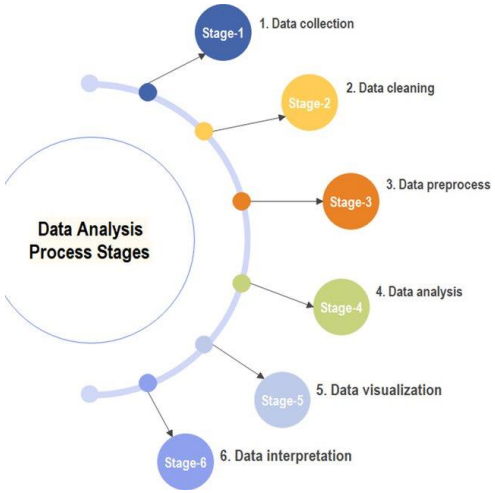
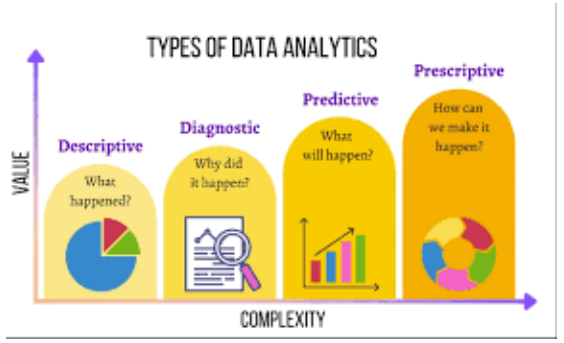
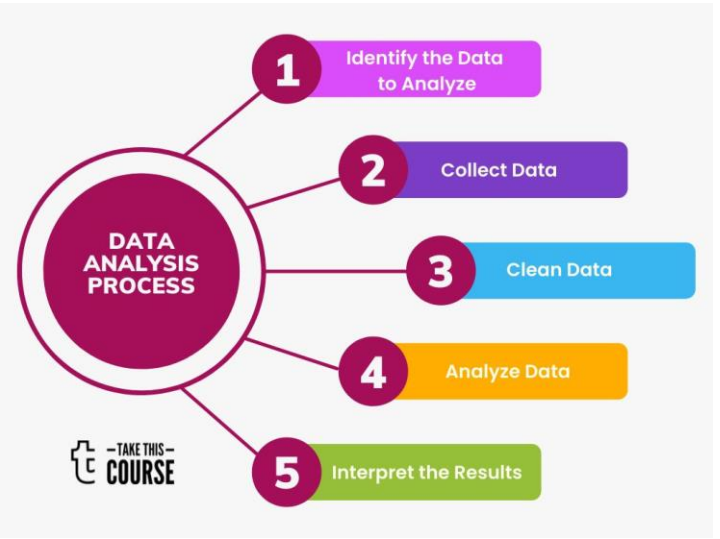




PHÂN TÍCH DỮ LIỆU

(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 6

THAO TÁC VỚI CÁC ĐỊNH DẠNG KHÁC NHAU CỦA TẬP TIN DỮ LIỆU

(Data Loading, Storage & File Formats)

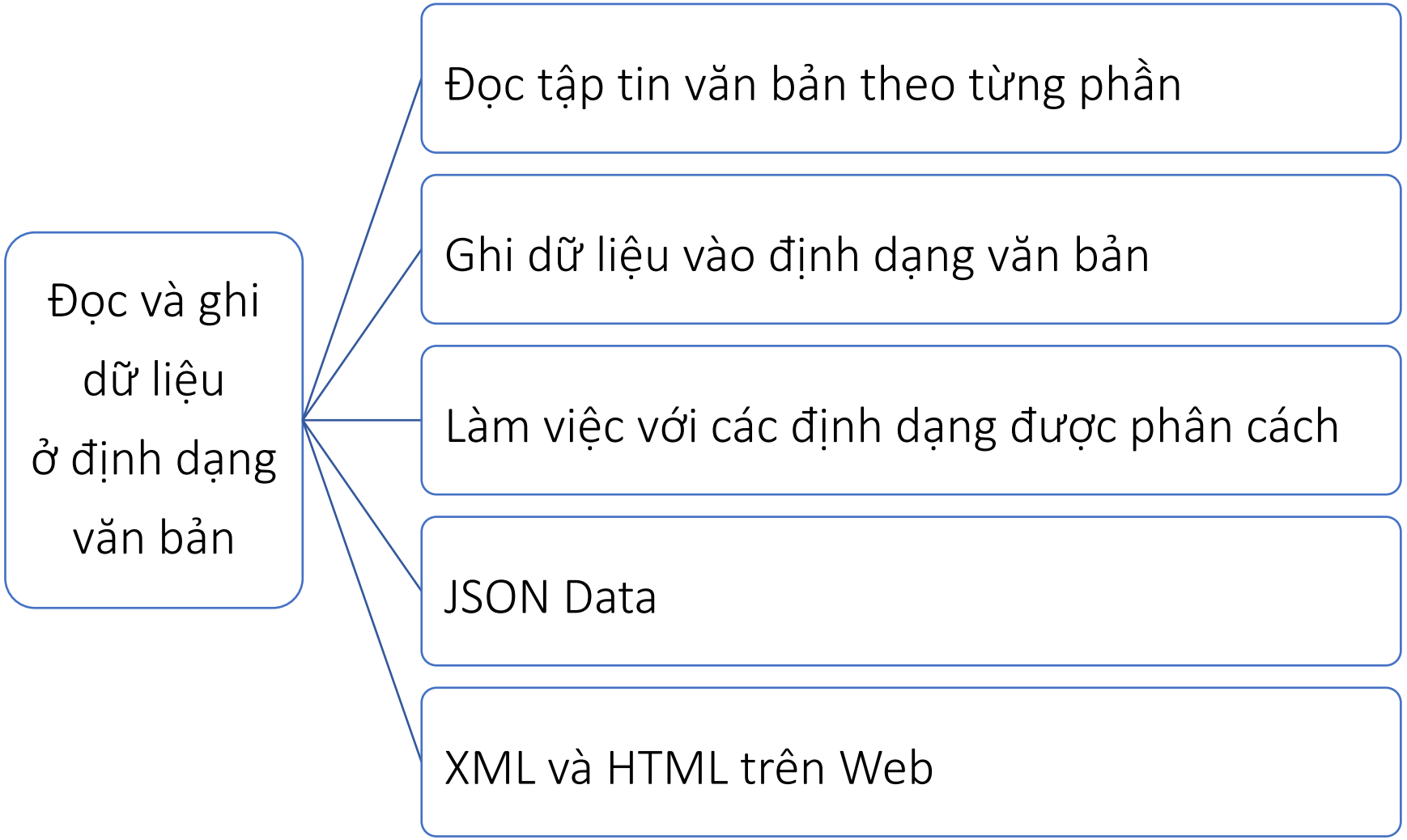
Lê Văn Hạnh
levanhanhvn@gmail.com

NỘI DUNG CHƯƠNG 6

1. Đọc và ghi dữ liệu ở định dạng văn bản (*Reading and Writing Data in Text Format*)
2. Định dạng dữ liệu nhị phân (*Binary Data Formats*)
3. Tương tác với API Web (*Interacting with Web APIs*)
4. Tương tác với Cơ sở dữ liệu (*Interacting with Databases*)

1. ĐỌC VÀ GHI DỮ LIỆU Ở ĐỊNH DẠNG VĂN BẢN

(*Reading and Writing Data in Text Format*)



1.1. Hàm trong pandas đọc dữ liệu dạng bảng sang đối tượng DataFrame

Function	*Description*
`read_csv`	Tải dữ liệu từ một tập tin, URL hoặc đối tượng giống như tập tin; mà những dữ liệu này mặc định sử dụng dấu phẩy (',') làm dấu phân cách
`read_table`	Tải dữ liệu từ một tập tin, URL hoặc đối tượng giống như tập tin; mà những dữ liệu này mặc định sử dụng dấu tab ('\t') làm dấu phân cách
`read_fwf`	Đọc dữ liệu ở định dạng cột có chiều rộng cố định (không có dấu phân cách)
`read_clipboard`	Phiên bản `read_table` đọc dữ liệu từ clipboard; hữu ích cho việc chuyển đổi bảng từ các trang web
`read_excel`	Đọc dữ liệu dạng bảng từ tập tin Excel XLS hoặc XLSX
`read_hdf`	Đọc tập tin HDF5 được viết bởi *pandas*
`read_html`	Đọc tất cả các bảng được tìm thấy trong tài liệu HTML đã cho
`read_json`	Đọc dữ liệu từ biểu diễn chuỗi *JSON* (JavaScript Object Notation)
`read_msgpack`	Đọc dữ liệu *pandas* được mã hóa bằng định dạng nhị phân *MessagePack*
`read_pickle`	Đọc một đối tượng tùy ý được lưu trữ ở định dạng *pickle Python*
`read_sas`	Đọc tập dữ liệu *SAS* được lưu trữ ở một trong các định dạng lưu trữ tùy chỉnh của hệ thống *SAS*
`read_sql`	Đọc kết quả của truy vấn *SQL* (sử dụng *SQLAlchemy*) dưới dạng *DataFrame* của *pandas*
`read_stata`	Đọc tập dữ liệu từ định dạng tập tin *Stata*
`read_feather`	Đọc định dạng tập tin nhị phân *Feather* (*Feather binary*)

1.2. Các đối số tùy chọn cho các hàm này có thể thuộc một số loại

- *Lập chỉ mục (Indexing):*Có thể coi một hoặc nhiều cột là *DataFrame* được trả về và liệu có lấy tên cột từ tập tin, người dùng hay không.
- *Suy luận kiểu và chuyển đổi dữ liệu (Type inference and data conversion):* bao gồm các chuyển đổi giá trị do người dùng xác định và danh sách tùy chỉnh các điểm đánh dấu giá trị bị thiếu.
- *Phân tích ngày giờ (Datetime parsing):* gồm khả năng kết hợp, gồm kết hợp thông tin ngày và giờ trải rộng trên nhiều cột thành một cột duy nhất trong kết quả.
- *Lặp lại (Iterating):* Hỗ trợ lặp lại các khối của tập tin rất lớn (*chunks of very large files*).
- *Vấn đề về dữ liệu không sạch (Unclean data issues):* Bỏ qua các hàng hoặc chân trang (footer), nhận xét (comments) hoặc những thứ nhỏ nhặt khác như dữ liệu số có dấu phân cách hàng ngàn được phân tách bằng dấu phẩy.

1.3. read_csv và read_table

- Một số đối số của hàm read_csv/read_table:

Argument	*Description*
path	Chuỗi biểu thị vị trí hệ thống tập tin, URL hoặc đối tượng giống tập tin
sep or delimiter	Chuỗi ký tự hoặc biểu thức chính quy được sử dụng để phân chia các trường trong mỗi hàng
header	Số thứ tự của hàng để sử dụng làm tên cột; mặc định là hàng 0 (hàng đầu tiên), nhưng sẽ là None nếu không có hàng tiêu đề
index_col	Số hoặc tên cột dùng làm chỉ mục hàng trong kết quả; có thể là một tên/số hoặc một danh sách để tạo chỉ mục phân cấp
names	Danh sách tên cột cho kết quả, kết hợp với header = None
skiprows	Số hàng ở đầu tập tin cần bỏ qua hoặc danh sách số hàng (bắt đầu từ 0) cần bỏ qua.
na_values	Chuỗi các giá trị thay thế bằng NA.
comment	(Các) ký tự để phân tách các nhận xét ở cuối dòng.
parse_dates	Cố gắng phân tích dữ liệu theo ngày giờ; False là giá trị mặc định. Nếu là True, sẽ cố gắng phân tích tất cả các cột. Ngược lại có thể chỉ định danh sách số cột hoặc tên để phân tích. Nếu phần tử của danh sách là tuple hoặc list, sẽ kết hợp nhiều cột lại với nhau và phân tích cú pháp theo ngày (ví dụ: nếu ngày/giờ được chia thành hai cột).
keep_date_col	Nếu nối các cột để phân tích ngày, hãy giữ các cột đã nối; False là giá trị mặc định.
converters	Dict chứa số cột ánh xạ tên tới các hàm (ví dụ: {'foo': f} sẽ áp dụng hàm f cho tất cả các giá trị trong cột 'foo').
dayfirst	Khi phân tích các ngày có thể không rõ ràng, hãy xử lý theo định dạng quốc tế (ví dụ: 6/7/2024 -> 7 tháng 6 năm 2024); False là giá trị mặc định.
date_parser	Hàm được sử dụng để phân tích ngày.
nrows	Số hàng cần đọc từ đầu tập tin.
iterator	Trả về một đối tượng TextParser để đọc từng phần của tập tin.
chunksize	Để lặp lại, kích thước của các đoạn tập tin (*size of file chunks*).
skip_footer	Số dòng cần bỏ qua ở cuối tập tin.
verbose	In thông tin đầu ra của trình phân tích cú pháp khác nhau, như số lượng giá trị bị thiếu được đặt trong các cột không phải là số.
encoding	Mã hóa văn bản cho Unicode (ví dụ: 'utf-8' cho văn bản được mã hóa UTF-8).
squeeze	Nếu dữ liệu được phân tích cú pháp chỉ chứa một cột, kết quả trả về Series.
thousands	Dấu phân cách cho hàng ngàn (ví dụ: ',' hoặc '.').

1.3. read_csv và read_table

- Có thể sử dụng lệnh type để xem nội dung file trong Jupyter notebook

```
In [1]: !type d:\ex1.csv
    a,b,c,d,message
    1,2,3,4,hello
    5,6,7,8,world
    9,10,11,12,foo
```

- Vì nội dung file được phân cách bằng dấu phẩy nên có thể sử dụng `read_csv` để đọc dữ liệu vào *DataFrame*:

```
In [2]: df = pd.read_csv('D:/ex1.csv')
In [3]: df
Out[3]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

- Cũng có thể sử dụng `read_table` và chỉ định dấu phân cách:

```
In [4]: pd.read_table('D:/ex1.csv', sep=',')
Out[4]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

1.3. read_csv và read_table

d:\ex2.csv
(không có tiêu đề cột)

```
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

- Đối với tập tin không có hàng tiêu đề (như file `d:\ex2.csv`). Để đọc tập tin này, có thể:
  - Cho phép *pandas* gán tên cột mặc định (theo số nguyên tính từ 0) :

```
In [5]: pd.read_csv('D:\ex2.csv', header=None)
Out[5]:
   0   1   2   3      4
0  1   2   3   4  hello
1  5   6   7   8  world
2  9  10  11  12    foo
```

  - Hoặc có thể tự chỉ định tên:

```
In [6]: pd.read_csv('D:\ex2.csv',
                    names=['a', 'b', 'c', 'd', 'message'])
Out[6]:
   a   b   c   d message
0  1   2   3   4   hello
1  5   6   7   8   world
2  9  10  11  12     foo
```

1.3. read_csv và read_table

- Đối với tập tin không có hàng tiêu đề (như file `d:\ex2.csv`). Để đọc tập tin này, có thể:
  - Tạo chỉ mục cho *DataFrame* được trả về:
    - Với chỉ mục chỉ gồm 1 cột:
      - Có thể sử dụng đối số `index_col` bằng một trong hai cách sau: **`index_col=4`** hoặc **`index_col='message'`**:
      - Giả sử muốn cột `message` là chỉ mục của *DataFrame* được trả về:
    - Trong trường hợp muốn tạo chỉ mục phân cấp từ nhiều cột, hãy chuyển danh sách số hoặc tên cột

```
In [7]: col_names = ['a', 'b', 'c', 'd', 'message']
In [8]: pd.read_csv('D:\ex2.csv', names=col_names,
                                  index_col='message')
Out[8]:
                a   b   c   d
    message
    hello       1   2   3   4
    world       5   6   7   8
    foo         9  10  11  12
```

```
In [9]: parsed = pd.read_csv('D:/ex2_multiindex.csv',
                             index_col=['key1', 'key2'])
In [10]: parsed
Out[10]:
                value1 value2
    key1 key2
    one     a       1      2
            b       3      4
            c       5      6
            d       7      8
    two     a       9     10
            b      11     12
            c      13     14
            d      15     16
```

1.3. read_csv và read_table

- Trong một số trường hợp, bảng có thể không có dấu phân cách cố định, sử dụng khoảng trắng hoặc một số mẫu khác để phân tách các trường.

Giả sử đã có file `d:\ex4.txt` (không có dấu phân cách cố định) với nội dung như sau. Do có thể dữ liệu giữa các cột không phân cách đều nhau 1 khoảng trắng mà có thể gồm một lượng khoảng trắng khác nhau. Trong những trường hợp này, có thể chuyển biểu thức chính quy (*regular expression*) làm dấu phân cách cho `read_table`. Điều này có thể được biểu diễn bằng biểu thức chính quy `\s+`, nhờ vậy kết quả sẽ thu được như sau:

d:\ex4.txt

```
' A B C\n',
'aaa -0.264438 -1.026059 -0.619500\n',
'bbb 0.927272 0.302904 -0.032399\n',
'ccc -0.264273 -0.386314 -0.217601\n',
'ddd -0.871858 -0.348382 1.100491\n'
```

Nội dung file D:/ex4.txt

```
In [11]: result = pd.read_table('examples/ex3.txt', sep='\s+')
In [12]: result
Out[12]:
                A         B         C
    aaa -0.264438 -1.026059 -0.619500
    bbb  0.927272  0.302904 -0.032399
    ccc -0.264273 -0.386314 -0.217601
    ddd -0.871858 -0.348382  1.100491
```

1.3. read_csv và read_table

- Trong một số trường hợp, bảng có thể không có dấu phân cách cố định, sử dụng khoảng trắng hoặc một số mẫu khác để phân tách các trường.

Giả sử đã có file `d:\ex4.txt` (không có dấu phân cách cố định) với nội dung như sau. Do có thể dữ liệu giữa các cột không phân cách đều nhau 1 khoảng trắng mà có thể gồm một lượng khoảng trắng khác nhau. Trong những trường hợp này, có thể chuyển biểu thức chính quy (*regular expression*) làm dấu phân cách cho `read_table`. Điều này có thể được biểu diễn bằng biểu thức chính quy `\s+`, nhờ vậy kết quả sẽ thu được như sau:

d:\ex4.txt

```
' A B C\n',
'aaa -0.264438 -1.026059 -0.619500\n',
'bbb 0.927272 0.302904 -0.032399\n',
'ccc -0.264273 -0.386314 -0.217601\n',
'ddd -0.871858 -0.348382 1.100491\n'
```

Nội dung file D:/ex4.txt

```
In [11]: result = pd.read_table('examples/ex3.txt', sep='\s+')
In [12]: result
Out[12]:
                A         B         C
    aaa -0.264438 -1.026059 -0.619500
    bbb  0.927272  0.302904 -0.032399
    ccc -0.264273 -0.386314 -0.217601
    ddd -0.871858 -0.348382  1.100491
```

1.3. read_csv và read_table

- Tham số `skiprows`: chỉ định các dòng cần bỏ qua.

  Ví dụ: có thể bỏ qua hàng đầu tiên, thứ ba và thứ tư của tập tin bằng cách sau:

```
In [14]: pd.read_csv('D:\ex1.csv',
                     skiprows=[0, 2, 3])
Out[14]:
    1,2,3,4,hello
```

d:\ex1.csv
a,b,c,d,message
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo

1.3. read_csv và read_table

- Xử lý các giá trị bị thiếu là một phần quan trọng và thường xuyên có nhiều sắc thái trong quá trình phân tích cú pháp tập tin.
  - Dữ liệu bị thiếu thường không có (chuỗi trống - empty string) hoặc được đánh dấu bằng một số giá trị trọng điểm (*sentinel* value). Theo mặc định, *pandas* sử dụng một tập hợp các giá trị chẳng hạn như *NA* và *NULL*

d:\ex5.csv

```
something,a,b,c,d,message
one,1,2,3,4,NA
two,5,6,,8,world
three,9,10,11,12,foo
```

```
In [16]: result = pd.read_csv('examples/ex5.csv')
In [17]: result
Out[17]:
   something  a   b     c   d  message
0        one  1   2   3.0   4      NaN
1        two  5   6   NaN   8    world
2      three  9  10  11.0  12      foo
In [18]: pd.isnull(result)
Out[18]:
   something      a      b      c      d  message
0        one  False  False  False  False     True
1        two  False  False   True  False    world
2      three  False  False  False  False      foo
```

1.3. read_csv và read_table

- Xử lý các giá trị bị thiếu:

d:\ex5.csv

```
something,a,b,c,d,message
one,1,2,3,4,NA
two,5,6,,8,world
three,9,10,11,12,foo
```

- Tùy chọn `na_values` có thể lấy danh sách hoặc tập hợp các chuỗi để xem xét thiếu giá trị:

```
In [19]: result = pd.read_csv('D:\ex5.csv', na_values=['NULL'])
In [20]: result
Out[20]:
   something  a   b     c  d message
0        one  1   2   3.0  4     NaN
1        two  5   6   NaN  8   world
2      three  9  10  11.0 12     foo
```

- Các trọng điểm NA khác nhau có thể được chỉ định cho mỗi cột trong một lệnh:

```
In [21]: sentinels={'message': ['foo','NA'], 'something':['two']}
In [22]: pd.read_csv('examples/ex5.csv', na_values=sentinels)
Out[22]:
   something  a   b     c  d message
0        one  1   2   3.0  4     NaN
1        NaN  5   6   NaN  8   world
2      three  9  10  11.0 12     NaN
```

1.4. Đọc tập tin văn bản theo từng phần

- Đọc dữ liệu từ file được chỉ định (D:/ex6.csv):

```
In [24]: result = pd.read_csv('D:/ex6.csv')
In [25]: result
Out[25]:
                one       two     three      four    key
        0  0.467976 -0.038649 -0.295344 -1.824726      L
        1 -0.358893  1.404453  0.704965 -0.200638      B
        2 -0.501840  0.659254 -0.421691 -0.057688      G
        3  0.204886  1.074134  1.388361 -0.982404      R
        4  0.354628 -0.133116  0.283763 -0.837063      Q
      ... ...       ...       ...       ..
     9995  2.311896 -0.417070 -1.409599 -0.515821      L
     9996 -0.479893 -0.650419  0.745152 -0.646038      E
     9997  0.523331  0.787112  0.486066  1.093156      K
     9998 -0.362559  0.598894 -1.843201  0.887292      G
     9999 -0.096376 -1.012999 -0.657431 -0.573315      0
     [10000 rows x 5 columns]
```

- Đối số nrows: Đối với các tập tin rất lớn, có thể phân tích viên chỉ muốn đọc hoặc lặp qua một phần nhỏ của tập tin.

  VD: đọc 5 hàng dữ liệu đầu tiên :

```
In [26]: pd.read_csv('examples/ex6.csv', nrows=5)
Out[26]:
                one       two     three      four    key
        0  0.467976 -0.038649 -0.295344 -1.824726      L
        1 -0.358893  1.404453  0.704965 -0.200638      B
        2 -0.501840  0.659254 -0.421691 -0.057688      G
        3  0.204886  1.074134  1.388361 -0.982404      R
        4  0.354628 -0.133116  0.283763 -0.837063      Q
```

1.4. Đọc tập tin văn bản theo từng phần

- Đọc dữ liệu từ file được chỉ định (D:/ex6.csv):

```
In [24]: result = pd.read_csv('D:/ex6.csv')
In [25]: result
Out[25]:
          one       two     three      four key
0    0.467976 -0.038649 -0.295344 -1.824726   L
1   -0.358893  1.404453  0.704965 -0.200638   B
2   -0.501840  0.659254 -0.421691 -0.057688   G
3    0.204886  1.074134  1.388361 -0.982404   R
4    0.354628 -0.133116  0.283763 -0.837063   Q
... ... ... ... ... ..
9995 2.311896 -0.417070 -1.409599 -0.515821   L
9996 -0.479893 -0.650419  0.745152 -0.646038   E
9997  0.523331  0.787112  0.486066  1.093156   K
9998 -0.362559  0.598894 -1.843201  0.887292   G
9999 -0.096376 -1.012999 -0.657431 -0.573315   0
[10000 rows x 5 columns]
```

- Đối số ***nrows***: Đối với các tập tin rất lớn, có thể phân tích viên chỉ muốn đọc hoặc lặp qua một phần nhỏ của tập tin.

  VD: đọc 5 hàng dữ liệu đầu tiên :

```
In [26]: pd.read_csv('examples/ex6.csv', nrows=5)
Out[26]:
          one       two     three      four key
0    0.467976 -0.038649 -0.295344 -1.824726   L
1   -0.358893  1.404453  0.704965 -0.200638   B
2   -0.501840  0.659254 -0.421691 -0.057688   G
3    0.204886  1.074134  1.388361 -0.982404   R
4    0.354628 -0.133116  0.283763 -0.837063   Q
```

1.4. Đọc tập tin văn bản theo từng phần

- Để đọc một tập tin theo từng phần, hãy chỉ định kích thước khối là một số hàng. Đối tượng TextParser được `read_csv` trả về cho phép lặp lại các phần của tập tin theo `chunksize`.

```
In [29]: chunker = pd.read_csv('D:\ex6.csv', chunksize=1000)
. . . .: tot = pd.Series([],dtype='float64')
. . . .: for piece in chunker:
. . . .: tot = tot.add(piece['key'].value_counts(), fill_value=0)
. . . .: tot = tot.sort_values(ascending=False)
. . . .: tot[:10]
Out[29]:
         E 368.0
         X 364.0
         L 346.0
         O 343.0
         Q 340.0
         M 338.0
         J 337.0
         F 335.0
         K 334.0
         H 330.0
         dtype: float64
```

1.5. Ghi dữ liệu vào định dạng văn bản *(Writing Data to Text Format)*

1.5.1. Phương thức `to_csv` của DataFrame

- Dùng để ghi dữ liệu ra tập tin:
  - Với phân tách là dấu phẩy:

```
In [30]: data = pd.read_csv('D:/ex5.csv')
In [31]: data
Out[31]:
   something  a   b     c   d  message
0        one  1   2   3.0   4      NaN
1        NaN  5   6   NaN   8    world
2      three  9  10  11.0  12      NaN

In [32]: data.to_csv('D:/out01.csv')
In [33]: !type D:/out01.csv
  ,something,a,b,c,d,message
  0,one,1,2,3.0,4,
  1,two,5,6,,8,world
  2,three,9,10,11.0,12,foo
```

  - Với phân tách là ký tự khác (ví dụ '|'): cần ghi vào `sys.stdout` để nó in kết quả văn bản ra bảng điều khiển:

```
In [34]: import sys
In [35]: data.to_csv(sys.stdout, sep='|')
  |something|a|b|c|d|message
  0|one|1|2|3.0|4|
  1|two|5|6||8|world
  2|three|9|10|11.0|12|foo
```

1.5. Ghi dữ liệu vào định dạng văn bản *(Writing Data to Text Format)*

1.5.1. Phương thức `to_csv` của DataFrame

- Đối số ***`na_rep`***: Theo mặc định, các giá trị bị thiếu sẽ xuất hiện dưới dạng chuỗi trống ở đầu ra. Có thể biểu thị chúng bằng một số giá trị trọng điểm (*sentinel value*) khác (như `'NULL'`):

```
In [36]: data.to_csv(sys.stdout, na_rep='NULL')
,something,a,b,c,d,message
0,one,1,2,3.0,4,NULL
1,two,5,6,NULL,8,world
2,three,9,10,11.0,12,foo
```

- Đối số ***`index`*** và ***`header`***: theo mặc định, cả nhãn của hàng (*row labels*) và cột (*column labels*) đều được ghi (=True). Để vô hiệu hóa việc ghi ra file nhãn của hàng và cột, cần thêm đối số với giá trị là False:

```
In [37]: data.to_csv(sys.stdout, index=False, header=False)
one,1,2,3.0,4,
two,5,6,,8,world
three,9,10,11.0,12,foo
```

1.5. Ghi dữ liệu vào định dạng văn bản *(Writing Data to Text Format)*

1.5.1. Phương thức `to_csv` của DataFrame

- Đối số **`columns`**: Cũng có thể chỉ viết một tập hợp con của các cột và theo thứ tự được chọn. Ví dụ với nội dung file *D:/ex1.csv* như sau, sau khi được đọc vào biến data, có thể ghi lại vào file csv khác với các cột được chỉ định:

```
In [41]: data.to_csv(sys.stdout, index=False,
                     columns=['a', 'b', 'c'])
a,b,c
1,2,3
5,6,7
9,10,11
```

d:/ex1.csv

```
a,b,c,d,message
1,2,3,4,hello
5,6,7,8,world
9,10,11,12,foo
```

1.5. Ghi dữ liệu vào định dạng văn bản *(Writing Data to Text Format)*

1.5.2. Phương thức `to_csv` của Series

```
In [42]: import pandas as pd
. . . .: # Create a Series with decimal values
. . . .: series = pd.Series({'A': 10, 'B': 20, 'C': 30, 'D': 40})
. . . .: # Write the Series to a CSV file
. . . .: series.to_csv('D:\out02.csv')
. . . .: !type D:/out02.csv
          ,0
          A,10
          B,20
          C,30
          D,40
```

1.6. Làm việc với các định dạng được phân cách

1.6.1. Các phân cách khác quy ước truyền thống

D:/ex7.csv
a @ b @ c
1 @ 2 @ 3
4 @ 5 @ 6

Không thể sử dụng module `csv` đối với các tập tin có dấu phân cách gồm nhiều ký tự cố định hoặc phức tạp. Trong những trường hợp đó, phải thực hiện việc tách dòng và dọn dẹp khác bằng phương thức phân tách của chuỗi hoặc phương thức biểu thức chính quy `re.split`.

Cho file dữ liệu như hình bên:

- Việc lặp qua trình đọc giống như một tập tin mang lại các bộ giá trị với bất kỳ ký tự trích dẫn nào bị loại bỏ:
- Từ đó, có thể thực hiện các thao tác cần thiết để đưa dữ liệu vào dạng mà ta cần. Hãy thực hiện từng bước này.
  - Đầu tiên, đọc tập tin thành danh sách các dòng:

```
In [43]: import csv
.......: f = open('D:/ex7.csv')
.......: reader = csv.reader(f)
.......: for line in reader:
.......:     print(line)
         ['a @ b @ c']
         ['1 @ 2 @ 3']
         ['1 @ 2 @ 3']
```

```
In [44]: with open('D:/ex7.csv') as f:
  ......:         lines = list(csv.reader(f))
  ......: lines
Out[44]: [['a @ b @ c'], ['1 @ 2 @ 3'], ['4 @ 5 @ 6']]
```

1.6. Làm việc với các định dạng được phân cách

1.6.1. Các phân cách khác quy ước truyền thống

D:/ex7.csv
a @ b @ c
1 @ 2 @ 3
4 @ 5 @ 6

Chia các dòng thành dòng tiêu đề và dòng dữ liệu:

```
In [45]: L=[]
         for index in range(len(lines)):
             L.append(list(lines[index][0].split('@')))
         header = [i.strip() for i in L[0]]
         header
Out[45]: ['a', 'b', 'c']
In [46]: values=[]
         for l in L[1:]:
             values.append([int(i) for i in l])
         values
Out[46]: [[1, 2, 3], [4, 5, 6]]
```

- Tạo một dictionary gồm các cột dữ liệu bằng cách sử dụng dictionary comprehension và biểu thức `zip(*values)`, chuyển hàng thành cột:

```
In [47]: data_dict = {h: v for h, v in zip(header, zip(*values))}
.......: data_dict
Out[47]: {'a': (1, 4), 'b': (2, 5), 'c': (3, 6)}
```

1.6. Làm việc với các định dạng được phân cách

1.6.2. csv.dialect

- Tập tin CSV có nhiều loại khác nhau. Để xác định một định dạng mới với dấu phân cách (*delimiter*), quy ước trích dẫn chuỗi (*string quoting convention*) hoặc dấu kết thúc dòng (*line terminator*) khác, có thể xác định một lớp con đơn giản của `csv.Dialect`:

```
class my_dialect(csv.Dialect):
      lineterminator = '\n'
      delimiter = ';'
      quotechar = '"'
      quoting = csv.QUOTE_MINIMAL

reader = csv.reader(f, dialect=my_dialect)
```

- Cũng có thể cung cấp các tham số dialect CSV riêng lẻ làm từ khóa cho `csv.reader` mà không cần phải xác định lớp con:

```
reader = csv.reader(f, dialect=my_dialect)
```

1.6. Làm việc với các định dạng được phân cách

1.6.2. csv.dialec

- Các tùy chọn của `csv.dialect`:

Argument	*Description*
`delimiter`	Chuỗi một ký tự để phân tách các fields. Mặc định là ','.
`lineterminator`	Ký tự kết thúc dòng. Mặc định là '\r\n'. Trình đọc bỏ qua điều này và nhận ra ký tự cuối dòng đa nền tảng (*cross-platform line terminators*).
`quotechar`	Ký tự trích dẫn cho các trường có ký tự đặc biệt (như dấu phân cách); mặc định là dấu nháy đôi ("").
`quoting`	Quy ước trích dẫn. Các tùy chọn bao gồm: • `csv.QUOTE_ALL` (trích dẫn tất cả các trường). • `csv.QUOTE_MINI MAL` (chỉ các trường có ký tự đặc biệt như dấu phân cách). • `csv.QUOTE_NONNUMERIC` và `csv.QUOTE_NONE` (không có trích dẫn). Mặc định là `QUOTE_MINIMAL`.
`skipinitialspace`	Bỏ qua khoảng trắng sau mỗi dấu phân cách; mặc định là `False`.
`doublequote`	Cách xử lý ký tự trích dẫn bên trong một field; nếu là `True`, nó sẽ được nhân đôi.
`escapechar`	Chuỗi để thoát dấu phân cách nếu `quoting` được đặt thành `csv.QUOTE_NONE`. Tùy chọn này sẽ bị tắt theo mặc định.

1.6.3. Ghi tập tin phân tách thủ công

Không thể sử dụng module `csv` đối với các tập tin có dấu phân cách gồm nhiều ký tự cố định hoặc phức tạp. Trong những trường hợp đó, phải thực hiện việc tách dòng và dọn dẹp khác bằng phương thức phân tách của chuỗi hoặc phương thức biểu thức chính quy `re.split`.

```
with open('mydata.csv', 'w') as f:
    writer = csv.writer(f, dialect=my_dialect)
    writer.writerow(('one', 'two', 'three'))
    writer.writerow(('1', '2', '3'))
    writer.writerow(('4', '5', '6'))
    writer.writerow(('7', '8', '9'))
```

1.7. JSON data

- JSON (*JavaScript Object Notation*) đã trở thành một trong những định dạng tiêu chuẩn để gửi dữ liệu theo yêu cầu HTTP giữa trình duyệt web và các ứng dụng khác. Đây là định dạng dữ liệu dạng tự do hơn nhiều so với dạng văn bản dạng bảng như CSV.

Ví dụ:

```
obj = """ {"name": "Wes",
           "places_lived": ["United States", "Spain", "Germany"],
           "pet": null,
           "siblings": [{"name":"Scott", "age":30, "pets":["Zeus","Zuko"]},
           {"name": "Katie", "age": 38,
           "pets": ["Sixes", "Stache", "Cisco"]}]
       }
       """
```

1.7. JSON data

- JSON gần như là mã *Python* hợp lệ, ngoại trừ giá trị `null` và một số sắc thái khác (chẳng hạn như không cho phép dấu phẩy ở cuối danh sách). Các kiểu dữ liệu cơ bản là các đối tượng (*dicts*), mảng (lists), chuỗi (strings), số (nembers), booleans và nulls. Tất cả các khóa trong một đối tượng phải là chuỗi. Có một số thư viện *Python* để đọc và ghi dữ liệu JSON. Trong phần này sẽ sử dụng module `json` vì nó được tích hợp vào thư viện chuẩn *Python*.
  - Để chuyển đổi chuỗi JSON thành dạng *Python*, hãy sử dụng `json.loads`:

```
In [48]: import json
In [49]: result = json.loads(obj)
In [50]: result
Out[50]:
   {'name': 'Wes',
    'pet': None,
    'places_lived': ['United States', 'Spain', 'Germany'],
    'siblings': [{'age': 30, 'name': 'Scott', 'pets': ['Zeus','Zuko']},
    {'age': 38, 'name': 'Katie', 'pets': ['Sixes', 'Stache','Cisco']}]}
```

  - Ngược lại, `json.dumps` có thể chuyển đổi một đối tượng *Python* trở lại về JSON:

```
In [51]: asjson = json.dumps(result)
```

1.7. JSON data

- Chuyển đổi một đối tượng JSON hoặc danh sách các đối tượng thành DataFrame hoặc một số cấu trúc dữ liệu khác để phân tích sẽ tùy thuộc vào nhu cầu của người lập trình.
  - Để thuận tiện, thông thường có thể chuyển danh sách các ký tự (trước đây là đối tượng JSON) cho hàm tạo DataFrame và chọn một tập hợp con của các trường dữ liệu:

```
In [48]: import json
In [49]: result = json.loads(obj)
In [50]: result
Out[50]:
   {'name': 'Wes',
    'pet': None,
    'places_lived': ['United States', 'Spain', 'Germany'],
    'siblings': [{'age': 30, 'name': 'Scott', 'pets': ['Zeus','Zuko']},
    {'age': 38, 'name': 'Katie', 'pets': ['Sixes', 'Stache','Cisco']}]}
```

```
In [52]: df = pd.DataFrame(result['siblings'], columns=['name','age'])
In [53]: df
Out[53]:
      name  age
   0  Scott   30
   1  Katie   38
```

1.7. JSON data

- `pandas.read_json` có thể tự động chuyển đổi các bộ dữ liệu JSON theo cách sắp xếp cụ thể thành *Series* hoặc *DataFrame*. Các tùy chọn mặc định cho `pandas.read_json` giả định rằng mỗi đối tượng trong mảng JSON là một hàng trong bảng
  - Ví dụ:

```
In [54]: !type D:\ex1.json
[{"a": 1, "b": 2, "c": 3},
 {"a": 4, "b": 5, "c": 6},
 {"a": 7, "b": 8, "c": 9}]
```

```
In [55]: data = pd.read_json('D:\ex1.json')
In [56]: data
Out[56]:
   a  b  c
0  1  2  3
1  4  5  6
2  7  8  9
```

- Để xuất dữ liệu từ *pandas* sang *JSON*, một trong các cách là sử dụng các phương thức `to_json` trên *Series* và *DataFrame*:

```
In [57]: print(data.to_json())
{"a":{"0":1,"1":4,"2":7},"b":{"0":2,"1":5,"2":8},"c":{"0":3,"1":6,"2":9}}
In [58]: print(data.to_json(orient='records'))
[{"a":1,"b":2,"c":3},{"a":4,"b":5,"c":6},{"a":7,"b":8,"c":9}]
```

1.8. XML và HTML: Web Scraping

1.8.1. HTML

- *Python* có nhiều thư viện để đọc và ghi dữ liệu ở các định dạng *HTML* và *XML* phổ biến. Các ví dụ bao gồm *lxml*, *Beautiful Soup* và *html5lib*. Mặc dù *lxml* nói chung tương đối nhanh hơn nhiều nhưng các thư viện khác có thể xử lý tốt hơn các tập tin *HTML* hoặc *XML* không đúng định dạng.
- *pandas* có một hàm dựng sẵn, ***read_html***, sử dụng các thư viện như *lxml* và *Beautiful Soup* để tự động phân tích cú pháp các bảng ra khỏi tập tin *HTML* dưới dạng đối tượng *DataFrame*.
  - Trước tiên, bạn phải cài đặt thêm một số thư viện được ***read_html*** sử dụng

```
conda install lxml # hoặc pip install lxml
pip install beautifulsoup4 html5lib
```

  - Hàm `pandas.read_html` có một số tùy chọn, nhưng theo mặc định, nó tìm kiếm và cố gắng phân tích tất cả dữ liệu dạng bảng có trong thẻ `<table>`. Kết quả là danh sách các đối tượng *DataFrame*

1.8. XML và HTML: Web Scraping

1.8.2. XML

- XML (*eXtensible Markup Language*) là một định dạng dữ liệu có cấu trúc phổ biến khác hỗ trợ dữ liệu lồng nhau, phân cấp với siêu dữ liệu.

- Hàm `pandas.read_html` sử dụng lxml hoặc Beautiful Soup để phân tích dữ liệu từ HTML. XML và HTML có cấu trúc tương tự nhau, nhưng XML tổng quát hơn.

D:/Performance.xml

```
<INDICATOR>
      <INDICATOR_SEQ>373889</INDICATOR_SEQ>
      <PARENT_SEQ></PARENT_SEQ>
      <AGENCY_NAME>Metro-North Railroad</AGENCY_NAME>
      <INDICATOR_NAME>Escalator Availability</INDICATOR_NAME>
      <DESCRIPTION>Percent of the time that escalators are operational systemwide.
      The availability rate is based on physical observations performed the
      morning of regular business days only. This is a new indicator the agency
      began reporting in 2009.
      </DESCRIPTION>
      <PERIOD_YEAR>2011</PERIOD_YEAR>
      <PERIOD_MONTH>12</PERIOD_MONTH>
      <CATEGORY>Service Indicators</CATEGORY>
      <FREQUENCY>M</FREQUENCY>
      <DESIRED_CHANGE>U</DESIRED_CHANGE>
      <INDICATOR_UNIT>%</INDICATOR_UNIT>
      <DECIMAL_PLACES>1</DECIMAL_PLACES>
      <YTD_TARGET>97.00</YTD_TARGET>
      <YTD_ACTUAL></YTD_ACTUAL>
      <MONTHLY_TARGET>97.00</MONTHLY_TARGET>
      <MONTHLY_ACTUAL></MONTHLY_ACTUAL>
</INDICATOR>
```

1.8.2. XML

- Ví dụ về cách sử dụng lxml để phân tích dữ liệu từ định dạng XML.
  - Sử dụng `lxml.objectify`, phân tích cú pháp tập tin và lấy tham chiếu đến nút gốc của tập tin *XML* bằng `getroot`:

```
from lxml import objectify
path = 'D:/Performance.xml'
parsed = objectify.parse(open(path))
root = parsed.getroot()
```

  - `root.INDICATOR` trả về một trình tạo (generator) mang lại từng phần tử `<INDICATOR>` XML. Đối với mỗi bản ghi, có thể điền một lệnh tên thẻ (như `YTD_ACTUAL`) vào các giá trị dữ liệu (không bao gồm một vài *tags*):

```
data = []
skip_fields=['PARENT_SEQ','INDICATOR_SEQ','DESIRED_CHANGE','DECIMAL_PLACES']
for elt in root.INDICATOR:
    el_data = {}
    for child in elt.getchildren():
    if child.tag in skip_fields:
        continue
    el_data[child.tag] = child.pyval
data.append(el_data)
```

1.8.2. XML

- Ví dụ (sử dụng lxml để phân tích dữ liệu từ định dạng XML)
  - Cuối cùng, chuyển đổi danh sách các ký tự này thành DataFrame: :

```
In [65]: perf = pd.DataFrame(data)
In [66]: perf.head()
Out[66]:
     Empty DataFrame
     Columns: []
     Index: []
```

2. ĐỊNH DẠNG DỮ LIỆU NHỊ PHÂN (*Binary Data Format*)

- Một trong những cách dễ nhất để lưu trữ dữ liệu (còn được gọi là tuần tự hóa - *serialization*) một cách hiệu quả ở định dạng nhị phân là sử dụng tính năng tuần tự hóa `pickle` tích hợp sẵn của Python. Tất cả các đối tượng pandas đều có phương thức `to_pickle` để ghi dữ liệu vào đĩa ở định dạng pickle:

```
In [70]: frame = pd.read_csv('D:/ex1.csv')
In [71]: frame
Out[71]:
          a    b    c    d  message
     0    1    2    3    4    hello
     1    5    6    7    8    world
     2    9   10   11   12      foo
In [72]: frame.to_pickle('D:/frame_pickle')
```

Có thể đọc bất kỳ đối tượng "pickle" nào được lưu trữ trong một tập tin bằng cách sử dụng trực tiếp pickle tích hợp (*built-in pickle*) hoặc thậm chí thuận tiện hơn bằng cách sử dụng `pandas.read_pickle`:

```
In [73]: pd.read_pickle(' D:/frame_pickle')
Out[73]:
          a    b    c    d  message
     0    1    2    3    4    hello
     1    5    6    7    8    world
     2    9   10   11   12      foo
```

2. Định dạng dữ liệu nhị phân (*Binary Data Format*)

- pandas có hỗ trợ tích hợp cho hai định dạng dữ liệu nhị phân khác: HDF5 và MessagePack. một số ví dụ về HDF5 sẽ được đưa ra trong phần tiếp theo, nhưng nên khám phá các định dạng tập tin khác nhau để xem chúng như thế nào và chúng hoạt động tốt như thế nào đối với phân tích đang thực hiện :
- Một số định dạng lưu trữ khác cho dữ liệu pandas hoặc NumPy bao gồm:
  - *bcolz:* Định dạng nhị phân hướng cột (*column-oriented binary*) có thể nén được dựa trên thư viện nén Blosc.
  - *Feather:* Một định dạng tập tin hướng theo cột đa ngôn ngữ (column-oriented binary). Feather sử dụng định dạng bộ nhớ cột Apache Arrow.

2.1. Sử dụng định dạng HDF5 *(Using HDF5 Format)*

- HDF5 (*Hierarchical Data Format* - định dạng dữ liệu phân cấp)
  - Mỗi tập tin HDF5 có thể lưu trữ nhiều bộ dữ liệu và siêu dữ liệu hỗ trợ (*supporting metadata*).
  - So với các định dạng đơn giản hơn, HDF5 hỗ trợ nén nhanh chóng với nhiều chế độ nén khác nhau, cho phép lưu trữ dữ liệu có mẫu lặp lại hiệu quả hơn.
  - HDF5 có thể là một lựa chọn tốt để làm việc với các tập dữ liệu rất lớn không vừa với bộ nhớ, vì có thể đọc và ghi các phần nhỏ của mảng lớn hơn nhiều một cách hiệu quả.
  - Có sẵn dưới dạng thư viện C và có giao diện bằng nhiều ngôn ngữ khác, bao gồm Java, Julia, MATLAB và Python

2.1. Sử dụng định dạng HDF5 *(Using HDF5 Format)*

- Mặc dù có thể truy cập trực tiếp các tập tin HDF5 bằng thư viện PyTables hoặc h5py, nhưng pandas cung cấp giao diện cấp cao giúp đơn giản hóa việc lưu trữ đối tượng Series và DataFrame. Lớp HDFStore hoạt động giống như một lệnh và xử lý các chi tiết cấp thấp:

```
In [74]: frame = pd.DataFrame({'a': np.random.randn(100)})
In [75]: store = pd.HDFStore('mydata.h5')
In [76]: store['obj1'] = frame
In [77]: store['obj1_col'] = frame['a']
In [78]: store
Out[78]:
```

```
<class 'pandas.io.pytables.HDFStore'>
File path: mydata.h5
/obj1      frame       (shape->[100,1])
/obj1_col  series      (shape->[100])
/obj2      frame_table (typ->appendable,nrows->100,ncols->1,indexers->[index])
/obj3      frame_table (typ->appendable,nrows->100,ncols->1,indexers-> [index])
```

- Sau đó, các đối tượng có trong tập tin HDF5 có thể được truy xuất

```
In [79]: store['obj1']
Out[79]:
                   a
0          -0.204708
1           0.478943
2          -0.519439
3          -0.555730
4           1.965781
..               ...
95          0.795253
96          0.118110
97         -0.748532
98          0.584970
99          0.152677
[100 rows x 1 columns]
```

2.1. Sử dụng định dạng HDF5 *(Using HDF5 Format)*

- HDFStore hỗ trợ hai lược đồ lưu trữ là 'fixed' và 'table'. Cái sau thường chậm hơn, nhưng nó hỗ trợ các hoạt động truy vấn bằng cú pháp đặc biệt. `Put` là phiên bản rõ ràng của phương thức `store['obj2'] = frame` nhưng cho phép đặt các tùy chọn khác như định dạng lưu trữ.

```
In [80]: store.put('obj2', frame, format='table')
In [81]: store.select('obj2', where=['index >= 10 and index <= 15'])
Out[82]:
                   a
     10     1.007189
     11    -1.296221
     12     0.274992
     13     0.228913
     14     1.352917
     15     0.886429
In [83]: store.close()
```

- Hàm `pandas.read_hdf` cung cấp lối tắt đến các công cụ này:

```
In [84]: frame.to_hdf('mydata.h5', 'obj3', format='table')
In [85]: pd.read_hdf('mydata.h5', 'obj3', where=['index < 5'])
Out[85]:
                   a
     0     -0.204708
     1      0.478943
     2     -0.519439
     3     -0.555730
     4      1.965781
```

2.1. Sử dụng định dạng HDF5 *(Using HDF5 Format)*

📖*Ghi chú*

- Nếu đang xử lý dữ liệu được lưu trữ trên các máy chủ từ xa, như Amazon S3 hoặc HDFS, thì việc sử dụng định dạng nhị phân khác được thiết kế cho bộ lưu trữ phân tán như Apache Parquet có thể phù hợp hơn.
- Nếu đang làm việc với số lượng lớn dữ liệu cục bộ, nên khám phá PyTables và h5py để xem chúng có thể phù hợp với nhu cầu hiện tại như thế nào. Vì nhiều vấn đề phân tích dữ liệu phụ thuộc vào I/O (chứ không phải CPU), việc sử dụng công cụ như HDF5 có thể tăng tốc đáng kể các ứng dụng.

📖***Thận trọng***:

- HDF5 không phải là cơ sở dữ liệu. HDF5 phù hợp nhất cho các bộ dữ liệu ghi một lần, đọc nhiều lần.
- Mặc dù dữ liệu có thể được thêm vào tập tin bất kỳ lúc nào, nhưng nếu nhiều người ghi đồng thời làm như vậy thì tập tin có thể bị hỏng.

2.2. Thao tác với tập tin Microsoft Excel *(Reading Microsoft Excel Files)*

pandas cũng hỗ trợ đọc dữ liệu dạng bảng được lưu trữ trong các tập tin *Excel 2003* (và cao hơn) bằng cách sử dụng lớp `ExcelFile` hoặc hàm `pandas.read_excel`. Bên trong, các công cụ này sử dụng các gói tiện ích bổ sung `xlrd` và `openpyxl` để đọc các tập tin *XLS* và *XLSX* tương ứng. Bạn có thể cần phải cài đặt thủ công bằng `pip` hoặc `conda`.

2.2. Thao tác với tập tin Microsoft Excel *(Reading Microsoft Excel Files)*

2.2.1. Đọc file

- ***Cách 1***:
  - Chuyển đường dẫn đến tập tin *xls* hoặc *xlsx*:

```
In [86]: xlsx = pd.ExcelFile('D:\ex1.xlsx')
```

  - Đọc dữ liệu của trang tính:

```
In [87]: DFrame = pd.read_excel(xlsx, 'Sheet1')
In [88]: DFrame = pd.read_excel(xlsx, 'Sheet1')
Out[88]:
```

	unnamed:	0	a	b	c	d	message
0		0	1	2	3	4	hello
1		1	5	6	7	8	world
2		2	9	10	11	12	foo

- ***Cách 2***:
  - Chuyển đường dẫn đến tập tin *xls* hoặc *xlsx*:

```
In [89]: DFrame = pd.read_excel('D:\ex1.xlsx', 'Sheet1')
In [90]: DFrame
Out[90]:
```

	unnamed:	0	a	b	c	d	message
0		0	1	2	3	4	hello
1		1	5	6	7	8	world
2		2	9	10	11	12	foo

2.2.2. Ghi file

- ***Cách 1***: Để ghi dữ liệu *pandas* sang định dạng *Excel*, trước tiên phải tạo `ExcelWriter`, sau đó ghi dữ liệu vào đó bằng phương thức `to_excel` của đối tượng *pandas*:

```
In [91]: writer = pd.ExcelWriter('D:\ex2.xlsx')
In [92]: frame.to_excel(writer, 'Sheet1')
In [93]: writer.save()
```

- ***Cách 2***: chuyển đường dẫn tập tin tới `to_excel` và tránh dùng `ExcelWriter`:

```
In [94]: frame.to_excel(' D:\ex2.xlsx')
```

3. TƯƠNG TÁC VỚI API Web *(Interacting with Web APIs)*

- Nhiều trang web có API công khai cung cấp nguồn cấp dữ liệu qua JSON hoặc một số định dạng khác. Có một số cách để truy cập các API này từ Python; một phương pháp dễ sử dụng là gói (package) requests.
- Giả sử, để tìm 30 vấn đề GitHub gần đây nhất đối với *pandas* trên GitHub, có thể thực hiện yêu cầu GET HTTP bằng cách bổ sung thư viện tiện ích `requests`:

```
In [95]: import requests
In [96]: url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
In [97]: resp = requests.get(url)
In [98]: resp
Out[98]: <Response [200]>
```

- Phương thức `json` của đối tượng Response sẽ trả về một dictionary chứa JSON được phân tích cú pháp thành các đối tượng Python gốc:

```
In [99]: data = resp.json()
In [100]: data[0]['title']
Out[101]: 'BUG: Pandas squashes 1-dimensional Numpy array with
shape (1,) down to a 0-dimensional array'
```

3. Tương tác với API Web *(Interacting with Web APIs)*

- Mỗi phần tử trong `data` là một dictionary chứa tất cả dữ liệu được tìm thấy trên trang GitHub issue (ngoại trừ các nhận xét - *comments*). Có thể truyền `data` trực tiếp tới DataFrame và trích xuất các trường quan tâm:
- Với một chút nỗ lực, có thể tạo một số giao diện cấp cao hơn cho các API web phổ biến trả về các đối tượng DataFrame để dễ dàng phân tích.

```
In [102]: issues = pd.DataFrame(data,
                                columns=['number', 'title', 'labels', 'state'])
In [103]: issues
Out[103]:
    number                                                title  \
0    17666       Period does not round down for frequencies les...
1    17665                DOC: improve docstring of function where
2    17664                    COMPAT: skip 32-bit test on int repr
3    17662                                implement Delegator class
4    17654       BUG: Fix series rename called with str alterin...
..     ...                                                   ...
25   17603       BUG: Correctly localize naive datetime strings...
26   17599                             core.dtypes.generic --> cython
27   17596        Merge cdate_range functionality into bdate_range
28   17587       Time Grouper bug fix when applied for list gro...
29   17583       BUG: fix tz-aware DatetimeIndex + TimedeltaInd...

                                               labels state
0                                                  []  open
1          [{'id': 134699, 'url': 'https://api.github.com...  open
2          [{'id': 563047854, 'url': 'https://api.github....  open
3                                                  []  open
4          [{'id': 76811, 'url': 'https://api.github.com/...  open
..                                                   ...   ...
25         [{'id': 76811, 'url': 'https://api.github.com/...  open
26         [{'id': 49094459, 'url': 'https://api.github.c...  open
27         [{'id': 35818298, 'url': 'https://api.github.c...  open
28         [{'id': 233160, 'url': 'https://api.github.com...  open
29         [{'id': 76811, 'url': 'https://api.github.com/...  open
[30 rows x 4 columns]
```

4. TƯƠNG TÁC VỚI CSDL *(Interacting with Databases)*

- Hiện tại, hầu hết dữ liệu của doanh nghiệp không được lưu trữ trong tập tin văn bản hoặc Excel. Cơ sở dữ liệu quan hệ dựa trên SQL (như SQL Server, PostgreSQL và MySQL) đang được sử dụng rộng rãi và nhiều cơ sở dữ liệu thay thế đã trở nên khá phổ biến. Việc lựa chọn cơ sở dữ liệu thường phụ thuộc vào hiệu suất, tính toàn vẹn dữ liệu và nhu cầu mở rộng của ứng dụng.
- Việc tải dữ liệu từ SQL vào DataFrame khá đơn giản và pandas có một số chức năng để đơn giản hóa quy trình.
  - Ví dụ: tạo cơ sở dữ liệu SQLite bằng trình điều khiển sqlite3 tích hợp của *Python*:

```
In [104]: import sqlite3
In [105]: query = """CREATE TABLE test
   .....: (a VARCHAR(20),
   .....: b VARCHAR(20),
   .....: c REAL, d INTEGER
   .....: );"""
In [106]: con = sqlite3.connect('mydata.sqlite')
In [107]: con.execute(query)
Out[107]: <sqlite3.Cursor at 0x7f6b12a50f10>
In [108]: con.commit()
```

4. Tương tác với CSDL *(Interacting with Databases)*

- Sau đó, chèn một vài hàng dữ liệu:

```
In [109]: data = [('Atlanta', 'Georgia', 1.25, 6),
 .....: ('Tallahassee', 'Florida', 2.6, 3),
 .....: ('Sacramento', 'California', 1.7, 5)]
In [110]: stmt = "INSERT INTO test VALUES(?, ?, ?, ?)"
In [111]: con.executemany(stmt, data)
Out[111]: <sqlite3.Cursor at 0x7f6b15c66ce0>
In [112]: con.commit()
```

Hầu hết các trình điều khiển Python SQL (PyODBC, psycopg2, MySQLdb, pymssql, v.v.) đều trả về list các tuples khi chọn dữ liệu từ một table:

```
In [113]: cursor = con.execute('select * from test')
In [114]: rows = cursor.fetchall()
In [115]: rows
Out[115]:
  [('Atlanta', 'Georgia', 1.25, 6),
   ('Tallahassee', 'Florida', 2.6, 3),
   ('Sacramento', 'California', 1.7, 5)]
```

4. Tương tác với CSDL *(Interacting with Databases)*

- Có thể chuyển list các tuples tới hàm tạo DataFrame, nhưng cũng cần tên cột (*column names*) có trong thuộc tính `description` của con trỏ (cursor):

```
In [116]: cursor.description
Out[116]:
        (('a', None, None, None, None, None, None),
         ('b', None, None, None, None, None, None),
         ('c', None, None, None, None, None, None),
         ('d', None, None, None, None, None, None))
In [117]: pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
Out[117]:
                    a           b     c    d
    0         Atlanta     Georgia  1.25    6
    1     Tallahassee     Florida  2.60    3
    2      Sacramento  California  1.70    5
```

4. Tương tác với CSDL *(Interacting with Databases)*

- Dự án *SQLAlchemy (http://www.sqlalchemy.org/)*:
  - Là một bộ công cụ *Python SQL* giúp loại bỏ nhiều điểm khác biệt phổ biến giữa các cơ sở dữ liệu SQL. *pandas* có hàm `read_sql` cho phép đọc dữ liệu dễ dàng từ kết nối *SQLAlchemy* chung. Tại đây, sẽ kết nối với cùng một cơ sở dữ liệu *SQLite* bằng *SQLAlchemy* và đọc dữ liệu từ bảng được tạo trước đó:

```
In [118]: import sqlalchemy as sqla
In [118]: db = sqla.create_engine('sqlite:///mydata.sqlite')
In [119]: pd.read_sql('select * from test', db)
Out[119]:
                a           b     c    d
    0      Atlanta     Georgia  1.25    6
    1  Tallahassee     Florida  2.60    3
    2   Sacramento  California  1.70    5
```


Q&A
How
Why
When
Where
What
Who