16. Nhóm dữ liệu (phần 2)

Bài học bao gồm:

+ Chuyển đổi (Transformation)

+ Lọc (Filtration)

Chuyển đổi (Transformation)

Thay vì sử dụng agg() ngay sau groupby(), ta có thể sử dụng phương thức transform(). Phương thức transform() này thay đổi trực tiếp thực thể của dataframe theo một hàm cụ thể được định nghĩa trước mà không làm thay đổi index. Vì vậy phương thức transform() trả về một đối tượng được lập chỉ mục (indexed) cùng kích cỡ như một nhóm đang được nhóm lại. Phương thức transform phải:

Trả về kết quả có cùng kích thước với group chunk hoặc có thể broadcast với kích thước của nhóm (ví dụ: một số, grouped.transform(lambda

x:

x.iloc[-1])).

Vận hành kiểu column-by-column trong một nhóm. Việc chuyển đổi được áp dụng cho group chunk đầu tiên sử dụng chunk.apply.

Không thực hiện các hoạt động tại chỗ trong group chunk. Các phần của nhóm nên được coi là không thay đổi, và các thay đổi đối với một group chunk có thể tạo ra kết quả không mong muốn. Ví dụ: khi sử dụng fillna, inplace = False (grouped.transform (lambda x: x.fillna (inplace = False))).

(Tùy chọn) hoạt động trên toàn bộ group chunk. Nếu được hỗ trợ, một đường dẫn nhanh được sử dụng bắt đầu từ đoạn thứ hai.

Chú ý đầu vào dữ liệu của transform là kiểu Series.

Ta cùng phân tích ví dụ sau: Nhóm dữ liệu theo “year” và chuẩn hóa dữ liệu trong năm tương ứng.

Tạo dữ liệu:

```
>>> import pandas as pd
>>> import numpy as np
>>> np.random.seed(1234)
>>> index = pd.date_range('10/1/1999', periods=1100)
>>> ts = pd.Series(np.random.normal(0.5, 2, 1100), index)
>>> ts = ts.rolling(window=100,min_periods=100).mean().dropna()
```

Kiểm tra dữ liệu. Ta sẽ thấy index theo kiểu datetime64.

```
>>> ts.head()
2000-01-08    0.570225
2000-01-09    0.566620
2000-01-10    0.601770
2000-01-11    0.583188
2000-01-12    0.595147
Freq: D, dtype: float64
>>> ts.tail()
2002-09-30    0.099543
2002-10-01    0.087771
2002-10-02    0.076343
2002-10-03    0.098357
2002-10-04    0.105230
Freq: D, dtype: float64
>>> ts.index
DatetimeIndex(['2000-01-08', '2000-01-09', '2000-01-10', '2000-01-11',
               '2000-01-12', '2000-01-13', '2000-01-14', '2000-01-15',
               '2000-01-16', '2000-01-17',
               ...
               '2002-09-25', '2002-09-26', '2002-09-27', '2002-09-28',
               '2002-09-29', '2002-09-30', '2002-10-01', '2002-10-02',
```

```
               '2002-10-03', '2002-10-04'],
              dtype='datetime64[ns]', length=1001, freq='D')
>>>
```

Sử dụng groupby. Ta có thể group dữ liệu theo ‘year’, ‘month’, ‘day’. Theo bài toán ta cần nhóm dữ liệu theo ‘year.’

```
>>> key = lambda x: x.year
>>> year_grouped= ts.groupby(key)
>>>
```

Tiến hành chuẩn hóa dữ liệu. Đây chính là bước transformation mà tôi muốn đề cập trong bài viết. Chú ý dữ liệu đầu vào cho zscore là kiểu Series.

```
>>> zscore = lambda x: (x - x.mean()) / x.std()
>>> transformed = year_grouped.transform(zscore)
>>>
```

Cùng xem lại kết quả trước khi transforming.

```
>>> print "Original Data"
Original Data
>>> year_grouped.mean()
2000    0.616392
2001    0.495943
2002    0.372166
dtype: float64
>>> year_grouped.std()
2000    0.123997
2001    0.192198
2002    0.127509
dtype: float64
>>>
```

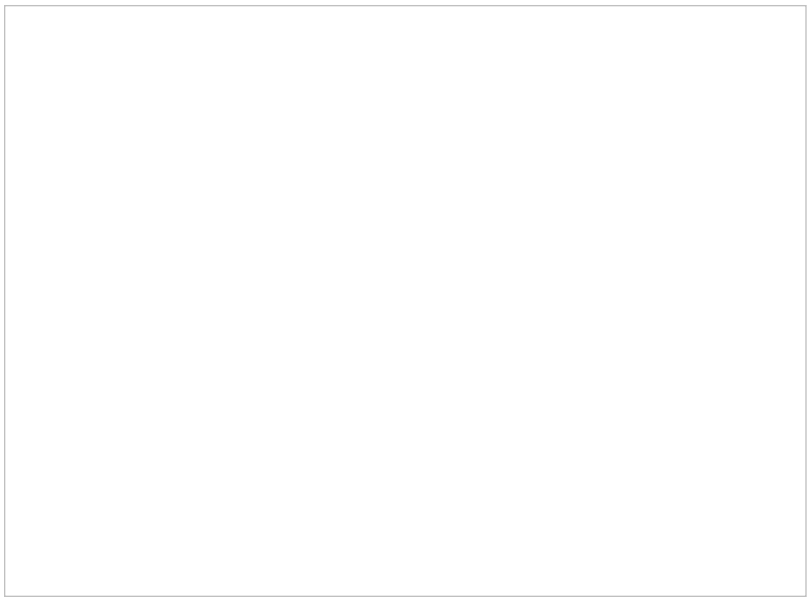
Kết quả sau khi transforming

```
>>> print ("Transformed Data")
Transformed Data
>>> grouped_trans = transformed.groupby(key)
>>> grouped_trans.mean()
2000    4.656752e-15
2001   -5.049233e-16
2002   -1.051706e-15
dtype: float64
>>> grouped_trans.std()
2000    1.0
2001    1.0
2002    1.0
dtype: float64
>>>
```

Bước cuối cùng, visualizing dữ liệu.

```
>>> compare = pd.DataFrame({'Original': ts, 'Transformed': transformed})
>>> compare.plot()
<matplotlib.axes._subplots.AxesSubplot object at 0x00000000075AFC50>
>>> import matplotlib.pyplot as plt
>>> plt.show()
```

```
>>>
```



Một số thao tác trên nhóm dữ liệu có thể không phù hợp với aggregate() hay transform(). Hoặc, bạn chỉ cần muốn GroupBy để làm thế nào kết hợp các kết quả theo một cách riêng. Đối với những điều này, sử dụng phương thức apply, có thể được thay thế cho cả “aggregate” và “tranfrom” mà trong nhiều trường hợp chúng có thể giải quyết. Tuy nhiên, apply() có thể xử lý một số trường hợp đặc biệt.

Có hai điểm khác biệt lớn giữa apply() và transform() là:

+ apply() dữ liệu đầu vào cho các custom function sử dụng trong apply() là tất cả các cột thuộc mỗi group (dạng dataframe). Trong khi transform() chỉ đưa vào một cột duy nhất thuộc mỗi group (dạng Series).

+ custom function sử dụng bởi apply() có thể trả về vô hướng (scalar), Series hoặc Dataframe. Trong dữ liệu trả về của custom function sử dụng bởi transform() bắt buộc phải là kiểu sequence (Series, list, array).

Vì vậy transform() chỉ làm việc với một Series ở một thời điểm trong khi apply() thì có thể làm việc với toàn bộ dataframe.

Sử dụng bộ dữ liệu sau:

```
>>> df = pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar','foo', 'bar', 'foo', 'foo'],'B' : ['one', 'one', 'two', 'three','two', 'two', 'one', 'three'],'C
>>> df
     A      B         C         D
0  foo    one -1.160609  0.852233
1  bar    one -0.542337  1.508110
2  foo    two -1.063359  1.074986
3  bar  three -0.236392 -0.078842
4  foo    two -0.651174 -0.239175
5  bar    two -0.310689  1.037463
6  foo    one  1.590951 -0.963769
7  foo  three -0.933333 -1.833742
>>>
```

Đầu vào của apply có thể cả cột C và cột D

```
>>> df.groupby('A').apply(lambda x: (x['C'] - x['D']))
A
bar  1   -2.050447
     3   -0.157550
     5   -1.348153
foo  0   -2.012842
     2   -2.138345
     4   -0.411999
     6    2.554720
```

```
     7    0.900410
dtype: float64
>>>
```

Ví dụ mở rộng hơn và phức tạp hơn:

```
>>> def custom_func(gr):
...     s = gr['C'].max() - gr['C'].min()
...     z = (gr['D'] - gr['D'].mean())/gr['D'].std()
...     return pd.DataFrame({'z(D)':z , 'Range in C':s})
...
>>> df.groupby('A').apply(custom_func)
   Range in C      z(D)
0    2.751560  0.878164
1    0.305945  0.841479
2    2.751560  1.060279
3    0.305945 -1.105528
4    2.751560 -0.014129
5    0.305945  0.264049
6    2.751560 -0.606529
7    2.751560 -1.317786
>>>
```

Lọc (Filteration)

Để minh tốt hơn cho phần hướng dẫn này tôi sẽ dùng bộ dữ liệu của https://gitlab.com/bambootran89/vimentor_data/blob/master/sales_14.csv như sau.

```
>>> sales = pd.read_csv("sales_14.csv",sep=',',index_col=0)
>>> sales
                            Company   Product  Units
Date
2/2/2015 8:30:00 AM              Hooli  Software      3
2/2/2015 9:00:00 PM          Mediacore  Hardware      9
2/3/2015 2:00:00 PM            Initech  Software     13
2/4/2015 3:30:00 PM          Streeplex  Software     13
2/4/2015 10:00:00 PM   Acme Coporation  Hardware     14
2/5/2015 2:00:00 AM    Acme Coporation  Software     19
2/5/2015 10:00:00 PM             Hooli   Service     10
2/7/2015 11:00:00 PM   Acme Coporation  Hardware      1
2/9/2015 9:00:00 AM          Streeplex   Service     19
2/9/2015 1:00:00 PM          Mediacore  Software      7
2/11/2015 8:00:00 PM           Initech  Software      7
2/11/2015 11:00:00 PM            Hooli  Software      4
2/16/2015 12:00:00 PM            Hooli  Software     10
2/19/2015 11:00:00 AM        Mediacore  Hardware     16
2/19/2015 4:00:00 PM         Mediacore   Service     10
2/21/2015 5:00:00 AM         Mediacore  Software      3
2/21/2015 8:30:00 PM             Hooli  Hardware      3
2/25/2015 12:30:00 AM          Initech   Service     10
2/26/2015 9:00:00 AM         Streeplex   Service      4
>>>
```

Yêu cầu: Sau khi thực hiện groupby theo ‘Company’ cần loại bỏ tất cả các hàng mà có tổng trên ‘Units’ nhỏ hơn hoặc bằng 35.

```
>>> sales.groupby('Company').filter(lambda g:g['Units'].sum() > 35)
                        Company   Product  Units
Date
2/2/2015 9:00:00 PM    Mediacore  Hardware      9
2/4/2015 3:30:00 PM    Streeplex  Software     13
2/9/2015 9:00:00 AM    Streeplex   Service     19
2/9/2015 1:00:00 PM    Mediacore  Software      7
2/19/2015 11:00:00 AM  Mediacore  Hardware     16
2/19/2015 4:00:00 PM   Mediacore   Service     10
2/21/2015 5:00:00 AM   Mediacore  Software      3
2/26/2015 9:00:00 AM   Streeplex   Service      4
>>>
```

Bạn đã thấy làm thế nào để nhóm theo một cột, hoặc nhiều cột. Đôi khi, bạn có thể muốn nhóm theo một chức năng/chuyển đổi của một cột. Chìa khóa ở đây là Series được lập chỉ mục giống như DataFrame. Bạn cũng có thể kết hợp phân nhóm cột theo nhóm Series.

Ví dụ: công việc của bạn là điều tra tỷ lệ sống sót của hành khách trên tàu Titanic theo ‘age’ và 'pclass'. Đặc biệt, mục đích là để tìm ra tỉ lệ của trẻ em dưới 10 sống sót trong mỗi 'pclass'. Bạn sẽ làm điều này bằng cách tạo ra một mảng boolean với True là hành khách dưới 10 tuổi và False là hành khách trên 10. Bạn sẽ sử dụng map() để thay đổi các giá trị này thành chuỗi. Cuối cùng, bạn sẽ nhóm theo nhóm dưới 10 và cột 'pclass' và tổng hợp cột 'survived'. Cột 'survived' có giá trị 1 nếu hành khách sống sót và 0 nếu không. Ý nghĩa của cột 'survived' là tỉ lệ của hành khách sống sót.

```
>>> titanic = pd.read_csv("https://raw.githubusercontent.com/vincentarelbundock/Rdatasets/master/csv/datasets/Titanic.csv",sep=',',index_col=0)
>>> under10 = (titanic['Age'] < 10).map({True:'under 10', False:'over 10'})
>>> survived_mean = titanic.groupby([under10,'PClass'])['Survived'].mean()
>>> print(survived_mean)
Age       PClass
over 10   *         0.000000
          1st       0.597484
          2nd       0.382239
          3rd       0.181818
under 10  1st       0.750000
          2nd       1.000000
          3rd       0.482759
Name: Survived, dtype: float64
>>>
```

Kết luận

Sau hai bài học chúng ta đã học được cách nhóm dữ liệu hiệu quả trong pandas và nó sẽ là một sự bổ sung mạnh mẽ vào các kĩ năng phân tích dữ liệu của bạn.