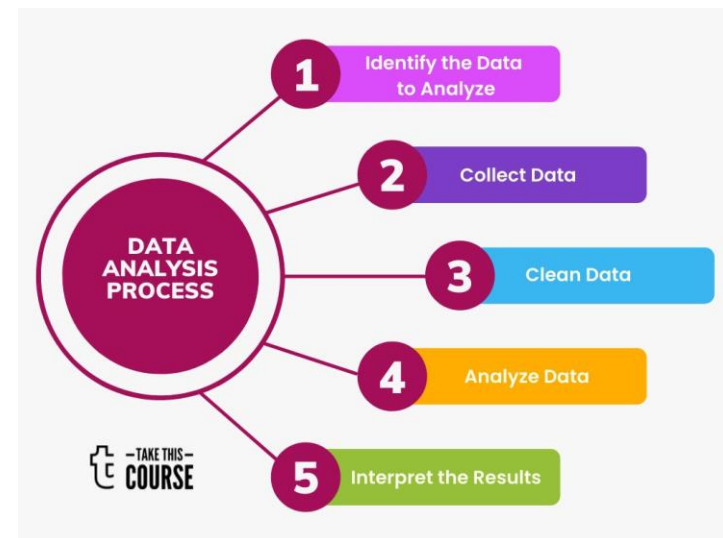
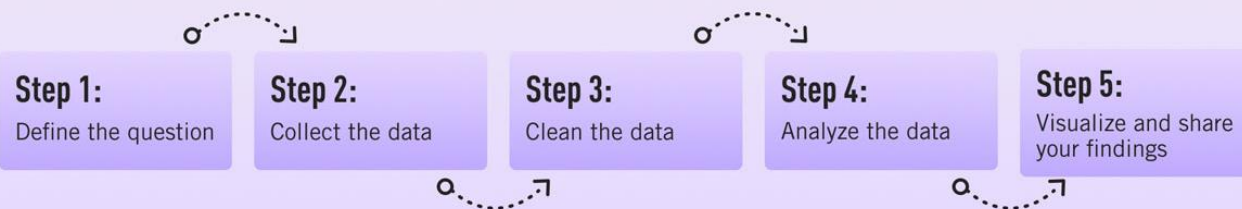


PHÂN TÍCH DỮ LIỆU

(Data Analysis)



THE DATA ANALYSIS PROCESS



Lê Văn Hạnh

levanhhanhvn@gmail.com

NỘI DUNG MÔN HỌC

PHẦN 1 TỔNG QUAN & THU THẬP DỮ LIỆU CHO VIỆC PHÂN TÍCH

1. Khoa học dữ liệu
2. Thu thập dữ liệu
3. Tìm hiểu dữ liệu

PHẦN 2: TIỀN XỬ LÝ DỮ LIỆU (*Data Preprocessing*)

4. Nhiệm vụ chính trong tiền xử lý dữ liệu
5. PANDAS
6. Thao tác với các định dạng khác nhau của tập tin dữ liệu
7. Làm sạch và Chuẩn bị dữ liệu
8. Sắp xếp dữ liệu: nối, kết hợp và định hình lại
9. Tổng hợp dữ liệu và các tác vụ trên nhóm

PHẦN 3 TRỰC QUAN HÓA DỮ LIỆU (*Data Visualization*)

10. Đồ thị và Biểu đồ
11. Vẽ đồ thị và Trực quan hóa

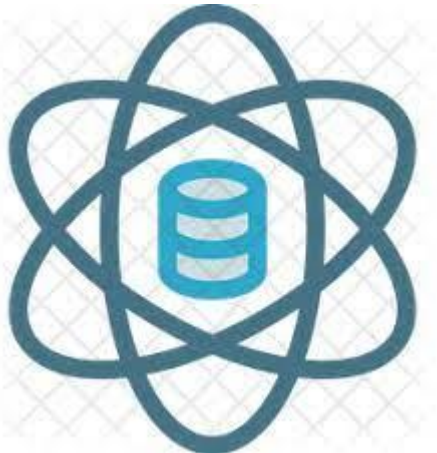
PHẦN 2

TIỀN XỬ LÝ DỮ LIỆU (*Data Preprocessing*)

Chương 9

TỔNG HỢP DỮ LIỆU & CÁC TÁC VỤ TRÊN NHÓM

(*Data Aggregation & Group Operations*)



Lê Văn Hạnh
levanhhanhvn@gmail.com

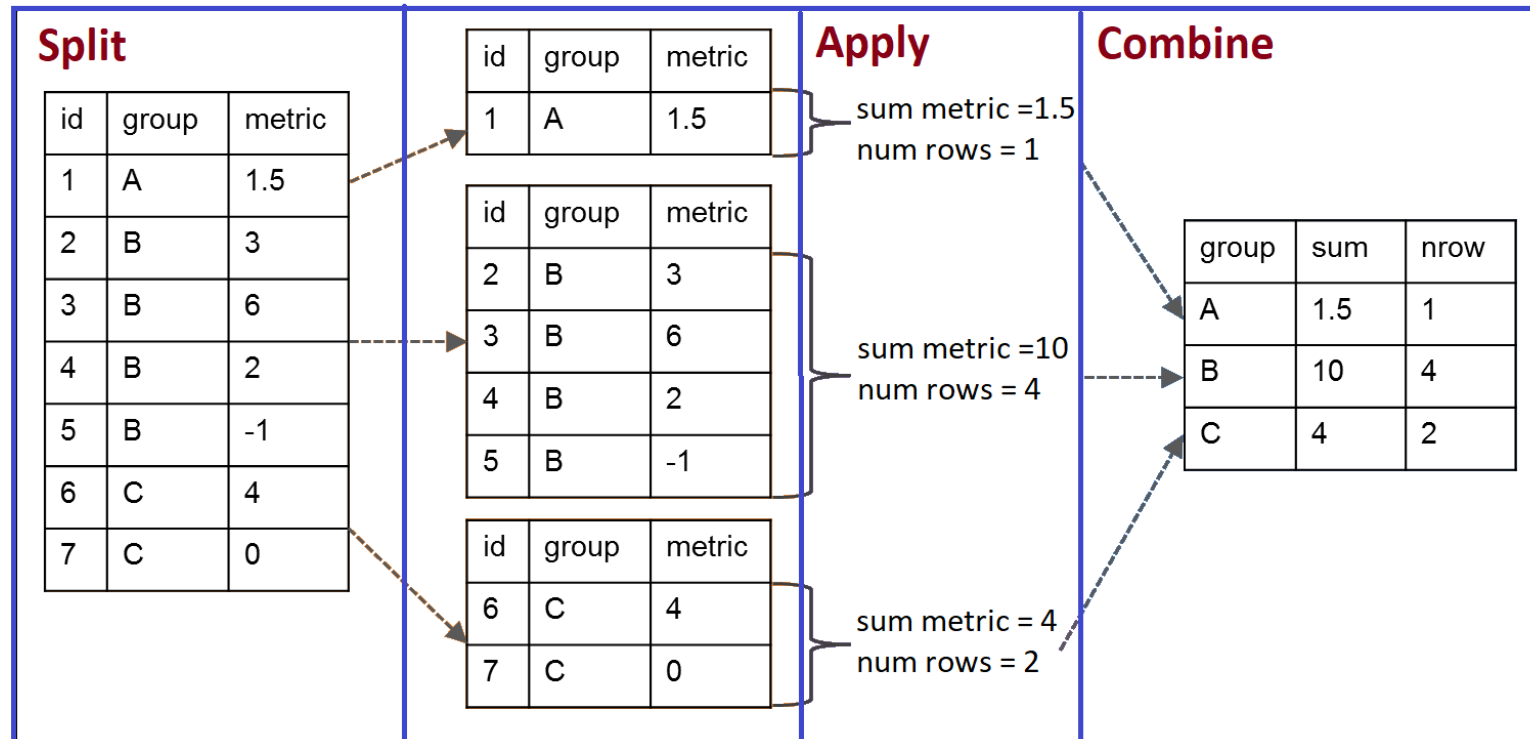
NỘI DUNG CHƯƠNG 9

1. Phân nhóm dữ liệu
2. Tổng hợp dữ liệu
3. Phương thức apply về split-apply-combine
4. Pivot Tables and Cross-Tabulation

1. PHÂN NHÓM DỮ LIỆU (*GroupBy Mechanics*)

Để tính toán trên dữ liệu chứa trong đối tượng *pandas*, cho dù là *Series*, *DataFrame* hay loại khác, Pandas cần thực hiện 3 bước:

- **Split**: dữ liệu được chia thành các nhóm dựa trên một hoặc nhiều khóa (dựa trên các hàng - `axis=0` - hoặc các cột - `axis=1`) mà ta cung cấp.
- **Apply**: thực hiện một hàm tính toán trên từng nhóm.
- **Combine**: tập hợp kết quả của tất cả các nhóm thành một đối tượng kết quả.



1. Phân nhóm dữ liệu (*GroupBy Mechanics*)

- Mỗi khóa của nhóm có thể thuộc nhiều dạng và các khóa không nhất thiết phải cùng loại:
 - List hoặc array các giá trị có cùng độ dài với trục được nhóm
 - Giá trị biểu thị tên cột trong *DataFrame*
 - Một *dict* hoặc *Series* đưa ra sự tương ứng giữa các giá trị trên trục được nhóm và tên nhóm
 - Một hàm được gọi trên chỉ mục trục (*axis index*) hoặc các nhãn riêng lẻ trong chỉ mục

Lưu ý rằng khóa của ba trường hợp sau là các lối tắt (*shortcuts*) để tạo ra một mảng giá trị dùng để phân chia đối tượng.

1. Phân nhóm dữ liệu (*GroupBy Mechanics*)

- Ví dụ: tính giá trị trung bình của cột `data1` bằng cách sử dụng các nhãn từ `key1`

```
In [1]: df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
.....: 'key2' : ['one', 'two', 'one', 'two', 'one'],
.....: 'data1' : np.random.randn(5),
.....: 'data2' : np.random.randn(5)})
```

```
In [2]: df
```

```
Out[2]:
```

	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

```
In [3]: grouped = df['data1'].groupby(df['key1'])
```

```
In [4]: grouped
```

```
Out[4]: <pandas.core.groupby.SeriesGroupBy object at 0x7faa31537390>
```

- Biến ***grouped*** hiện là một đối tượng *GroupBy*. Nó chưa thực sự tính toán bất cứ điều gì ngoại trừ một số dữ liệu trung gian về khóa của nhóm `df['key1']`. Ý tưởng là đối tượng này có tất cả thông tin cần thiết để áp dụng một số tác vụ cho từng nhóm. Ví dụ: để tính trung bình của nhóm có nghĩa là cần gọi phương thức ***mean*** của biến `grouped`:

```
In [5]: grouped.mean()
```

```
Out[5]: key1
a      0.746672
b     -0.537585|
Name: data1, dtype: float64
```

1. Phân nhóm dữ liệu (GroupBy Mechanics)

- Dữ liệu (Series) đã được tạo ra dựa trên việc tổng hợp theo khóa của nhóm:

```
In [5]: grouped.mean()
Out[5]:
key1
a      0.746672
b     -0.537585
Name: data1, dtype: float64
```

kết quả group by trên 1 field

```
In [6]: means = df['data1'].groupby([df['key1'],
                                     df['key2']]).mean()
```

```
In [7]: means
```

```
Out[7]:
key1  key2
a      one    0.880536
      two    0.478943
b      one   -0.519439
      two   -0.555730
Name: data1, dtype: float64
```

kết quả group by trên 2 fields

```
df
  key1  key2  data1  data2
0    a   one -0.204708  1.393406
1    a   two  0.478943  0.092908
2    b   one -0.519439  0.281746
3    b   two -0.555730  0.769023
4    a   one  1.965781  1.246435
```

```
In [8]: means.unstack()
```

```
Out[8]:
key2      one      two
key1
a      0.880536  0.478943
b     -0.519439 -0.555730
```

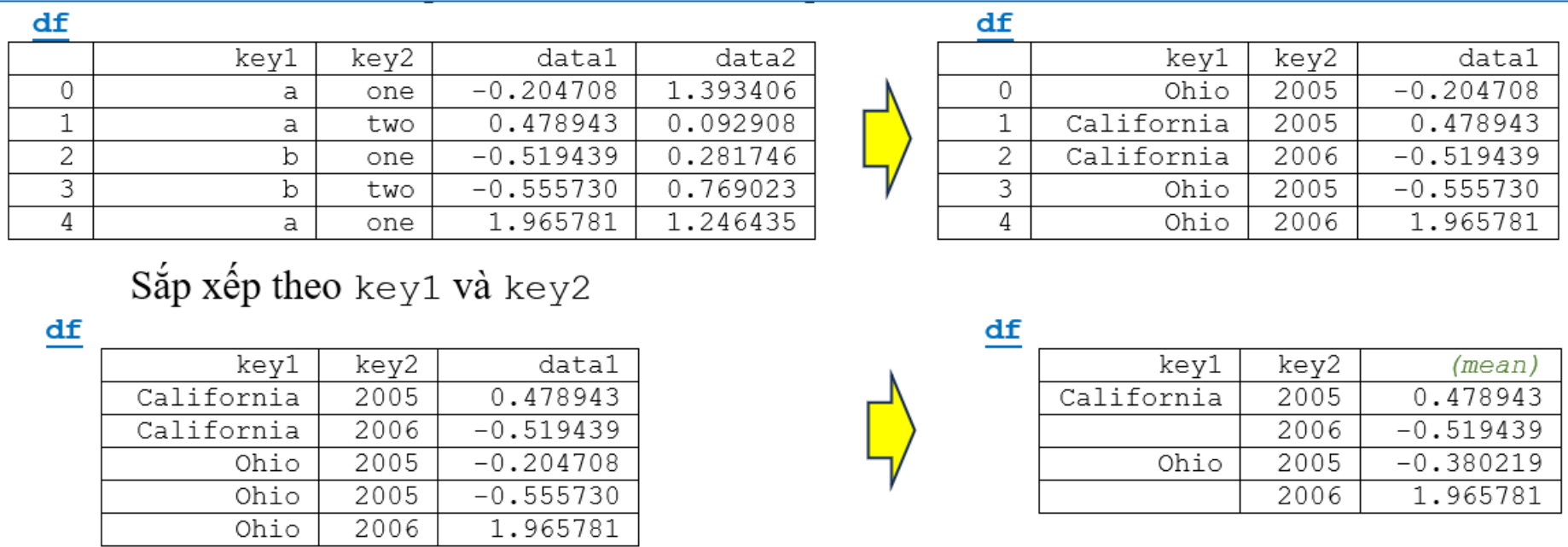

1. Phân nhóm dữ liệu (GroupBy Mechanics)

- Khóa nhóm có thể là bất kỳ mảng nào có độ dài phù hợp. Trong minh họa sau, khóa nhóm là Series:

	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

```
In [9]: states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
In [10]: years = np.array([2005, 2005, 2006, 2005, 2006])
In [11]: df['data1'].groupby([states, years]).mean()
Out[11]:
California      2005      0.478943
                2006     -0.519439
Ohio            2005     -0.380219
                2006      1.965781
Name: data1, dtype: float64
```

Có thể minh họa quá trình thực hiện kết quả trên như hình bên:



1. Phân nhóm dữ liệu (*GroupBy Mechanics*)

- Khóa nhóm có thể là chuỗi, số hoặc các đối tượng *Python* khác. Theo mặc định, tất cả các cột số được tổng hợp.

df	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

- Phương thức ***size***: trả về một *Series* chứa kích thước các nhóm (tương tự hàm ***count*** trong SQL).

```
In [12]: df.groupby('key1').mean()
```

```
Out[12]:
```

	data1	data2
key1		
a	0.746672	0.910916
b	-0.537585	0.525384

```
In [13]: df.groupby(['key1', 'key2']).mean()
```

```
Out[13]:
```

		data1	data2
key1	key2		
a	one	0.880536	1.319920
	two	0.478943	0.092908
b	one	-0.519439	0.281746
	two	-0.555730	0.769023

```
In [14]: df.groupby(['key1', 'key2']).size()
```

```
Out[14]:
```

	key1	key2
a	one	2
	two	1
b	one	1
	two	1

dtype: int64

1. Phân nhóm dữ liệu (GroupBy Mechanics)

1.1. Lặp trên các nhóm

- Đối tượng *GroupBy* hỗ trợ phép lặp, tạo ra một chuỗi gồm 2 bộ chứa tên nhóm cùng với dữ liệu của nhóm.

df	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

```
# khóa chỉ gồm 1 giá trị
In [15]: for name, group in df.groupby('key1'):
        ....:     print(name)
        ....:     print(group)
a
      key1  key2  data1  data2
0      a   one -0.204708 1.393406
1      a   two  0.478943 0.092908
4      a   one  1.965781 1.246435
b
      key1  key2  data1  data2
2      b   one -0.519439 0.281746
3      b   two -0.555730 0.769023
```

```
# khóa gồm nhiều giá trị
In [16]: for (k1, k2), group in df.groupby(['key1', 'key2']):
        ....:     print((k1, k2))
        ....:     print(group)
('a', 'one')
      key1  key2  data1  data2
0      a   one -0.204708 1.393406
4      a   one  1.965781 1.246435
('a', 'two')
      key1  key2  data1  data2
1      a   two  0.478943 0.092908
('b', 'one')
      key1  key2  data1  data2
2      b   one -0.519439 0.281746
('b', 'two')
      key1  key2  data1  data2
3      b   two -0.55573 0.769023
```

1.1. Lặp trên các nhóm (Iterating Over Groups)

- Có thể chọn làm bất cứ điều gì với các phần dữ liệu. Một cách có thể hữu ích là tính toán một bản tóm tắt các phần dữ liệu dưới dạng một dòng (*one-liner*):

```
In [17]: pieces = dict(list(df.groupby('key1')))  
.....: pieces  
{'a': key1 key2  data1  data2  
0  a one -0.903567  1.551300  
1  a two -0.029337  1.498718  
4  a one -0.925419 -0.092673,  
'b': key1 key2  data1  data2  
2  b one  0.140246  0.113323  
3  b two  1.443575 -1.263956}
```

```
In [18]: pieces['b']  
Out[18]:
```

	data1	data2	key1	key2
2	-0.519439	0.281746	b	one
3	-0.555730	0.769023	b	two

1. Phân nhóm dữ liệu (*GroupBy Mechanics*)

1.1. Lặp trên các nhóm (*Iterating Over Groups*)

- Theo mặc định, nhóm theo nhóm trên trục=0, nhưng có thể nhóm trên bất kỳ trục nào khác.
- Ví dụ:
 - Có thể nhóm các cột của **df** ở đây theo **dtype**:
 - Có thể in ra các nhóm:

```
In [19]: df.dtypes
```

```
Out[19]:
```

```
data1    float64
data2    float64
key1      object
key2      object
dtype: object
```

```
In [20]: grouped = df.groupby(df.dtypes, axis=1)
```

```
In [21]: for dtype, group in grouped:
```

```
.....:     print(dtype)
```

```
.....:     print(group)
```

float64

	data1	data2
0	-0.204708	1.393406
1	0.478943	0.092908
2	-0.519439	0.281746
3	-0.555730	0.769023
4	1.965781	1.246435

object

	key1	key2
0	a	one
1	a	two
2	b	one
3	b	two
4	a	one

1.2. Chọn một cột hoặc tập hợp con các cột (*Selecting a Column or Subset of Columns*)

- Việc lập chỉ mục một đối tượng *GroupBy* được tạo từ *DataFrame* với tên cột hoặc mảng các tên cột có tác dụng tập hợp lại cột để tổng hợp. Điều này có nghĩa rằng 2 cách sử dụng sau là tương đương nhau:

```
df.groupby('key1')['data1']      ≈ df['data1'].groupby(df['key1'])  
df.groupby('key1')[['data2']]   ≈ df[['data2']].groupby(df['key1'])
```

1. Phân nhóm dữ liệu (*GroupBy Mechanics*)

1.2. Chọn một cột hoặc tập hợp con các cột

- Đặc biệt đối với các tập dữ liệu lớn, có thể chỉ nên tổng hợp một vài cột.

Ví dụ: tính mean chỉ cho cột *data2* và nhận kết quả dưới dạng *DataFrame*

```
In [22]: df.groupby(['key1', 'key2'])[['data2']].mean()  
Out[22]:
```

	key1	key2	data2
	a	one	1.319920
		two	0.092908
	b	one	0.281746
		two	0.769023

- Đối tượng được trả về bởi thao tác lập chỉ mục này là một *DataFrame* được nhóm nếu đối tượng được truyền (list hoặc array hay *Series*) đã được nhóm với chỉ một tên cột duy nhất được truyền dưới dạng vô hướng (*scalar*)

```
In [23]: s_grouped = df.groupby(['key1', 'key2'])['data2']  
In [24]: s_grouped  
Out[24]: <pandas.core.groupby.SeriesGroupBy object at 0x7faa30c78da0>  
In [25]: s_grouped.mean()  
Out[25]:  
   key1  key2    data2  
a     one  1.319920  
      two  0.092908  
b     one  0.281746  
      two  0.769023  
Name: data2, dtype: float64
```

1.3. Phân nhóm với Dicts và Series (*Grouping with Dicts and Series*)

- Thông tin nhóm có thể tồn tại ở dạng khác ngoài mảng.

```
In [26]: people = pd.DataFrame(np.random.randn(5, 5),
.....:                        columns=['a', 'b', 'c', 'd', 'e'],
.....:                        index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
.....: people
```

	a	b	c	d	e
Joe	-0.168187	-0.463033	0.087715	1.445575	0.908136
Steve	-0.878599	0.754252	-1.332088	0.302420	0.404372
Wes	-1.644471	-0.881902	-1.674201	-0.798130	-0.067222
Jim	-0.525094	-0.730765	0.103663	-0.293637	0.284511
Travis	-0.462649	0.387363	-1.013089	-0.017257	-0.195096

```
In [27]: people.iloc[2:3, [1, 2]] = np.nan # Add a few NA values
In [28]: people
Out[28]:
```

	a	b	c	d	e
Joe	-0.168187	-0.463033	0.087715	1.445575	0.908136
Steve	-0.878599	0.754252	-1.332088	0.302420	0.404372
Wes	-1.644471	NaN	NaN	-0.798130	-0.067222
Jim	-0.525094	-0.730765	0.103663	-0.293637	0.284511
Travis	-0.462649	0.387363	-1.013089	-0.017257	-0.195096

1.3. Phân nhóm với Dicts và Series (*Grouping with Dicts and Series*)

Giả sử có sự tương ứng nhóm cho các cột và muốn tổng hợp các cột theo nhóm:

```
In [29]: mapping = {'a': 'red', 'b': 'red', 'c': 'blue',  
.....:             'd': 'blue', 'e': 'red', 'f': 'orange'}
```

Có thể tạo một mảng từ lệnh này để chuyển sang groupby, việc thêm item 'f': 'orange' (key= 'f') để nhấn mạnh rằng các khóa nhóm không sử dụng đều ổn

```
In [30]: by_column = people.groupby(mapping, axis=1)  
.....: by_column  
Out[30]: <pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000175E208DCD0>  
In [31]: by_column.sum()  
Out[31]:
```

	blue	red
Joe	1.533290	0.276916
Steve	-1.029668	0.280025
Wes	-0.798130	-1.711693
Jim	-0.189974	-0.971348
Travis	-1.030346	-0.270382

1.3. Phân nhóm với Dicts và Series (*Grouping with Dicts and Series*)

- Chức năng tương tự cũng áp dụng cho *Series*, có thể được xem dưới dạng ánh xạ có kích thước cố định:

```
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',  
           'd': 'blue', 'e': 'red', 'f': 'orange'}
```

```
In [32]: map_series = pd.Series(mapping)
```

```
In [33]: map_series
```

```
Out[33]:
```

```
    a      red  
    b      red  
    c     blue  
    d     blue  
    e      red  
    f  orange  
dtype: object
```

```
In [34]: people.groupby(map_series, axis=1).count()
```

```
Out[34]:
```

```
      blue  red  
Joe       2   3  
Steve     2   3  
Wes       1   2  
Jim       2   3  
Travis    2   3
```

1.4. Phân nhóm với các hàm (Grouping with Functions)

- Sử dụng các hàm *Python* là một cách tổng quát hơn để xác định ánh xạ nhóm so với *dict* hoặc *Series*. Bất kỳ hàm nào được truyền dưới dạng khóa nhóm (*group key*) sẽ được gọi một lần cho mỗi giá trị chỉ mục, với các giá trị trả về của hàm được sử dụng làm tên nhóm.

people					
	a	b	c	d	e
Joe	-0.168187	-0.463033	0.087715	1.445575	0.908136
Steve	-0.878599	0.754252	-1.332088	0.302420	0.404372
Wes	-1.644471	-0.881902	-1.674201	-0.798130	-0.067222
Jim	-0.525094	-0.730765	0.103663	-0.293637	0.284511
Travis	-0.462649	0.387363	-1.013089	-0.017257	-0.195096
In [35]: people.groupby(len).sum()					
Out[35]:					
	a	b	c	d	e
3	-2.337752	-0.193798	2.191378	0.353808	1.125425
5	-0.878599	0.754252	-1.332088	0.302420	0.404372
6	-0.462649	0.387363	-1.013089	-0.017257	-0.195096

1.4. Phân nhóm với các hàm (*Grouping with Functions*)

- Việc trộn các hàm với *arrays*, *Dicts* hoặc *Series* không phải là vấn đề vì mọi thứ đều được chuyển đổi thành *arrays* bên trong

```
In [36]: key_list = ['one', 'one', 'one', 'two', 'two']
In [37]: people.groupby([len, key_list]).min()
Out[37]:
```

		a	b	c	d	e
3	one	-1.644471	-0.463033	0.087715	-0.798130	-0.067222
	two	-0.525094	-0.730765	0.103663	-0.293637	0.284511
5	one	-0.878599	0.754252	-1.332088	0.302420	0.404372
6	two	-0.462649	0.387363	-1.013089	-0.017257	-0.195096

1.5. Phân nhóm theo các mức của chỉ mục (*Grouping by Index Levels*)

- Các tập dữ liệu cũng có thể được phân nhóm dựa trên một trong các cấp độ của chỉ mục trục (*levels of an axis index*)

```
In [38]: Columns = pd.MultiIndex.from_arrays([[ 'US', 'US', 'US', 'JP', 'JP'],
.....:                                     [1, 3, 5, 1, 3]],
.....:                                     names=[ 'cty', 'tenor'])
.....: Columns
Out[38]:
MultiIndex([ ('US', 1),
              ('US', 3),
              ('US', 5),
              ('JP', 1),
              ('JP', 3)],
            names=[ 'cty', 'tenor'])
In [39]: hier_df = pd.DataFrame(np.random.randn(4, 5), columns=Columns)
.....: hier_df
Out[39]:
```

	US			JP	
cty					
tenor	1	3	5	1	3
0	0.221647	0.171463	0.996575	-2.058284	-0.073196
1	-0.273768	0.736584	0.625347	-0.282391	-0.002228
2	-0.682168	0.574300	0.479721	-0.130622	0.179321
3	-1.483553	0.174865	0.493978	0.234410	-0.891909

1. Phân nhóm dữ liệu (GroupBy Mechanics)

1.5. Phân nhóm theo các mức của chỉ mục (Grouping by Index Levels)

- Các tập dữ liệu cũng có thể được phân nhóm dựa trên một trong các cấp độ của chỉ mục trực

hier_df					
cty	US			JP	
tenor	1	3	5	1	3
0	0.221647	0.171463	0.996575	-2.058284	-0.073196
1	-0.273768	0.736584	0.625347	-0.282391	-0.002228
2	-0.682168	0.574300	0.479721	-0.130622	0.179321
3	-1.483553	0.174865	0.493978	0.234410	-0.891909

```
In [40]: hier_df.iloc[2:3, [1, 2]] = np.nan
.... : hier_df.iloc[0:2, 3] = np.nan
.... : hier_df
```

cty	US			JP	
tenor	1	3	5	1	3
0	0.221647	0.171463	0.996575	NaN	-0.073196
1	-0.273768	0.736584	0.625347	NaN	-0.002228
2	-0.682168	NaN	NaN	-0.130622	0.179321
3	-1.483553	0.174865	0.493978	0.234410	-0.891909

Để nhóm theo cấp độ, hãy chuyển số cấp độ hoặc tên cấp độ bằng từ khóa cấp độ:

```
In [41]: hier_df.groupby(level='cty', axis=1).count()
Out[41]:
```

cty	JP	US
0	1	3
1	1	3
2	2	1
3	2	3

2. TỔNG HỢP DỮ LIỆU (*Data Aggregation*)

Tập hợp đề cập đến bất kỳ chuyển đổi dữ liệu nào tạo ra các giá trị vô hướng từ mảng (bao gồm giá trị trung bình –`mean`–, đếm –`count`–, giá trị nhỏ nhất –`min`– và tổng –`sum`–).

Những hàm tập hợp phổ biến như trong bảng sau:

Các hàm tập hợp đã được tối ưu hóa

<i>Function name</i>	<i>Description</i>
<code>count</code>	Đếm số lượng các giá trị khác null (not-NA) trong nhóm
<code>sum</code>	Tính tổng các giá trị khác null (not-NA) trong nhóm
<code>mean</code>	Tính trung bình các giá trị khác null (not-NA) trong nhóm
<code>median</code>	Tìm trung vị của các giá trị khác null (not-NA) trong nhóm
<code>std, var</code>	Tính độ lệch chuẩn và phương sai
<code>min, max</code>	Tìm giá trị tối thiểu và tối đa khác null (not-NA) trong nhóm
<code>prod</code>	Tính tích của các giá trị khác null (not-NA) trong nhóm
<code>first, last</code>	Tìm giá trị khác null (not-NA) đầu tiên và cuối cùng trong nhóm

2. Tổng hợp dữ liệu (Data Aggregation)

- Tuy nhiên, *Python* không bị giới hạn chỉ với những hàm này. Có thể sử dụng các tập hợp tùy ý và gọi thêm bất kỳ phương thức nào trên đối tượng được nhóm. Ví dụ: có thể tính toán lượng tử (*quantile*) của mẫu trên các cột trong *Series* hoặc *DataFrame*.
- Mặc dù *quantile* không được triển khai rõ ràng cho *GroupBy*, nhưng đây là một phương thức của *Series* và do đó có thể sử dụng. Trong nội bộ, *GroupBy* chia nhỏ *Series* một cách hiệu quả, gọi ***piece.quantile(0.9)*** cho mỗi phần và sau đó tập hợp các kết quả đó lại với nhau thành đối tượng kết quả.

<u>df</u>	key1	key2	data1	data2
0	a	one	-0.204708	1.393406
1	a	two	0.478943	0.092908
2	b	one	-0.519439	0.281746
3	b	two	-0.555730	0.769023
4	a	one	1.965781	1.246435

```
In [42]: grouped = df.groupby('key1')
In [43]: grouped['data1'].quantile(0.9)
Out[43]:
      key1
a      1.668413
b     -0.523068
Name: data1, dtype: float64
```


2. Tổng hợp dữ liệu (Data Aggregation)

- Để sử dụng các hàm tổng hợp của riêng người dùng, hãy chuyển tên hàm đó làm đối số cho 1 trong 2 phương thức **aggregate** hoặc **agg**. Lưu ý rằng *các hàm tổng hợp của người dùng thường chậm hơn nhiều so với các hàm được tối ưu hóa*

```
In [44]: def peak_to_peak(arr):
        ....:     return arr.max() - arr.min()
In [45]: grouped.agg(peak_to_peak)
Out[45]:
```

		data1	data2
	key1		
	a	2.170488	1.300498
	b	0.036291	0.487277

```
In [46]: grouped.describe()
Out[46]:
```

	data1							
	count	mean	std	min	25%	50%	75%	max
key1								
a	3.0	0.746672	1.109736	-0.204708	0.137118	0.478943	1.222362	1.965780
b	2.0	-0.537584	0.025662	-0.555730	-0.546657	-0.537584	-0.528512	-0.519439

	data2							
	count	mean	std	min	25%	50%	75%	max
key1								
a	0.910918	0.712218	0.092908	0.669674	1.246440	1.319923	1.393406	0.910918
b	0.525385	0.344557	0.281746	0.403565	0.525385	0.647204	0.769023	0.525385

2. Tổng hợp dữ liệu (Data Aggregation)

2.1. Tổng hợp bằng nhiều hàm khác nhau trên cùng một cột (Column-Wise and Multiple Function Application)

- Việc tổng hợp một *Series* hoặc tất cả các cột của *DataFrame* là vấn đề sử dụng *aggregate* với các hàm của người dùng hoặc gọi một phương thức như *mean* (giá trị trung bình) hoặc *std* (độ lệch chuẩn). Tuy nhiên, có thể thực hiện tổng hợp bằng một (hoặc nhiều) hàm khác tùy thuộc vào kiểu dữ liệu của cột.

```
tips #trong đó tips['tip_pct'] = tips['tip'] / tips['total_bill']
total_bill  tip  smoker  day  time  size  tip_pct
0      16.99   1.01     No  Sun  Dinner    2   0.059447
1      10.34   1.66     No  Sun  Dinner    3   0.160542
2      21.01   3.50     No  Sun  Dinner    3   0.166587
3      23.68   3.31     No  Sun  Dinner    2   0.139780
4      24.59   3.61     No  Sun  Dinner    4   0.146808
5      25.29   4.71     No  Sun  Dinner    4   0.186240
```

```
In [50]: grouped = tips.groupby(['day', 'smoker'])
In [51]: grouped_pct = grouped['tip_pct']
In [52]: grouped_pct.agg('mean')
Out[52]:
day  smoker
Fri      No    0.151650
        Yes    0.174783
Sat      No    0.158048
        Yes    0.147906
Sun      No    0.160113
        Yes    0.187250
Thur     No    0.160298
        Yes    0.163863
Name: tip_pct, dtype: float64
```

2. Tổng hợp dữ liệu (Data Aggregation)

2.1. Tổng hợp bằng nhiều hàm khác nhau trên cùng một cột (Column-Wise and Multiple Function Application)

Thay vào đó, nếu chuyển danh sách các hàm hoặc tên hàm, sẽ nhận được một DataFrame với các tên cột được lấy từ các hàm:

```
tips #trong đó tips['tip_pct'] = tips['tip'] / tips['total_bill']
total_bill  tip  smoker  day  time  size  tip_pct
0      16.99  1.01     No  Sun  Dinner    2  0.059447
1      10.34  1.66     No  Sun  Dinner    3  0.160542
2      21.01  3.50     No  Sun  Dinner    3  0.166587
3      23.68  3.31     No  Sun  Dinner    2  0.139780
4      24.59  3.61     No  Sun  Dinner    4  0.146808
5      25.29  4.71     No  Sun  Dinner    4  0.186240
```



```
In [50]: grouped = tips.groupby(['day', 'smoker'])
In [51]: grouped_pct = grouped['tip_pct']
In [52]: grouped_pct.agg('mean')
Out[52]:
day  smoker
Fri      No  0.151650
        Yes  0.174783
Sat      No  0.158048
        Yes  0.147906
Sun      No  0.160113
        Yes  0.187250
Thur     No  0.160298
        Yes  0.163863
Name: tip_pct, dtype: float64
```



```
In [53]: grouped_pct.agg(['mean', 'std', peak_to_peak])
Out[53]:
          mean      std  peak_to_peak
day  smoker
Fri      No  0.151650  0.028123    0.067349
        Yes  0.174783  0.051293    0.159925
Sat      No  0.158048  0.039767    0.235193
        Yes  0.147906  0.061375    0.290095
Sun      No  0.160113  0.042347    0.193226
        Yes  0.187250  0.154134    0.644685
Thur     No  0.160298  0.038774    0.193350
        Yes  0.163863  0.039389    0.151240
```

2. Tổng hợp dữ liệu (Data Aggregation)

2.1. Tổng hợp bằng nhiều hàm khác nhau trên cùng một cột (Column-Wise and Multiple Function Application)

- Nếu chuyển một list các tuples gồm (name, function), phần tử đầu tiên của mỗi bộ dữ liệu sẽ được sử dụng làm tên cột *DataFrame* (có thể coi list gồm 2-tuples như một ánh xạ có thứ tự - *ordered mapping*)

```
In [54]: grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

```
Out[54]:
```

		foo	bar
day	smoker		
Fri	No	0.151650	0.028123
	Yes	0.174783	0.051293
Sat	No	0.158048	0.039767
	Yes	0.147906	0.061375
Sun	No	0.160113	0.042347
	Yes	0.187250	0.154134
Thur	No	0.160298	0.038774
	Yes	0.163863	0.039389

2. Tổng hợp dữ liệu (Data Aggregation)

2.1. Tổng hợp bằng nhiều hàm khác nhau trên cùng một cột (Column-Wise and Multiple Function Application)

- Với *DataFrame*, có nhiều tùy chọn hơn vì có thể chỉ định danh sách các hàm để áp dụng cho tất cả các cột hoặc các hàm khác nhau trên mỗi cột.

Giả sử muốn tính ba số liệu thống kê giống nhau cho cột `tip_pct` và `total_bill`:

```
tips #trong đó tips['tip_pct'] = tips['tip'] / tips['total_bill']
total_bill  tip smoker  day   time  size  tip_pct
0      16.99   1.01   No  Sun  Dinner     2   0.059447
1      10.34   1.66   No  Sun  Dinner     3   0.160542
2      21.01   3.50   No  Sun  Dinner     3   0.166587
3      23.68   3.31   No  Sun  Dinner     2   0.139780
4      24.59   3.61   No  Sun  Dinner     4   0.146808
5      25.29   4.71   No  Sun  Dinner     4   0.186240
```

```
grouped = tips.groupby(['day', 'smoker'])
```

```
In [55]: functions = ['count', 'mean', 'max']
In [56]: result = grouped['tip_pct', 'total_bill'].agg(functions)
In [57]: result
Out[57]:
```

		tip_pct			total_bill		
		count	mean	max	count	mean	max
day	smoker						
Fri	No	4	0.151650	0.187735	4	18.420000	22.75
	Yes	15	0.174783	0.263480	15	16.813333	40.17
Sat	No	45	0.158048	0.291990	45	19.661778	48.33
	Yes	42	0.147906	0.325733	42	21.276667	50.81
Sun	No	57	0.160113	0.252672	57	20.506667	48.17
	Yes	19	0.187250	0.710345	19	24.120000	45.35
Thur	No	45	0.160298	0.266312	45	17.113111	41.19
	Yes	17	0.163863	0.241255	17	19.190588	43.11

2. Tổng hợp dữ liệu (Data Aggregation)

2.1. Tổng hợp bằng nhiều hàm khác nhau trên cùng một cột (Column-Wise and Multiple Function Application)

Từ kết quả trên, có thể thấy *DataFrame* kết quả có các cột phân cấp, giống như cách tổng hợp từng cột riêng biệt và sử dụng hàm concat để dán các kết quả lại với nhau bằng cách sử dụng tên cột làm đối số keys:

```
In [55]: functions = ['count', 'mean', 'max']
In [56]: result = grouped['tip_pct', 'total_bill'].agg(functions)
In [57]: result
Out[57]:
```

		tip_pct			total_bill		
		count	mean	max	count	mean	max
day	smoker						
Fri	No	4	0.151650	0.187735	4	18.420000	22.75
	Yes	15	0.174783	0.263480	15	16.813333	40.17
Sat	No	45	0.158048	0.291990	45	19.661778	48.33
	Yes	42	0.147906	0.325733	42	21.276667	50.81
Sun	No	57	0.160113	0.252672	57	20.506667	48.17
	Yes	19	0.187250	0.710345	19	24.120000	45.35
Thur	No	45	0.160298	0.266312	45	17.113111	41.19
	Yes	17	0.163863	0.241255	17	19.190588	43.11

```
In [58]: result['tip_pct']
Out[58]:
```

		count	mean	max
day	smoker			
Fri	No	4	0.151650	0.187735
	Yes	15	0.174783	0.263480
Sat	No	45	0.158048	0.291990
	Yes	42	0.147906	0.325733
Sun	No	57	0.160113	0.252672
	Yes	19	0.187250	0.710345
Thur	No	45	0.160298	0.266312
	Yes	17	0.163863	0.241255

```
In [59]: result[' total_bill ']
Out[59]:
```

		count	mean	max
day	smoker			
Fri	No	4	18.420000	22.75
	Yes	15	16.813333	40.17
Sat	No	45	19.661778	48.33
	Yes	42	21.276667	50.81
Sun	No	57	20.506667	48.17
	Yes	19	24.120000	45.35
Thur	No	45	17.113111	41.19
	Yes	17	19.190588	43.11

2. Tổng hợp dữ liệu (Data Aggregation)

2.1. Tổng hợp bằng nhiều hàm khác nhau trên cùng một cột (Column-Wise and Multiple Function Application)

- Có thể chuyển một list các tuples chứa tên tùy chỉnh cho cột và hàm sẽ áp dụng cho cột đó:

`tips` #trong đó `tips['tip_pct'] = tips['tip'] / tips['total_bill']`

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240



```
grouped = tips.groupby(['day', 'smoker'])
```



In [60]: `ftuples = [('Durchschnitt', 'mean'), ('Abweichung', np.var)]`
`...: grouped['tip_pct', 'total_bill'].agg(ftuples)`
Out [60]:

		tip_pct		total_bill	
		Durchschnitt	Abweichung	Durchschnitt	Abweichung
day	smoker				
Fri	No	0.151650	0.000791	18.420000	25.596333
	Yes	0.174783	0.002631	16.813333	82.562438
Sat	No	0.158048	0.001581	19.661778	79.908965
	Yes	0.147906	0.003767	21.276667	101.387535
Sun	No	0.160113	0.001793	20.506667	66.099980
	Yes	0.187250	0.023757	24.120000	109.046044
Thur	No	0.160298	0.001503	17.113111	59.625081
	Yes	0.163863	0.001551	19.190588	69.808518

2. Tổng hợp dữ liệu (Data Aggregation)

2.1. Tổng hợp bằng nhiều hàm khác nhau trên cùng một cột

- Áp dụng các hàm khác nhau cho một hoặc nhiều cột:
- Chuyển một dict tới agg. Trong đó mỗi item của dict cần chỉ ra key là tên cột sẽ được tính toán và value là tên hàm sẽ thực hiện tính toán.
- Nếu cần tính toán nhiều hàm trên cùng 1 cột thì đưa danh sách các hàm vào trong 1 list:

```
In [61]: grouped.agg({'tip' : np.max, 'size' : 'sum'})
Out[61]:
```

		tip	size
day	smoker		
Fri	No	3.50	9
	Yes	4.73	31
Sat	No	9.00	115
	Yes	10.00	104
Sun	No	6.00	167
	Yes	6.50	49
Thur	No	6.70	112
	Yes	5.00	40

```
In [62]: grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
.....:                'size' : 'sum'})
Out[62]:
```

		tip_pct				size
		min	max	mean	std	sum
day	smoker					
Fri	No	0.120385	0.187735	0.151650	0.028123	9
	Yes	0.103555	0.263480	0.174783	0.051293	31
Sat	No	0.056797	0.291990	0.158048	0.039767	115
	Yes	0.035638	0.325733	0.147906	0.061375	104
Sun	No	0.059447	0.252672	0.160113	0.042347	167
	Yes	0.065660	0.710345	0.187250	0.154134	49
Thur	No	0.072961	0.266312	0.160298	0.038774	112
	Yes	0.090014	0.241255	0.163863	0.039389	40

2.2. Trả về dữ liệu tổng hợp không có chỉ mục hàng (Returning Aggregated Data Without Row Indexes)

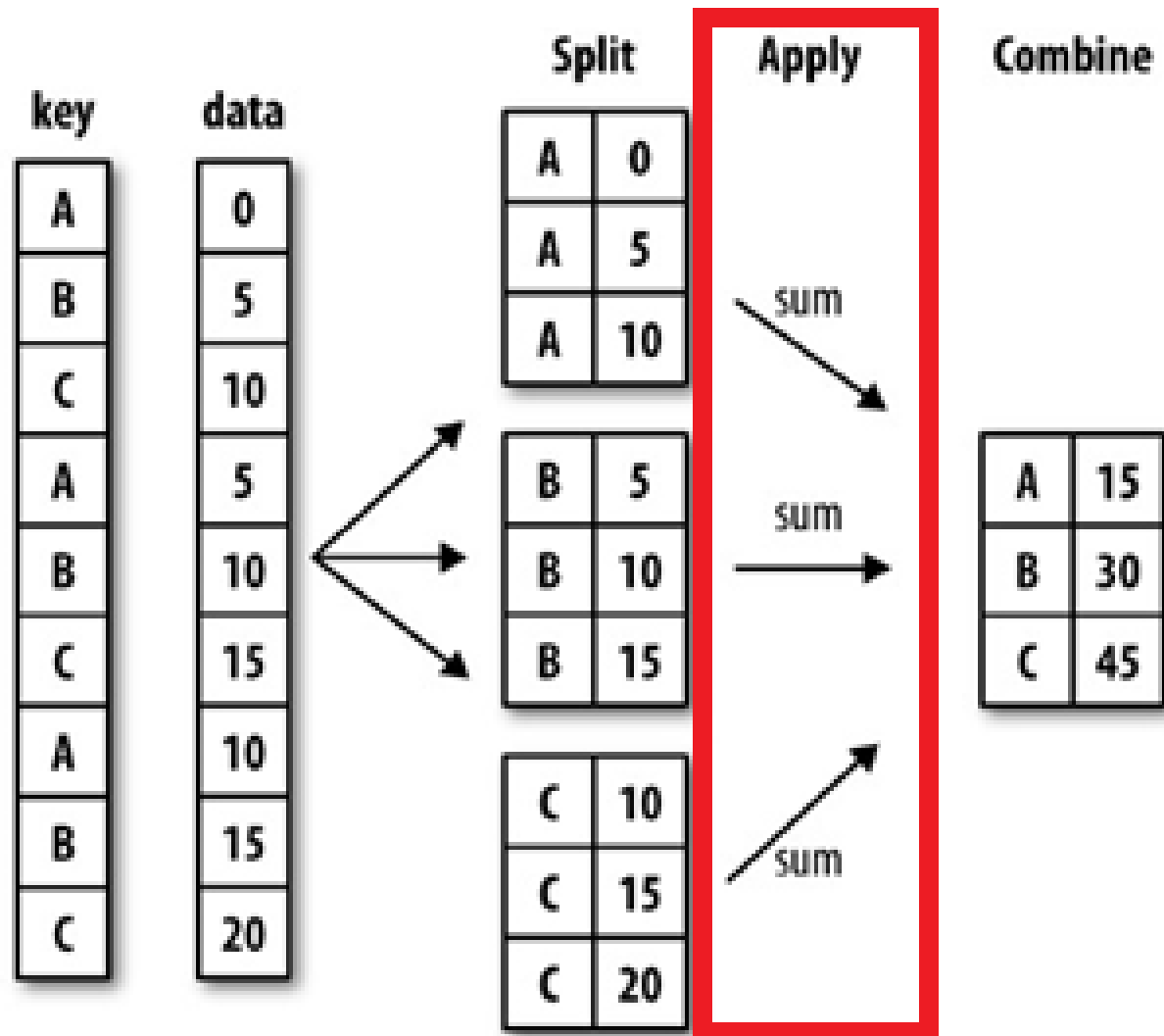
Theo mặc định, dữ liệu tổng hợp đều có một chỉ mục, có khả năng phân cấp, được tạo từ các tổ hợp khóa nhóm (*group key*) duy nhất. Có thể vô hiệu hóa hành vi này trong hầu hết các trường hợp bằng cách thêm đối số `as_index=False` vào phương thức `groupby`

```
In [63]: tips.groupby(['day', 'smoker'], as_index=False).mean()
Out [63]:
```

	day	smoker	total_bill	tip	size	tip_pct
0	Fri	No	18.420000	2.812500	2.250000	0.151650
1	Fri	Yes	16.813333	2.714000	2.066667	0.174783
2	Sat	No	19.661778	3.102889	2.555556	0.158048
3	Sat	Yes	21.276667	2.875476	2.476190	0.147906
4	Sun	No	20.506667	3.167895	2.929825	0.160113
5	Sun	Yes	24.120000	3.516842	2.578947	0.187250
6	Thur	No	17.113111	2.673778	2.488889	0.160298
7	Thur	Yes	19.190588	3.030000	2.352941	0.163863

3. PHƯƠNG THỨC apply DÙNG CHO split-apply-combine

(Apply: General split-apply-combine)



3. Phương thức apply dùng cho split-apply-combine (Apply: General split-apply-combine)

Giả sử muốn chọn năm giá trị lớn nhất theo nhóm `smoker` trong cột `tip_pct` của dữ liệu *tips* sau:

- **B1**: Đầu tiên, viết hàm chọn các hàng có giá trị lớn nhất trong cột `tip_pct` :

```
tips #trong đó tips['tip_pct'] = tips['tip'] / tips['total_bill']
total_bill  tip  smoker  day  time  size  tip_pct
0      16.99  1.01    No   Sun  Dinner    2  0.059447
1      10.34  1.66    No   Sun  Dinner    3  0.160542
2      21.01  3.50    No   Sun  Dinner    3  0.166587
3      23.68  3.31    No   Sun  Dinner    2  0.139780
4      24.59  3.61    No   Sun  Dinner    4  0.146808
5      25.29  4.71    No   Sun  Dinner    4  0.18624

In [64]: def top(df, n=5, column='tip_pct'):
        ....:     return df.sort_values(by=column)[-n:]
In [65]: top(tips, n=10)
Out[65]:
      total_bill  tip  smoker  day  time  size  tip_pct
88          24.71  5.85     No  Thur  Lunch    2  0.23675
185          20.69   5      No   Sun  Dinner    5  0.24166
51          10.29  2.6      No   Sun  Dinner    2  0.25267
149           7.51   2      No  Thur  Lunch    2  0.26631
109          14.31   4      Yes  Sat  Dinner    2  0.27953
183          23.17  6.5      Yes  Sun  Dinner    4  0.28054
232          11.61  3.39     No  Sat  Dinner    2  0.29199
67           3.07   1      Yes  Sat  Dinner    1  0.32573
178           9.6   4      Yes  Sun  Dinner    2  0.41667
172           7.25  5.15     Yes  Sun  Dinner    2  0.71035
```

3. Phương thức apply dùng cho split-apply-combine (Apply: General split-apply-combine)

- **B2**: gọi phương thức kèm tên field
 - Khi phương thức **groupby** thực hiện trên 1 field: nhóm theo **smoker** và gọi **apply** với hàm **top** này, sẽ nhận được kết quả sau:

tips #trong đó `tips['tip_pct'] = tips['tip'] / tips['total_bill']`

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

- Hàm `top` được gọi trên mỗi nhóm hàng từ `DataFrame`
- Sau đó các kết quả được nối lại với nhau bằng `pandas.concat`
- Gắn nhãn các phần bằng tên nhóm để cho kết quả cuối cùng

```
In [66]: tips.groupby('smoker').apply(top)
Out[66]:
```

		total_bill	tip	smoker	day	time	size	tip_pct	
smoker	No	88	24.71	5.85	No	Thur	Lunch	2	0.236746
		185	20.69	5.00	No	Sun	Dinner	5	0.241663
		51	10.29	2.60	No	Sun	Dinner	2	0.252672
		149	7.51	2.00	No	Thur	Lunch	2	0.266312
		232	11.61	3.39	No	Sat	Dinner	2	0.291990
Yes	109	14.31	4.00	Yes	Sat	Dinner	2	0.279525	
		183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
		67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
		178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
		172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

3. Phương thức apply dùng cho split-apply-combine (Apply: General split-apply-combine)

- **B2**: gọi phương thức kèm tên field
 - Khi phương thức `groupby` thực hiện trên nhiều field: sẽ tạo ra phân cấp thêm cho kết quả và có thể truyền thêm cho hàm `apply` các đối số hoặc từ khóa khác tùy theo nhu cầu sử dụng:

`tips` #trong đó `tips['tip_pct'] = tips['tip'] / tips['total_bill']`

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

```
In [67]: tips.groupby(['smoker', 'day']).apply(top, n=1,
                                                column='total_bill')
```

Out [67]:

			total_bill	tip	smoker	day	time	size	tip_pct
smoker		day							
No	Fri	94	22.75	3.25	No	Fri	Dinner	2	0.142857
		Sat	212	48.33	No	Sat	Dinner	4	0.186220
		Sun	156	48.17	No	Sun	Dinner	6	0.103799
		Thur	142	41.19	No	Thur	Lunch	5	0.121389
Yes	Fri	95	40.17	4.73	Yes	Fri	Dinner	4	0.117750
		Sat	170	50.81	Yes	Sat	Dinner	3	0.196812
		Sun	182	45.35	Yes	Sun	Dinner	3	0.077178
		Thur	197	43.11	Yes	Thur	Lunch	4	0.115982

3. Phương thức apply dùng cho split-apply-combine (Apply: General split-apply-combine)

- Có thể nhớ lại rằng trước đây đã gọi describe trên đối tượng `GroupBy`:

```
In [68]: result = tips.groupby('smoker')['tip_pct'].describe()
In [69]: result
Out[69]:
```

	count	mean	std	min	25%	50%	75%	max
No	151.0	0.159328	0.039910	0.056797	0.136906	0.155625	0.185014	0.291990
Yes	93.0	0.163196	0.085119	0.035638	0.106771	0.153846	0.195059	0.710345

```
In [70]: result.unstack('smoker')
Out[70]:
```

smoker		
count	No	151.000000
	Yes	93.000000
mean	No	0.159328
	Yes	0.163196
std	No	0.039910
	Yes	0.085119
min	No	0.056797
	Yes	0.035638
25%	No	0.136906
	Yes	0.106771
50%	No	0.155625
	Yes	0.153846
75%	No	0.185014
	Yes	0.195059
max	No	0.291990
	Yes	0.710345
dtype:		float64

Bên trong `GroupBy`, khi gọi một phương thức như `describe`, nó thực sự chỉ là một lối tắt (*shortcut*) cho hàm lambda sau:

```
f = lambda x: x.describe() grouped.apply(f)
```

3.1. Loại bỏ các khóa nhóm (Suppressing the Group Keys)

- Có thể tắt tính năng này bằng cách thêm đối số `group_keys=False` cho phương thức `groupby`:

```
In [71]: tips.groupby('smoker', group_keys=False).apply(top)
Out[71]:
```

	total_bill	tip	smoker	day	time	size	tip_pct
88	24.71	5.85	No	Thur	Lunch	2	0.236746
185	20.69	5.00	No	Sun	Dinner	5	0.241663
51	10.29	2.60	No	Sun	Dinner	2	0.252672
149	7.51	2.00	No	Thur	Lunch	2	0.266312
232	11.61	3.39	No	Sat	Dinner	2	0.291990
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

```
In [72]: tips.groupby('smoker').apply(top)
Out[72]:
```

smoker		total_bill	tip	smoker	day	time	size	tip_pct
No	88	24.71	5.85	No	Thur	Lunch	2	0.236746
	185	20.69	5.00	No	Sun	Dinner	5	0.241663
	51	10.29	2.60	No	Sun	Dinner	2	0.252672
	149	7.51	2.00	No	Thur	Lunch	2	0.266312
Yes	232	11.61	3.39	No	Sat	Dinner	2	0.291990
	109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
	183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
	67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
	178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
	172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

3.2. Phân tích lượng tử và nhóm (Quantile and Bucket Analysis)

pandas có một số công cụ, đặc biệt là `cut` và `qcut`, để cắt dữ liệu thành các nhóm với các nhóm (hay ngăn - *bins*) do người dùng tự chọn hoặc theo lượng tử mẫu (*sample quantiles*).

Việc kết hợp các chức năng này với `groupby` giúp việc thực hiện phân tích nhóm (*bucket*) hoặc lượng tử (*quantile*) trên tập dữ liệu trở nên thuận tiện.

Hãy xem xét một tập dữ liệu ngẫu nhiên đơn giản và phân loại nhóm có độ dài bằng nhau (*equal-length bucket categorization*) bằng cách sử dụng `cut`:

frame		
	data1	data2
0	-0.976109	0.323824
1	-1.562182	1.255462
2	0.441154	0.719108
3	-1.327498	1.831213
4	-1.119932	-1.680169
...
995	-2.086971	-0.096300
996	1.456853	0.587568
997	0.633794	0.794055
998	0.530703	-0.937737
999	0.023687	-1.125411
1000 rows x 2 columns		

```
In [73]: quartiles = pd.cut(frame.data1, 4)
        ....: quartiles
Out[72]:
0      (-1.36,0.381]
1      (-3.108,-1.36]
2      (0.381,2.121]
3      (-1.36,0.381]
4      (-1.36,0.381]
...
995     (-3.108,-1.36]
996     (0.381,2.121]
997     (0.381,2.121]
998     (0.381,2.121]
999     (-1.36,0.381]

Name: data1, Length: 1000, dtype: category
Categories (4, interval[float64, right]): [(-3.108, -1.36] <
                                           (-1.36, 0.381] < (0.381, 2.121] < (2.121, 3.862]]
```


3. Phương thức apply dùng cho split-apply-combine (*Apply: General split-apply-combine*)

3.2. Phân tích lượng tử và nhóm (Quantile and Bucket Analysis)

pandas có một số công cụ, đặc biệt là `cut` và `qcut`, để cắt dữ liệu thành các nhóm với các nhóm (hay ngăn - *bins*) do người dùng tự chọn hoặc theo lượng tử mẫu (*sample quantiles*).

```
In [73]: quartiles = pd.cut(frame.data1, 4)
        ....: quartiles

Out[72]:
0      (-1.36,0.381]
1      (-3.108,-1.36]
2      (0.381,2.121]
3      (-1.36,0.381]
4      (-1.36,0.381]
...
995    (-3.108,-1.36]
996    (0.381,2.121]
997    (0.381,2.121]
998    (0.381,2.121]
999    (-1.36,0.381]

Name: data1, Length: 1000, dtype: category
Categories (4, interval[float64, right]): [(-3.108, -1.36] <
                                           (-1.36, 0.381] < (0.381, 2.121] < (2.121, 3.862]]
```

```
In [74]: quartiles[:10]
Out[74]:
```

0	(-1.36, 0.381]
1	(-3.108, -1.36]
2	(0.381, 2.121]
3	(-1.36, 0.381]
4	(-1.36, 0.381]
5	(0.381, 2.121]
6	(-3.108, -1.36]
7	(-1.36, 0.381]
8	(0.381, 2.121]
9	(-1.36, 0.381]

Name: data1, dtype: category
Categories (4, interval[float64, right]):
[(-3.108, -1.36] < (-1.36, 0.381] <
(0.381, 2.121] < (2.121, 3.862]]

3.2. Phân tích lượng tử và nhóm (*Quantile and Bucket Analysis*)

Đối tượng *Categorical* được trả về bởi `cut` (tạo nhóm (*buckets*) có miền giá trị bằng nhau) có thể được chuyển trực tiếp tới `groupby`. Vì vậy, có thể tính toán một tập hợp số liệu thống kê cho cột `data2` như sau:

```
In [75]: def get_stats(group):
.....:     return {'min': group.min(), 'max': group.max(),
.....:             'count': group.count(), 'mean': group.mean()}
In [76]: grouped = frame.data2.groupby(quantiles)
In [77]: grouped.apply(get_stats).unstack()
Out[77]:
```

	min	max	count	mean
data1				
(-3.108, -1.36]	-2.549418	2.394429	93.0	0.105303
(-1.36, 0.381]	-3.311799	3.135589	564.0	-0.013696
(0.381, 2.121]	-2.330625	3.235061	324.0	0.063328
(2.121, 3.862]	-0.843283	1.458427	19.0	0.176194

3.2. Phân tích lượng tử và nhóm (*Quantile and Bucket Analysis*)

Để tính toán các nhóm có kích thước (số lượng phần tử) bằng nhau dựa trên lượng tử mẫu, hãy sử dụng `qcut`. Lúc này, sử dụng đối số **`labels=False`** để chỉ số lượng tử cần lấy:

```
# Return quantile numbers
In [78]: grouping = pd.qcut(frame.data1, 10, labels=False)
In [79]: grouped = frame.data2.groupby(grouping)
In [80]: grouped.apply(get_stats).unstack()
Out[80]:
```

	min	max	count	mean
data1				
0	-2.549418	2.394429	100.0	0.149586
1	-3.311799	2.785687	100.0	-0.060164
2	-3.232371	3.135589	100.0	-0.119662
3	-2.264888	3.108391	100.0	0.084592
4	-2.352826	2.133101	100.0	-0.138631
5	-2.921217	2.309757	100.0	0.008078
6	-1.955823	2.220635	100.0	0.100089
7	-2.304838	2.254004	100.0	0.115845
8	-2.330625	3.235061	100.0	0.009124
9	-1.979998	2.778011	100.0	0.110489

3.2. Phân tích lượng tử và nhóm (*Quantile and Bucket Analysis*)

Để tính toán các nhóm có kích thước (số lượng phần tử) bằng nhau dựa trên lượng tử mẫu, hãy sử dụng `qcut`. Lúc này, sử dụng đối số **`labels=False`** để chỉ số lượng tử cần lấy:

```
# Return quantile numbers
In [78]: grouping = pd.qcut(frame.data1, 10, labels=False)
In [79]: grouped = frame.data2.groupby(grouping)
In [80]: grouped.apply(get_stats).unstack()
Out[80]:
```

	min	max	count	mean
data1				
0	-2.549418	2.394429	100.0	0.149586
1	-3.311799	2.785687	100.0	-0.060164
2	-3.232371	3.135589	100.0	-0.119662
3	-2.264888	3.108391	100.0	0.084592
4	-2.352826	2.133101	100.0	-0.138631
5	-2.921217	2.309757	100.0	0.008078
6	-1.955823	2.220635	100.0	0.100089
7	-2.304838	2.254004	100.0	0.115845
8	-2.330625	3.235061	100.0	0.009124
9	-1.979998	2.778011	100.0	0.110489

3.3. Một số ví dụ

- Ví dụ 1: Điền các giá trị còn thiếu bằng các giá trị cụ thể theo từng nhóm

- Sử dụng **fillna**:

- Điền giá trị NA bằng giá trị trung bình (*mean*) cho tất cả các phần tử:

```
s
0      NaN
1    1.101768
2      NaN
3   -0.324564
4      NaN
5   -0.345106
dtype: float64
```



```
In [84]: S2 = s.fillna(s.mean())
.....: S2
Out[84]:
0      0.144033
1    1.101768
2      0.144033
3   -0.324564
4      0.144033
5   -0.345106
dtype: float64
```

- Điền giá trị thay đổi theo nhóm

```
data
Ohio      0.243966
New York  -1.141501
Vermont      NaN
Florida   -0.556632
Oregon     1.089663
Nevada      NaN
California 0.166817
Idaho      NaN
dtype: float64
```

```
group_key
['East', 'East', 'East', 'East', 'West', 'West', 'West', 'West']
```



```
In [91]: data.groupby(group_key).mean()
Out[91]:
East    -0.358967
West     0.485801
dtype: float64
```

3. Phương thức apply dùng cho split-apply-combine (*Apply: General split-apply-combine*)

3.3. Một số ví dụ

- Ví dụ 1: Điền các giá trị còn thiếu bằng các giá trị cụ thể theo từng nhóm
 - Có thể điền các giá trị NA bằng cách sử dụng hàm tự tạo của người
 - Điền giá trị NA bằng giá trị trung bình (*mean*) cho tất cả các phần tử:

```
data
Ohio      0.243966
New York  -1.141501
Vermont    NaN
Florida   -0.556632
Oregon     1.089663
Nevada     NaN
California 0.166817
Idaho      NaN
dtype: float64
```

```
group_key
```

```
['East', 'East', 'East', 'East', 'West', 'West', 'West', 'West']
```



```
In [92]: fill_mean = lambda g: g.fillna(g.mean())
In [93]: data.groupby(group_key).apply(fill_mean)
Out[93]:
Ohio      0.243966
New York| -1.141501
Vermont    -0.484722
Florida   -0.556632
Oregon     1.089663
Nevada     0.628240
California 0.166817
Idaho      0.628240
dtype: float64
```

3. Phương thức apply dùng cho split-apply-combine (Apply: General split-apply-combine)

3.3. Một số ví dụ

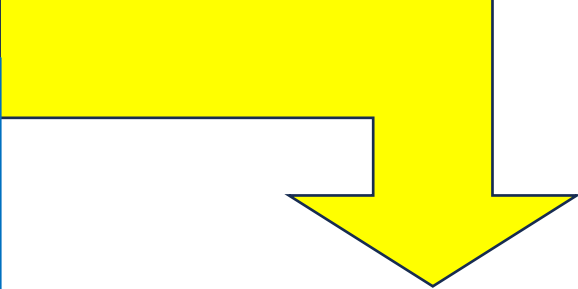
- Ví dụ 1: Điền các giá trị còn thiếu bằng các giá trị cụ thể theo từng nhóm
 - Có thể điền các giá trị NA bằng cách sử dụng hàm tự tạo của người
 - Có thể chỉ định giá trị sẽ thay thế cho từng nhóm

group_key

['East', 'East', 'East', 'East', 'West', 'West', 'West', 'West']

data

Ohio	0.243966
New York	-1.141501
Vermont	NaN
Florida	-0.556632
Oregon	1.089663
Nevada	NaN
California	0.166817
Idaho	NaN



In [94]: fill_values = {'East': 0.5, 'West': -1}

In [95]: fill_func = lambda g: g.fillna(fill_values[g.name])

In [96]: data.groupby(group_key).apply(fill_func)

Out[96]:

Ohio	0.243966
New York	-1.141501
Vermont	0.500000
Florida	-0.556632
Oregon	1.089663
Nevada	-1.000000
California	0.166817
Idaho	-1.000000

3.3. Một số ví dụ

- Ví dụ 2: Lấy mẫu ngẫu nhiên và hoán vị (*Random Sampling and Permutation*)

Giả sử muốn lấy một mẫu ngẫu nhiên (có hoặc không thay thế) từ một tập dữ liệu lớn cho mục đích mô phỏng *Monte Carlo* hoặc một số ứng dụng khác. Có một số cách để thực hiện việc “rút thăm” (“*draws*”); ở đây sử dụng phương thức `sample` cho *Series*.

Đây là cách thường dùng để xây dựng một bộ bài kiểu Anh:

```
# Hearts, Spades, Clubs, Diamonds
suits = ['H', 'S', 'C', 'D']
card_val = (list(range(1, 11)) + [10] * 3) * 4
.....: group_key
''' tạo list chứa 52 giá trị:
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,
     1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,
     1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10,
     1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10, 10, 10] '''
base_names = ['A'] + list(range(2, 11)) + ['J', 'K', 'Q']
''' tạo list chứa 13 giá trị:
    ['A', 2, 3, 4, 5, 6, 7, 8, 9, 10, 'J', 'K', 'Q'] '''
cards = []
```


3. Phương thức apply dùng cho split-apply-combine (*Apply: General split-apply-combine*)

3.3. Một số ví dụ

- Ví dụ 2: Lấy mẫu ngẫu nhiên và hoán vị (*Random Sampling and Permutation*)

Đây là cách thường dùng để xây dựng một bộ bài kiểu Anh:

```
''' Tạo list cards chứa 52 giá trị đại diện cho 52 lá bài
    ['AH', '2H', '3H', '4H', '5H', '6H', '7H', '8H', '9H', '10H', 'JH', 'KH', 'QH',
    'AS', '2S', '3S', '4S', '5S', '6S', '7S', '8S', '9S', '10S', 'JS', 'KS', 'QS',
    'AC', '2C', '3C', '4C', '5C', '6C', '7C', '8C', '9C', '10C', 'JC', 'KC', 'QC',
    'AD', '2D', '3D', '4D', '5D', '6D', '7D', '8D', '9D', '10D', 'JD', 'KD', 'QD']
'''

for suit in ['H', 'S', 'C', 'D']:
    cards.extend(str(num) + suit for num in base_names)

''' Tạo Series "deck" chứa có độ dài 52 có chỉ mục chứa tên và giá
trị của từng quân bài

    AH    1    AS    1    AC    1    AD    1
    2H    2    2S    2    2C    2    2D    2
    3H    3    3S    3    3C    3    3D    3
    4H    4    4S    4    4C    4    4D    4
    5H    5    5S    5    5C    5    5D    5
    6H    6    6S    6    6C    6    6D    6
    7H    7    7S    7    7C    7    7D    7
    8H    8    8S    8    8C    8    8D    8
    9H    9    9S    9    9C    9    9D    9
    10H   10   10S   10   10C   10   10D   10
    JH    10   JS    10   JC    10   JD    10
    KH    10   KS    10   KC    10   KD    10
    QH    10   QS    10   QC    10   QD    10

dtype: int64 '''
deck = pd.Series(card_val, index=cards)
```

3. Phương thức apply dùng cho split-apply-combine (*Apply: General split-apply-combine*)

3.3. Một số ví dụ

- Ví dụ 2: Lấy mẫu ngẫu nhiên và hoán vị (*Random Sampling and Permutation*)

Vậy là đã có một *Series* có độ dài 52 có chỉ mục chứa tên và giá trị quân bài. Dựa vào đó, việc rút một tay gồm năm lá bài từ bộ bài có thể được viết là:

```
In [97]: def draw(deck, n=5):
        .....:     return deck.sample(n)
In [98]: draw(deck)
Out[98]:
    5H    5
    QH   10
    9C    9
    KH   10
    KC   10
dtype: int64
```

```
''' Tạo Series "deck" chứa có độ dài 52 có chỉ mục chứa tên và giá
trị của từng quân bài
    AH    1   AS    1   AC    1   AD    1
    2H    2   2S    2   2C    2   2D    2
    3H    3   3S    3   3C    3   3D    3
    4H    4   4S    4   4C    4   4D    4
    5H    5   5S    5   5C    5   5D    5
    6H    6   6S    6   6C    6   6D    6
    7H    7   7S    7   7C    7   7D    7
    8H    8   8S    8   8C    8   8D    8
    9H    9   9S    9   9C    9   9D    9
    10H   10  10S   10  10C   10  10D   10
    JH    10   JS    10   JC    10   JD    10
    KH    10   KS    10   KC    10   KD    10
    QH    10   QS    10   QC    10   QD    10
dtype: int64 '''
deck = pd.Series(card_val, index=cards)
```

3. Phương thức apply dùng cho split-apply-combine (*Apply: General split-apply-combine*)

3.3. Một số ví dụ

- Ví dụ 2: Lấy mẫu ngẫu nhiên và hoán vị (*Random Sampling and Permutation*)

Giả sử muốn có hai lá bài ngẫu nhiên từ mỗi chất (suit, có 4 suit gồm **Hearts**, **Spades**, **Clubs**, **Diamonds**). Vì chất là ký tự cuối cùng của mỗi tên lá bài nên có thể nhóm dựa trên điều này và sử dụng `apply`. Có thể sử dụng 1 trong 2 cách sau:

```
In [99]: get_suit = lambda card: card[-1] # last letter is suit
In [100]: deck.groupby(get_suit).apply(draw, n=2)
Out[101]:
   C 6C   6
   AC   1
   D KD  10
   QD  10
   H 9H   9
   AH   1
   S JS  10
   7S   7
dtype: int64
```

```
In [102]: deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
Out[102]:
   2C   2
   9C   9
   5D   5
   4D   4
   KH  10
   2H   2
   KS  10
   2S   2
dtype: int64
```

3. Phương thức apply dùng cho split-apply-combine (*Apply: General split-apply-combine*)

3.3. Một số ví dụ

- Ví dụ 3: Trung bình trọng số nhóm và mối tương quan (*Group Weighted Average and Correlation*)

Theo mô hình *phân tách-áp dụng-kết hợp* (*split – apply - combine*) của `groupby`, các thao tác giữa các cột trong *DataFrame* hoặc hai *Series*, chẳng hạn như tính mức trung bình có trọng số của nhóm, đều có thể thực hiện được.

```
In [103]: df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',  
.....:                                'b', 'b', 'b', 'b'],  
.....:                      'data': np.random.randn(8),  
.....:                      'weights': np.random.rand(8)})
```

```
In [104]: df
```

```
Out[104]:
```

	category	data	weights
0	a	1.561587	0.957515
1	a	1.219984	0.347267
2	a	-0.482239	0.581362
3	a	0.315667	0.217091
4	b	-0.047852	0.894406
5	b	-0.454145	0.918564
6	b	-0.556774	0.277825
7	b	0.253321	0.955905

```
In [105]: grouped = df.groupby('category')
```

```
In [106]: get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
```

```
In [107]: grouped.apply(get_wavg)
```

```
Out[107]:
```

```
category  
a      0.811643  
b     -0.122262  
dtype: float64
```

3.3. Một số ví dụ

- Ví dụ 4: Hồi quy tuyến tính theo nhóm (*Group-Wise Linear Regression*)

Có thể sử dụng `groupby` để thực hiện phân tích thống kê theo nhóm phức tạp hơn, miễn là hàm trả về đối tượng *pandas* hoặc giá trị vô hướng (*scalar value*).

Có thể xác định hàm hồi quy sau (sử dụng thư viện thống kê kinh tế lượng - *the statsmodels econometrics library*), thực hiện hồi quy bình phương nhỏ nhất thông thường (*Ordinary Least Squares regression* - OLS) trên mỗi đoạn (*chunk*) của dữ liệu:

```
close_px[-4:]
```

	AAPL	MSFT	XOM	SPX
2011-10-11	400.29	27.00	76.27	1195.54
2011-10-12	402.19	26.96	77.16	1207.25
2011-10-13	408.43	27.18	76.37	1203.66
2011-10-14	422.00	27.27	78.11	1224.58

```
import statsmodels.api as sm
def regress(data, yvar, xvars):
    Y = data[yvar]
    X = data[xvars]
    X['intercept'] = 1.
    result = sm.OLS(Y, X).fit()
    return result.params
```

```
In [117]: by_year.apply(regress, 'AAPL', ['SPX'])
Out[117]:
```

	SPX	intercept
2003	1.195406	0.000710
2004	1.363463	0.004201
2005	1.766415	0.003246
2006	1.645496	0.000080
2007	1.198761	0.003438
2008	0.968016	-0.001110

4. PIVOT TABLES & CROSS-TABULATION

4.1. Bảng tổng hợp (*pivot table*)

pandas có hàm ***pandas.pivot_table*** cấp cao nhất và *DataFrame* cũng có phương thức ***pivot_table*** riêng cho mình. Ngoài việc cung cấp giao diện thuận tiện cho việc *groupby*, ***pivot_table*** còn có thể thêm một phần tổng, còn được gọi là lề (*margins*).

Các tùy chọn của pivot_table

<i>Đối số</i>	<i>Mô tả</i>
values	Tên cột hoặc tên để tổng hợp; theo mặc định tổng hợp tất cả các cột số
index	Tên cột hoặc các khóa nhóm khác để nhóm trên các hàng của bảng tổng hợp kết quả
columns	Tên cột hoặc các khóa nhóm khác để nhóm trên các cột của bảng tổng hợp kết quả
aggfunc	Hàm tổng hợp hoặc danh sách các hàm (mặc định là hàm 'mean'); có thể là bất kỳ hàm nào hợp lệ trong bối cảnh nhóm
fill_value	Thay thế các giá trị còn thiếu trong bảng kết quả
dropna	Nếu là <code>True</code> , không bao gồm các cột có mục nhập đều là <code>NA</code>
margins	Thêm tổng phụ hàng/cột và tổng cuối (giá trị mặc định là <code>False</code>)

4.1. Bảng tổng hợp (*pivot table*)

pandas có hàm ***pandas.pivot_table*** cấp cao nhất và *DataFrame* cũng có phương thức ***pivot_table*** riêng cho mình. Ngoài việc cung cấp giao diện thuận tiện cho việc *groupby*, ***pivot_table*** còn có thể thêm một phần tổng, còn được gọi là lề (*margins*).

Giả sử muốn tính một *pivot_table* mặc định được sắp xếp theo *day* và *smoker* trên các hàng của *DataFrame* ***tips***

tips #trong đó tips['tip_pct'] = tips['tip'] / tips['total_bill']

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

In [118]: tips.pivot_table(index=['day', 'smoker'])
Out[118]:

		size	tip	tip_pct	total_bill
day	smoker				
Fri	No	2.250000	2.812500	0.151650	18.420000
	Yes	2.066667	2.714000	0.174783	16.813333
Sat	No	2.555556	3.102889	0.158048	19.661778
	Yes	2.476190	2.875476	0.147906	21.276667
Sun	No	2.929825	3.167895	0.160113	20.506667
	Yes	2.578947	3.516842	0.187250	24.120000
Thur	No	2.488889	2.673778	0.160298	17.113111
	Yes	2.352941	3.030000	0.163863	19.190588

4.1. Bảng tổng hợp (pivot table)

Giả sử chỉ muốn tổng hợp tip_pct và size, đồng thời nhóm thêm theo time. Ta sẽ đặt smoker vào đối số columns và time cùng day vào đối số index:

`tips` #trong đó `tips['tip_pct'] = tips['tip'] / tips['total_bill']`

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

```
In [119]: tips.pivot_table(['tip_pct', 'size'],
.....:                    index=['time', 'day'], columns='smoker')
Out[119]:
```

		size		tip_pct		
smoker		No	Yes	No	Yes	
time	day					
	Dinner	Fri	2.000000	2.222222	0.139622	0.165347
		Sat	2.555556	2.476190	0.158048	0.147906
		Sun	2.929825	2.578947	0.160113	0.187250
Lunch		Thur	2.000000	NaN	0.159744	NaN
	Fri	3.000000	1.833333	0.187735	0.188937	
	Thur	2.500000	2.352941	0.160311	0.163863	

4.1. Bảng tổng hợp (pivot table)

Có thể bổ sung vào bảng này dòng tổng cộng và cột tổng cộng bằng cách thêm đối số *margins=True*. Điều này có tác dụng thêm nhãn **All** cho mỗi hàng và mỗi cột, với các giá trị tương ứng là số liệu thống kê nhóm cho tất cả dữ liệu trong một cấp duy nhất:

`tips`

#trong đó `tips['tip_pct'] = tips['tip'] / tips['total_bill']`

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

```
In [119]: tips.pivot_table(['tip_pct', 'size'],
.....:                    index=['time', 'day'], columns='smoker')
Out[119]:
```

		size		tip_pct	
smoker		No	Yes	No	Yes
time	day				
Dinner	Fri	2.000000	2.222222	0.139622	0.165347
	Sat	2.555556	2.476190	0.158048	0.147906
	Sun	2.929825	2.578947	0.160113	0.187250
	Thur	2.000000	NaN	0.159744	NaN
Lunch	Fri	3.000000	1.833333	0.187735	0.188937
	Thur	2.500000	2.352941	0.160311	0.163863

```
In [120]: tips.pivot_table(['tip_pct', 'size'],
.....:                    index=['time', 'day'], columns='smoker', margins=True)
Out[120]:
```

		size			tip_pct		
smoker		No	Yes	All	No	Yes	All
time	day						
Dinner	Fri	2.000000	2.222222	2.166667	0.139622	0.165347	0.158916
	Sat	2.555556	2.476190	2.517241	0.158048	0.147906	0.153152
	Sun	2.929825	2.578947	2.842105	0.160113	0.187250	0.166897
	Thur	2.000000	NaN	2.000000	0.159744	NaN	0.159744
Lunch	Fri	3.000000	1.833333	2.000000	0.187735	0.188937	0.188765
	Thur	2.500000	2.352941	2.459016	0.160311	0.163863	0.161301
All		2.668874	2.408602	2.569672	0.159328	0.163196	0.160803

4.1. Bảng tổng hợp (pivot table)

Để sử dụng hàm tổng hợp khác, hãy truyền hàm đó qua đối số *aggfunc*. Ví dụ: *count* hoặc *len* sẽ cung cấp bảng tham chiếu chéo (cross-tabulation) về số lượng hoặc tần suất của nhóm:

```
tips #trong đó tips['tip_pct'] = tips['tip'] / tips['total_bill']
total_bill  tip  smoker  day  time  size  tip_pct
0      16.99  1.01    No   Sun  Dinner    2   0.059447
1      10.34  1.66    No   Sun  Dinner    3   0.160542
2      21.01  3.50    No   Sun  Dinner    3   0.166587
3      23.68  3.31    No   Sun  Dinner    2   0.139780
4      24.59  3.61    No   Sun  Dinner    4   0.146808
5      25.29  4.71    No   Sun  Dinner    4   0.186240
```

```
In [121]: tips.pivot_table('tip_pct', index = ['time', 'smoker'],
.....:                    columns = 'day', aggfunc = len, margins = True)
Out[121]:
```

day	Fri	Sat	Sun	Thur	All	
time						
Dinner						
	No	3.0	45.0	57.0	1.0	106.0
	Yes	9.0	42.0	19.0	NaN	70.0
Lunch						
	No	1.0	NaN	NaN	44.0	45.0
	Yes	6.0	NaN	NaN	17.0	23.0
All	19.0	87.0	76.0	62.0	244.0	

4.1. Bảng tổng hợp (pivot table)

Có thể sử dụng đối số *fill_value* cho một số kết hợp trống (NA),

`tips`

#trong đó `tips['tip_pct'] = tips['tip'] / tips['total_bill']`

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

```
In [122]: tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
.....: columns='day', aggfunc = 'mean', fill_value = 0)
Out[122]:
```

			Fri	Sat	Sun	Thur	
day	time	size	smoker				
Dinner	Dinner	1	No	0.000000	0.137931	0.000000	0.000000
			Yes	0.000000	0.325733	0.000000	0.000000
		2	No	0.139622	0.162705	0.168859	0.159744
			Yes	0.171297	0.148668	0.207893	0.000000
		3	No	0.000000	0.154661	0.152663	0.000000
			Yes	0.000000	0.144995	0.152660	0.000000
	Lunch	4	No	0.000000	0.150096	0.148143	0.000000
			Yes	0.117750	0.124515	0.193370	0.000000
		5	No	0.000000	0.000000	0.206928	0.000000
			Yes	0.000000	0.106572	0.065660	0.000000
		6	No	0.000000	0.000000	0.000000	0.000000
			Yes	0.000000	0.000000	0.000000	0.000000
...			
Lunch	Lunch	1	No	0.000000	0.000000	0.000000	0.181728
			Yes	0.223776	0.000000	0.000000	0.000000
		2	No	0.000000	0.000000	0.000000	0.166005
			Yes	0.181969	0.000000	0.000000	0.158843
		3	No	0.187735	0.000000	0.000000	0.084246
			Yes	0.000000	0.000000	0.000000	0.204952
	Dinner	4	No	0.000000	0.000000	0.000000	0.138919
			Yes	0.000000	0.000000	0.000000	0.155410
		5	No	0.000000	0.000000	0.000000	0.121389
			Yes	0.000000	0.000000	0.000000	0.000000
		6	No	0.000000	0.000000	0.000000	0.173706
			Yes	0.000000	0.000000	0.000000	0.000000

[21 rows x 4 columns]

4.2. Bảng tham chiếu chéo (*Cross-Tabulations*, viết tắt là *Crosstab*)

Lập *Crosstab* là trường hợp đặc biệt của *pivot table* trong việc tính toán tần suất nhóm.

Giả sử có nhu cầu tóm tắt dữ liệu này theo quốc tịch và độ thuận tay. Có thể sử dụng *pivot_table* để thực hiện việc này, nhưng hàm *pandas.crosstab* có thể thuận tiện hơn:

data			
	Sample	Nationality	Handedness
0	1	USA	Right-handed
1	2	Japan	Left-handed
2	3	USA	Right-handed
3	4	Japan	Right-handed
4	5	Japan	Left-handed
5	6	Japan	Right-handed
6	7	USA	Right-handed
7	8	USA	Left-handed
9		9 Japan	Right-handed
9	10	USA	Right-handed

```
In [124]: pd.crosstab(data.Nationality, data.Handedness, margins=True)
Out[124]:
```

Handedness	Left-handed	Right-handed	All
Nationality			
Japan	2	3	5
USA	1	4	5
All	3	7	10

```
In [125]: pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
Out[125]:
```

		smoker		
		No	Yes	All
Dinner	day			
	Fri	3	9	12
	Sat	45	42	87
	Sun	57	19	76
Lunch	Thur	1	0	1
	Fri	1	6	7
	Thur	44	17	61
All		151	93	244

Hai đối số đầu tiên của *crosstab* có thể là một *array* hoặc *Series* hoặc một *list* các *arrays*.

