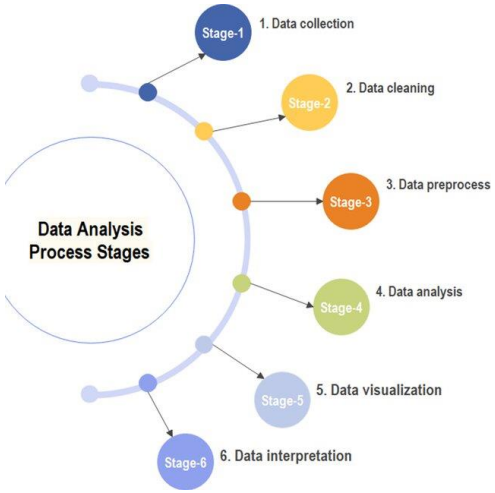
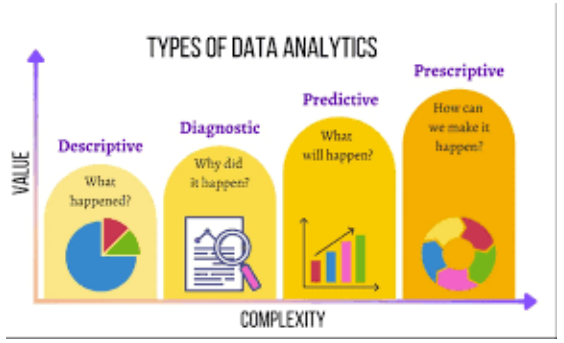
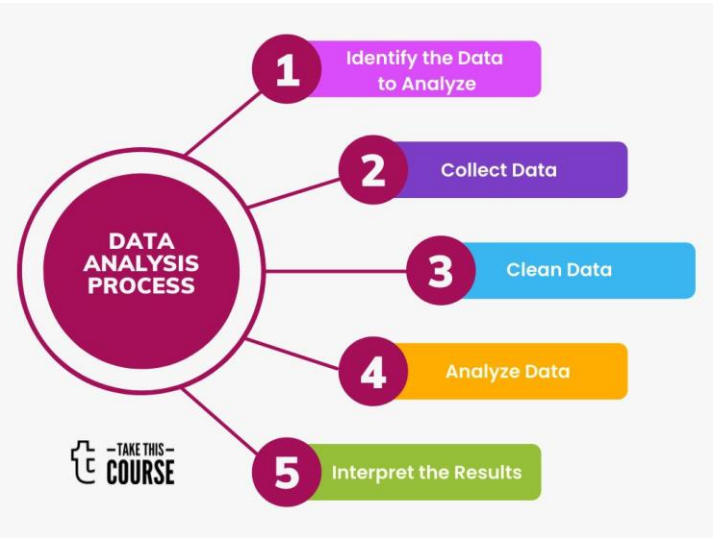




PHÂN TÍCH DỮ LIỆU

(Data Analysis)



Lê Văn Hạnh
levanhanhvn@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 5

PANDAS

Lê Văn Hạnh
levanhanhvn@gmail.com

NỘI DUNG CHƯƠNG 5

1. Giới thiệu

2. Các cấu trúc dữ liệu trong pandas

2. Một số chức năng thiết yếu

3. Tóm tắt và tính toán thống kê mô tả

1. GIỚI THIỆU

- *pandas*
  - Chứa các cấu trúc dữ liệu và các công cụ thao tác dữ liệu được thiết kế để giúp việc làm sạch và phân tích dữ liệu nhanh chóng, dễ dàng trong *Python*.
  - Thường được sử dụng song song với:
    - Các công cụ tính toán số như *NumPy* và *SciPy*
    - Các thư viện phân tích như *statsmodels* và *scikit-learn*
    - Các thư viện trực quan hóa dữ liệu như *matplotlib*.
- *pandas & numpy*
  - *pandas* áp dụng các phần quan trọng của phong cách tính toán dựa trên mảng đặc trưng của *NumPy*, đặc biệt là các hàm dựa trên mảng và ưu tiên xử lý dữ liệu mà không cần vòng lặp *for*.
  - Điểm khác biệt lớn nhất là *pandas* được thiết kế để làm việc với dữ liệu dạng bảng hoặc kiểu dữ liệu không đồng nhất. Ngược lại, *NumPy* phù hợp nhất để làm việc với dữ liệu mảng số đồng nhất.
- Trong suốt phần còn lại của *tài liệu*, sẽ sử dụng quy ước import sau đây cho *pandas*:

```
import pandas as pd
```

2. CÁC CẤU TRÚC DỮ LIỆU TRONG PANDAS

2.1.- Series

- *Series* là một đối tượng giống như mảng một chiều chứa một chuỗi các giá trị (có kiểu tương tự như kiểu *NumPy*) và một mảng nhãn dữ liệu (*data labels*) liên quan, được gọi là chỉ mục của nó.
- Biểu diễn chuỗi của *Series* được hiển thị tương tác hiển thị chỉ mục ở bên trái và các giá trị ở bên phải.

```
In [3]: obj = pd.Series([4, 7, -5, 9])
In [4]: obj
Out[4]: 0    4
        1    7
        2   -5
        3    9
        dtype: int64
```

2.1.- Series

- Chỉ mục trong *Series*:

- Mặc định sẽ được đánh số từ 0 đến N-1 (với N là độ dài của dữ liệu). Có thể lấy đối tượng chỉ mục và biểu diễn mảng của *Series* thông qua các giá trị và thuộc tính index:

```
In [5]: obj.values
Out[5]: array([ 4,  7, -5,  9])
In [6]: obj.index # like range(4)
Out[6]: RangeIndex(start=0, stop=4, step=1)
```

- Thông thường, nên tạo một *Series* có chỉ mục xác định từng điểm dữ liệu bằng nhãn:

```
In [7]: obj2=pd.Series([4,7,-5,9], index=['d','b','a','c'])
In [8]: obj2
Out[8]: d    4
        b    7
        a   -5
        c    9
        dtype: int64
In [9]: obj2.index
Out[9]: Index(['d', 'b', 'a', 'c'], dtype='object')
```

2.1.- Series

- **Chỉ mục trong *Series*:**
  - So với mảng *NumPy*, có thể sử dụng nhãn trong chỉ mục khi chọn các giá trị đơn lẻ hoặc một tập hợp các giá trị:

```
In [10]: obj2['a']
Out[10]: -5
In [11]: obj2['d'] = 6
In [12]: obj2[['c', 'a', 'd']]
Out[12]: c   9
         a  -5
         d   6
         dtype: int64
```

Ở đây `['c', 'a', 'd']` được hiểu là danh sách các chỉ mục, mặc dù nó chứa các chuỗi thay vì số nguyên.

- **Tính toán và sử dụng hàm trong *Series*:**
  - Được thực hiện dựa trên sẽ duy trì liên kết chỉ số-giá trị (*index-value*), chẳng hạn như:
    - Lọc bằng mảng Boolean
    - Phép nhân vô hướng
    - Áp dụng các hàm toán học

```
In [13]: obj2[obj2 > 0]
Out[13]:
     d   6
     b   7
     c   9
     dtype: int64
In [14]: obj2 * 2
Out[14]: d     12
         b     14
         a    -10
         c     6
         dtype: int64
In [15]: np.sum(obj2)
Out[16]: 15
```

2.1.- Series

- *Series và ordered dict :*
  - Có thể xem *Series* là dưới dạng một *ordered dict* có độ dài cố định, vì nó là ánh xạ các giá trị chỉ mục (*key*) tới các giá trị dữ liệu (*value*)

```
In [17]: 'b' in obj2
Out[17]: True
In [18]: 'e' in obj2
Out[18]: False
```

- **Tạo *Series* từ *iterator object (list, dictionary, ...)* của *Python*:**

```
In [19]: mydict = {'Ohio': 35000, 'Texas': 71000,
                   'Oregon': 16000, 'Utah': 5000}
In [19]: myseries1 = pd.Series(mydict)
In [20]: myseries1
Out[20]: Ohio     35000
         Oregon   16000
         Texas    71000
         Utah     5000
         dtype: int64
In [21]: mylist=list(range(2,14,3))
In [22]: myseries2 = pd.Series(mylist)
In [23]: myseries2
Out[23]: 0  2
         1  5
         2  8
         3  11
         dtype: int64
```

2.1.- Series

- **Tạo *Series* từ *iterator object (list, dictionary, ...)* của *Python*:**
  - Tham số thứ 1 của hàm *Series* là đối tượng cần chuyển thành *Series* (X), tham số thứ 2 (Y) là tùy chọn cho biết cách đánh chỉ mục mới sẽ lấy theo danh sách được chỉ định trong Y.
  - Do đó, đối tượng *Series* vừa tạo ra sẽ:
    - Nếu item trong X nhưng key không tồn tại trong Y, thì item này sẽ bị loại bỏ (như *Utah* trong VD minh họa)
    - Những key có trong Y nếu không tồn tại trong X thì value sẽ nhận giá trị là *NaN*(như *California* trong VD minh họa)

```
In [24]: mydict = {'Ohio': 35000, 'Texas': 71000,
                                   'Oregon': 16000, 'Utah': 5000}
In [25]: myseries1 = pd.Series(mydict)
In [26]: myseries1
Out[26]: Ohio      35000
         Texas     71000
         Oregon    16000
         Utah       5000
         dtype: int64
In [27]: mylist = ['Texas', 'Ohio', 'California', 'Oregon']
In [28]: myseries2 = pd.Series(mydict, index=mylist)
In [29]: myseries2
Out[29]: Texas       71000.0
         Ohio        35000.0
         California      NaN
         Oregon      16000.0
         dtype: float64
```

2.1.- Series

- **Tạo *Series* từ *iterator object (list, dictionary, ...)* của *Python*:**
  - Nên sử dụng các hàm *isnull* và *notnull* trong *pandas* để phát hiện dữ liệu dạng này:

```
In [30]: pd.isnull(myseries2)
Out[30]: California    True
         Ohio          False
         Oregon        False
         Texas         False
         dtype: bool
In [31]: pd.notnull(myseries2)
Out[31]: California False
         Ohio True
         Oregon True
         Texas True
         dtype: bool
```

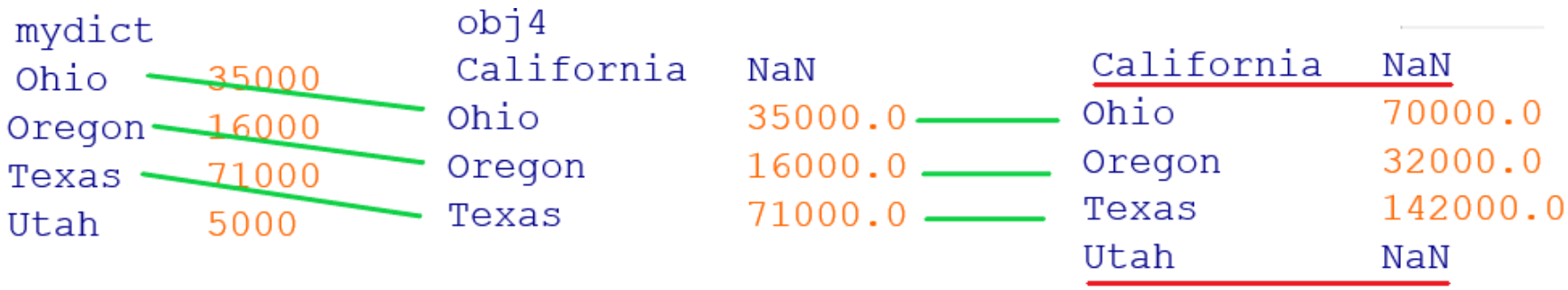
  - Bản thân các đối tượng *Series* cũng có những phương thức này *isnull* và *notnull*:

```
In [32]: myseries2.isnull()
Out[32]: California    True
         Ohio          False
         Oregon        False
         Texas         False
         dtype: bool
```

2.1.- Series

- ***Series và phép tính số học*** *(tương tự Join operation trong CSDL)*

```
In [33]: mydict
Out[33]: Ohio     35000
         Oregon   16000
         Texas    71000
         Utah     5000
         dtype:   int64
In [34]: obj4
Out[34]: California   NaN
         Ohio         35000.0
         Oregon       16000.0
         Texas        71000.0
         dtype:       float64
In [35]: obj3 + obj4
Out[35]: California   NaN
         Ohio         70000.0
         Oregon       32000.0
         Texas        142000.0
         Utah         NaN
         dtype:       float64
```



2.1.- Series

- Thuộc tính *name* của đối tượng *Series* và chỉ mục của nó

```
In [36]: myseries1.name = 'population'
In [37]: myseries1.name
Out[37]: 'population'
In [38]: myseries1.index.name = 'state'
In [39]: myseries1.index.name
Out[39]: 'state'
In [40]: myseries1
Out[40]: state
         California        NaN
         Ohio          35000.0
         Oregon        16000.0
         Texas         71000.0
         Name: population, dtype: float64
```

- Chỉ mục của *Series* có thể được thay đổi tại chỗ bằng cách gán:

```
In [41]: obj = pd.Series([4, 7, -5, 9])
In [42]: obj
Out[42]: 0   4
         1   7
         2   -5
         3   9
       dtype: int64
In [43]: obj.index = ['Bob', 'Steve', 'Jeff', 'Ryan']
In [44]: obj
Out[44]: Bob    4
         Steve  7
         Jeff   -5
         Ryan   3
         dtype: int64
```

2.2.- DataFrame

2.2.1.- Giới thiệu

- *DataFrame* biểu thị một bảng dữ liệu hình chữ nhật và chứa một tập hợp các cột được sắp xếp, mỗi cột có thể là một loại giá trị khác nhau (số, chuỗi, boolean, v.v.).
- *DataFrame* có cả chỉ mục cho hàng và cột; nó có thể được coi như một dictionary của *Series* (dict of *Series*), tất cả đều có chung một chỉ mục.

2.2.- DataFrame

2.2.2. Tạo DataFrame

- Có nhiều cách để xây dựng *DataFrame*, mặc dù một trong những cách phổ biến nhất là từ một *dict* gồm các *list* có độ dài bằng nhau hoặc *NumPy arrays*:

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002, 2003],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9, 3.2]}
frame = pd.DataFrame(data)

        In [45]: frame
        Out[45]:
                pop      state   year
             0  1.5       Ohio   2000
             1  1.7       Ohio   2001
             2  3.6       Ohio   2002
             3  2.4     Nevada   2001
             4  2.9     Nevada   2002
             5  3.2     Nevada   2003
```

2.2.- DataFrame

2.2.2. Tạo DataFrame

- Tạo DataFrame từ một *dict* lồng nhau của các *dict*. Khi đó, các *keys* của *dict* bên ngoài là các cột và các *keys* bên trong là chỉ mục hàng:

```
In [65]: pop = {'Nevada': {2001: 2.4, 2002: 2.9},
                'Ohio': {2000: 1.5, 2001: 1.7, 2002: 3.6}}
```

```
In [66]: frame3 = pd.DataFrame(pop)
In [67]: frame3
Out[67]:
      Nevada  Ohio
2000     NaN   1.5
2001     2.4   1.7
2002     2.9   3.6
```

- Các keys bên trong của dict được kết hợp và sắp xếp để tạo thành chỉ mục trong kết quả. Điều này sẽ không đúng nếu một chỉ mục rõ ràng được chỉ định:

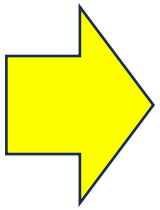
```
In [69]: pd.DataFrame(pop, index=[2001, 2002, 2003])
Out[69]:
      Nevada  Ohio
2001     2.4   1.7
2002     2.9   3.6
2003     NaN   NaN
```

2.2.- DataFrame

2.2.2. Tạo DataFrame

- *Dicts* của các *Series* cũng được xử lý theo cách tương tự:

```
frame3
      Nevada  Ohio
2000     NaN   1.5
2001     2.4   1.7
2002     2.9   3.6
```



```
In [70]: pdata = {  'Ohio': frame3['Ohio'][:-1],
                    'Nevada': frame3['Nevada'][:2]}
In [71]: pd.DataFrame(pdata)
Out[71]:
           Nevada  Ohio
     2000     NaN   1.5
     2001     2.4   1.7
```

- Nếu chỉ mục và cột của *DataFrame* được đặt thuộc tính tên thì chúng cũng sẽ được hiển thị:

```
In [72]: frame3.index.name = 'year';
         frame3.columns.name = 'state'
In [73]: frame3
Out[73]:
     state  Nevada  Ohio
     year
     2000      NaN   1.5
     2001      2.4   1.7
     2002      2.9   3.6
```

2.2.- DataFrame

2.2.2. Tạo DataFrame

- Danh sách những kiểu dữ liệu có thể truyền vào hàm tạo *DataFrame*:

Kiểu dữ liệu	Diễn giải
2D ndarray	Ma trận dữ liệu, tùy chọn việc chuyển nhãn của hàng và cột
dict of arrays, lists, or tuples	Mỗi chuỗi trở thành một cột trong *DataFrame*; tất cả các chuỗi phải có cùng độ dài
NumPy structured/ record array	Được coi là trường hợp *"dict of arrays"*
dict of Series	Mỗi giá trị sẽ trở thành một cột; các chỉ mục từ mỗi *Series* được kết hợp với nhau để tạo thành chỉ mục hàng của kết quả (nếu không có chỉ định rõ ràng chỉ mục).
dict of dicts	Mỗi lệnh bên trong sẽ trở thành một cột; các *keys* được liên kết để tạo thành chỉ mục hàng như trong trường hợp *"dict of Series"*
List of dicts or Series	Mỗi mục (*item*) sẽ trở thành một hàng trong *DataFrame*; sự kết hợp của các *keys* của *dict* hoặc chỉ mục của *Series* trở thành nhãn cột của *DataFrame*
List of lists or tuples	Được xử lý như trường hợp *"2D ndarray"*
Another DataFrame	Các chỉ mục của *DataFrame* được sử dụng trừ khi có chỉ định rõ ràng về chỉ mục
NumPy MaskedArray	Giống như trường hợp *"2D ndarray"* ngoại trừ các giá trị bị che sẽ trở thành *NA/missing* trong kết quả *DataFrame*

2.2.3. *Sử dụng*

- Đối với các *DataFrame* lớn, phương thức *head* chỉ chọn năm hàng đầu tiên:

- Nếu chỉ định một chuỗi các cột thì các cột của *DataFrame* sẽ được sắp xếp theo thứ tự liệt kê đó:

- Nếu cột không có trong *dict*, nó sẽ xuất hiện với các giá trị là NAN trong kết quả:

```
In [46]: frame.head()
Out[46]:
    state  year  pop
0    Ohio  2000  1.5
1    Ohio  2001  1.7
2    Ohio  2002  3.6
3  Nevada  2001  2.4
4  Nevada  2002  2.9
```

```
In [47]: pd.DataFrame(data,
         columns=['year', 'state', 'pop'])
Out[47]:
   year   state  pop
0  2000    Ohio  1.5
1  2001    Ohio  1.7
2  2002    Ohio  3.6
3  2001  Nevada  2.4
4  2002  Nevada  2.9
5  2003  Nevada  3.2
```

```
In [48]: frame2 = pd.DataFrame(data,
         columns=['year','state','pop','debt'],
index=['one','two','three','four','five','six'])
In [49]: frame2
Out[49]:
       year   state  pop debt
one    2000    Ohio  1.5  NaN
two    2001    Ohio  1.7  NaN
three  2002    Ohio  3.6  NaN
four   2001  Nevada  2.4  NaN
five   2002  Nevada  2.9  NaN
six    2003  Nevada  3.2  NaN
```

2.2.3. Sử dụng

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

- Sử dụng thuộc tính columns để xem tên cột có trong DataFrame

```
In [50]: frame2.columns
Out[50]: Index(['year', 'state', 'pop', 'debt']
                                        , dtype='object')
```

- Một cột trong *DataFrame* có thể được truy xuất dưới dạng *Series* bằng ký hiệu giống như ký pháp của *dict* hoặc theo thuộc tính:

  Ghi chú: *dataFrame_Name[column_Name]* (như *frame2[column]*) hoạt động với bất kỳ tên cột nào, nhưng *dataFrame_Name.column_Name* chỉ hoạt động khi tên cột là tên biến *Python* hợp lệ.

```
In [51]: frame2['state']
Out[51]:
one      Ohio
two      Ohio
three    Ohio
four   Nevada
five   Nevada
six    Nevada
Name: state, dtype: object
In [52]: frame2.year
Out[52]:
one      2000
two      2001
three    2002
four     2001
five     2002
six      2003
Name: year, dtype: int64
```

2.2.- DataFrame

2.2.3. Sử dụng

- Lưu ý:
  - *Series* được trả về có cùng chỉ mục với *DataFrame* và thuộc tính tên của chúng đã được đặt.
  - Các hàng cũng có thể được truy xuất theo vị trí hoặc tên bằng thuộc tính *loc*:

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

```
In [53]: frame2.loc['three']
Out[53]:
    year   2002
    state  Ohio
    pop     3.6
    debt    NaN
    Name: three, dtype: object
```

- Có thể sửa đổi giá trị trên từng cột

```
In [54]: frame2['debt'] = 16.5
In [55]: frame2
Out[55]:
```

	year	state	pop	debt
one	2000	Ohio	1.5	16.5
two	2001	Ohio	1.7	16.5
three	2002	Ohio	3.6	16.5
four	2001	Nevada	2.4	16.5
five	2002	Nevada	2.9	16.5
six	2003	Nevada	3.2	16.5

```
In [56]: frame2['debt'] = np.arange(6.)
In [57]: frame2
Out[57]:
```

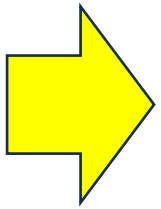
	year	state	pop	debt
one	2000	Ohio	1.5	0.0
two	2001	Ohio	1.7	1.0
three	2002	Ohio	3.6	2.0
four	2001	Nevada	2.4	3.0
five	2002	Nevada	2.9	4.0
six	2003	Nevada	3.2	5.0

2.2.- DataFrame

2.2.3. Sử dụng

- Thuộc tính values:
  - Giống như *Series*, thuộc tính *values* trả về dữ liệu có trong *DataFrame* dưới dạng *ndarray* hai chiều.

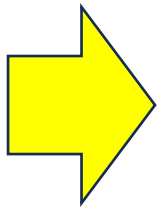
```
frame3
      Nevada  Ohio
2000     NaN   1.5
2001     2.4   1.7
2002     2.9   3.6
```



```
In [74]: frame3.values
Out[74]: array([[ nan, 1.5],
                [ 2.4, 1.7],
                [ 2.9, 3.6]])
```

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

```
frame2
       Year   state  pop  debt
one    2000    Ohio  1.5   NaN
two    2001    Ohio  1.7  -1.2
three  2002    Ohio  3.6   NaN
four   2001  Nevada  2.4  -1.5
five   2002  Nevada  2.9  -1.7
six    2003  Nevada  3.2   NaN
```



```
In [75]: frame2.values
Out[75]: array([ [2000, 'Ohio', 1.5, nan],
                 [2001, 'Ohio', 1.7, -1.2],
                 [2002, 'Ohio', 3.6, nan],
                 [2001, 'Nevada', 2.4, -1.5],
                 [2002, 'Nevada', 2.9, -1.7],
                 [2003, 'Nevada', 3.2, nan]], dtype=object)
```

2.2.- DataFrame

2.2.3. Sử dụng

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

- Gán giá trị cho cột:

```
In [54]: frame2['debt'] = 16.5
In [55]: frame2
Out[55]:
        year   state  pop  debt
one     2000    Ohio  1.5  16.5
two     2001    Ohio  1.7  16.5
three   2002    Ohio  3.6  16.5
four    2001  Nevada  2.4  16.5
five    2002  Nevada  2.9  16.5
six     2003  Nevada  3.2  16.5
```

```
In [56]: frame2['debt'] = np.arange(6.)
In [57]: frame2
Out[57]:
        year   state  pop  debt
one     2000    Ohio  1.5   0.0
two     2001    Ohio  1.7   1.0
three   2002    Ohio  3.6   2.0
four    2001  Nevada  2.4   3.0
five    2002  Nevada  2.9   4.0
six     2003  Nevada  3.2   5.0
```

- Khi gán *list* hoặc *array* cho một cột, độ dài của giá trị phải khớp với độ dài của *DataFrame*.

2.2.3. Sử dụng

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

- Khi gán một *Series*, các nhãn của nó sẽ được so sánh (tương tự JOIN operation trong CSDL) lại chính xác theo chỉ mục của *DataFrame*, và chèn các giá trị *NaN* vào bất kỳ chỉ mục nào không khớp.

```
In [58]: val = pd.Series([-1.2,-1.5,-1.7],
                         index=['two','four','five'])
In [59]: frame2['debt'] = val
In [60]: frame2
Out[60]:
          Year    state      pop      debt
one       2000     Ohio      1.5       NaN
two       2001     Ohio      1.7      -1.2
three     2002     Ohio      3.6       NaN
four      2001   Nevada      2.4      -1.5
five      2002   Nevada      2.9      -1.7
six       2003   Nevada      3.2       NaN
```

- Việc gán một cột không tồn tại sẽ tạo ra một cột mới.
- Từ khóa *del* được dùng sẽ xóa các cột như với một *dict*.

```
In [61]: frame2['eastern'] = frame2.state == 'Ohio'
In [62]: frame2
Out[62]:
       Year    state    pop   debt  eastern
one    2000     Ohio    1.5    NaN     True
two    2001     Ohio    1.7   -1.2     True
three  2002     Ohio    3.6    NaN     True
four   2001   Nevada    2.4   -1.5    False
five   2002   Nevada    2.9   -1.7    False
six    2003   Nevada    3.2    NaN    False
```

2.2.3. Sử dụng

- Có thể chuyển đổi *DataFrame* (hoán đổi hàng và cột) với cú pháp tương tự như mảng NumPy

	year	state	pop	debt
one	2000	Ohio	1.5	NaN
two	2001	Ohio	1.7	NaN
three	2002	Ohio	3.6	NaN
four	2001	Nevada	2.4	NaN
five	2002	Nevada	2.9	NaN
six	2003	Nevada	3.2	NaN

```
In [68]: frame3.T
Out[68]:
        2000  2001  2002
Nevada   NaN   2.4   2.9
Ohio     1.5   1.7   3.6
```

2.3.- Index object

2.3.1.- Giới thiệu

- Các đối tượng chỉ mục (*Index*) của *pandas* chịu trách nhiệm giữ nhãn của các trục (*axis labels*) và dữ liệu khác (như tên của index hoặc tên trục).
- Bất kỳ *array* hoặc danh sách nhãn nào khác được sử dụng khi xây dựng *Series* hoặc *DataFrame* đều được chuyển đổi nội bộ thành chỉ mục:

```
In [76]: obj = pd.Series(range(3), index=['a', 'b', 'c'])
In [77]: myIndex = obj.index
In [78]: myIndex
Out[78]: Index(['a', 'b', 'c'], dtype='object')
In [79]: myIndex[1:]
Out[79]: Index(['b', 'c'], dtype='object')
```

2.3.- Index object

2.3.2.- Một số đặc điểm của chỉ mục

- Một số người dùng thường không tận dụng được các khả năng do chỉ mục cung cấp, nhưng vì một số thao tác sẽ mang lại kết quả chứa dữ liệu được lập chỉ mục nên điều quan trọng là phải hiểu cách chúng hoạt động.
- Các đối tượng chỉ mục là bất biến và do đó người dùng không thể sửa đổi:

```
index[1] = 'd' # TypeError
```

- Tính bất biến giúp việc chia sẻ các đối tượng *Index* giữa các cấu trúc dữ liệu trở nên an toàn hơn:

```
In [80]: labels = pd.Index(np.arange(3))
In [81]: labels
Out[81]: Int64Index([0, 1, 2], dtype='int64')
In [82]: obj2 = pd.Series([1.5, -2.5, 0], index=labels)
In [83]: obj2
Out[83]:    0     1.5
            1    -2.5
            2     0.0
            dtype: float64
In [84]: obj2.index is labels
Out[84]: True
```

2.3.- Index object

2.3.2.- Một số đặc điểm của chỉ mục

- Ngoài việc giống như *array*, *Index* còn hoạt động giống như một *set* có kích thước cố định:

```
In [85]: frame3
Out[85]: state   Nevada   Ohio
          year
          2000      NaN    1.5
          2001      2.4    1.7
          2002      2.9    3.6
In [86]: frame3.columns
Out[86]: Index(['Nevada', 'Ohio'], dtype='object', name='state')
In [87]: 'Ohio' in frame3.columns
Out[87]: True
In [88]: 2003 in frame3.index
Out[88]: False
```

- *index* trong *pandas* có thể chứa các nhãn trùng lặp:

```
In [89]: dup_labels = pd.Index(['foo', 'foo', 'bar', 'bar'])
In [90]: dup_labels
Out[90]: Index(['foo', 'foo', 'bar', 'bar'], dtype='object')
```

Các lựa chọn có nhãn trùng lặp sẽ chọn tất cả các lần xuất hiện của nhãn đó.

2.3.- Index object

2.3.3.- *Một số phương thức và thuộc tính của index*

Method	*Diễn giải*
append	Kết hợp các Index hiện có với các đối tượng Index bổ sung, tạo ra một Index mới
difference	Tính toán sự khác biệt tập hợp dưới dạng chỉ mục
intersection	Tính toán phần chung (giao) của tập hợp
union	Tính toán phần hội (hợp) của tập hợp
isin	Tính toán mảng boolean cho biết liệu mỗi giá trị có được chứa trong bộ danh sách (collection) đã truyền hay không?
delete	Tính Index mới với phần tử tại chỉ mục thứ i đã bị xóa
drop	Tính Index mới bằng cách xóa các giá trị đã được truyền
insert	Tính Index mới bằng cách chèn phần tử vào chỉ mục thứ i
is_monotonic	Trả về True nếu mỗi phần tử lớn hơn hoặc bằng phần tử trước đó
is_unique	Trả về True nếu Index không có giá trị trùng lặp
unique	Tính toán mảng các giá trị duy nhất trong Index

3. MỘT SỐ CHỨC NĂNG THIẾT YẾU

3.1.- Reindexing

- Một phương thức quan trọng trên các đối tượng *pandas* là *reindex*, có nghĩa là tạo một đối tượng mới với dữ liệu tuân theo một chỉ mục mới.

  Gọi *reindex* trên *Series* sẽ sắp xếp lại dữ liệu theo chỉ mục mới, đưa ra các giá trị bị thiếu nếu bất kỳ giá trị chỉ mục nào chưa có:

```
In [91]: obj = pd.Series([4.5, 7.2, -5.3, 3.6],
                         index=['d', 'b', 'a', 'c'])
In [92]: obj
Out[92]: d  4.5
         b    7.2
         a   -5.3
         c    3.6
         dtype: float64
In [93]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
In [94]: obj2
Out[94]: a   -5.3
         b    7.2
         c    3.6
         d    4.5
         e    NaN
         dtype: float64
```

3.1.- Reindexing

- **Đối với *Series***(dữ liệu được sắp xếp)*:* có thể nên thực hiện một số phép nội suy hoặc điền các giá trị khi lập chỉ mục lại. Tùy chọn *method* của hàm này cho phép thực hiện việc này bằng cách đạt tùy chọn *method = ffill*, phương thức này sẽ điền các giá trị của những index đã có cho những index liền ngay sau nhưng không có trong danh sách các index cũ:

```
In [95]: obj3 = pd.Series(['blue', 'purple', 'yellow'],
                                              index=[0, 2, 4])
In [96]: obj3
Out[96]: 0        blue
         2      purple
         4      yellow
         dtype: object
In [97]: obj3.reindex(range(8), method='ffill')
Out[97]: 0        blue
         1        blue
         2      purple
         3      purple
         4      yellow
         5      yellow
         6      yellow
         7      yellow
         dtype: object
```

3.1.- Reindexing

- **Đối với** *DataFrame*: *reindex* có thể thay đổi chỉ mục (hàng), cột hoặc cả hai. Khi chỉ truyền một danh sách (*sequence*), *reindex* sẽ lập chỉ mục lại các hàng trong kết quả

```
In [98]: frame = pd.DataFrame(np.arange(9).reshape((3, 3)),
                              index=['a', 'c', 'd'],
                              columns=['Ohio', 'Texas', 'California'])
In [99]: frame
Out[99]:
   Ohio  Texas  California
a     0      1           2
c     3      4           5
d     6      7           8
In [100]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])
In [101]: frame2
Out[101]:
   Ohio  Texas  California
a   0.0    1.0         2.0
b   NaN    NaN         NaN
c   3.0    4.0         5.0
d   6.0    7.0         8.0
```

- Các cột có thể được lập chỉ mục lại bằng từ khóa *columns*:

```
In [104]: frame.loc[['a', 'b', 'c', 'd'], states]
Out[104]:
   Texas  Utah  California
a    1.0   NaN         2.0
b    NaN   NaN         NaN
c    4.0   NaN         5.0
d    7.0   NaN         8.0
```

3.1.- Reindexing

- Khi khám phá chi tiết hơn, có thể lập chỉ mục lại ngắn gọn hơn bằng cách lập chỉ mục nhãn (*label-indexing*) bằng *loc*

```
In [104]: frame.loc[['a', 'b', 'c', 'd'], states]
Out[104]:
   Texas  Utah  California
a    1.0   NaN         2.0
b    NaN   NaN         NaN
c    4.0   NaN         5.0
d    7.0   NaN         8.0
```

3.1.- Reindexing

- Các đối số của *reindex*

Argument	Description
`index`	Sử dụng sequence mới làm chỉ mục. Có thể là phiên bản *Index* hoặc bất kỳ cấu trúc dữ liệu *Python* dạng danh sách (*sequence-like*) nào khác. Chỉ mục sẽ được sử dụng chính xác mà không cần sao chép.
`method`	Phương thức nội suy (*fill*); *'ffill'* sẽ thực hiện "lấp đầy" về phía trước, trong khi *'bfill'* sẽ thực hiện "lấp đầy" phía sau.
`fill_value`	Giá trị thay thế sẽ được sử dụng khi dữ liệu bị thiếu (missing) khi lập lại chỉ mục.
`limit`	Khi thực hiện "lấp đầy" (về phía trước hoặc phía sau), đối số này cho biết kích thước tối đa (về số phần tử) cần lấp đầy.
`tolerance`	Khi thực hiện "lấp đầy" (về phía trước hoặc phía sau), đối số này cho biết khoảng cách kích thước tối đa (trong khoảng cách số tuyệt đối) sẽ lấp đầy cho các kết quả khớp không chính xác.
`level`	Ghép chỉ mục đơn giản theo cấp độ của *MultiIndex*; nếu không hãy chọn tập hợp con của chúng.
`copy`	Nếu là *True*, luôn sao chép dữ liệu cơ bản ngay cả khi chỉ mục mới tương đương với chỉ mục cũ; ngược lại, nếu là *False*, không sao chép dữ liệu khi các chỉ mục tương đương.

3.2.- Loại bỏ một hoặc nhiều mục từ một trục

- Phương thức *drop* sẽ trả về một đối tượng mới với giá trị được chỉ định hoặc các giá trị bị xóa khỏi một trục

- **Đối với *Series***:

```
In [105]: obj = pd.Series(np.arange(5.),
          index=['a', 'b', 'c', 'd', 'e'])
In [106]: obj
Out[106]:
        a    0.0
        b    1.0
        c    2.0
        d    3.0
        e    4.0
        dtype: float64
In [107]: new_obj = obj.drop('c')
In [108]: new_obj
Out[108]:
        a    0.0
        b    1.0
        d    3.0
        e    4.0
        dtype: float64
```

```
In [109]: """tạo ra đối tượng mới từ obj
     sau khi đã xóa 'c' và 'd', do đó obj không
     bị ảnh hưởng bởi việc xóa"""
          obj.drop(['d', 'c'])
Out[109]:
        a    0.0
        b    1.0
        e    4.0
        dtype: float64
In [110]: obj """không bị thay đổi bởi phương
thức drop trước đó"""
Out[110]:
        a    0.0
        b    1.0
        c    2.0
        d    3.0
        e    4.0
        dtype: float64
```

3.2.- Loại bỏ một hoặc nhiều mục từ một trục

- **Đối với *DataFrame*:**
  - Sử dụng *drop* để xóa các giá trị chỉ mục khỏi một trong hai trục:
    - Đối với các hàng (*axis=0* – giá trị mặc định): cung cấp một danh sách các nhãn (sequence of labels) sẽ bị được chỉ ra trong danh sách:
    - Đối với các cột: chuyển thêm đối số *axis=1* hoặc *axis='columns'*:

```
In [112]: data.drop(['Colorado', 'Ohio'])
Out[112]:
          one  two  three  four
Utah        8    9     10    11
New York   12   13     14    15
In [113]: data.drop('two', axis=1)
Out[113]:
          one  three  four
Ohio        0      2     3
Colorado    4      6     7
Utah        8     10    11
New York   12     14    15
```

```
In [112]: data.drop(['Colorado', 'Ohio'])
Out[112]:
          one  two  three  four
Utah        8    9     10    11
New York   12   13     14    15
In [114]: data.drop(['two', 'four'], axis='columns')
Out[114]:
          one  four
Ohio        0     3
Colorado    4     7
Utah        8    11
New York   12    15
```

3.2.- Loại bỏ một hoặc nhiều mục từ một trục

- **Đối với *DataFrame*:**
  - Sử dụng *drop* để xóa các giá trị chỉ mục khỏi một trong hai trục:
    - Đối với các hàng (*axis=0* – giá trị mặc định): cung cấp một danh sách các nhãn (sequence of labels) sẽ bị được chỉ ra trong danh sách:
    - Đối với các cột: chuyển thêm đối số *axis=1* hoặc *axis='columns'*:

```
In [112]: data.drop(['Colorado', 'Ohio'])
Out[112]:
          one  two  three  four
Utah        8    9     10    11
New York   12   13     14    15
In [113]: data.drop('two', axis=1)
Out[113]:
          one  three  four
Ohio        0      2     3
Colorado    4      6     7
Utah        8     10    11
New York   12     14    15
```

```
In [112]: data.drop(['Colorado', 'Ohio'])
Out[112]:
          one  two  three  four
Utah        8    9     10    11
New York   12   13     14    15
In [114]: data.drop(['two', 'four'], axis='columns')
Out[114]:
          one  four
Ohio        0     3
Colorado    4     7
Utah        8    11
New York   12    15
```

3.2.- Loại bỏ một hoặc nhiều mục từ một trục

- **Đối với *DataFrame*:**
  - Nhiều hàm, chẳng hạn như *drop*, sửa đổi kích thước hoặc hình dạng của *Series* hoặc *DataFrame*, có thể thao tác trực tiếp trên một đối tượng mà không trả về đối tượng mới. Do đó cần cẩn thận khi sử dụng:

```
In [105]: obj = pd.Series(np.arange(5.),
                          index=['a', 'b', 'c', 'd', 'e'])
In [106]: obj
Out[106]:
          a   0.0
          b   1.0
          c   2.0
          d   3.0
          e   4.0
          dtype: float64
In [115]: obj.drop('c', inplace=True)
In [116]: obj
Out[116]:  a   0.0
           b   1.0
           d   3.0
           e   4.0
           dtype: float64
```

3.3.- Indexing, Selection, và Filtering

3.3.1.- Indexing, Selection, và Filtering

- **Đối với *Series*:**
  - Lập chỉ mục trên *Series* hoạt động tương tự như lập chỉ mục *array* của *NumPy*, ngoại trừ việc có thể sử dụng các giá trị chỉ mục của *Series* thay vì chỉ số nguyên.

```
In [117]: obj = pd.Series(np.arange(4.),
                          index=['a', 'b', 'c', 'd'])
In [118]: obj
Out[118]:  a 0.0
           b 1.0
           c 2.0
           d 3.0
           dtype: float64
In [119]: obj['b']
Out[120]: 1.0
In [121]: obj[2:4]
Out[121]:  c 2.0
           d 3.0
           dtype: float64
```

```
In [122]: obj[['b', 'a', 'd']]
Out[122]:  b  1.0
           a  0.0
           d  3.0
           dtype: float64
In [123]: obj[[1, 3]]
Out[123]:  b  1.0
           d  3.0
           dtype: float64
In [124]: obj[obj < 2]
Out[124]:  a 0.0
           b 1.0
           dtype: float64
```

3.3.- Indexing, Selection, và Filtering

3.3.1.- Indexing, Selection, và Filtering

- **Đối với *Series***:
  - Việc cắt bằng nhãn hoạt động khác với việc cắt bằng *Python* thông thường ở chỗ việc cắt bằng nhãn bao gồm luôn điểm cuối (endpoint):

```
In [117]: obj = pd.Series(np.arange(4.),
                          index=['a', 'b', 'c', 'd'])
In [118]: obj
Out[118]:  a 0.0
           b 1.0
           c 2.0
           d 3.0
           dtype: float64
# cắt các hàng từ 'b' đến 'c'
In [125]: obj['b':'c']
Out[125]:  b 1.0
           c 2.0
           dtype: float64
# Gán giá trị 5 cho các hàng từ 'b' đến 'c'
In [126]: obj['b':'c'] = 5
In [127]: obj
Out[127]:  a 0.0
           b 5.0
           c 5.0
           d 3.0
           dtype: float64
```

3.3.- Indexing, Selection, và Filtering

3.3.1.- Indexing, Selection, và Filtering

- **Đối với *DataFrame*:**
  - Lập chỉ mục trên *DataFrame* là để truy xuất một hoặc nhiều cột với một giá trị hoặc chuỗi (sequence) duy nhất:

```
In [128 data = pd.DataFrame(np.arange(16).reshape((4, 4)),
           index=['Ohio', 'Colorado', 'Utah', 'New York'],
           columns=['one', 'two', 'three', 'four'])
In [129]: data
Out[129]:
                one   two  three   four
Ohio              0     1      2      3
Colorado          4     5      6      7
Utah              8     9     10     11
New York         12    13     14     15
In [130]: data['two']
Out[130]:
                two
Ohio              1
Colorado          5
Utah              9
New York         13
Name: two, dtype: int64
In [131]: data[['three', 'one']]
Out[131]:
                three   one
Ohio                2     0
Colorado            6     4
Utah               10     8
New York           14    12
```

3.3.- Indexing, Selection, và Filtering

3.3.1.- Indexing, Selection, và Filtering

- **Đối với *DataFrame*:**

  - Lập chỉ mục trên *DataFrame* là để truy xuất một hoặc nhiều cột với một giá trị hoặc chuỗi (sequence) duy nhất:

```
In [129]: data
Out[129]:
          one   two  three  four
Ohio        0     1      2     3
Colorado    4     5      6     7
Utah        8     9     10    11
New York   12    13     14    15
```

- Cắt (*slice*) hoặc chọn (*selection*) 2 hàng đầu của bảng dữ liệu:

```
In [132]: data[:2]
Out[132]:
          one   two  three  four
Ohio        0     1      2     3
Colorado    4     5      6     7
```

- Cắt (*slice*) hoặc chọn (*selection*) dữ liệu bằng so sánh dựa trên giá trị của cột:

```
In [133]: data[data['three'] > 5]
Out[133]:
          one   two  three  four
Colorado    4     5      6     7
Utah        8     9     10    11
New York   12    13     14    15
```

3.3.- Indexing, Selection, và Filtering

3.3.1.- Indexing, Selection, và Filtering

- **Đối với *DataFrame*:**

  - Tạo *DataFrame* boolean, bằng so sánh vô hướng:

  - Thay đổi giá trị trong *DataFrame*

```
In [128 data = pd.DataFrame(np.arange(16).reshape((4, 4)),
           index=['Ohio', 'Colorado', 'Utah', 'New York'],
           columns=['one', 'two', 'three', 'four'])
In [129]: data
Out[129]:
                one    two  three   four
Ohio              0      1      2      3
Colorado          4      5      6      7
Utah              8      9     10     11
New York         12     13     14     15
'''So sánh từng giá trị trong data với 5, sẽ cho DataFrame
boolean'''
In [134] data < 5
Out[134]:
                 one    two  three   four
Ohio            True   True   True   True
Colorado        True  False  False  False
Utah           False  False  False  False
New York       False  False  False  False
'''Những giá trị trong data <5 sẽ được gán bằng 0'''
In [135]: data[data < 5] = 0
In [136]: data
Out[136]:
                one    two  three   four
Ohio              0      0      0      0
Colorado          0      5      6      7
Utah              8      9     10     11
New York         12     13     14     15
```

3.3.- Indexing, Selection, và Filtering

3.3.2.- Selection with loc and iloc

- Để lập chỉ mục nhãn *DataFrame* trên các hàng, các toán tử lập chỉ mục đặc biệt là *loc* và *iloc*. Chúng cho phép chọn một tập hợp con các hàng và cột từ *DataFrame* với ký hiệu giống NumPy bằng cách sử dụng nhãn trục (*loc*) hoặc số nguyên (*iloc*)
- Ví dụ:

```
# Chọn dữ liệu của hàng thứ 2 bằng iloc:
In [139]: data.iloc[2]
Out[139]:
one    8
two    9
three 10
four  11
Name: Utah, dtype: int64
```

data

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
# Chọn dữ liệu trên 3 cột 3,0 và 1 của hàng thứ 2 bằng iloc:
In [138]: data.iloc[2, [3, 0, 1]]
Out[138]:
four  11
one    8
two    9
Name: Utah, dtype: int64
```

data

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

3.3.- Indexing, Selection, và Filtering

3.3.2.- Selection with loc and iloc

- Ví dụ:

```
# Chọn một hàng và nhiều cột theo nhãn:
 In [137]: data.loc['Colorado', ['two', 'three']]
 Out[137]:
   two    5
   three  6
   Name: Colorado, dtype: int64
```

data

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
""" loc và iloc đều hoạt động với các lát ngoài
các nhãn đơn lẻ hoặc danh sách nhãn.
VD: Lấy từ hàng đầu đến hết hàng Utah của cột two' """
    In [141]: data.loc[:'Utah', 'two']
    Out[141]:
      Ohio     1
      Colorado 5
      Utah     9
      Name: two, dtype: int64
```

data

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

3.3.- Indexing, Selection, và Filtering

3.3.2.- Selection with loc and iloc

- Ví dụ:

```
# Chọn tất cả các hàng từ cột đầu đến hết cột thứ 2
```

```
In [142]: data.iloc[:, :3]
Out[142]:
          one  two  three
Ohio        0    1      2
Colorado    4    5      6
Utah        8    9     10
New York   12   13     14
```

data

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
# sau khi chọn xong vùng dữ liệu, thực hiện lọc  những giá trị >5:
```

```
In [143]: data.iloc[:, :3][data.three > 5]
Out[142]:
          one  two  three
Colorado    4    5      6
Utah        8    9     10
New York   12   13     14
```

data

	one	two	three
~~Ohio~~	~~0~~	~~1~~	~~2~~
Colorado	4	5	6
Utah	8	9	10
New York	12	13	14

3.3.3.- Chỉ mục theo số nguyên (Integer Indexes)

- Làm việc với các đối tượng *pandas* được lập chỉ mục theo số nguyên là điều thường gây khó chịu cho người dùng mới do một số khác biệt với ngữ nghĩa lập chỉ mục trên các cấu trúc dữ liệu *Python* tích hợp như lists và tuples

```
# Lỗi
In [144]: ser = pd.Series(np.arange(3.));
. . .   : ser
Out[144]:
   0 0.0
   1 1.0
   2 2.0
   dtype: float64
In [145]: ser[-1]  # ValueError: -1 is not in range
# KHÔNG gây lỗi
In [146]: ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])
. . .   : # hoặc ser2 = pd.Series(np.arange(3.), index=[1, 2, 3])
. . .   : ser
Out[146]:
          0 0.0
          1 1.0
          2 2.0
          dtype: float64
In [147]: ser2[-1]
Out[147]: 2.0
```

3.3.3.- Chỉ mục theo số nguyên (Integer Indexes)

- Để xử lý chính xác hơn, hãy sử dụng *loc* (đối với nhãn) hoặc *iloc* (đối với số nguyên)

```
In [148]: ser[:1]
Out[148]:
  0 0.0
  dtype: float64
In [149]: ser.loc[:1]
Out[149]:
  0   0.0
  1   1.0
  dtype: float64
In [150]: ser.iloc[:1]
Out[150]:
  0   0.0
  dtype: float64
```

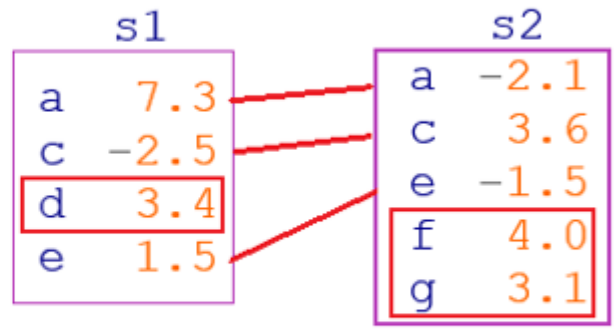
3.4.- Phép toán số học trên các đối tượng

- Khi cộng các đối tượng lại với nhau, nếu bất kỳ cặp chỉ mục nào không giống nhau thì chỉ mục tương ứng trong kết quả sẽ là sự kết hợp của các cặp chỉ mục. Đối với người dùng có trải nghiệm về cơ sở dữ liệu, điều này tương tự như phép nối ngoài (`outer join`) tự động trên nhãn chỉ mục.
- **Đối với *Series***

```
In [150]: s1 = pd.Series([7.3, -2.5, 3.4, 1.5],
                         index=['a', 'c', 'd', 'e'])
In [151]: s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
                         index=['a', 'c', 'e', 'f', 'g'])
In [152]: s1
Out[152]:
  a   7.3
  c  -2.5
  d   3.4
  e   1.5
  dtype: float64
In [153]: s2
Out[153]:
  a  -2.1
  c   3.6
  e  -1.5
  f   4.0
  g   3.1
  dtype: float64
```

Cộng 2 *Series* này lại với nhau sẽ mang lại kết quả:

```
In [154]: s1 + s2
Out[154]:
  a   5.2
  c   1.1
  d   NaN
  e   0.0
  f   NaN
  g   NaN
  dtype: float64
```



3.4.- Phép toán số học trên các đối tượng

- **Đối với** *DataFrame:* việc căn chỉnh được thực hiện trên cả hàng và cột

```
In [155]: df1 = pd.DataFrame(np.arange(9.).reshape((3, 3)),
 columns=list('bcd'), .....: index=['Ohio', 'Texas', 'Colorado'])
In [156]: df2 = pd.DataFrame(np.arange(12.).reshape((4, 3)),
  columns=list('bde'), index=['Utah', 'Ohio', 'Texas', 'Oregon'])
```

df1

	b	c	d
Ohio	0.0	1.0	2.0
Texas	3.0	4.0	5.0
Colorado	6.0	7.0	8.0

df2

	b	d	e
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0

```
In [159]: df1 + df2
Out[159]:
              b   c     d   e
Colorado    NaN NaN   NaN NaN
Ohio        3.0 NaN   6.0 NaN
Oregon      NaN NaN   NaN NaN
Texas       9.0 NaN  12.0 NaN
Utah        NaN NaN   NaN NaN
```

3.4.- Phép toán số học trên các đối tượng

- **Đối với *DataFrame:***
  - Nếu thêm các đối tượng *DataFrame* không có nhãn cột hoặc hàng chung, kết quả sẽ chứa tất cả các giá trị rỗng

```
In [160]: df1 = pd.DataFrame({'A': [1, 2]})
In [161]: df2 = pd.DataFrame({'B': [3, 4]})

In [164]: df1 - df2
Out[164]:
     A    B
0  NaN  NaN
1  NaN  NaN
```

	A
0	1
1	2

	B
0	3
1	4

3.4.- Phép toán số học trên các đối tượng

3.4.1. Sử dụng các phương thức số học để điền (fill) các giá trị với tham số fill_value

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

```
In [164]: df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
                             columns=list('abcd'))
In [165]: df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
                             columns=list('abcde'))
```

Việc cộng các giá trị này lại với nhau sẽ tạo ra giá trị *NaN* ở những vị trí không trùng nhau hoặc 1 bên có giá trị là *NaN*:

```
In [169]: df1 + df2
Out[169]:
     a    b     c    d   e
0  0.0  2.0   4.0  6.0 NaN
1  9.0  NaN  13.0 15.0 NaN
2 18.0 20.0  22.0 24.0 NaN
   NaN  NaN   NaN  NaN NaN
```

df1

	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

df2

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	NaN	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

3.4.- Phép toán số học trên các đối tượng

3.4.1. Sử dụng các phương thức số học để điền (fill) các giá trị với tham số fill_value

```
In [169]: df1 + df2
Out[169]:
     a    b     c    d    e
0  0.0  2.0   4.0  6.0  NaN
1  9.0  NaN  13.0 15.0  NaN
2 18.0 20.0  22.0 24.0  NaN
   NaN  NaN   NaN  NaN  NaN
```

df1

	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

df2

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	NaN	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

Sử dụng phương thức add trên df1, với 2 đối số là df2 và fill_value:

```
In [170]: df1.add(df2, fill_value=0)
Out[170]:
      a     b     c     d     e
0   0.0   2.0   4.0   6.0   4.0
1   9.0   5.0  13.0  15.0   9.0
2  18.0  20.0  22.0  24.0  14.0
3  15.0  16.0  17.0  18.0  19.0
```

3.4.- Phép toán số học trên các đối tượng

3.4.1. Sử dụng các phương thức số học để điền (fill) các giá trị với tham số fill_value

- Các phương thức số học

Method	*Description*
`add, radd`	Phương thức cộng (+)
`sub, rsub`	Phương thức trừ (-)
`div, rdiv`	Phương thức chia (/)
`floordiv, rfloordiv`	Phương thức chia lấy phần nguyên (//)
`mul, rmul`	Phương thức nhân (*)
`pow, rpow`	Phương thức lũy thừa (**)

- Mỗi phương thức "gốc" đó đều có một phương thức nghịch đảo (tức là =1/phương thức "gốc"), với tên tương tự nhưng được thêm ký tự r ngay trước tên phương thức "gốc". Vì vậy, hai lệnh sau là tương đương:

```
df1
     a    b     c     d
0  0.0  1.0   2.0   3.0
1  4.0  5.0   6.0   7.0
2  8.0  9.0  10.0  11.0
```

```
In [171]: 1 / df1
Out[171]:
          a         b         c         d
0       inf  1.000000  0.500000  0.333333
1  0.250000  0.200000  0.166667  0.142857
2  0.125000  0.111111  0.100000  0.090909
In [172]: df1.rdiv(1)
Out[172]:
          a         b         c         d
0       inf  1.000000  0.500000  0.333333
1  0.250000  0.200000  0.166667  0.142857
2  0.125000  0.111111  0.100000  0.090909
```

3.4.- Phép toán số học trên các đối tượng

3.4.1. Sử dụng các phương thức số học để điền (fill) các giá trị với tham số fill_value

- Khi lập chỉ mục lại *Series* hoặc *DataFrame*, cũng có thể chỉ định một giá trị điền khác:

df1

	a	b	c	d
0	0.0	1.0	2.0	3.0
1	4.0	5.0	6.0	7.0
2	8.0	9.0	10.0	11.0

df2

	a	b	c	d	e
0	0.0	1.0	2.0	3.0	4.0
1	5.0	NaN	7.0	8.0	9.0
2	10.0	11.0	12.0	13.0	14.0
3	15.0	16.0	17.0	18.0	19.0

```
In [173]: df1.reindex(columns=df2.columns, fill_value=0)
Out[173]:
     a    b     c     d  e
0  0.0  1.0   2.0   3.0  0
1  4.0  5.0   6.0   7.0  0
2  8.0  9.0  10.0  11.0  0
```

3.4.- Phép toán số học trên các đối tượng

3.4.2. Các phép toán số học giữa DataFrame và Series

- *broadcasting* trong NumPy: Khi trừ mảng hai chiều cho mảng 1 chiều, NumPy sẽ lần lượt trừ mỗi hàng trong mảng 2 chiều với mảng 1 chiều.

```
In [174]: arr = np.arange(12.).reshape((3, 4))
In [175]: arr
Out[175]:
  array([  [ 0.,  1.,  2.,  3.],
           [ 4.,  5.,  6.,  7.],
           [ 8.,  9., 10., 11.]])
In [176]: arr[0]
Out[176]:
  array([ 0.,  1.,  2.,  3.])
In [177]: arr - arr[0]
Out[177]:
  array([  [ 0.,  0.,  0.,  0.],
           [ 4.,  4.,  4.,  4.],
           [ 8.,  8.,  8.,  8.]])
```

3.4.- Phép toán số học trên các đối tượng

3.4.2. Các phép toán số học giữa DataFrame và Series

- Các phép toán giữa *DataFrame* và *Series:* Giống như mảng NumPy có kích thước khác nhau, các phép toán số học giữa *DataFrame* và *Series* cũng được xác định. Theo mặc định, việc thực hiện phép toán số học giữa *DataFrame* và *Series* khớp với chỉ mục của *Series* trên các cột của *DataFrame*, truyền xuống các hàng

```
In [178]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                               columns=list('bde'),
                               index=['Utah', 'Ohio', 'Texas', 'Oregon'])
In [179]: series = frame.iloc[0]

In [182]: frame - series
Out[182]:
             b    d    e
Utah       0.0  0.0  0.0
Ohio       3.0  3.0  3.0
Texas      6.0  6.0  6.0
Oregon     9.0  9.0  9.0
```

```
frame
             b     d     e
Utah       0.0   1.0   2.0
Ohio       3.0   4.0   5.0
Texas      6.0   7.0   8.0
Oregon     9.0  10.0  11.0
```

```
series
b 0.0
d 1.0
e 2.0
```

3.4.- Phép toán số học trên các đối tượng

3.4.2. Các phép toán số học giữa DataFrame và Series

- *"broadcast"* trên các hàng của *DataFrame*: Kết quả thực hiện các phép toán số học giữa *DataFrame* và *Series*:
  - Nếu chỉ mục trong *DataFrame* và *Series* khớp nhau: Giá trị của chỉ mục bên Series sẽ được cộng cho tất cả các hàng trong DataFrame.
  - Ngược lại, khi chỉ mục có trong *Series* nhưng không có trong các giá trị chỉ mục trong các cột của *DataFrame* sẽ được nối vào sau trong kết quả với giá trị được điền vào kết quả sẽ là *NaN*.

```
In [183]: series2 = pd.Series(range(3), index=['b', 'e', 'f'])
In [184]: frame + series2
Out[184]:
             b    d     e   f
  Utah     0.0  NaN   3.0 NaN
  Ohio     3.0  NaN   6.0 NaN
  Texas    6.0  NaN   9.0 NaN
  Oregon   9.0  NaN  12.0 NaN
```

frame

	b	d	e
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0

series2

b	0
e	1
f	2

3.4.- Phép toán số học trên các đối tượng

3.4.2. Các phép toán số học giữa DataFrame và Series

- *"broadcast" trên các cột của DataFrame*: phải sử dụng một trong các phương thức số học.

  Ví dụ:

Trong trường hợp này, muốn khớp với chỉ mục hàng của *DataFrame*, cần phải dùng `axis='index'` hoặc `axis=0`.

```
In [185]: series3 = frame['d']
In [186]: frame
Out[186]:
          b     d     e
Utah    0.0   1.0   2.0
Ohio    3.0   4.0   5.0
Texas   6.0   7.0   8.0
Oregon  9.0  10.0  11.0
In [187]: series3
Out[187]:
Utah     1.0
Ohio     4.0
Texas    7.0
Oregon  10.0
Name: d, dtype: float64
In [188]: frame.sub(series3, axis='index')
Out[188]:
          b     d     e
Utah   -1.0   0.0   1.0
Ohio   -1.0   0.0   1.0
Texas  -1.0   0.0   1.0
Oregon -1.0   0.0   1.0
```

3.5.- *Ứng dụng hàm và mapping*

- NumPy's ufuncs (*Universal Functions* - các phương thức mảng theo phần tử) cũng hoạt động với các đối tượng *pandas*:

  Ví dụ:

```
In [189]: frame = pd.DataFrame(np.random.randn(4, 3),
  columns=list('bde'), index=['Utah','Ohio','Texas','Oregon'])
In [190]: frame
Out[190]:
               b         d         e
Utah   -0.204708  0.478943 -0.519439
Ohio   -0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189 -1.296221
In [191]: np.abs(frame)
Out[191]:
               b         d         e
Utah    0.204708  0.478943  0.519439
Ohio    0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189  1.296221
```

3.5.- Ứng dụng hàm và mapping

- Phương thức `apply` của *DataFrame*
  - Áp dụng hàm trên mảng một chiều cho mỗi cột hoặc hàng của DataFrame.

Ví dụ:

```
In [192]: f = lambda x: x.max() - x.min()
In [193]: frame.apply(f)
Out[193]:
b 1.802165
d 1.684034
e 2.689627
dtype: float64
```

frame

	b	d	e
Utah	-0.204708	0.478943	-0.519439
Ohio	-0.555730	1.965781	1.393406
Texas	0.092908	0.281746	0.769023
Oregon	1.246435	1.007189	-1.296221
xmax-xmin=	1.802165	1.684034	2.689627

Nếu có tham số `axis='columns'`, hàm sẽ thực hiện tính `x.max - x.min` trên mỗi hàng:

```
In [194]: frame.apply(f, axis='columns')
Out[194]:
Utah    0.998382
Ohio    2.521511
Texas   0.676115
Oregon  2.542656
dtype: float64
```

frame

	b	d	e	xmax-xmin
Utah	-0.204708	0.478943	-0.519439	0.998382
Ohio	-0.555730	1.965781	1.393406	2.521511
Texas	0.092908	0.281746	0.769023	0.676115
Oregon	1.246435	1.007189	-1.296221	2.542656

3.5.- Ứng dụng hàm và mapping

- Phương thức `apply` của *DataFrame*
  - Nhiều số liệu thống kê trên mảng phổ biến (như *`sum`* và *`mean`*) là các phương thức của `DataFrame`, vì vậy việc sử dụng phương thức *`apply`* là không cần thiết.
  - Hàm được truyền cho `apply` không nhất thiết phải trả về giá trị vô hướng; nó cũng có thể trả về một `Series` gồm nhiều giá trị:

```
In [195]: def f(x):
              return pd.Series([x.min(), x.max()], index=['min', 'max'])

In [196]: frame.apply(f)
Out[196]:
             b         d         e
min  -0.555730  0.281746 -1.296221
max   1.246435  1.965781  1.393406
```

frame

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

3.5.- Ứng dụng hàm và mapping

- Phương thức `applymap` của *DataFrame*
  - `applymap` cho phép tác động lên từng phần tử. Lý do có tên là `applymap` vì `Series` đã có phương thức `map` để áp dụng hàm theo từng phần tử.
  - Ví dụ sau chỉ thay đổi giá trị hiển thị, giá trị thực tế của frame là không đổi

▫ Định dạng mỗi giá trị trong `frame` chỉ gồm 2 số lẻ:

```
In [197]: format = lambda x: '%.2f' % x
In [198]: frame.applymap(format)
Out[198]:
             b     d     e
Utah     -0.20  0.48 -0.52
Ohio     -0.56  1.97  1.39
Texas     0.09  0.28  0.77
Oregon    1.25  1.01 -1.30
```

frame

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

▫ Chỉ thực hiện định dạng trên cột 'e':

```
In [199]: frame['e'].map(format)
Out[199]:
Utah     -0.52
Ohio      1.39
Texas     0.77
Oregon   -1.30
Name: e, dtype: object
```

3.6.- *Sắp xếp và xếp hạng (Sorting and Ranking)*

- Phương thức **`Sort_index`**: trả về một đối tượng mới được sắp xếp theo thứ tự từ điển theo chỉ mục hàng hoặc cột

```
#Series
In [200]: obj = pd.Series(range(4), index=['d', 'a', 'b', 'c'])
In [201]: obj.sort_index()
Out[201]:
      a    1
      b    2
      c    3
      d    0
      dtype: int64
#Có thể sắp xếp DataFrame theo chỉ mục trên một trong hai trục:
In [202]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
           index=['three', 'one'], columns=['d', 'a', 'b', 'c'])
In [203]: frame.sort_index()
Out[203]
              d    a    b    c
      one     4    5    6    7
      three   0    1    2    3
In [204]: frame.sort_index(axis=1)
Out[204]:
              a    b    c    d
      three   1    2    3    0
      one     5    6    7    4
```

3.6.- Sắp xếp và xếp hạng (Sorting and Ranking)

- Phương thức **`Sort_index`**:

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

```
In [205]: frame.sort_index(axis=1, ascending=False)
Out[205]:
       d  c  b  a
three  0  3  2  1
one    4  7  6  5
```

3.6.- Sắp xếp và xếp hạng (Sorting and Ranking)

- Phương thức **`Sort_values:`**
  - Với `Series:`

```
In [206]: obj = pd.Series([4, 7, -3, 2])
In [207]: obj.sort_values()
Out[207]:
     2      -3
     3       2
     0       4
     1       7
     dtype: int64
```

Theo mặc định, mọi giá trị bị thiếu đều được sắp xếp vào cuối `Series:`

```
In [208]: obj = pd.Series([4, np.nan, 7, np.nan, -3, 2])
In [209]: obj.sort_values()
Out[209]:
     4    -3.0
     5     2.0
     0     4.0
     2     7.0
     1     NaN
     3     NaN
     dtype: float64
```

3.6.- Sắp xếp và xếp hạng (Sorting and Ranking)

- Phương thức **`Sort_values:`**
  - Với `DataFrame`:
    - Có thể sử dụng dữ liệu trong một hoặc nhiều cột làm khóa sắp xếp. Để làm như vậy, hãy chuyển một hoặc nhiều tên cột vào tùy chọn `by` của `sort_values`:

```
In [210]: frame = pd.DataFrame({'b': [4, 7, -3, 2],
                                'a': [0, 1, 0, 1]})
In [211]: frame
Out[211]:
   a  b
0  0  4
1  1  7
2  0 -3
3  1  2
In [212]: frame.sort_values(by='b')
Out[212]:
   a  b
2  0 -3
3  1  2
0  0  4
1  1  7
```

    - Để sắp xếp theo nhiều cột, hãy chuyển danh sách thứ tự các tên cột theo sau tham số `by`:

```
In [213]: frame.sort_values(by=['a', 'b'])
Out[213]:
   a  b
2  0 -3
0  0  4
3  1  2
1  1  7
```

3.6.- Sắp xếp và xếp hạng (Sorting and Ranking)

- Phương thức **`Rank:`**
  - Xếp hạng chỉ định thứ hạng từ một đến số lượng điểm dữ liệu hợp lệ trong một mảng.
  - Theo mặc định, `rank` phá vỡ mối quan hệ bằng cách gán cho mỗi nhóm thứ hạng trung bình
  - Các giá trị có thể dùng với tham số method của phương thức *`rank`*

Method	*Description*
`'average'`	Giá trị mặc định. Gán thứ hạng trung bình cho mỗi mục trong nhóm bằng nhau
`'min'`	Sử dụng thứ hạng tối thiểu cho cả nhóm
`'max'`	Sử dụng thứ hạng tối đa cho cả nhóm
`'first'`	Gán thứ hạng theo thứ tự các giá trị xuất hiện trong dữ liệu
`'dense'`	Giống như method ='min', nhưng thứ hạng luôn tăng thêm 1 ở giữa các nhóm thay vì số lượng phần tử bằng nhau trong một nhóm

3.6.- Sắp xếp và xếp hạng (Sorting and Ranking)

- Phương thức **`Rank`**:
  - Với Series

```
In [214]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
In [215]: obj.rank()
Out[215]:
0    6.5
1    1.0
2    6.5
3    4.5
4    3.0
5    2.0
6    4.5
dtype: float64
```

```
7
-5
7
4
2
0
4
```

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

```
In [216]: obj.rank(method='first')
Out[216]:
0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

```
7
-5
7
4
2
0
4
```

3.6.- Sắp xếp và xếp hạng (Sorting and Ranking)

- Phương thức **`Rank:`**

Cũng có thể xếp hạng theo thứ tự giảm dần:

```
# Assign tie values the maximum rank in the group
In [217]: obj.rank(ascending=False, method='max')
Out[217]:
   0   2.0          7
   1   7.0         -5
   2   2.0          7
   3   4.0          4
   4   5.0          2
   5   6.0          0
   6   4.0          4
   dtype: float64
```

3.6.- Sắp xếp và xếp hạng (Sorting and Ranking)

- Phương thức **`Rank:`**
  - Với DataFrame

Có thể tính thứ hạng theo hàng hoặc cột:

```
In [218]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2],
                                'a': [0, 1, 0, 1],
                                'c': [-2, 5, 8, -2.5]})

In [219]: frame
Out[219]:
   a    b    c
0  0  4.3 -2.0
1  1  7.0  5.0
2  0 -3.0  8.0
3  1  2.0 -2.5
```

```
In [220]: frame.rank(axis='columns')
Out[220]:
     a    b    c
0  2.0  3.0  1.0
1  1.0  3.0  2.0
2  2.0  1.0  3.0
3  2.0  3.0  1.0
In [221]: frame.rank(axis='rows')
Out[221]:
     a    b    c
0  3.0  1.5  2.0
1  4.0  3.5  3.0
2  1.0  1.5  4.0
3  2.0  3.5  1.0
```

3.7.- Chỉ mục trên trục có giá trị trùng lặp

- Mặc dù nhiều chức năng của *pandas* (như *reindex*) yêu cầu các nhãn phải là duy nhất nhưng điều đó không bắt buộc.
- Ví dụ về một *Series* có các chỉ số trùng lặp:

```
In [221]: obj = pd.Series(range(5),
                          index=['a', 'a', 'b', 'b', 'c'])
In [222]: obj
Out[222]:
  a    0
  a    1
  b    2
  b    3
  c    4
  dtype: int64
```

- Thuộc tính `is_unique` của chỉ mục có thể cho biết nhãn của nó có phải là duy nhất hay không:

```
In [223]: obj.index.is_unique
Out[223]: False
```

3.7.- Chỉ mục trên trục có giá trị trùng lặp

- Việc chọn dữ liệu trên `Series` khi:
  - *Chỉ mục trùng nhau*: kết quả trả về là một `Series`.
  - *Chỉ mục không trùng nhau*: kết quả trả về là một giá trị vô hướng.

```
In [224]: obj['a']
Out[224]:
  a    0
  a    1
  dtype: int64
In [225]: obj['c']
Out[225]: 4
```

obj

a	0
a	1
b	2
b	3
c	4

Điều này có thể làm cho mã phức tạp hơn vì kiểu của đầu ra từ việc lập chỉ mục có thể khác nhau tùy thuộc vào việc nhãn có bị trùng hay không.

3.7.- Chỉ mục trên trục có giá trị trùng lặp

- Logic tương tự áp dụng cho việc lập chỉ mục các hàng trong *DataFrame*:

```
In [226]: df = pd.DataFrame(np.random.randn(4, 3),
                            index=['a', 'a', 'b', 'b'])
In [227]: df
Out[227]:
          0         1         2
a  0.274992  0.228913  1.352917
a  0.886429 -2.001637 -0.371843
b  1.669025 -0.438570 -0.539741
b  0.476985  3.248944 -1.021228
In [228]: df.loc['b']
Out[228]:
          0         1         2
b  1.669025 -0.438570 -0.539741
b  0.476985  3.248944 -1.021228
```

4. TÓM TẮT VÀ TÍNH TOÁN THỐNG KÊ MÔ TẢ
(Summarizing and Computing Descriptive Statistics)

4.1. Thống kê mô tả và tóm tắt

- Các đối tượng *pandas* được trang bị một bộ phương thức toán học và thống kê phổ biến. Hầu hết trong số này thuộc:
  - Danh mục rút gọn (*category of reductions*)
  - Hoặc thống kê tóm tắt (*summary statistics*)
  - Hoặc các phương thức trích xuất một giá trị duy nhất (như *sum* hoặc *mean*) từ một *Series* hoặc *Series* các giá trị từ các hàng hoặc cột của *DataFrame*.

4.1. Thống kê mô tả và tóm tắt

- Các phương thức thống kê mô tả và tóm tắt (*Descriptive and summary statistics*) trên *Series* và *DataFrame*

Method	*Description*
`argmin, argmax`	Tính toán các vị trí chỉ mục (số nguyên) tại đó thu được giá trị tối thiểu hoặc tối đa tương ứng
`count`	Đếm số lượng giá trị không phải NA (non-NA)
`cumsum`	Tổng giá trị tích lũy
`cummin, cummax`	Tích lũy tối thiểu hoặc tối đa của các giá trị tương ứng
`cumprod`	Tích lũy của tích các giá trị
`describe`	Tính toán các giá trị thống kê tóm tắt (count, mean, std, min 25%, 50%, 75%, max) cho *Series* hoặc từng cột trong *DataFrame*
`diff`	Tính sai phân số học đầu tiên (hữu ích cho chuỗi thời gian - time *Series*)
`idxmin, idxmax`	Tính toán các nhãn chỉ số tương ứng đạt được giá trị tối thiểu hoặc tối đa
`kurt`	Độ nhọn mẫu (fourth moment) của các giá trị

4.1. Thống kê mô tả và tóm tắt

- Các phương thức thống kê mô tả và tóm tắt (*Descriptive and summary statistics*) trên *Series* và *DataFrame*

Method	*Description*
`min, max`	Tính giá trị tối thiểu và tối đa
`mad`	Độ lệch tuyệt đối trung bình so với giá trị trung bình
`mean`	Giá trị trung bình
`median`	Trung vị số học (50% quantile) của các giá trị
`quantile`	Tính toán các giá trị lượng tử của mẫu trong khoảng từ 0 đến 1
`pct_change`	Tính phần trăm thay đổi
`prod`	Tích của mọi giá trị
`skew`	Độ lệch mẫu (third moment) của các giá trị
`std`	Độ lệch chuẩn mẫu của các giá trị
`sum`	Tổng các giá trị
`var`	Phương sai mẫu của các giá trị

4.1. Thống kê mô tả và tóm tắt

- Các phương thức thống kê mô tả và tóm tắt (*Descriptive and summary statistics*) trên *Series* và *DataFrame*
- *Các tùy chọn cho phương thức reduction*

Method	*Description*
`axis`	Trục để thực hiện việc giảm; 0 cho các hàng của *DataFrame* và 1 cho các cột
`skipna`	Loại trừ các giá trị bị thiếu; `True` là giá trị mặc định
`level`	Giảm nhóm theo cấp độ nếu trục được lập chỉ mục theo thứ bậc (hierarchically indexed - MultiIndex)

4.1. Thống kê mô tả và tóm tắt

- Phương thức trích xuất một giá trị duy nhất

- **Phương thức `sum`**: của `DataFrame` trả về một `Series`

```
In [229]: df = pd.DataFrame([[1.4, np.nan], [7.1, -4.5],
 [np.nan, np.nan], [0.75, -1.3]], index=['a', 'b', 'c', 'd'],
                                          columns=['one', 'two'])
In [230]: df
Out[230]:
            one      two
      a    1.40      NaN
      b    7.10     -4.5
      c     NaN      NaN
      d    0.75     -1.3
```

- Tính tổng cột:

```
In [231]: df.sum()
Out[231]:
      one  9.25
      two -5.80
      dtype: float64
```

- Tính tổng dòng: chuyển thêm đối số `axis='columns'` hoặc `axis=1`:

```
In [232]: df.sum(axis='columns')
Out[232]:
      a    1.40
      b    2.60
      c     NaN
      d   -0.55
      dtype: float64
```

4.1. Thống kê mô tả và tóm tắt

- *Phương thức trích xuất một giá trị duy nhất*

- **Phương thức sum**: của `DataFrame` trả về một `Series`
  - Giá trị *NA* bị loại trừ trừ khi toàn bộ lát cắt (hàng hoặc cột) là *NA*. Điều này có thể bị vô hiệu hóa bằng tùy chọn *skipna*:

```
In [233]: df.mean(axis='columns', skipna=False)
Out[233]:
a      NaN
b    1.300
c      NaN
d   -0.275
dtype: float64
```

```
df
        one      two
a      1.40      NaN
b      7.10     -4.5
c       NaN      NaN
d      0.75     -1.3
```

4.1. Thống kê mô tả và tóm tắt

- ***Phương thức trả về số liệu thống kê gián tiếp***
  - ***idxmin*** *và* ***idxmax*** *(*giá trị chỉ mục nơi đạt được giá trị tối thiểu hoặc tối đa *)*

```
In [234]: df.idxmax()
Out[234]:
one      b
two      d
dtype: object
```

```
df
      one      two
a    1.40      NaN
b    7.10     -4.5
c     NaN      NaN
d    0.75     -1.3
```

- ***Phương thức tính tích lũy*** *(accumulations)*
  - ***cumsum*** (tổng giá trị tích lũy)

```
In [235]: df.cumsum()
Out[235]:
     one      two
a   1.40      NaN
b   8.50     -4.5
c    NaN      NaN
d   9.25     -5.8
```

```
df
      one      two
a    1.40      NaN
b    7.10     -4.5
c     NaN      NaN
d    0.75     -1.3
```

4.1. Thống kê mô tả và tóm tắt

- **Phương thức không phải là giảm bớt cũng không phải là tích lũy**
  - **`describe`**

▫ *Trên dữ liệu số*: tạo ra nhiều số liệu thống kê tóm tắt trong một lần:

```
In [236]: df.describe()
Out[236]:
                one         two
count      3.000000    2.000000
mean       3.083333   -2.900000
std        3.493685    2.262742
min        0.750000   -4.500000
25%        1.075000   -3.700000
50%        1.400000   -2.900000
75%        4.250000   -2.100000
max        7.100000   -1.300000
```

▫ *Trên dữ liệu không phải số*: `describe` tạo ra số liệu thống kê tóm tắt thay thế:

```
In [237]: obj = pd.Series(['a', 'a', 'b', 'c'] * 4)
In [238]: obj.describe()
Out[238]:
count     16
unique     3
top        a
freq       8
dtype: object
```

4.2. Unique Values, Value Counts và Membership

- Một lớp phương thức liên quan khác trích xuất thông tin về các giá trị có trong *Series* một chiều (one-dimensional *Series*).

Các phương thức `Unique`, `value counts`, và `set membership`

Method	*Description*
`isin`	Tính toán mảng boolean cho biết mỗi giá trị `Series` có nằm trong chuỗi (sequence) giá trị được truyền hay không
`match`	Tính các chỉ số nguyên cho từng giá trị trong một mảng thành một mảng khác có các giá trị riêng biệt; hữu ích cho việc căn chỉnh dữ liệu và các tác vụ có kiểu kết nối (join-type operations)
`unique`	Tính toán mảng các giá trị duy nhất trong `Series`, được trả về theo thứ tự được quan sát
`value_counts`	Trả về một `Series` chứa các giá trị duy nhất làm chỉ mục và tần suất làm giá trị của nó, được sắp xếp theo thứ tự giảm dần

4.2. Unique Values, Value Counts và Membership

- **`unique`**:

Cung cấp một mảng các giá trị duy nhất trong *Series*. Các giá trị duy nhất không nhất thiết phải được trả về theo thứ tự đã sắp xếp nhưng có thể được sắp xếp theo thực tế nếu cần (`uniques.sort()`)

```
In [248]: obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b',
                           'b', 'c', 'c'])
In [249]: uniques = obj.unique()
In [250]: uniques
Out[250]: array(['c', 'a', 'd', 'b'], dtype=object)
In [251]: uniques.sort()
Out[251]: array(['a', 'b', 'c', 'd'], dtype=object)
```

4.2. Unique Values, Value Counts và Membership

- **`value_counts`**: tính toán một `Series` chứa tần suất giá trị:
  - Theo mặc định `Series` sắp xếp các giá trị đếm được theo thứ tự giảm dần.

```
In [252]: obj.value_counts()
Out[252]:
     c   3
     a   3
     b   2
     d   1
     dtype: int64
```

  - Để sắp xếp theo giá trị có trong `Series`, cần sử dụng trực tiếp phương thức `value_counts` của `pandas`. Phương thức này có thể được sử dụng với bất kỳ `array` hoặc `sequence` nào:

```
In [253]: pd.value_counts(obj.values, sort=False)
Out[253]:
     a   3
     b   2
     c   3
     d   1
     dtype: int64
```

4.2. Unique Values, Value Counts và Membership

- **`isin`:** thực hiện kiểm tra thành viên tập hợp được vectơ hóa và có thể hữu ích trong việc lọc tập dữ liệu xuống tập hợp con các giá trị trong `Series` hoặc cột trong `DataFrame`:

```
In [255]: mask = obj.isin(['b', 'c'])
In [256]: mask
Out[256]:
0     True
1    False
2    False
3    False
4    False
5     True
6     True
7     True
8     True
dtype: bool
```

	obj
0	c
1	a
2	d
3	a
4	a
5	b
6	b
7	c
8	c

```
In [257]: obj[mask]
Out[257]:
0    c
5    b
6    b
7    c
8    c
dtype: object
```

4.2. Unique Values, Value Counts và Membership

- **`Index.get_indexer`**: phương thức này cung cấp một mảng chỉ mục từ một mảng các giá trị có thể không phân biệt (*non-distinct values*) thành một mảng giá trị riêng biệt (*distinct values*) khác

```
In [258]: to_match = pd.Series(['c', 'a', 'b', 'b', 'c', 'a'])
In [259]: unique_vals = pd.Series(['c', 'b', 'a'])
In [260]: pd.Index(unique_vals).get_indexer(to_match)
Out[260]: array([0, 2, 1, 1, 0, 2])
```

4.2. Unique Values, Value Counts và Membership

- Trong một số trường hợp, có thể muốn tính biểu đồ (histogram) trên nhiều cột có liên quan trong `DataFrame`.

Truyền `pandas.value_counts` cho hàm `apply` của *DataFrame* sẽ được mang lại kết quả mong muốn. Ở đây, nhãn của các hàng trong kết quả là các giá trị riêng biệt xuất hiện trong tất cả các cột. Các giá trị là số lượng tương ứng của các giá trị này trong mỗi cột.

```
In [263]: result = data.apply(pd.value_counts).fillna(0)
In [264]: result
Out[264]:
```

	Qu1	Qu2	Qu3
1	1.0	1.0	1.0
2	0.0	2.0	1.0
3	2.0	2.0	0.0
4	2.0	0.0	2.0
5	0.0	0.0	1.0

data

	Qu1	Qu2	Qu3
0	1	2	1
1	3	3	5
2	4	1	2
3	3	2	4
4	4	3	4


Q&A
How
Why
When
Where
What
Who