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

———————— * ————————

BÁO CÁO HỌC PHẦN

TIỀN XỬ LÝ DỮ LIỆU VỚI PYTHON

HỘI ĐỒNG: Khoa học máy tính
GVHD: ThS. Nguyễn Mạnh Hùng
GVPB: ThS. Nguyễn Mạnh Hùng
—o0o—

SVTH 1: Trần Nguyễn Quang Lâm - 20139040
SVTH 2: Tô Gia Huy - 20139003
SVTH 3: Nguyễn Trí Ban - 20139064
SVTH 4: Trần Đức Hiếu - 20139074
SVTH 5: Phạm Thanh Hà - 20139073

TP. HỒ CHÍ MINH, 11/2022

Tóm tắt nội dung

Phần nhiều khối lượng công việc của phân tích và mô hình hóa dữ liệu nằm ở việc chuẩn bị, hay còn gọi là tiền xử lý dữ liệu: vận hành, sàng lọc, biến đổi và sắp xếp dữ liệu. Cách thức mà dữ liệu được lưu trữ và sắp xếp trong tập tin hoặc database không phải lúc nào cũng thuận tiện hoàn hảo cho ứng dụng xử lý dữ liệu. Cách xử lý dữ liệu theo kiểu chuyển đổi định dạng được sử dụng rộng rãi bởi mọi người với các ngôn ngữ đa dụng như Python, Perl, R hoặc Java. May mắn thay Python có một thư viện chuẩn tên Pandas cung cấp cho người dùng các công cụ hiệu quả và linh hoạt bậc cao, áp dụng cho nhiều thuật toán, hỗ trợ người dùng xử lý dữ liệu theo nhu cầu một cách dễ dàng. Phần báo cáo này sẽ cung cấp cho người đọc các phương pháp cơ bản với tiền xử lý dữ liệu sử dụng thư viện Pandas của Python.

Mục lục

Danh sách bảng

Chương 1

KẾT HỢP VÀ GỘP CÁC TẬP DỮ LIỆU

Dữ liệu lưu trữ trong các đối tượng pandas có thể được gộp lại với nhiều cách đã được tích hợp sẵn trong thư viện này:

•**pandas.merge** nối các hàng trong các DataFrames dựa trên một hoặc nhiều từ khóa. Cách kết nối này khá quen thuộc với người sử dụng SQL hoặc các cơ sở dữ liệu kiểu khác vì cách dùng toán tử join của cơ sở dữ liệu .

•**pandas.concat** nối hoặc xếp các đối tượng dọc theo một cột.

•**combinefirst** phương thức cho cắt lát các dữ liệu chồng lặp để tìm ra giá trị cho các ô trống trong một đối tượng với giá trị của các đối tượng khác.

Phần báo cáo này sẽ đề cập đến tất cả các phương thức trên và đưa ra các ví dụ. Tất cả đều được tối ưu trong trong phần báo cáo này.

1.1 Gộp các tập dữ liệu DataFrame

Các phương thức Merge hoặc Join kết hợp các tập dữ liệu bằng cách kết nối các hàng sử dụng một hoặc nhiều từ khóa.Các toán tử này là trung tâm của các tập dữ liệu tương quan. Hàm **merge** trong pandas là cách chính để sử dụng các thuật toán này trong dữ liệu của người dùng.

Hãy bắt đầu với một ví dụ đơn giản:

```
In [15]: df1 = DataFrame('key': ['b','b','a','c','c','a','a','b'], 'data1': range(7))
In [16]: df2 = DataFrame('key': ['b','a','d'], 'data2': range(3))
```

Đây là một ví dụ của trường hợp gộp nhiều tập dữ liệu thành một; tập dữ liệu trong df1 có nhiều hàng được gán nhãn a và b, trong khi df2 có mỗi 1 hàng cho mỗi giá trị trong cột key của nó. Gọi **merge** với mỗi đối tượng trên ta nhận được

```
In [19]: pd.merge(df1, df2)
Out [19]:
   data1 key   data2
0      2   a       0
1      4   a       0
2      5   a       0
3      0   b       1
4      1   b       1
5      6   b       1
```

Để ý rằng ta không xác định cụ thể cột nào để dồn chúng lại. Nếu không xác định cụ thể, **merge** sử dụng các cột có tên lặp chồng làm các từ khóa. Đây là một bài tập tốt để xác định

sự khác biệt, ví dụ:

```
In [20]: pd.merge(df1, df2,on='key')
Out [20]:
   data1 key   data2
0      2   a      0
1      4   a      0
2      5   a      0
3      0   b      1
4      1   b      1
5      6   b      1
```

Nếu tên các cột khác nhau ở mỗi đối tượng, ta có thể xác định chúng riêng biệt:

```
In [21]: df3 = DataFrame('key': ['b','b','a','c','c','a','a','b'], 'data1': range(7))
In [22]: df4 = DataFrame('rkey': ['a','b','d'], 'data2': range(3))
In [23]: pd.merge(df3, df4, leftOn='lkey', rightOn='rkey')
Out [23]:
   data1 lkey   data2    rkey
0      2   a      0        a
1      4   a      0        a
2      5   a      0        a
3      0   b      1        b
4      1   b      1        b
5      6   b      1        b
```

Ta nhận ra 2 khóa 'c' và 'd' và các dữ liệu liên quan đều bị thiếu trong phần đầu ra. Bởi **merge** thực hiện dồn trong **'inner'**; từ khóa trong kết quả đầu ra là các phần giao thoa của 2 tập dữ liệu. Các lựa chọn có thể khác đó là **'left'**, **'right'**, và **'outer'**. Lựa chọn gộp **'outer'** hay nói cách khác là gộp kiểu ngoài, là kiểu gộp lấy phần gộp của các từ khóa, kết hợp gộp cả 2 tập dữ liệu lại:

```
In [24]: pd.merge(df1 df2, how='outer')
Out[24]:
   data1 key   data2
0      2   a      0
1      4   a      0
2      5   a      0
3      0   b      1
4      1   b      1
5      6   b      1
6      3   b    NaN
7    NaN   c      2
```

Để xác định kiểu kết hợp key nào sẽ hiển thị ở trong phần kết quả đầu ra phụ thuộc vào cách chọn trong phương thức merge, xét trường hợp nhiều từ khóa như việc hình thành một mảng các tuples để sử dụng như một từ khóa gộp riêng biệt(mặc dù cách thức gộp không hẳn là như vậy).

Vấn đề cuối cùng cần xem xét trong khi dùng các phương thức gộp đó là cách xử lý các cột có tên trùng lặp. Trong khi ta có thể chỉ đến vị trí trùng lặp một cách thủ công(trong phần đổi tên các cột ở sau này), **merge** có một lựa chọn cho chỉ cụ thể các chuỗi mang giá trị tên trùng lặp trong các DataFrame được gộp:

```
In [34]: pd.merge(df1 df2, how='outer')
Out[34]:
```

```
   key1 key2x  lval  key2y  rval
0   bar   one     3    one     6
1   bar   one     3    two     7
2   foo   one     1    one     4
3   foo   one     1    one     5
4   foo   two     2    one     4
5   foo   two     2    one     5
In [35]: pd.merge(left, right, on='key1', suffixes=('Left','Right'))
Out[35]:
   key1 key2Left  lval  key2Right  rval
0   bar      one     3        one     6
1   bar      one     3        two     7
2   foo      one     1        one     4
3   foo      one     1        one     5
4   foo      two     2        one     4
5   foo      two     2        one     5
```

Coi Bảng **1.1** để hiểu thêm về tham chiếu các đối số trong hàm **merge**. Gộp theo chỉ số sẽ là chủ để của phần tiếp theo

Tham số	**Mô tả**
left	DataFrame được gộp ở bên trái(đầu tiên)
right	DataFrame được gộp ở bên phải(sau)
how	Cách gộp, inner hay outer, left hay right, inner là mặc định
on	Tên cột bắt đầu gộp, phải có ở cả 2 DataFrame.
left_on	Cột ở DataFrame bên trái để bắt đầu gộp
right_on	Tương tự như left_on đối với DataFrame bên phải
left_index	Chỉ số của cột DataFrame bên trái để bắt đầu gộp
right_index	Tương tự như left_index
sort	Sắp xếp và lọc dữ liệu dựa theo các key
suffixes	Các tupple chứa dữ liệu để thêm vào tên các cột bị chồng lặp
copy	Nếu giá trị này False thì tránh copy dữ liệu vào trong cấu trúc kết quả

Bảng 1.1: Các tham số của hàm **merge**

1.2 Gộp theo chỉ số

Trong một số trường hợp, từ khóa gộp hay các từ khóa trong DataFrames sẽ là các chỉ số của các cột. Trong trường hợp này, ta có thể dùng **leftIndex=True** hoặc **rightIndex=True** hoặc cả hai để chỉ ra rằng chỉ số của các cột nên được dùng là các key:

```
In [36]: left1 = DataFrame('key': ['a','b','a','a','a','b','c'],'value': range(6))
In [37]: right1 = DataFrame('group_val' : [3.5,7], index = ['a','b'])
In [38]: left1
In [39]: right1
.....:
```

Out[38]:

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

Out[39]:

	groupVal
a	3.5
a	7.0

In [40]: pd.merge(left1, right1, leftOn='key', rightIndex=True)

Out[40]:

	key	value	groupVal
0	a	0	3.5
1	b	1	3.5
2	a	2	3.5
3	a	3	7.0
4	b	4	7.0
5	c	5	NaN

Kỹ thuật đánh dấu dữ liệu theo cấp độ sẽ phức tạp hơn:

In [42]: lefth = DataFrame('key1': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
.....: 'key2': [2000, 2001, 2002, 2001, 2002],
.....: 'data': np.arange(5.)

In [44]: lefth

n [45]: righth

Out[44]:

	data	key1	key2
0	0	Ohio	2000
1	1	Ohio	2000
2	2	Ohio	2000
3	3	Ohio	2000
4	4	Ohio	2000

Out[45]:

		event1	event2
Nevada	2001	0	1
Nevada	2000	0	3
Ohio	2000	0	5
Ohio	2000	0	7
Ohio	2001	0	9
Ohio	2002	10	11

Trong trường hợp này, ta cần xác định nhiều cột để gộp thành một list(chú ý đến cách xử lý các chỉ số lặp):

In [46]: $pd.merge(lefth, righth, left_on = ['key1','key2'], right_index = True)$

Out[46]:

	data	key1	key2	event1	event2
3	3	Nevada	2001	0	1
0	0	Ohio	2000	4	5
0	0	Ohio	2000	6	7
1	1	Ohio	2000	8	9
2	2	Ohio	2000	10	11

In [47]: $pd.merge(lefth, righth, left_on = ['key1','key2'], right_index = True, how = 'outer')$

Out[47]:

	data	key1	key2	event1	event2
4	NaN	Nevada	2000	2	3
3	3	Nevada	2001	0	1
4	4	Nevada	2002	NaN	NaN
0	0	Ohio	2000	4	5
0	0	Ohio	2000	6	7
1	1	Ohio	2001	8	9
2	2	Ohio	2002	10	11

Sử dụng chỉ số của cả hai mảng hợp cũng không phải là một vấn đề quá khó:

In [48]: left2 = DataFrame([[1., 2.], [3., 4.], [5., 6.]], index=['a', 'c', 'e'],columns=['Ohio', 'Nevada'])

In [49]: right2 = DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],index=['b', 'c', 'd', 'e'], columns=['Missouri', 'Alabama'])

In [50]: left2

Out[50]:

	Ohio	Nevada
a	1	2
c	3	4
e	5	6

In [51]: right2

Out[51]:

	Missouri	Alabama
b	7	8
c	9	10
d	11	12

DataFrame có nhiều biến thể của **join** bằng cách gộp theo chỉ số. Ta cũng có thể gộp nhiều DataFrame bằng cách có nhiều chỉ số tương tự nhưng không chồng lặp các cột. Ở ví dụ trước, ta cũng có thể viết theo cách sau:

In [53]: left2.join(right2, how='outer')

Out[53]:

	Ohio	Nevada	Missouri	Alabama
a	1	2	b	7	8
c	3	4	c	9	10
e	5	6	d	11	12

Trong phần nguyên nhân thuộc tính, phương pháp **join** của DataFrame thực hiện gộp DataFrame đầu tiên theo các từ khóa. Phương thức này cũng hỗ trợ gộp theo chỉ số của DataFrame đã được tham chiếu đến dựa trên các cột của DataFrame được gọi:

In [54]: left1.join(right1, on='key')

Out[54]:

	key	value	groupVal
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

Cuối cùng, để gộp chỉ số theo chỉ số ta cần truyền vào một list các DataFrame để gộp theo cách tương tự dùng phương thức gộp chung được mô tả dưới đây:

```
In [55]: another = DataFrame([[7., 8.], [9., 10.], [11., 12.], [16., 17.]],
    ....:                    index=['a', 'c', 'e', 'f'], columns=['New York', 'Oregon']),

In [56]: left2.join([right2, another])
Out[56]:
```

Out[56]:

	Ohio	Nevada	Missouri	Alabama	New York	Oregon
a	1	2	NaN	NaN	7	8
c	3	4	9	10	9	10
e	5	6	13	14	11	12

1.3 Nối các tập dữ liệu theo cột

Một cách khác để kết hợp dữ liệu là ghép nối, liên kết hoặc xếp chồng. NumPy có chức năng nối để thực hiện việc này với các mảng NumPy thô:

```
In [58]: arr = np.arange(12).reshape((3, 4))
In [59]: arr
Out[59]:
array([[0, 1, 2, 3],
       [4, 5, 6, 7],
       [8, 9, 10, 11]])
In [60]: np.concatenate([arr, arr], axis=1)
Out[60]:
array([[ 0, 1, 2, 3, 0, 1, 2, 3],
       [4, 5, 6, 7, 4, 5, 6, 7],
       [8, 9, 10, 11, 8, 9, 10, 11]])
```

Trong ngữ cảnh của các đối tượng gấu trúc như Sê-ri và DataFrame, việc có các trục được gắn nhãn cho phép bạn khái quát hóa hơn nữa phép nối mảng. Đặc biệt, bạn có một số điều bổ sung để suy nghĩ về:

- Nếu các đối tượng được lập chỉ mục khác nhau trên các trục khác, thì bộ sưu tập của các trục được hợp nhất hoặc giao nhau?
- Các nhóm có cần được xác định trong đối tượng kết quả không
- Trục nối có quan trọng không?

Hàm concat trong pandas cung cấp một cách nhất quán để giải quyết từng mối quan tâm này. Tôi sẽ đưa ra một số ví dụ để minh họa cách nó hoạt động. Giả sử chúng ta có ba Sê-ri không có chỉ số trùng nhau:

```
In [61]: s1 = Series([0, 1], index=['a', 'b'])
In [62]: s2 = Series([2, 3, 4], index=['c', 'd', 'e'])
In [63]: s3 = Series([5, 6], index=['f', 'g'])
```

Gọi concat với các đối tượng này trong danh sách sẽ dán các giá trị và chỉ mục lại với nhau:

```
In [64]: pd.concat([s1, s2, s3])
Out[64]:
a 0
b 1
c 2
d 3
```

e 4
f 5
g 6

Theo mặc định, concat hoạt động dọc theo trục = 0, tạo ra một series khác. Nếu bạn vượt qua axis=1, thay vào đó, kết quả sẽ là một DataFrame (trục = 1 là các cột):

```
In [65]: pd.concat([s1, s2, s3], axis=1)
Out[65]:
     0    1    2
a    0  NaN  NaN
b    1  NaN  NaN
c  NaN    2  NaN
d  NaN    3  NaN
e  NaN    4  NaN
f  NaN  NaN    5
g  NaN  NaN    6
```

Trong trường hợp này, không có sự trùng lặp trên trục khác, như bạn có thể thấy là trục đã được sắp xếp union (nối 'bên ngoài') của các chỉ mục. Thay vào đó, bạn có thể cắt chúng bằng cách đi qua tham gia = 'bên trong':

```
In [66]: s4 = pd.concat([s1 * 5, s3])
In [67]: pd.concat([s1, s4], axis=1)
Out[67]:
   0 1
a  0 0
b  1 5
f NaN 5
g NaN 6
```

```
In [68]: pd.concat([s1, s4], axis=1, join='inner')
Out[68]:
  0 1
a 0 0
b 1 5
```

Bạn thậm chí có thể chỉ định các trục sẽ được sử dụng trên các trục khác với $join_axes$:

In [69]: $pd.concat([s1, s4], axis = 1, join_axes = [['a','c','b','e']])$

```
Out[69]:
   0   1
a  0   0
c NaN NaN
b  1   5
e NaN NaN
```

Một vấn đề là các phần được nối không thể xác định được trong kết quả. Giả sử thay vào đó, bạn muốn tạo một chỉ mục phân cấp trên trục nối. Để làm điều này,sử dụng đối số keys :

```
In [70]: result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])
In [71]: result
Out[71]:
one   a  0
      b  1
two   a  0
      b  1
three f  5
      g  6
```

In [72]: $result.unstack()$

```
Out[72]:
          a    b    f    g
```

```
one      0    1   NaN   NaN
two      0    1   NaN   NaN
three  NaN  NaN     5     6
```

Trong trường hợp kết hợp series dọc theo trục=1, các khóa trở thành tiêu đề cột DataFrame:

```
In [73]: pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three'])
Out[73]:
   one two three
a   0   NaN   NaN
b   1   NaN   NaN
c NaN     2   NaN
d NaN     3   NaN
e NaN     4   NaN
f NaN   NaN     5
g NaN   NaN     6
```

Logic tương tự mở rộng cho các đối tượng DataFrame:

```
In [74]: df1 = DataFrame(np.arange(6).reshape(3, 2), index=['a', 'b', 'c'],
....:                    columns=['one', 'two'])
In [75]: df2 = DataFrame(5 + np.arange(4).reshape(2, 2), index=['a', 'c'],
....:                    columns=['three', 'four'])
In [76]: pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
Out[76]:
        level1       level2
           one          two        three         four
a            0            1            5            6
b            2            3          NaN          NaN
c            4            5            7            8
```

Nếu bạn chuyển một lệnh của các đối tượng thay vì một danh sách, các phím của lệnh đó sẽ được sử dụng cho tùy chọn phím :

```
In [77]: pd.concat('level1': df1, 'level2': df2, axis=1)
Out[77]:
      level1 level2
      one   two   three    four
a      0      1   5          6
b      2      3   NaN       NaN
c      4      5   7          8
```

Có một số đối số bổ sung chi phối cách tạo chỉ mục phân cấp (xem Bảng 7-2):

```
In [78]: pd.concat([df1, df2], axis=1, keys=['level1', 'level2'],
....:       names=['upper', 'lower'])
Out[78]:
upper level1 level2
lower one   two   three    four
a      0      1   5          6
b      2      3   NaN       NaN
c      4      5   7          8
```

Cân nhắc cuối cùng liên quan đến DataFrames trong đó chỉ mục hàng không có ý nghĩa trong ngữ cảnh phân tích:

```
In [79]: df1 = DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])
In [80]: df2 = DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
```

In [81]: df1 In [82]: df2
Out[81]: Out[82]:

	a	b	c	d		b	d	a
0	-0.204708	0.478943	-0.519439	-0.555730	0	0.274992	0.228913	1.352917
1	1.965781	1.393406	0.092908	0.281746	1	0.886429	-2.001637	-0.371843
2	0.769023	1.246435	1.007189	-1.296221				

Trong trường hợp này, bạn có thể bỏ qua $ignore_index = True$:
In [83]: $pd.concat([df1, df2], ignore_index = True)$
Out[83]:

	a	b	c	d
0	-0.204708	0.478943	-0.519439	-0.555730
1	1.965781	1.393406	0.092908	0.281746
2	0.769023	1.246435	1.007189	-1.296221
3	1.352917	0.274992	NaN	0.228913
4	-0.371843	0.886429	NaN	-2.001637

Bảng 7-2: tham số của hàm concat

Tham số	**Mô tả**
objs	Danh sách các dictionary của các đối tượng pandas để concat
axis	Cột để bắt đầu concat, mặc định là cột 0
join	Cách concat: inner hoặc outer, mặc định là outer
join_axes	Chỉ số cụ thể để dùng cho các cột n-1 khác ngoại trừ các hàm lấy phần chung.
keys	Các giá trị để tham gia vào đối tượng được concat
levels	Các chỉ số cụ thể để sắp xếp theo thứ bậc
names	Tên cho các bậc chỉ số nếu key được truyền vào
verify_integrity	Kiểm tra nếu cột trong đối tượng được concat có bị lặp
ignore_index	Không đảo ngược chỉ số theo các cột được concat, mà trả về range(total_lenght)

Bảng 1.2: Các tham số của hàm **concat**

1.4 Kết hợp dữ liệu chéo

Tình huống kết hợp dữ liệu khác không thể được biểu thị dưới dạng thao tác hợp nhất hoặc nối quốc gia. Bạn có thể có hai tập dữ liệu có chỉ mục trùng nhau toàn bộ hoặc một phần. Như một ví dụ thúc đẩy, hãy xem xét hàm where của NumPy, biểu thị một if-else được vector hóa:

```
In [84]: a = Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],
   ....:       index=['f', 'e', 'd', 'c', 'b', 'a'])
In [85]: b = Series(np.arange(len(a), dtype=np.float64),
   ....:       index=['f', 'e', 'd', 'c', 'b', 'a'])
In [86]: b[-1] = np.nan
In [87]: a          In [88]: b          In [89]: np.where(pd.isnull(a), b, a)
Out[87]:            Out[88]:            Out[89]:
```

```
f NaN          f 0          f 0.0
e 2.5          e 1          e 2.5
d NaN          d 2          d 2.0
c 3.5          c 3          c 3.5
b 4.5          b 4          b 4.5
a NaN          a NaN        a NaN
```

Series có phương thức kết hợp đầu tiên , thực hiện tương đương với thao tác này cộng với căn chỉnh dữ liệu:

```
In [90]: b[:-2].combine_first(a[2:])
Out[90]:
a    NaN
b    4.5
c    3.0
d    2.0
e    1.0
f    0.0
```

Với DataFrames, tổ hợp first tự nhiên thực hiện cùng một việc theo từng cột, vì vậy bạn có thể coi đó là "vá" dữ liệu bị thiếu trong đối tượng gọi bằng dữ liệu từ đối tượng bạn chuyển:

```
In [91]: df1 = DataFrame('a': [1., np.nan, 5., np.nan],
   ....:                  'b': [np.nan, 2., np.nan, 6.],
   ....:                  'c': range(2, 18, 4))
In [92]: df2 = DataFrame('a': [5., 4., np.nan, 3., 7.],
   ....:                  'b': [np.nan, 3., 4., 6., 8.])
In [93]: df1.combine_first(df2)
Out[93]:
     a   b   c
0    1 NaN   2
1    4   2   6
2    5   4  10
3    3   6  14
4    7   8 NaN
```

Chương 2

Tái định dạng và xoay dữ liệu

Có một số thao tác cơ bản để sắp xếp lại dữ liệu dạng bảng. Chúng được gọi luân phiên là các hoạt động định hình lại hoặc trục .

2.1 Tái định dạng với đánh dấu theo kiểu thứ bậc

Lập chỉ mục theo cấp bậc cung cấp một cách nhất quán để sắp xếp lại dữ liệu trong DataFrame.

Có hai hành động chính:

- stack: xoay từ các cột trong dữ liệu sang các hàng
- unstack: xoay từ các hàng vào các cột

Ta sẽ minh họa các hoạt động này thông qua một loạt các ví dụ. Hãy xem xét một DataFrame nhỏ với các mảng chuỗi dưới dạng chỉ mục hàng và cột:

```
In [94]: data = DataFrame(np.arange(6).reshape((2, 3)),
    ....:                 index=pd.Index(['Ohio', 'Colorado'], name='state'),
    ....:                 columns=pd.Index(['one', 'two', 'three'], name='number'))
In [95]: data
Out[95]:
number one two three
state
Ohio      0   1   2
Colorado  3   4   5
```

Sử dụng phương thức ngăn xếp trên dữ liệu này sẽ chuyển các cột thành các hàng, tạo ra một series:

```
In [96]: result = data.stack()
In [97]: result
Out[97]:
state     number
Ohio      one       0
          two       1
          three     2
Colorado  one       3
          two       4
          three     5
```

Từ Series được lập chỉ mục theo cấp bậc, bạn có thể sắp xếp lại dữ liệu trở lại DataFrame bằng cách hủy ngăn xếp :

```
In [98]: result.unstack()
Out[98]:
number one   two   three
state
Ohio      0     1     2
Colorado 3      4     5
```

Theo mặc định, mức trong cùng không được xếp chồng lên nhau (tương tự với ngăn xếp). Bạn có thể hủy xếp chồng một cấp độ khác bằng cách chuyển số hoặc tên cấp độ:

```
In [99]: result.unstack(0)
Out[99]:
state Ohio Colorado
number
one      0   3
two      1   4
three   2   5
```

```
In [100]: result.unstack('state')
Out[100]:
 state Ohio Colorado
number
 one 0   3
 two 1   4
three 2   5
```

Unstacking có thể dẫn đến dữ liệu bị thiếu nếu tất cả các giá trị trong cấp độ không được tìm thấy trong mỗi nhóm con:

```
In [101]: s1 = Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
In [102]: s2 = Series([4, 5, 6], index=['c', 'd', 'e'])
In [103]: data2 = pd.concat([s1, s2], keys=['one', 'two'])
In [104]: data2.unstack()
Out[104]:
        a   b   c   d   e
one     0   1   2   3 NaN
two NaN NaN  4   5   6
```

Tính năng xếp chồng lọc ra dữ liệu bị thiếu theo mặc định, vì vậy hoạt động này dễ dàng đảo ngược:

```
In [105]: data2.unstack().stack()
Out[105]:
 one   a   0
       b   1
       c   2
       d   3
two    c   4
       d   5
       e   6
```

```
In [106]: data2.unstack().stack(dropna=False)
Out[106]:
  one   a   0
        b   1
        c   2
        d   3
        e   NaN
two     a   NaN
        b   NaN
        c   4
        d   5
        e   6
```

When unstacking in a DataFrame, the level unstacked becomes the lowest level in the result:

```
In [107]: df = DataFrame('left': result, 'right': result + 5,
   .....:                columns=pd.Index(['left', 'right'], name='side'))
In [108]: df
Out[108]:
side                left right
```

```
state       number
Ohio        one      0   5
            two      1   6
            three    2   7
Colorado    one      3   8
            two      4   9
            three    5   10
```

In [109]: df.unstack('state')
Out[109]:

side	left		right	
state	Ohio	Colorado	Ohio	Colorado
number				
one	0	3	5	8
two	1	4	6	9
three	2	5	7	10

In [110]: df.unstack('state').stack('side')
Out[110]:

	state	Ohio	Colorado
number	side		
one	left	0	3
	right	5	8
two	left	1	4
	right	6	9
three	left	2	5
	right	7	10

2.2 Xoay từ định dạng "dài" sang "rộng"

Một cách phổ biến để lưu trữ nhiều chuỗi thời gian trong cơ sở dữ liệu và CSV được gọi là định dạng dài hoặc xếp chồng :

In [116]: ldata[:10]
Out[116]:

	date	item	value
0	1959-03-31 00:00:00	realgdp	2710.349
1	1959-03-31 00:00:00	infl	0.000
2	1959-03-31 00:00:00	unemp	5.800
3	1959-06-30 00:00:00	realgdp	2778.801
4	1959-06-30 00:00:00	infl	2.340
5	1959-06-30 00:00:00	unemp	5.100
6	1959-09-30 00:00:00	realgdp	2775.488
7	1959-09-30 00:00:00	infl	2.740
8	1959-09-30 00:00:00	unemp	5.300
9	1959-12-31 00:00:00	realgdp	2785.204

Dữ liệu thường được lưu trữ theo cách này trong cơ sở dữ liệu quan hệ như MySQL dưới dạng lược đồ cố định (tên cột và kiểu dữ liệu) cho phép số lượng giá trị riêng biệt trong cột mục tăng hoặc giảm khi dữ liệu được thêm hoặc xóa trong bảng. Trong ví dụ trên , ngày và mục thường sẽ là khóa chính (theo cách nói của cơ sở dữ liệu quan hệ), cung cấp cả tính toàn vẹn của quan hệ cũng như các truy vấn lập trình và tham gia dễ dàng hơn trong nhiều trường hợp. Tất nhiên, nhược điểm là dữ liệu có thể không dễ làm việc với định dạng dài; bạn có thể muốn có một Khung dữ liệu chứa một cột cho mỗi giá trị mục riêng biệt được lập chỉ mục theo dấu thời gian trong cột ngày . Phương pháp trục của DataFrame trên mỗi biểu mẫu chính xác là phép biến đổi này: Trong [117]: pivoted = ldata.pivot('date', 'item', 'value')

In [117]: pivoted = ldata.pivot('date', 'item', 'value')
In [118]: pivoted.head()
Out[118]:

```
item           infl   realgdp   unemp
date
1959-03-31   0.00   2710.349   5.8
1959-06-30   2.34   2778.801   5.1
1959-09-30   2.74   2775.488   5.3
1959-12-31   0.27   2785.204   5.6
1960-03-31   2.31   2847.699   5.2
```

Hai giá trị đầu tiên được chuyển là các cột được sử dụng làm chỉ mục hàng và cột, cuối cùng là cột giá trị tùy chọn để điền vào Khung dữ liệu. Giả sử bạn có hai cột giá trị mà bạn muốn định hình lại đồng thời:

```
In [119]: ldata['value2'] = np.random.randn(len(ldata))
In [120]: ldata[:10]
Out[120]:
                  date     item     value     value2
0   1959-03-31 00:00:00   realgdp   2710.349 1.669025
1   1959-03-31 00:00:00   infl       0.000   -0.438570
2   1959-03-31 00:00:00   unemp     5.800   -0.539741
3   1959-06-30 00:00:00   realgdp   2778.801 0.476985
4   1959-06-30 00:00:00   infl       2.340   3.248944
5   1959-06-30 00:00:00   unemp     5.100   -1.021228
6   1959-09-30 00:00:00   realgdp   2775.488 -0.577087
7   1959-09-30 00:00:00   infl       2.740   0.124121
8   1959-09-30 00:00:00   unemp     5.300   0.302614
9   1959-12-31 00:00:00   realgdp   2785.204   0.523772
```

Bằng cách bỏ qua đối số cuối cùng, bạn có được DataFrame với các cột phân cấp:

```
In [121]: pivoted = ldata.pivot('date', 'item')
In [122]: pivoted[:5]
Out[122]:
              value                    value2
item          infl   realgdp   unemp   infl       realgdp       unemp
date
1959-03-31   0.00   2710.349   5.8   -0.438570   1.669025   -0.539741
1959-06-30   2.34   2778.801   5.1   3.248944   0.476985   -1.021228
1959-09-30   2.74   2775.488   5.3   0.124121   -0.577087   0.302614
1959-12-31   0.27   2785.204   5.6   0.000940   0.523772   1.343810
1960-03-31   2.31   2847.699   5.2   -0.831154   -0.713544   -2.370232
In [123]: pivoted['value'][:5]
Out[123]:
item        infl      realgdp   unemp
date
1959-03-31 0.00   2710.349   5.8
1959-06-30 2.34   2778.801   5.1
1959-09-30 2.74   2775.488   5.3
1959-12-31 0.27   2785.204   5.6
1960-03-31 2.31   2847.699   5.2
```

Lưu ý rằng trục xoay chỉ là lối tắt để tạo chỉ mục phân cấp bằng cách sử dụng set_index và định hình lại bằng unstack:

```
In [124]: unstacked = ldata.set_index(['date', 'item']).unstack('item')
```

In [125]: unstacked[:7]
Out[125]:

	value			value2		
item	infl	realgdp	unemp	infl	realgdp	unemp
date						
1959-03-31	0.00	2710.349	5.8	-0.438570	1.669025	-0.539741
1959-06-30	2.34	2778.801	5.1	3.248944	0.476985	-1.021228
1959-09-30	2.74	2775.488	5.3	0.124121	-0.577087	0.302614
1959-12-31	0.27	2785.204	5.6	0.000940	0.523772	1.343810
1960-03-31	2.31	2847.699	5.2	-0.831154	-0.713544	-2.370232
1960-06-30	0.14	2834.390	5.2	-0.860757	-1.860761	0.560145
1960-09-30	2.70	2839.022	5.6	0.119827	-1.265934	-1.063512

Chương 3

Chuyển đổi dữ liệu

Cho đến giờ trong chương này, chúng ta đã quan tâm đến việc sắp xếp lại dữ liệu. Lọc, làm sạch, và các phép biến đổi khác là một loại hoạt động quan trọng khác.

3.1 Loại bỏ trùng lặp

Các hàng trùng lặp có thể được tìm thấy trong DataFrame vì bất kỳ lý do nào. Đây là một thí dụ:

```
In [126]: data = DataFrame('k1': ['one'] * 3 + ['two'] * 4,
 .....:                    'k2': [1, 1, 2, 3, 3, 4, 4])
In [127]: data
Out[127]:
   k1   k2
0 one    1
1 one    1
2 one    2
3 two    3
4 two    3
5 two    4
6 two    4
```

Phương thức DataFrame được sao chép trả về một Sê-ri boolean cho biết liệu mỗi hàng có trùng lặp hay không:

```
In [128]: data.duplicated()
Out[128]:
0   False
1   True
2   False
3   False
4   True
5   False
6   True
```

Liên quan, drop_duplicates trả về một DataFrame trong đó mảng trùng lặp là True:

```
In [129]: data.drop_duplicates()
Out[129]:
  k1   k2
```

```
0 one    1
2 one    2
3 one    3
5 two    4
```

Theo mặc định, cả hai phương pháp này đều xem xét tất cả các cột; cách khác bạn có thể chỉ định bất kỳ tập hợp con nào của chúng để phát hiện các bản sao. Giả sử chúng ta có thêm một cột của các giá trị và chỉ muốn lọc các giá trị trùng lặp dựa trên cột 'k1':

```
In [130]: data['v1'] = range(7)
In [131]: data.drop_duplicates(['k1'])
Out[131]:
   k1 k2 v1
0 one 1 0
3 two 3 3
```

Duplicate và drop-uplicates theo mặc định giữ kết hợp giá trị được quan sát đầu tiên. Vượt qua take_last=True sẽ trả về cái cuối cùng:

```
 In [132]: data.drop_duplicates(['k1', 'k2'], take_last=True)
Out[132]:
   k1 k2 v1
1 one 1 1
2 one 2 2
4 two 3 4
6 two 4 6
```

3.2 Biến đổi dữ liệu sử dụng chức năng ánh xạ

Đối với nhiều tập dữ liệu, bạn có thể muốn thực hiện một số chuyển đổi dựa trên các giá trị trong một mảng, Sê-ri hoặc cột trong DataFrame. Hãy xem xét các dữ liệu giả định sau đây sưu tầm về một số loại thịt:

```
In [133]: data = DataFrame('food': ['bacon', 'pulled pork', 'bacon', 'Pastrami',
 .....:                             corned beef', 'Bacon', 'pastrami', 'honey ham',
 .....:                            'nova lox'],
 .....:                             'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6])
In [134]: data
Out[134]:
        food      ounces
0    bacon          4.0
1    pulled pork    3.0
2    bacon         12.0
3    Pastrami       6.0
4    corned beef    8.0
5    Bacon          7.5
6    pastrami       3.0
7    honey ham      5.0
8    nova lox       6.0
```

Giả sử bạn muốn thêm một cột cho biết loại động vật mà mỗi loại thực phẩm đến từ. Hãy viết ra một ánh xạ của từng loại thịt riêng biệt cho loại động vật:

```
 meat_to_animal = {
```

```
    'bacon': 'pig',
    'pulled pork': 'pig',
    'pastrami': 'cow',
    'corned beef': 'cow',
    'honey ham': 'pig',
    'nova lox': 'salmon'
```

Phương thức map trên Sê-ri chấp nhận một hàm hoặc đối tượng giống như dict chứa ánh xạ, nhưng ở đây chúng ta có một vấn đề nhỏ là một số loại thịt ở trên được viết hoa và những người khác thì không. Vì vậy, chúng ta cũng cần chuyển đổi từng giá trị thành chữ thường:

```
In [136]: data['animal'] = data['food'].map(str.lower).map(meat_to_animal)
In [137]: data
Out[137]:
        food  ounces  animal
0  bacon        4.0     pig
1  pulled pork  3.0     pig
2  bacon       12.0     pig
3  Pastrami     6.0     cow
4  corned beef  7.5     cow
5  Bacon        8.0     pig
6  pastrami     3.0     cow
7  honey ham    5.0     pig
8  nova lox     6.0     salmon
```

Chúng ta cũng có thể truyền vào một hàm thực hiện tất cả công việc:

```
In [138]: data['food'].map(lambda x: meat_to_animal[x.lower()])
Out[138]:
0       pig
1       pig
1       pig
2       pig
3       cow
4       cow
5       pig
6       cow
7       pig
8       salmon
Name: food
```

Sử dụng bản đồ là một cách thuận tiện để thực hiện các phép biến đổi phần tử và dữ liệu khác các hoạt động liên quan đến vệ sinh.

3.3 Thay thế giá trị

Điền vào dữ liệu còn thiếu bằng phương thức fillna có thể được coi là một trường hợp đặc biệt của thay thế giá trị tổng quát hơn. Trong khi bản đồ, như bạn đã thấy ở trên, có thể được sử dụng để sửa đổi một tập hợp con các giá trị trong một đối tượng, thay thế cung cấp một cách đơn giản và linh hoạt hơn để thực hiện vì thế. Hãy xem xét Series này:

```
In [139]: data = Series([1., -999., 2., -999., -1000., 3.])
In [140]: data
```

Out[140]:
0 1
1 -999
2 2
3 -999
4 -1000
5 3

Các giá trị -999 có thể là giá trị trọng điểm cho dữ liệu bị thiếu. Để thay thế chúng bằng NaN các giá trị mà gấu trúc hiểu được, chúng ta có thể sử dụng thay thế, tạo ra một Sê-ri mới:

In [141]: data.replace(-999, np.nan)
Out[141]:
0 1
1 NaN
2 2
3 NaN
4 -1000
5 3

Nếu bạn muốn thay thế nhiều giá trị cùng một lúc, thay vào đó, bạn chuyển một danh sách rồi thay thế giá trị:

In [142]: data.replace([-999, -1000], np.nan)
Out[142]:
0 1
1 NaN
2 2
3 NaN
4 NaN
5 3

Để sử dụng một thay thế khác cho mỗi giá trị, hãy chuyển danh sách thay thế:

In [143]: data.replace([-999, -1000], [np.nan, 0])
Out[143]:
0 1
1 NaN
2 2
3 NaN
4 0
5 3

Đối số được thông qua cũng có thể là một lệnh:

In [144]: data.replace(-999: np.nan, -1000: 0)
Out[144]:
0 1
1 NaN
2 2
3 NaN
4 0
5 3

3.4 Đổi tên trục Indexes

Giống như các giá trị trong Sê-ri, nhãn trục có thể được chuyển đổi tương tự bằng hàm hoặc ánh xạ của một số dạng để tạo ra các đối tượng mới, được dán nhãn khác nhau. Các trục cũng có thể được sửa đổi tại chỗ mà không cần tạo cấu trúc dữ liệu mới. Đây là một ví dụ đơn giản:

```
In [145]: data = DataFrame(np.arange(12).reshape((3, 4)),
 .....:                    index=['Ohio', 'Colorado', 'New York'],
 .....:                    columns=['one', 'two', 'three', 'four'])
```

Giống như một Sê-ri, các chỉ mục trục có một phương thức bản đồ:

```
In [146]: data.index.map(str.upper)
Out[146]: array([OHIO, COLORADO, NEW YORK], dtype=object)
```

Bạn có thể gán cho chỉ mục, sửa đổi DataFrame tại chỗ:

```
In [147]: data.index = data.index.map(str.upper)
In [148]: data
Out[148]:
          one two three four
OHIO        0   1     2    3
COLORADO    4   5     6    7
NEW YORK    8   9    10   11
```

Nếu bạn muốn tạo phiên bản đã chuyển đổi của tập dữ liệu mà không sửa đổi bản gốc, một phương pháp hữu ích là đổi tên:

```
In [149]: data.rename(index=str.title, columns=str.upper)
Out[149]:
          ONE TWO THREE FOUR
OHIO        0   1     2    3
COLORADO    4   5     6    7
NEW YORK    8   9    10   11
```

Đáng chú ý, đổi tên có thể được sử dụng cùng với một đối tượng giống như dict cung cấp các giá trị mới cho một tập hợp con của các nhãn trục:

```
In [150]: data.rename(index= { 'OHIO': 'INDIANA'} ,
 .....:                columns='three': 'peekaboo')
Out[150]:
          one  two peekaboo  four
INDIANA     0    1        2     3
COLORADO    4    5        6     7
NEW YORK    8    9       10    11
```

đổi tên tiết kiệm phải sao chép DataFrame theo cách thủ công và gán cho chỉ mục và col của nó thuộc tính umns. Nếu bạn muốn sửa đổi tập dữ liệu tại chỗ, hãy chuyển inplace=True:

Luôn trả về tham chiếu đến DataFrame

```
In [151]: _data.rename(index='OHIO': 'INDIANA', inplace=True)
In [152]: data
Out[152]:
          one   two   three   four
INDIANA     0     1       2      3
COLORADO    4     5       6      7
NEW YORK    8     9      10     11
```

3.5 Rời rạc hóa và Binning

Dữ liệu liên tục thường được rời rạc hóa hoặc được tách thành các "thùng" để phân tích. Giả sử bạn có dữ liệu về một nhóm người trong một nghiên cứu và bạn muốn nhóm họ thành các nhóm tuổi rời rạc:

```
In [153]: ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
```

Hãy chia chúng thành các thùng từ 18 đến 25, 26 đến 35, 35 đến 60 và cuối cùng là 60 trở lên. Đến làm như vậy, bạn phải sử dụng hàm cut trong pandas:

```
In [154]: bins = [18, 25, 35, 60, 100]
In [155]: cats = pd.cut(ages, bins)
In [156]: cats
Out[156]:
Categorical:
array([(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], (18, 25],
       (35, 60], (25, 35], (60, 100], (35, 60], (35, 60], (25, 35]], dtype=object)
 Levels (4): Index([(18, 25], (25, 35], (35, 60], (60, 100]], dtype=object)
```

Đối tượng pandas trả về là một đối tượng Phân loại đặc biệt. Bạn có thể coi nó như một mảng của các chuỗi cho biết tên bin; bên trong nó chứa một mảng cấp độ cho biết tên danh mục riêng biệt cùng với nhãn cho dữ liệu độ tuổi trong thuộc tính nhãn:

```
In [157]: cats.labels
Out[157]: array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1])
In [158]: cats.levels
Out[158]: Index([(18, 25], (25, 35], (35, 60], (60, 100]], dtype=object)
In [159]: pd.value_counts(cats)
Out[159]:
(18, 25]    5
(35, 60]    3
(25, 35]    3
(60, 100]   1
```

Phù hợp với ký hiệu toán học cho các khoảng, dấu ngoặc đơn có nghĩa là cạnh đang mở trong khi dấu ngoặc vuông có nghĩa là nó đã đóng (bao gồm). Có thể đóng cửa bên nào được thay đổi bằng cách chuyển sang phải=Sai:

```
In [160]: pd.cut(ages, [18, 26, 36, 61, 100], right=False)
Out[160]:
Categorical:
array([[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), [18, 26),
       [36, 61), [26, 36), [61, 100), [36, 61), [36, 61), [26, 36)], dtype=object)
Levels (4): Index([[18, 26), [26, 36), [36, 61), [61, 100)], dtype=object)
```

Bạn cũng có thể chuyển tên thùng rác của riêng mình bằng cách chuyển danh sách hoặc mảng vào tùy chọn nhãn:

```
In [161]: group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
In [162]: pd.cut(ages, bins, labels=group_names)
Out[162]:
Categorical:
array([Youth, Youth, Youth, YoungAdult, Youth, Youth, MiddleAged
       YoungAdult, Senior, MiddleAged, MiddleAged, YoungAdult], dtype=object)
Levels (4): Index([Youth, YoungAdult, MiddleAged, Senior], dtype=object)
```

Nếu bạn vượt qua cắt một số lượng thùng thay vì các cạnh thùng rõ ràng, nó sẽ tính toán các thùng có chiều dài bằng nhau dựa trên các giá trị tối thiểu và tối đa trong dữ liệu. Xem xét trường hợp một số dữ liệu được phân phối đồng đều được chia thành phần tư:

```
In [163]: data = np.random.rand(20)
In [164]: pd.cut(data, 4, precision=2)
Out[164]:
Categorical:
array([(0.45, 0.67], (0.23, 0.45], (0.0037, 0.23], (0.45, 0.67],
       (0.67, 0.9], (0.45, 0.67], (0.67, 0.9], (0.23, 0.45], (0.23, 0.45],
       (0.67, 0.9], (0.67, 0.9], (0.67, 0.9], (0.23, 0.45], (0.23, 0.45],
       (0.23, 0.45], (0.67, 0.9], (0.0037, 0.23], (0.0037, 0.23],
       (0.23, 0.45], (0.23, 0.45]], dtype=object)
Levels (4): Index([(0.0037, 0.23], (0.23, 0.45], (0.45, 0.67],
                   (0.67, 0.9]], dtype=object)
```

Một chức năng liên quan chặt chẽ, qcut, xử lý dữ liệu dựa trên các lượng tử mẫu. Tùy về phân phối dữ liệu, sử dụng cắt thường sẽ không dẫn đến việc mỗi ngăn có cùng số điểm dữ liệu. Vì qcut sử dụng lượng tử mẫu thay thế, nên theo định nghĩa bạn sẽ thu được các thùng có kích thước gần bằng nhau:

```
In [165]: data = np.random.randn(1000)
In [166]: cats = pd.qcut(data, 4)
In [167]: cats
Out[167]:
Categorical:
array([(-0.022, 0.641], [-3.745, -0.635], (0.641, 3.26], ...,
       (-0.635, -0.022], (0.641, 3.26], (-0.635, -0.022]], dtype=object)
Levels (4): Index([[-3.745, -0.635], (-0.635, -0.022], (-0.022, 0.641],
                   (0.641, 3.26]], dtype=object)
In [168]: pd.value_counts(cats)
[-3.745, -0.635]   250
(0.641, 3.26]      250
(-0.635, -0.022]   250
(-0.022, 0.641]    250
```

Tương tự như cắt, bạn có thể chuyển các lượng tử của riêng mình (bao gồm các số từ 0 đến 1):

```
In [169]: pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.])
Out[169]:
Categorical:
array([(-0.022, 1.302], (-1.266, -0.022], (-0.022, 1.302], ...,
       (-1.266, -0.022], (-0.022, 1.302], (-1.266, -0.022]], dtype=object)
Levels (4): Index([[-3.745, -1.266], (-1.266, -0.022], (-0.022, 1.302],
                   (1.302, 3.26]], dtype=object)
```

Chúng ta sẽ quay lại cut và qcut sau trong chương về phép toán tổng hợp và nhóm, vì các hàm rời rạc này đặc biệt hữu ích cho phân tích nhóm và lượng tử.

3.6 Phát hiện và lọc các ngoại lệ

Lọc hoặc chuyển đổi các ngoại lệ phần lớn là vấn đề áp dụng các hoạt động mảng. Consider một DataFrame với một số dữ liệu được phân phối bình thường:

```
In [170]: np.random.seed(12345)
In [171]: data = DataFrame(np.random.randn(1000, 4))
In [172]: data.describe()
Out[172]:
                0            1            2            3
count 1000.000000 1000.000000 1000.000000 1000.000000
mean -0.067684     0.067924     0.025598    -0.002298
std    0.998035     0.992106     1.006835     0.996794
min -3.428254    -3.548824    -3.184377    -3.745356
25% -0.774890     -0.591841     -0.641675     -0.644144
50% -0.116401     0.101143     0.002073     -0.013611
75% 0.616366     0.780282     0.680391     0.654328
max 3.366626     2.653656     3.260383     3.927528
```

Giả sử bạn muốn tìm các giá trị ở một trong các cột vượt quá ba độ lớn:

```
In [173]: col = data[3]
In [174]: col[np.abs(col) > 3]
Out[174]:
97     3.927528
305   -3.399312
400    -3.745356
Name: 3
```

Để chọn tất cả các hàng có giá trị vượt quá 3 hoặc -3, bạn có thể sử dụng bất kỳ phương pháp nào trên một Khung dữ liệu boolean:

```
In [175]: data[(np.abs(data) > 3).any(1)]
Out[175]:
         0            1            2            3
5 -0.539741     0.476985     3.248944    -1.021228
97 -0.774363    0.552936    0.106061    3.927528
102 -0.655054    -0.565230    3.176873    0.959533
305 -2.315555    0.457246    -0.025907    -3.399312
324 0.050188    1.951312    3.260383    0.963301
400 0.146326    0.508391    -0.196713    -3.745356
499 -0.293333    -0.242459    -3.056990    1.918403
523 -3.428254    -0.296336    -0.439938    -0.867165
586 0.275144    1.179227    -3.184377    1.369891
808 -0.362528    -3.548824    1.553205    -2.186301
900 3.366626    -2.372214    0.851010    1.332846
```

Các giá trị có thể dễ dàng được thiết lập dựa trên các tiêu chí này. Đây là mã giới hạn các giá trị bên ngoài khoảng -3 đến 3:

```
In [176]: data[np.abs(data) > 3] = np.sign(data) * 3
In [177]: data.describe()
Out[177]:
                     0            1            2            3
count 1000.000000 1000.000000 1000.000000 1000.000000
```

```
mean    -0.067623    0.068473    0.025153    -0.002081
std      0.995485    0.990253     1.003977    0.989736
min     -3.000000    -3.000000    -3.000000    -3.000000
25%     -0.774890     -0.591841    -0.641675    -0.644144
50%    -0.116401    0.101143    0.002073      -0.013611
75%     0.616366    0.780282    0.680391      0.654328
max     3.000000    2.653656    3.000000      3.000000
```

ufunc np.sign trả về một mảng 1 và -1 tùy thuộc vào dấu của các giá trị.

3.7 Hoán vị và lấy mẫu ngẫu nhiên

Việc hoán vị (sắp xếp lại ngẫu nhiên) một Sê-ri hoặc các hàng trong DataFrame rất dễ thực hiện bằng cách sử dụng chức năng numpy.random.permutation. Gọi hoán vị với độ dài của trục bạn muốn hoán vị tạo ra một mảng các số nguyên biểu thị thứ tự mới:

```
In [178]: df = DataFrame(np.arange(5 * 4).reshape(5, 4))
In [179]: sampler = np.random.permutation(5)
In [180]: sampler
Out[180]: array([1, 0, 2, 3, 4])
```

Sau đó, mảng đó có thể được sử dụng trong lập chỉ mục dựa trên ix hoặc hàm take:

```
In [181]: df
Out[181]:
    0   1   2   3
0   0   1   2   3
1   4   5   6   7
2   8   9   10  11
3   12  13  14  15
4   16  17  18  19
```

```
In [182]: df.take(sampler)
Out[182]:
    0   1   2   3
1   4   5   6   7
0   0   1   2   3
2   8   9   10  11
3   12  13  14  15
4   16  17  18  19
```

Để chọn một tập hợp con ngẫu nhiên mà không cần thay thế, một cách là cắt bỏ phần tử k đầu tiên ment của mảng được trả về bằng phép hoán vị, trong đó k là kích thước tập hợp con mong muốn. Ở đó là các thuật toán lấy mẫu không cần thay thế hiệu quả hơn nhiều, nhưng đây là một thuật toán dễ chiến lược sử dụng các công cụ sẵn có:

```
In [183]: df.take(np.random.permutation(len(df))[:3])
Out[183]:
    0   1   2   3
1   4   5   6   7
3   12  13  14  15
4   16  17  18  19
```

Để tạo mẫu có thay thế, cách nhanh nhất là sử dụng np.random.randint để vẽ số nguyên ngẫu nhiên:

```
In [184]: bag = np.array([5, 7, -1, 6, 4])
In [185]: sampler = np.random.randint(0, len(bag), size=10)
In [186]: sampler
Out[186]: array([4, 4, 2, 2, 2, 0, 3, 0, 4, 1])
In [187]: draws = bag.take(sampler)
In [188]: draws
Out[188]: array([ 4, 4, -1, -1, -1, 5, 6, 5, 4, 7])
```

Chương 4

Các thao tác với chuỗi

Python từ lâu đã là một ngôn ngữ trộn dữ liệu phổ biến một phần do nó dễ sử dụng để xử lý chuỗi và văn bản. Hầu hết các thao tác văn bản được thực hiện đơn giản với các phương thức tích hợp sẵn của đối tượng chuỗi. Đối với các thao tác văn bản và đối sánh mẫu phức tạp hơn, có thể cần đến các biểu thức thông thường. Pandas thêm vào hỗn hợp bằng cách cho phép bạn áp dụng chuỗi và biểu thức chính quy một cách chính xác trên toàn bộ mảng dữ liệu, ngoài ra còn xử lý sự khó chịu của dữ liệu bị thiếu

4.1 Các phương thức đối tượng của String

Trong nhiều ứng dụng trộn chuỗi và tạo tập lệnh, các phương thức chuỗi tích hợp sẵn là đủ. Ví dụ: một chuỗi được phân tách bằng dấu phẩy có thể được chia thành nhiều phần bằng cách chia:

In [208]: val = 'a,b, guido'
In [209]: val.split(',')
Out[209]: ['a', 'b', ' guido']

spilit thường được kết hợp với strip để xóa khoảng (kể cả dòng mới):

In [210]: pieces = [x.strip() for x in val.split(',')]
In [211]: pieces
Out[211]: ['a', 'b', 'guido']

Các chuỗi con này có thể được nối với nhau bằng dấu phân cách hai dấu hai chấm bằng cách sử dụng điều kiện bổ sung:

In [212]: first, second, third = pieces
In [213]: first + '::' + second + '::' + third
Out[213]: 'a::b::guido'

Tuy nhiên, đây không phải là một phương pháp chung thực tế. Một cách nhanh hơn và nhiều Pythonic hơn là chuyển một danh sách hoặc bộ dữ liệu sang phương thức nối trên chuỗi '::' :

In [214]: '::'.join(pieces)
Out[214]: 'a::b::guido'

Các phương pháp khác liên quan đến việc định vị các chuỗi con. Sử dụng từ khóa trong Python là cách tốt nhất để phát hiện một chuỗi con, mặc dù cũng có thể sử dụng chỉ mục và tìm kiếm

In [215]: 'guido' in val
Out[215]: True
In [216]: val.index(',') In [217]: val.find(':')
Out[216]: 1 Out[217]: -1

Lưu ý sự khác biệt giữa tìm và chỉ mục là chỉ mục đưa ra một ngoại lệ nếu không tìm thấy chuỗi (so với trả về -1):

```
In [218]: val.index(':')
---------------------------------------------------------------
ValueError Traceback (most recent call last)
<ipython-input-218-280f8b2856ce> in <module>()
—> 1 val.index(':')
ValueError: substring not found
```

Liên quan, số lượng trả về số lần xuất hiện của một chuỗi con cụ thể:

```
In [219]: val.count(',')
Out[219]: 2
```

replace sẽ thay thế các lần xuất hiện của mẫu này cho mẫu khác. Điều này cũng thường được sử dụng để xóa các mẫu bằng cách chuyển một chuỗi trống:

```
In [220]: val.replace(',', '::')
Out[220]: 'a::b:: guido'

In [221]: val.replace(',', '')
Out[221]: 'ab guido'
```

Biểu thức chính quy cũng có thể được sử dụng với nhiều thao tác như bạn sẽ thấy bên dưới.

Bảng 7-3. Các phương thức chuỗi tích hợp trong Python

Argument	Description
count	Trả về số lần xuất hiện không chồng lấp của chuỗi con trong chuỗi.
endswith, startswith	Trả về Đúng nếu chuỗi kết thúc bằng hậu tố (bắt đầu bằng tiền tố).
join	Sử dụng chuỗi làm dấu phân cách để nối một chuỗi các chuỗi khác.
index	Trả về vị trí của ký tự đầu tiên trong chuỗi con nếu tìm thấy trong chuỗi. Tăng ValueError nếu không tìm thấy.
find	Trả về vị trí của ký tự đầu tiên xuất hiện đầu tiên của chuỗi con trong chuỗi. Giống như index, nhưng trả về -1 nếu không tìm thấy.
rfind	Trả về vị trí của ký tự đầu tiên xuất hiện cuối cùng của chuỗi con trong chuỗi. Trả về -1 nếu không tìm thấy.
replace	Thay thế các lần xuất hiện của chuỗi bằng một chuỗi khác.
strip, rstrip, lstrip	Cắt khoảng trắng, bao gồm cả dòng mới; tương đương với x.strip() (and rstrip, lstrip, respectively) cho mỗi phần tử.
split	Ngắt chuỗi thành danh sách các chuỗi con bằng cách sử dụng dấu phân cách đã truyền.
lower, upper	Chuyển đổi các ký tự bảng chữ cái thành chữ thường hoặc chữ hoa tương ứng.
ljust, rjust	Căn trái hoặc căn phải tương ứng. Đệm phía đối diện của chuỗi bằng dấu cách (hoặc một số ký tự điền khác) để trả về chuỗi có chiều rộng tối thiểu.

Bảng 4.1: Các tham số của hàm **merge**

4.2 Biểu thức chính quy

Biểu thức chính quy cung cấp một cách linh hoạt để tìm kiếm hoặc so khớp các mẫu chuỗi trong văn bản. Một biểu thức đơn, thường được gọi là biểu thức chính quy, là một chuỗi được tạo theo ngôn ngữ biểu thức chính quy. Mô-đun re tích hợp sẵn của Python chịu trách nhiệm áp dụng các biểu thức chính quy cho các chuỗi; Tôi sẽ đưa ra một số ví dụ về việc sử dụng nó ở đây.
Các chức năng của mô-đun lại được chia thành ba loại: so khớp mẫu, thay thế và tách. Đương nhiên, tất cả những điều này đều có liên quan với nhau; một biểu thức chính quy mô tả một mẫu để xác định vị trí trong văn bản, sau đó có thể được sử dụng cho nhiều mục đích. Hãy xem một ví dụ đơn giản: giả sử tôi muốn tách một chuỗi có số ký tự khoảng trắng thay đổi (tab, dấu cách và dòng mới). Biểu thức chính quy mô tả một hoặc nhiều ký tự khoảng trắng là $\backslash s+$:

In [223]: text = "$foobar\backslash tbaz\backslash tqux$"
In [224]: re.split($'\backslash s+'$,$text$)
Out[224]: ['foo', 'bar', 'baz', 'qux']

Khi bạn gọi re.split($'\backslash s+'$,$text$) trước tiên, biểu thức chính quy được biên dịch, sau đó phương thức tách của nó được gọi trên văn bản đã truyền. Bạn có thể tự biên dịch biểu thức chính quy với re.compile, tạo thành một đối tượng biểu thức chính quy có thể tái sử dụng:

In [225]: regex = re.compile('$\backslash s+$')
In [226]: regex.split(text)
Out[226]: ['foo', 'bar', 'baz', 'qux']

Thay vào đó, nếu bạn muốn lấy danh sách tất cả các mẫu khớp với biểu thức chính quy, bạn có thể sử dụng phương thức findall:

In [227]: regex.findall(text)
Out[227]: [' ', '\ t ', ' \ t']

Tạo một đối tượng regex với re.compile rất được khuyến khích nếu bạn có ý định áp dụng cùng một biểu thức cho nhiều chuỗi; làm như vậy sẽ tiết kiệm chu kỳ CPU.match và tìm kiếm có liên quan chặt chẽ với findall. Trong khi findall trả về tất cả các kết quả khớp trong một chuỗi, tìm kiếm chỉ trả về kết quả khớp đầu tiên. Khéo léo hơn, match chỉ khớp ở đầu chuỗi. Như một ví dụ ít tầm thường hơn, hãy xem xét một khối văn bản và một biểu thức chính quy có khả năng xác định hầu hết các địa chỉ email:

text = """$Davedave@google.com$
$Stevesteve@gmail.com$
$Robrob@gmail.com$
$Ryanryan@yahoo.com$
"""
$pattern = r'[A-Z0-9._\%+-]+@[A-Z0-9.-]+\.[A-Z]2,4'$
$\#re.IGNORECASEmakestheregexcase-insensitive$
$regex = re.compile(pattern, flags = re.IGNORECASE)$

Sử dụng findall trên văn bản sẽ tạo ra một danh sách các địa chỉ e-mail:

In [229]: $regex.findall(text)$
Out[229]: $['dave@google.com','steve@gmail.com','rob@gmail.com','ryan@yahoo.com']$

Tìm kiếm trả về một đối tượng khớp đặc biệt cho địa chỉ email đầu tiên trong văn bản. Đối với biểu thức chính quy trên, đối tượng khớp chỉ có thể cho chúng ta biết vị trí bắt đầu và kết thúc của mẫu trong chuỗi:

In [230]: m = regex.search(text)
In [231]: m

Out[231]: <sre.SREMatch at 0x10a05de00>
In [232]: text[m.start():m.end()]
Out[232]: $'dave@google.com'$

regex.match trả về Không, vì nó chỉ khớp nếu mẫu xuất hiện ở đầu chuỗi:

In [233]: print regex.match(text)
None

Liên quan, sub sẽ trả về một chuỗi mới với các lần xuất hiện của mẫu được thay thế bằng chuỗi mới:

In [234]: print regex.sub('REDACTED', text)
Dave REDACTED
Steve
Rob REDACTED
Ryan REDACTED

Giả sử bạn muốn tìm địa chỉ email và đồng thời phân chia từng địa chỉ thành 3 thành phần: tên người dùng, tên miền và hậu tố miền. Để thực hiện việc này, hãy đặt dấu ngoặc đơn xung quanh các phần của mẫu để phân đoạn:

In [235]: $pattern = r'([A-Z0-9.\%+-]+)@([A-Z0-9.-]+)\.([A-Z]2,4)'$
In [236]: regex = re.compile(pattern,flags=re.IGNORECASE)

Một đối tượng khớp được tạo bởi biểu thức chính quy đã sửa đổi này trả về một bộ gồm các thành phần mẫu với phương thức nhóm của nó:

In [237]: m = regex.match('wesm@bright.net')
In [238]: m.groups()
Out[238]: ('wesm', 'bright', 'net')

findall trả về một danh sách các bộ khi mẫu có các nhóm:

In [239]: regex.findall(text)
Out[239]:
[('dave', 'google', 'com'),
('steve', 'gmail', 'com'),
('rob', 'gmail', 'com'),
('ryan', 'yahoo', 'com')]

sub cũng có quyền truy cập vào các nhóm trong mỗi trận đấu bằng các ký hiệu đặc biệt như \ 1, \ 2, v.v.:

In [240]: print regex.sub(r'Username: \1, Domain: \2, Suffix: \3', text)
Dave Username: dave, Domain: google, Suffix: com
Steve Username: steve, Domain: gmail, Suffix: com
Rob Username: rob, Domain: gmail, Suffix: com
Ryan Username: ryan, Domain: yahoo, Suffix: com

Còn nhiều điều nữa về các biểu thức chính quy trong Python, hầu hết trong số đó nằm ngoài phạm vi của cuốn sách. Để cung cấp cho bạn một hương vị, một biến thể trên biểu thức chính quy email ở trên đặt tên cho các nhóm phù hợp:

$regex = re.compile(r"""(?P<username>[A-Z0-9.\%+-]+)@(?P<domain>[A-Z0-9.-]+)\.(?P<suffix>[A-Z]2,4)""", flags = re.IGNORECASE|re.VERBOSE)$

Đối tượng khớp được tạo bởi một biểu thức chính quy như vậy có thể tạo ra một lệnh tiện dụng với các tên nhóm được chỉ định:

In [242]: m = regex.match('wesm@bright.net')
In [243]: m.groupdict()
Out[243]: 'domain': 'bright', 'suffix': 'net', 'username': 'wesm'

Argument	Description
findall, finditer	Trả về tất cả các mẫu đối sánh không chồng chéo trong một chuỗi. findall trả về một danh sách tất cả các mẫu trong khi finditer trả về từng mẫu một từ một trình vòng lặp.
match	Khớp ở đầu chuỗi và tùy chọn phân đoạn các thành phần mẫu thành các nhóm. Nếu mẫu phù hợp, trả về một đối tượng phù hợp, nếu không thì Không.
search	Quét chuỗi để khớp với mẫu; trả lại một đối tượng phù hợp nếu vậy. Không giống như đối sánh, đối sánh có thể ở bất kỳ đâu trong chuỗi thay vì chỉ ở đầu.
split	Chia chuỗi thành từng mảnh ở mỗi lần xuất hiện mẫu.
sub, subn	Replace all (sub) or first n occurrences (subn) of pattern in string with replacement expression. Use symbols \1, \2, ... to refer to match group elements in the replacement string.

Bảng 4.2: Phương thức biểu thức chính quy

4.3 Vector hóa chuỗi trong pandas

Việc dọn dẹp một tập hợp dữ liệu lộn xộn để phân tích thường đòi hỏi rất nhiều thao tác trộn chuỗi và chuẩn hóa. Để làm phức tạp thêm vấn đề, một cột chứa các chuỗi đôi khi sẽ bị thiếu dữ liệu:

```
In [244]: data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
     .....:               'Rob': 'rob@gmail.com', 'Wes': np.nannan}
In [245]:  data = Series(data)
In [246]:  data                          In [247]:  data.isnull()
Out[246]:                                Out[247]:
  Dave    dave@google.com                  Dave    False
  Rob       rob@gmail.com                  Rob     False
  Steve   steve@gmail.com                  Steve   False
  Wes                   NaN                Wes      True
```

Các phương thức chuỗi và biểu thức chính quy có thể được áp dụng (chuyển lambda hoặc hàm khác) cho từng giá trị bằng cách sử dụng data.map, nhưng nó sẽ không thành công trên NA. Để giải quyết vấn đề này, Sê-ri có các phương thức ngắn gọn cho các thao tác chuỗi bỏ qua các giá trị NA. Chúng được truy cập thông qua thuộc tính str của Series; ví dụ: chúng tôi có thể kiểm tra xem từng địa chỉ email có 'gmail' trong đó với str.contains:

```
In [248]:  data.str.contains('gmail')
Out[248]:
Dave False
Rob True
Steve True
Wes NaNNaN
```

Cũng có thể sử dụng các biểu thức chính quy, cùng với bất k ỳ tùy chọn lại nào như IGNORECASE:

```
In [249]:  pattern
Out[249]:  '([A-Z0-9.%+-]+)@([A-Z0-9.-]+).([A-Z]{2,4})
In [250]:  data.str.findall(pattern, flags=re.IGNORECASE)
```

```
Out[250]:
Dave[('dave', 'google', 'com')]
Rob[('rob', 'gmail', 'com')]
Steve[('steve', 'gmail', 'com')]
Wes NaN
```

Có một số cách để thực hiện truy xuất phần tử được vector hóa. Sử dụng str.get hoặc lập chỉ mục vào thuộc tính str:

```
In [251]: matches = data.str.match(pattern, flags=re.IGNORECASE)
In [252]: matches
Out[252]:
Dave ('dave', 'google', 'com')
Rob ('rob', 'gmail', 'com')
Steve ('steve', 'gmail', 'com')
Wes
```

Method	**Description**
cat	Nối các chuỗi theo từng phần tử với dấu phân cách tùy chọn
contains	Trả về mảng boolean nếu mỗi chuỗi chứa pattern/regex
count	Đếm số lần xuất hiện của mẫu
endswith, startswith	Tương đương với x.endswith(pattern) hoặc x.startswith(pattern) cho mỗi phần tử.
findall	Tính toán danh sách tất cả các lần xuất hiện của mẫu/regex cho mỗi chuỗi
get	Lập chỉ mục vào từng phần tử (truy xuất phần tử thứ i)
join	Tham gia các chuỗi trong từng phần tử của Sê-ri với dấu phân cách đã qua
len	Tính độ dài của mỗi chuỗi
lower, upper	Chuyển đổi trường hợp; tương đương với x.lower() hoặc x.upper() cho mỗi phần tử.
match	Sử dụng re.match với biểu thức chính quy đã truyền trên mỗi phần tử, trả về các nhóm phù hợp dưới dạng danh sách.
pad	Thêm khoảng trắng vào bên trái, bên phải hoặc cả hai bên của chuỗi
center	Tương đương với pad(side=' both')
repeat	Nhân đôi giá trị; ví dụ s.str.repeat(3) tương đương với x * 3 cho mỗi chuỗi.
replace	Thay thế các lần xuất hiện của mẫu/regex bằng một số chuỗi khác
slice	Cắt từng chuỗi trong Sê-ri.
split	Tách chuỗi trên dấu phân cách hoặc biểu thức chính quy
strip, rstrip, lstrip	Cắt khoảng trắng, bao gồm cả dòng mới; tương đương với x.strip() (và rstrip,lstrip, tương ứng) cho mỗi phần tử.

Bảng 4.3: Các phương thức chuỗi được vector hóa

Chương 5

Example: USDA Food Database

Bộ Nông Nghiệp Hoa Kỳ cung cấp cơ sở dữ liệu về thông tin dinh dưỡng thực phẩm. Ashley Williams, một hacker người Anh, đã cung cấp một phiên bản của cơ sở dữ liệu này ở định dạng JSON (http://ashleyw.co.uk/project/food-nutrient-database).


```
{
  "id": 21441,
  "description": "KENTUCKY FRIED CHICKEN, Fried Chicken, EXTRA CRISPY, Wing, meat and skin with breading",
  "tags": [ "KFC"],
  "manufacturer": "Kentucky Fried Chicken",
  "group": "Fast Foods",
  "portions": [
    {
      "amount": 1,
      "unit": "wing, with skin",
      "grams": 68.0++
    },
    ...
  ],
  "nutrients": [

      "value": 20.8,
      "units": "g",
      "description": "Protein",
      "group": "Composition"
    },
  ...
  ]
```


Mỗi thực phẩm có một số thuộc tính xác định,hai danh sách các chất dinh dưỡng và kích thước bộ phận. Dạng dữ liệu này chưa phù hợp để phân tích, ta cần sắp xếp lại dữ liệu một cách tốt hơn

Tải về và giải nén dữ liệu từ link trên, sử dụng module Json Python tích hợp

```
In [256]: import json
In [257]: db = json.load(open('ch07/foods-2011-10-03.json'))
In [258]: len(db)
Out[258]: 6636
```

Mỗi mục trong db là một lệnh chứa tất cả dữ liệu cho một loại thực phẩm. Lĩnh vực 'nutrients' là một danh sách các lệnh, một lệnh cho mỗi nutrient:

```
db[0].keys()
Out[259:
[u'portions', u'description',
u'tags', u'tags', u'nutrients', u'group',
 u'id', u'manufaturer']
In [261]: nutrients = DataFrame(db[0]['nutrients'])
In [262]: nutrients[:7]
Out[262]:
```

	description	group	units	value
1	Protein	Composition	g	25.18
2	Total lipid(fat)	Composition	g	29.20
3	Carbohydrate, by difference	Composition	g	3.06
4	Ash	Other	g	3.28
5	Energy	Energy	kcal	376.00
6	Water	Composition	g	39.28
7	Energy	Energy	kJ	1573.00

Khi thành dataframe, ta sẽ trích xuất đích dành các trường dữ liệu cần lấy

```
In[263]: info_keys = ['description','group','id','manufacturer']
In[264]: info= DataFrame(db,columns=info_keys)
In[265]: info[:5]
Out[265]:
```

	description	group	id	manufacturer
0	Cheese, caraway	Dairy and Egg Products	1008	
1	Cheese, cheddar	Dairy and Egg Products	1009	
2	Cheese, edam	Dairy and Egg Products	1018	
3	Cheese, feta	Dairy and Egg Products	1019	
4	Cheese, mozzarella, part skim milk	Dairy and Egg Products	1028	

Ta có thể thấy sự phân bố của các nhóm thực phẩm với value_counts:

```
In[267]: pd.value_counts(info.group)[:10]
Out[267]:
```

Vegetables and Vegetable Products	812
Beef Products	618
Baked Products	496
Breakfast Cereals	403
Legumes and Legume Products	365
Fast Foods	365
Lamb, Veal, and Game Products	345
Sweets	341
Pork Products	328
Fruits and Fruit Juices	328

Bây giờ ta sẽ phân tích tất cả dữ liệu về dinh dưỡng, bằng cách tập hợp các chất dinh dưỡng cho mỗi thực phẩm.Đầu tiên chuyển đổi danh sách chất dinh dưỡng thành DataFrame,nối thêm ID, sử dụng Concat:

```
nutrients = []
for rec in db:
  fnuts = DataFrame(rec['nutrients'])
```

```
    fnuts['id'] = rec['id']
    nutrients.append(fnuts)
  nutrients = pd.concat(nutrients, ignore_index=True)
```

Khi sắp xếp thành công, bảng các chất dinh dưỡng::

```
In [269]: nutrients
Out[269]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 389355 entries, 0 to 389354
Data columns:
 description                             389355  non-null  values
 group                                   389355  non-null  values
 units                                   389355  non-null  values
 value                                   389355  non-null  values
 id                                      389355  non-null  values
 dtypes: float64(1), int64(1), object(3)
```

Ta có thể mô tả 2 thông tin 'group' và 'description' một cách rõ ràng hơn

```
In[272]: col_mapping = {'description' : 'food',: 'group' : 'fgroup'}
   .....
In [273]: info = info.rename(columns=col_mapping, copy=False)
In [274]: info
Out[274]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6636 entries, 0 to 6635
Data columns:
 food                    6636  non-null  values
 fgroup                  6636  non-null  values
 id                      6636  non-null  values
 manufacturer            5195  non-null  values
 dtypes: int64(1), object(3)
```

Sau các bước trên,ta hợp nhất thông tin với các chất dinh dưỡng

```
In [278]: ndata = pd.merge(nutrients, info, on='id', how='outer')
In [279]: ndata
Out[279]:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 375176 entries, 0 to 375175
Data columns:
 nutrient                                375176  non-null  values
 nutgroup                                375176  non-null  values
 units                                   375176  non-null  values
 value                                   375176  non-null  values
 id                                      375176  non-null  values
 food                                    375176  non-null  values
 fgroup                                  375176  non-null  values
 manufacturer                            293054  non-null  values
 dtypes: float64(1), int64(1), object(6)
In [280]: ndata.ix[30000]
Out[280]:
```

nutrient	Folic acid
nutgroup	Vitamins
units	mcg
value	0
id	5658
food	Ostrich, top loin, cooked
fgroup	Poultry Products
manufacturer	
Name: 30000	

Cuối cùng, ta ẽ biểu đồ giá trị trung bình theo Nhóm thực phẩm và loại dinh dưỡng, ở hình Figure 7-1

```
In[281]: result = ndata.groupby(['nutrient', 'fgroup'])['value'].quantile(0.5)
In[282]: result['Zinc,Zn'].order().plot(kind='barh')
```

Ta có thể tìm thực phẩm đậm đặc chất dinh dưỡng nhất:

```
by_nutrient = ndata.groupby(['nutgroup', 'nutrient'])
get_maximum =lambda x:x.xs(x.value.idxmax())
get_minimum = lambda x: x.xs(x.value.idxmin())
max_foods = by_nutrient.apply(get_maximum)[['value', 'food']]
max_foods.food = max_foods.food.str[:50]
```

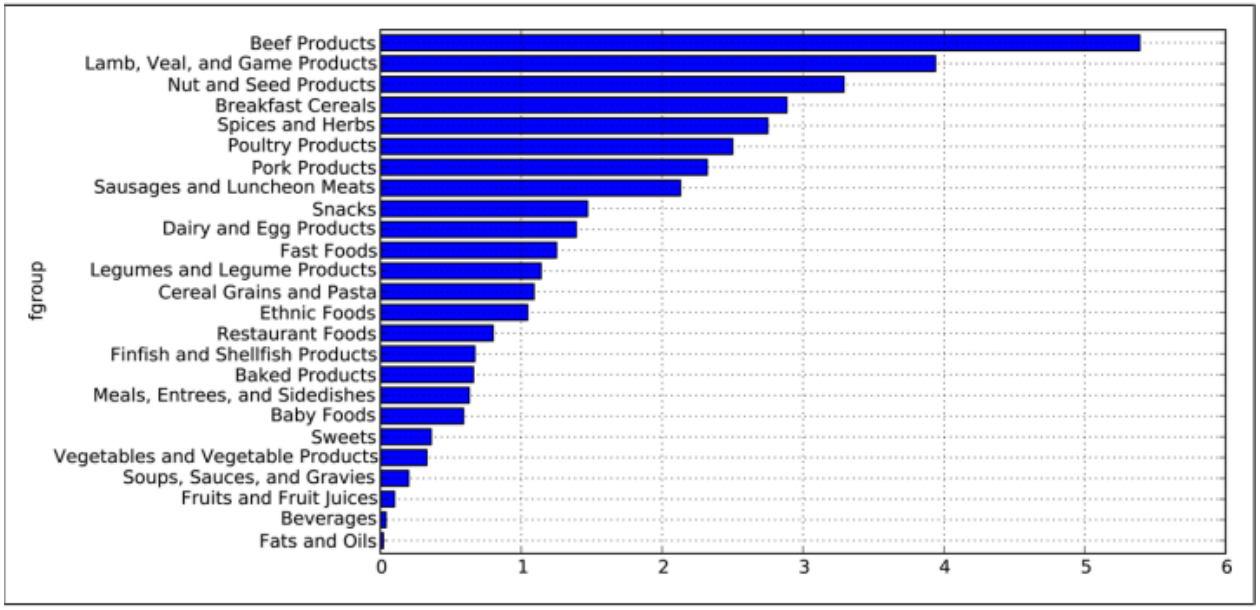

Figure 7-1. Median Zinc values by nutrient group

Hiển thị chất dinh dưỡng Animo Acid's ở DataFrame

```
In[284]: max_foods.ix['Amino Acids']['food']
Out[284]:
```

nutrient	
Alanine	Gelatins, dry powder, unsweetened
Arginine	Seeds, sesame flour, low-fat
Aspartic	acid Soy protein isolate
Cystine	Seeds, cottonseed flour, low fat (glandless)
Glutamic acid	Soy protein isolate
Glycine	Gelatins, dry powder, unsweetened
Histidine	Whale, beluga, meat, dried (Alaska Native)
Hydroxyproline	KENTUCKY FRIED CHICKEN, Fried Chicken, ORIGINAL R
Isoleucine	Soy protein isolate, PROTEIN TECHNOLOGIES INTERNA
Leucine	Soy protein isolate, PROTEIN TECHNOLOGIES INTERNA
Lysine	Seal, bearded (Oogruk), meat, dried (Alaska Nativ
Methionine	Fish, cod, Atlantic, dried and salted
Phenylalanine	Soy protein isolate, PROTEIN TECHNOLOGIES INTERNA
Proline	Gelatins, dry powder, unsweetened
Serine	Soy protein isolate, PROTEIN TECHNOLOGIES INTERNA
Threonine	Soy protein isolate, PROTEIN TECHNOLOGIES INTERNA
Tryptophan	Sea lion, Steller, meat with fat (Alaska Native)
Tyrosine	Soy protein isolate, PROTEIN TECHNOLOGIES INTERNA
Valine	Soy protein isolate, PROTEIN TECHNOLOGIES INTERNA
Name: food	

Tài liệu tham khảo

[1] Wes McKinney. Data wrangling: Clean, transform, merge, reshape. In *Python For Data Analysis*, pages 177–217, Oct. 2012.