các số liệu thống kê tóm tắt và các phương pháp liên quan.

Bảng 5-10. Thống kê mô tả và tóm tắt.

Phương thức	**Mô tả**
describe	Thống kê tóm tắt cho Series hoặc cột của Dataframe
min,max	Tính giá trị lớn nhất và nhỏ nhất
argmin,argmax	Tính các giá trị chỉ mục(số nguyên) mà tại đó giá trị tối thiểu hoặ tối đa tương ứng
idxmin, idxmax	Tính các giá trị chỉ mục(số nguyên) mà tại đó giá trị tối thiểu hoặ tối đa tương ứng
quantile	Lấy mẫu trong khoảng từ 0 đến 1
sum	Tổng các giá trị
mean	Trung bình
median	Trung bình số học của các giá trị.
var	Phương sai mẫu của giá trị
std	Độ lệch chuẩn
skew	Độ lệch mẫu của các gia trị
kurt	Độ nhọn mẫu của các giá trị
cumsum	Tổng giá trị tích lũy
cummin, cummax	Giá trị tối thiểu hoặc tối da tương ứng
diff	Tính toán sự khác biệt đầu tiên

3.1 Correlation and Covariance

Một số thống kê tóm tắt, như tương quan và hiệp phương sai, được tính toán từ các cặp lập luận. Chúng ta thử xem xét một số DataFrames về giá cổ phiếu và khối lượng thu được tức các trang web Yahoo! và Finance:

```
import pandas.io.data as web
all_data = {}
for ticker in ['AAPL', 'IBM', 'MSFT', 'GOOG']:
    all_data[ticker] = web.get_data_yahoo(ticker, '1/1/2000', '1/1/2010')

price = DataFrame({tic: data['Adj Close']
                   for tic, data in all_data.iteritems()})
volume = DataFrame({tic: data['Volume']
                   for tic, data in all_data.iteritems()})
```

Tiếp theo, ta tính toán sự thay đổi về tỉ lệ phần trăm mà giá trị price thay đổi:

```
In [209]: returns = price.pct_change()
In [210]: returns.tail()
Out[210]:
            AAPL      GOOG       IBM        MSFT
Date
2009-12-24  0.034339  0.011117  0.004420   0.002747
2009-12-28  0.012294  0.007098  0.013282   0.005479
2009-12-29  -0.011861 -0.005571 -0.003474  0.006812
```

```
2009-12-30  0.012147  0.005376  0.005468  -0.013532
2009-12-31  -0.004300 -0.004416 -0.012609 -0.015432
```

Phương thức corr của Series tính toán sự tương quan mối overlapping, các giá trị không phải là NA được căn chỉnh theo các chỉ mục trong hai Series.Tương tự phương thức cov tính toán giá trị hiệp phương sai :

```
In [211]: returns.MSFT.corr(returns.IBM)
Out[211]: 0.49609291822168838
In [212]: returns.MSFT.cov(returns.IBM)
Out[212]: 0.00021600332437329015
```

Mặt khác, các phương pháp corr và cov của DataFrame trả về mối tương quan đầy đủ hoặc ma trận hiệp phương sai dưới dạng DataFrame, tương ứng:

```
In [213]: returns.corr()
Out[213]:
          AAPL       GOOG       IBM        MSFT
AAPL  1.000000   0.470660   0.410648   0.424550
GOOG  0.470660   1.000000   0.390692   0.443334
IBM   0.410648   0.390692   1.000000   0.496093
MSFT  0.424550   0.443334   0.496093   1.000000

In [214]: returns.cov()
Out[214]:
          AAPL        GOOG       IBM        MSFT
 AAPL   0.001028    0.000303  0.000252    0.000309
 GOOG   0.000303    0.000580  0.000142    0.000205
 IBM    0.000252    0.000142  0.000367    0.000216
 MSFT   0.000309    0.000205  0.000216     0.000516
```

3.2 Unique Values, Value Counts, and Membership

Một lớp phương thức liên quan khác trích xuất thông tin về các giá trị có trong một Series một chiều. Để minh họa những điều này, hãy xem xét ví dụ này.Hàm đầu tiên là unique , cung cấp cho bạn một mảng các giá trị duy nhất trong Series:

```
In [217]: obj = Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c'])
In [218]: uniques = obj.unique()

In [219]: uniques
Out[219]: array([c, a, d, b], dtype=object)
```

Các giá trị duy nhất không nhất thiết phải được trả về theo thứ tự đã sắp xếp, nhưng có thể được sắp xếp theo một Series chứa tần số giá trị :

```
In [220]: obj.value_counts()
Out[220]:
c    3
a    3
b    2
d    1
```

Series được sắp xếp theo giá trị theo thứ tự giảm dần như một sự tiện lợi 'valuecounts' cũng là có sẵn dưới dạng phương pháp Panes cấp cao nhất có thể được sử dụng với bất kỳ mảng hoặc chuỗi nào:

```
In [221]: pd.value_counts(obj.values, sort=False)
Out[221]:
a    3
b    2
c    3
d    1
```

Cuối cùng, isin chịu trách nhiệm về tư cách thành viên tập hợp vectơ và có thể rất hữu ích trong lọc tập dữ liệu xuống một tập hợp con các giá trị trong Series hoặc cột trong DataFrame :

```
In [222]: mask = obj.isin(['b', 'c'])
In [223]: mask                  In [224]: obj[mask]
Out[223]:                       Out[224]:
0     True                      0   c
1     False                     5   b
2     False                     6   b
3     False                     7   c
4     False                     8   c
5     True
6     True
```

Xem Bảng 5-11 để tham khảo về các phương pháp này:

Bảng 5-11. Duy nhất, giá trị đếm và phương pháp binning

Phương thức	**Mô tả**
isin	Tính toán mảng boolean cho biết mỗi giá trị series đã truyền hay chưa
unique	Tính toán mảng giá trị duy nhất trả về theo thứ tự được quan sát
value counts	Trả về Series chứa số lượng các giá trị duy nhất .

Trong một số trường hợp, bạn có thể muốn tính biểu đồ tần suất trên nhiều cột liên quan trong một DataFrame. Dưới đây là một ví dụ:

```
In [225]: data = DataFrame({'Qu1': [1, 3, 4, 3, 4],
.....:                      'Qu2': [2, 3, 1, 2, 3],
.....:                      'Qu3': [1, 5, 2, 4, 4]})

In [226]: data
Out[226]:
  Qu1  Qu2  Qu3
0   1    2    1
1   3    3    5
2   4    1    2
3   3    2    4
4   4    3    4
```

Đưa đối số pd.valuecounts của DataFrame cho hàm apply() :

```
In [227]: result = data.apply(pd.value_counts).fillna(0)

In [228]: result
Out[228]:
  Qu1  Qu2   Qu3
1   1    1     1
2   0    2     1
3   2    2     0
4   2    0     2
5   0    0     1
```

4 Handling Missing Data

Thiếu dữ liệu là phổ biến trong hầu hết các ứng dụng phân tích dữ liệu. Một trong những mục tiêu trong việc thiết kế gấu trúc là làm cho việc làm việc với dữ liệu bị thiếu trở nên dễ dàng nhất có thể.
Ví dụ: tất cả các số liệu thống kê mô tả về các đối tượng gấu trúc đều loại trừ dữ liệu bị thiếu dưới dạng bạn đã thấy trước đó trong chương pandas trúc sử dụng giá trị dấu phẩy động NaN (Không phải số) để biểu diễn dữ liệu bị thiếu trong cả hai đều nổi cũng như trong các mảng dấu phẩy động. Nó chỉ được sử dụng như một sentinel có thể dễ dàng bị phát hiện:

```
In [229]: string_data = Series(['aardvark', 'artichoke', np.nan, 'avocado'])

In [230]: string_data
Out[230]:
0     aardvark
1    artichoke
2          NaN
3      avocado

In [231]: string_data.isnull()
Out[231]:
0     False
1     False
2     True
3     False
```

Giá trị None được định nghĩa trong Python cũng được coi là NA trong các mảng đối tượng:

```
In [232]: string_data[0] = None
In [233]: string_data.isnull()
Out[233]:
0     True
1     False
2     True
3     False
```

Bảng 5-12. Phương pháp xử lý NA.

Đối số	Mô tả
dropna	Lọc các nhãn lục dựa trên thiếu hay không và các ngưỡng dữ liệu bị thiếu có thể chấp nhận được
fillna	Điền giá trị còn thiếu bằng một số giá trị hoặc sử dụng phương pháp nội suy
isnull	Trả về giá trị cùng loại chứa các giá trị boolean và cho biết giá trị nào bị thiếu NA
notnull	Phủ định của isnull

4.1 Filtering Out Missing Data

Bạn có một số tùy chọn để lọc ra dữ liệu bị thiếu. Trong khi làm điều đó bằng tay là luôn luôn là một lựa chọn, dropna có thể rất hữu ích. Trên một Series, nó trả về Series chỉ với các giá trị chỉ mục và dữ liệu không rỗng:

```
In [234]: from numpy import nan as NA
In [235]: data = Series([1, NA, 3.5, NA, 7])
In [236]: data.dropna()
Out[236]:

0    1.0
2    3.5
4    7.0
```

Đương nhiên, bạn có thể tự tính toán điều này bằng cách lập chỉ mục boolean:

```
In [237]: data[data.notnull()]
Out[237]:
0    1.0
2    3.5
4    7.0
```

Với các đối tượng DataFrame, chúng phức tạp hơn một chút. Bạn có thể muốn thả hàng hoặc các cột đều là NA hoặc chỉ những cột chứa bất kỳ NA nào. dropna theo mặc định giảm bất kỳ hàng nào chứa giá trị bị thiếu:

```
In [238]: data = DataFrame([[1., 6.5, 3.], [1., NA, NA],
   .....:                   [NA, NA, NA], [NA, 6.5, 3.]])
In [239]: cleaned = data.dropna()

In [240]: data          In [241]: cleaned
Out[240]:               Out[241]:
    0    1   2              0    1  2
0   1  6.5   3           0  1  6.5  3
1   1  NaN NaN
2 NaN  NaN NaN
3 NaN  6.5   3
```

Tham số how='all' sẽ chỉ loại bỏ các hàng đều là NaN, nhưng không làm thay đổi chỉ số index của DataFrame.

```
In [242]: data.dropna(how='all')
Out[242]:
    0     1   2
0   1   6.5   3
1   1   NaN NaN
3   NaN 6.5   3
```

Tương tự chúng ta có thể loại bỏ các cột khi trong cột đều là giá trị NaN , tham số axis = 1 là tham số truyền vào hàm dropna().

```
In [243]: data[4] = NA
In [244]: data
Out[244]:
    0    1   2   4
0   1  6.5   3 NaN
1   1  NaN NaN NaN
2 NaN  NaN NaN NaN
3 NaN  6.5   3 NaN
```

```
In [245]: data.dropna(axis=1, how='all')
Out[245]:
    0    1   2
0   1  6.5   3
1   1  NaN NaN
2 NaN  NaN NaN
3 NaN  6.5   3
```

Để lọc ra các hàng DataFrame có xu hướng liên quan đến dữ liệu time series. Giả thiết bạn chỉ muốn giữ lại các hàng chứa một số lượng quan sát nhất định. Chúng ta có thể lấy ra những dữ liệu với đối số thresh. Thesh = N yêu cầu ít nhất N giá trị không phải là NaN , điều kiện này là đủ để hàng hoặc cột có thể được in ra .

```
In [246]: df = DataFrame(np.random.randn(7, 3))
In [247]: df.iloc[:4, 1] = NA; df.iloc[:2, 2] = NA
In [248]: df
Out[248]:
          0         1         2
0 -0.577087       NaN       NaN
1  0.523772       NaN       NaN
2 -0.713544       NaN       NaN
3 -1.860761       NaN  0.560145
4 -1.265934       NaN -1.063512
5  0.332883 -2.359419 -0.199543
6 -1.541996 -0.970736 -1.307030
```

```
In [249]: df.dropna(thresh=3)
Out[249]:
          0         1         2
5  0.332883 -2.359419 -0.199543
6 -1.541996 -0.970736 -1.307030
```

4.2 Filling in Missing Data

Thay vì lọc ra dữ liệu bị thiếu (và có khả năng loại bỏ dữ liệu khác cùng với nó), bạn có thể muốn điền vào các chỗ trống theo bất kỳ cách nào. Chúng ta có thể gọi hàm fillna liên tục để điền vào các giá trị đó:

```
In [250]: df.fillna(0)
Out[250]:
          0         1         2
0 -0.577087  0.000000  0.000000
1  0.523772  0.000000  0.000000
2 -0.713544  0.000000  0.000000
3 -1.860761  0.000000  0.560145
4 -1.265934  0.000000 -1.063512
5  0.332883 -2.359419 -0.199543
6 -1.541996 -0.970736 -1.307030
```

Gọi fillna bằng một dict, bạn có thể sử dụng một giá trị điền khác nhau cho mỗi cột:

```
In [250]: df.fillna(0)
Out[250]:
          0         1        2
0 -0.577087  0.500000       NaN
1  0.523772  0.500000       NaN
2 -0.713544  0.500000       NaN
3 -1.860761  0.500000  0.560145
4 -1.265934  0.500000 -1.063512
5  0.332883 -2.359419 -0.199543
6 -1.541996 -0.970736 -1.307030
```

Fillna trả về một đối tượng mới, nhưng chúng ta có thể sửa đổi đối tượng hiện tại với tham số implace = True:

```
In [252]: _ = df.fillna(0, inplace=True)
In [253]: df
Out[253]:
          0         1         2
0 -0.577087  0.000000  0.000000
1  0.523772  0.000000  0.000000
2 -0.713544  0.000000  0.000000
3 -1.860761  0.000000  0.560145
4 -1.265934  0.000000 -1.063512
5  0.332883 -2.359419 -0.199543
6 -1.541996 -0.970736 -1.307030
```

Ta cũng có thể tùy chỉnh các giá trị index để phù hợp với việc truy cập vào các giá trị NaN và tại các NaN ta cũng có rất nhiều phương pháp để thay thế bằng những giá trị lân cận hoặc bằng các giá trị trị trung bình,..

```
In [254]: df = DataFrame(np.random.randn(6, 3))
In [255]: df.iloc[2:, 1] = NA; df.iloc[4:, 2] = NA
In [256]: df
Out[256]:
          0         1         2
0  0.286350 0.377984 -0.753887
1  0.331286 1.349742  0.069877
2  0.246674       NaN  1.004812
3  1.327195       NaN -1.549106
4  0.022185       NaN       NaN
5  0.862580       NaN       NaN
```

```
In [257]: df.fillna(method='ffill')
Out[257]:
          0         1         2
0  0.286350  0.377984 -0.753887
1  0.331286  1.349742  0.069877
2  0.246674  1.349742  1.004812
3  1.327195  1.349742 -1.549106
4  0.022185  1.349742 -1.549106
5  0.862580  1.349742 -1.549106
```

```
In [258]: df.fillna(method='ffill', limit=2)
Out[258]:
          0         1         2
0  0.286350 0.377984 -0.753887
1  0.331286 1.349742  0.069877
2  0.246674 1.349742  1.004812
3  1.327195 1.349742 -1.549106
4  0.022185       NaN -1.549106
5  0.862580       NaN -1.549106
```

Với fillna, bạn có thể làm rất nhiều thứ khác với một chút sáng tạo. Ví dụ: bạn có thể vượt qua giá trị trung bình hoặc giá trị trung bình của một Seres:

```
In    [259]: data = Series([1., NA, 3.5, NA, 7])
In    [260]: data.fillna(data.mean())
Out[260]:
0   1.000000
1   3.833333
2   3.500000
3   3.833333
4   7.000000
```

Xem Bảng 5-13 để tham khảo về fillna.

Bảng 5-13. Đối số hàm Fillna .

Đối số	**Mô tả**
value	Giá trị vô hướng giống như dict để điền vào các giá trị còn thiếu
method	Nội suy mặc định điền nếu hàm được gọi mà không có đối số nào khác
axis	Trục để điền vào , trục mặc định bằng 0
inplace	Sửa đổi đối tượng gọi mà không tạo bản sao
limit	Để điền tiến và lùi , số khoảng thời gian liên tiếp tối đa cần điền.

5 Hierarchical Indexing

Hierarchical indexing là một trong những tính năng quan trọng của pandas cho phép đa dạng (một hoặc nhiều) index levels trên một trục. Một cách trừu tượng, nó cho phép làm việc với dữ liệu nhiều cấp bậc hơn trong một cấu trúc ít phức tạp hơn.

```
In [261]: data = Series(np.random.randn(10),
.....:                  index=[['a', 'a', 'a', 'b', 'b', 'b', 'c', 'c', 'd', 'd'],
.....:                         [1, 2, 3, 1, 2, 3, 1, 2, 2, 3]])
In [262]: data
Out[262]:
a 1     0.670216
  2     0.852965
  3    -0.955869
b 1    -0.023493
  2    -2.304234
  3    -0.652469
c 1    -1.218302
  2    -1.332610
d 2     1.074623
  3     0.723642
```

Hãy bắt đầu với một ví dụ đơn giản; tạo một Series với một nhiều kiểu dữ liệu lists nằm trong một list với chỉ số tương ứng:

```
In [263]: data.index
Out[263]:
MultiIndex
[('a', 1) ('a', 2) ('a', 3) ('b', 1) ('b', 2) ('b', 3) ('c', 1)
('c', 2) ('d', 2) ('d', 3)]
In [262]: data
Out[262]:
a 1     0.670216
  2     0.852965
  3    -0.955869
b 1    -0.023493
  2    -2.304234
  3    -0.652469
c 1    -1.218302
  2    -1.332610
d 2     1.074623
  3     0.723642
```

Series với `MultiIndex` làm index đầu vào. Khoảng trống ở index có nghĩa là "dùng nhãn trực tiếp ở phía trên".

```
In [263]: data.index
Out[263]:
MultiIndex
[('a', 1) ('a', 2) ('a', 3) ('b', 1) ('b', 2) ('b', 3) ('c', 1)
('c', 2) ('d', 2) ('d', 3)]
```

Với một đối tượng được lập index phân cấp, có thể được gọi là lập index từng phần, chia một cách chính xác các tập hợp con của dữ liệu.

```
In [264]: data['b']
Out[264]:
1 -0.023493
2 -2.304234
3 -0.652469
```

```
In [265]: data['b':'c']
Out[265]:
b  1    -0.023493
   2    -2.304234
   3    -0.652469
c  1    -1.218302
   2    -1.332610
```

```
In [266]: data.ix[['b', 'd']]
Out[266]:
b  1    -0.023493
   2    -2.304234
   3    -0.652469
d  2     1.074623
   3     0.723642
```

Ta còn có thể lựa chọn cấp độ trong một số trường hợp từ cấp độ "inner":

```
In [267]: data[:, 2]
Out[267]:
    a   0.852965
    b  -2.304234
    c  -1.332610
    d   1.074623
```

Hierarchical indexing đóng một vai trò quan trọng trong việc định hình lại dữ liệu và các hoạt động dựa trên nhóm, giống như tạo một bảng tổng hợp. Ví dụ: dữ liệu này có thể được sắp xếp lại thành DataFrame bằng cách sử dụng phương pháp `unstack`:

```
In [268]: data.unstack()
Out[268]:
          1         2          3
a  0.670216  0.852965  -0.955869
b -0.023493 -2.304234  -0.652469
c -1.218302 -1.332610        NaN
d       NaN  1.074623   0.723642
```

Nếu muốn quay trở lại nguyên trạng, ta sử dụng `unstack` đi kèm cùng `stack`:

```
In [269]: data.unstack().stack()
Out[269]:
a  1    0.670216
   2    0.852965
   3   -0.955869
b  1   -0.023493
   2   -2.304234
   3   -0.652469
c  1   -1.218302
   2   -1.332610
d  2    1.074623
   3    0.723642
```

`stack` và `unstack` sẽ được trình bày chi tiết hơn ở chương 7.

Với DataFrame, một trong hai trục có thể có index phân cấp:

```
In [270]: frame = DataFrame(np.arange(12).reshape((4, 3)),
   .....:                   index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
   .....:                   columns=[['Ohio', 'Ohio', 'Colorado'],
   .....:                            ['Green', 'Red', 'Green']])
In [271]: frame
Out[271]:
      Ohio        Colorado
      Green  Red     Green
a 1       0    1         2
  2       3    4         5
b 1       6    7         8
  2       9   10        11
```

Các cấp phân cấp có thể có tên (dưới dạng chuỗi hoặc bất kỳ đối tượng Python nào) sẽ hiển thị trong đầu ra (đừng nhầm lẫn tên index với axis label):

```
In [272]: frame.index.names = ['key1', 'key2']
In [273]: frame.columns.names = ['state', 'color']
In [274]: frame
Out[274]:
state       Ohio        Colorado
color       Green  Red     Green
key1 key2
a    1          0    1         2
     2          3    4         5
b    1          6    7         8
     2          9    10       11
```

Với lập index một phần cột, có thể chọn các nhóm cột một cách tương tự:

```
In [275]: frame['Ohio']
Out[275]:
color       Green  Red
key1 key2
a    1          0    1
     2          3    4
b    1          6    7
     2          9   10
```

Một MultiIndex có thể được tạo ra bởi chính nó và sau đó được sử dụng lại; các cột trong DataFrame ở trên với tên cấp có thể được tạo như thế này:

```
MultiIndex.from_arrays([['Ohio', 'Ohio', 'Colorado'], ['Green', 'Red', 'Green']],
                       names=['state', 'color'])
```

5.1 Reordering and Sorting Levels

Đôi khi sẽ cần phải sắp xếp lại thứ tự của các cấp trên một trục hoặc sắp xếp dữ liệu bởi các giá trị trong một cấp cụ thể. Swaplevel có hai cấp số hoặc tên và trả về một đối tượng mới với các mức được hoán đổi cho nhau (nhưng dữ liệu thì không thay đổi):

```
In [276]: frame.swaplevel('key1', 'key2')
Out[276]:
state       Ohio        Colorado
color       Green  Red     Green
key2 key1
1    a          0    1         2
2    a          3    4         5
1    b          6    7         8
2    b          9    10       11
```

Mặt khác, `sortlevel` sắp xếp dữ liệu (ổn định) chỉ sử dụng các giá trị trong một cấp độ. Khi hoán đổi cấp độ, không có gì lạ khi cũng sử dụng `sortlevel` như kết quả được sắp xếp dưới đây:

```
In [277]: frame.sortlevel(1)
Out[277]:
state       Ohio        Colorado
color       Green  Red     Green
key1 key2
a    1          0    1         2
b    1          6    7         8
a    2          3    4         5
b    2          9    10        11
```

```
In [278]: frame.swaplevel(0, 1).sortlevel(0)
Out[278]:
state         Ohio        Colorado
color        Green   Red     Green
key2 key1
1    a           0      1        2
     b           6      7        8
2    a           3      4        5
     b           9      10       11
```

Hiệu suất lựa chọn dữ liệu tốt hơn nhiều khi được lập index theo cấp bậc các đối tượng, nếu các index được sắp xếp theo thứ tự từ điển và bắt đầu từ cấp cao nhất, đó chính là kết quả của việc gọi `sort_level()` hoặc `sort_index()`.

5.2 Summary Statistics by Level

Nhiều thống kê mô tả và tóm tắt trên DataFrame và Series có tùy chọn cấp độ trong đó ta có thể chỉ định mức bạn muốn tính tổng trên một trục cụ thể. Xem xét DataFrame ở trên; chúng ta có thể tính tổng theo cấp độ trên các hàng hoặc cột như sau:

```
In [279]: frame.sum(level='key2')
Out[279]:
state   Ohio        Colorado
color   Green  Red     Green
```

```
key2
1           6    8        10
2           12   14       16
In [280]: frame.sum(level='color', axis=1)
Out[280]:
color       Green  Red
key1 key2
a    1          2    1
     2          8    4
b    1         14    7
     2         20   10
```

5.3 Using a DataFrame's Columns

Khi muốn sử dụng một hoặc nhiều cột từ DataFrame giống như là index đầu vào của hàng; hay muốn di chuyển index hàng vào các cột của DataFrame, ta sẽ sẽ sử dụng Dataframes columns. Đây là một ví dụ về DataFrame:

```
In [281]: frame = DataFrame({'a': range(7), 'b': range(7, 0, -1),
    .....:                                   'c': ['one', 'one', 'one', 'two', 'two',
    .....:                                   'd': [0, 1, 2, 0, 1, 2, 3]})
In [282]: frame
Out[282]:
  a b   c d
0 0 7 one 0
1 1 6 one 1
2 2 5 one 2
3 3 4 two 0
4 4 3 two 1
5 5 2 two 2
6 6 1 two 3
```

Hàm `set_index()` của DataFrame sẽ tạo thành một DataFrame mới sử dụng một hoặc nhiều cột của nó để làm index:

```
In [283]: frame2 = frame.set_index(['c', 'd'])
In [284]: frame2
Out[284]:
        a b
c   d
one 0   0 7
    1   1 6
    2   2 5
two 0   3 4
    1   4 3
    2   5 2
    3   6 1
```

Mặc định thì các hàng được xóa mất DataFrame cũ, nhưng thông qua hàm `set_index` ta cũng có thể giữ nguyên nó:

```
In [285]: frame.set_index(['c', 'd'], drop=False)
Out[285]:
```

```
         a  b     c  d
c   d
one 0    0  7   one  0
    1    1  6   one  1
    2    2  5   one  2
two 0    3  4   two  0
    1    4  3   two  1
    2    5  2   two  2
    3    6  1   two  3
```

Hàm `reset_index` () trái ngược với hàm `set_index` (); các cấp bậc index được di chuyển vào trong các cột:

```
In [286]: frame2.reset_index()
Out[286]:
     c d a b
0  one 0 0 7
1  one 1 1 6
2  one 2 2 5
3  two 0 3 4
4  two 1 4 3
5  two 2 5 2
6  two 3 6 1
```

5.4 Other pandas Topics

Một số chủ đề bổ sung có thể sử dụng làm hành trang ở hành trình làm việc với dữ liệu trong tương lai.

Integer Indexing

Làm việc với các đối tượng Pandas được lập index bởi các số nguyên thường gây một số rắc rối cho người chưa quen sử dụng do một số khác biệt với ngữ nghĩa, hay việc lập index trên cấu trúc dữ liệu của Python như kiểu dữ liệu list hay tuples. Ví dụ, ta sẽ gặp phải các dòng code báo lỗi như dưới đây:

```
ser = Series(np.arange(3.))
ser[-1]
```

Trong trường hợp này, Pandas có thể "quay trở lại" khi lập index số nguyên, nhưng không an toàn và cách chung để làm điều này mà không đưa ra các lỗi nhỏ. Ở đây ta có một index chứa 0, 1, 2, nhưng suy ra những gì người dùng muốn (lập chỉ mục dựa trên nhãn hoặc dựa trên vị trí) sẽ gây khó khăn hơn:

```
In [288]: ser
Out[288]:
0   0
1   1
2   2
```

Trong một trường hợp khác, với các index không phải số nguyên, khả năng chạy sai của câu lệnh được giảm thiểu tối đa:

```
In [289]: ser2 = Series(np.arange(3.), index=['a', 'b', 'c'])
In [290]: ser2[-1]
Out[290]: 2.0
```

Để giữ mọi thứ nhất quán, nếu có một trục index chứa các index khác, và lựa chọn dữ liệu với số nguyên sẽ luôn là phù hợp nhất. Điều này bao gồm cắt với ix:

```
In [291]: ser.ix[:1]
Out[291]:
0   0
1   1
```

Trong trường hợp cần các index ở vị trí đáng tin cậy bất kể ở kiểu index nào, ta có thể dùng phương thức `iget_value()` từ Series và `irow()` và `icol()` từ DataFrame:

```
In [292]: ser3 = Series(range(3), index=[-5, 1, 3])
In [293]: ser3.iget_value(2)
Out[293]: 2
In [294]: frame = DataFrame(np.arange(6).reshape(3, 2)), index=[2, 0, 1])
In [295]: frame.irow(0)
Out[295]:
0   0
1   1
Name: 2
```

Panel Data

Mặc dù không phải là chủ đề quan trọng của cuốn sách này, nhưng `pandas` cũng có chứa một cấu trúc dữ liệu tên là `Panel`, bạn có thể dùng nó tương tự như một không gian ba chiều của DataFrame. Hầu hết trọng tâm phát triển của Pandas được tập trung vào thao tác dạng bảng vì chúng dễ lý luận hơn, và sự phân cấp index theo thứ bâc thực sự khiến việc sử dụng mảng N chiều là thực sự không cần thiết trong nhiều trường hợp.

Để tạo một `Panel`, bạn có thể khai báo một chuỗi các đối tượng của DatFrame mang kiểu dữ liệu dict hoặc một mảng 3 chiều:

```
import pandas.io.data as web
pdata = pd.Panel(dict((stk, web.get_data_yahoo(stk, '1/1/2009', '6/1/2012'))
                      for stk in ['AAPL', 'GOOG', 'MSFT', 'DELL']))
```

Mỗi item (tương tự với cột trong DataFrame) trong `Panel` là một DataFrame:

```
In [297]: pdata
Out[297]:
<class 'pandas.core.panel.Panel'>
Dimensions: 4 (items) x 861 (major) x 6 (minor)
Items: AAPL to MSFT
Major axis: 2009-01-02 00:00:00 to 2012-06-01 00:00:00
Minor axis: Open to Adj Close

In [298]: pdata = pdata.swapaxes('items', 'minor')

In [299]: pdata['Adj Close']
Out[299]:
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 861 entries, 2009-01-02 00:00:00 to 2012-06-01 00:00:00
Data columns:
AAPL
861 non-null values
DELL
861 non-null values
GOOG
861 non-null values
MSFT
861 non-null values
dtypes: float64(4)
```

Truy xuất nhãn index bằng `index` duyệt ba đối số truyền vào, vì vậy ta có thể lựa chọn tất cả dữ liệu ở một ngày cụ thể hay một khoảng thời giạn, như ví dụ dưới:

```
In [300]: pdata.ix[:, '6/1/2012', :]
Out[300]:
          Open    High     Low     CloseVolume   Adj Close
AAPL    569.16  572.65  560.52        18606700      560.99
MSFT     12.15   12.30   12.05        19396700       12.07
GOOG    571.79  572.65  568.35         3057900      570.98
MSFT     28.76   28.96   28.44        56634300       28.45
In [301]: pdata.ix['Adj Close', '5/22/2012':, :]
Out[301]:
               AAPL     DELL    GOOG    MSFT
Date
2012-05-22   556.97    15.08  600.80   29.76
2012-05-23   570.56    12.49  609.46   29.11
2012-05-24   565.32    12.45  603.66   29.07
2012-05-25   562.29    12.46  591.53   29.06
2012-05-29   572.27    12.66  594.34   29.56
2012-05-30   579.17    12.56  588.23   29.34
2012-05-31   577.73    12.33  580.86   29.19
2012-06-01   560.99    12.07  570.98   28.45
```

Một cách khác để biểu diễn dữ liệu của `Panel`, đặc biệt phù hợp với các mô hình thống kê, nằm trong khung "stacked" của DataFrame:

```
In [302]: stacked = pdata.ix[:, '5/30/2012':, :].to_frame()
In [303]: stacked
Out[303]:
                     Open    High     Low    Close     Volume  Adj Close
major       minor
2012-05-30  AAPL   569.16  579.99  566.56   579.17   18908200     579.17
            MSFT    12.59   12.70   12.46    12.56   19787800      12.56
            GOOG   588.16  591.90  583.53   588.23    1906700     588.23
            MSFT    29.35   29.48   29.12    29.34   41585500      29.34
2012-05-31  AAPL   580.74  581.50  571.46   577.73   17559800     577.73
            MSFT    12.53   12.54   12.33    12.33   19955500      12.33
            GOOG   588.72  590.00  579.00   580.86    2968300     580.86
            MSFT    29.30   29.42   28.94    29.19   39134000      29.19
```

```
2012-06-01   AAPL  569.16  572.65  560.52    560.99  18606700     560.99
             MSFT   12.15   12.30   12.05     12.07  19396700      12.07
             GOOG  571.79  572.65  568.35    570.98   3057900     570.98
             MSFT   28.76   28.96   28.44    28.45   56634300      28.45
```

DataFrame có một phương thức liên quan `to_panel`, là sự đảo ngược của phương thức `to_frame`:

```
In [304]: stacked.to_panel()
Out[304]:
<class 'pandas.core.panel.Panel'>
Dimensions: 6 (items) x 3 (major) x 4 (minor)
Items: Open to Adj Close
Major axis: 2012-05-30 00:00:00 to 2012-06-01 00:00:00
Minor axis: AAPL to MSFT
```

TÀI LIỆU THAM KHẢO

Wes McKinney , ***Python for Data Analysis***