

# Ngôn Ngữ Lập Trình Python

## Các thư viện phổ biến (2)

Trịnh Tấn Đạt

Đại Học Sài Gòn

[trinhtandat@sgu.edu.vn](mailto:trinhtandat@sgu.edu.vn)

<http://sites.google.com/site/ttdat88>



# Nội Dung

- Giới thiệu và cài đặt
- Cấu trúc dữ liệu của pandas
- Series và Dataframe
- Bài tập

# Cài đặt

- “pandas” là thư viện mở rộng từ numpy, chuyên để xử lý dữ liệu cấu trúc dạng bảng (có thể dùng để đọc file excel hoặc csv)
- Tên “pandas” là viết tắt từ “panel data”
- Để cài đặt module pandas dùng lệnh:

`pip install pandas`

- [https://pandas.pydata.org/docs/user\\_guide/index.html](https://pandas.pydata.org/docs/user_guide/index.html)
- <https://pandas.pydata.org/docs/reference/index.html>

# Đặc điểm

- Đọc dữ liệu từ nhiều định dạng
- Liên kết dữ liệu và tích hợp xử lý dữ liệu bị thiếu
- Xoay và chuyển đổi chiều của dữ liệu dễ dàng
- Tách, đánh chỉ mục và chia nhỏ các tập dữ liệu lớn dựa trên nhãn
- Có thể nhóm dữ liệu cho các mục đích hợp nhất và chuyển đổi
- Lọc dữ liệu và thực hiện query trên dữ liệu
- Xử lý dữ liệu chuỗi thời gian và lấy mẫu

# Cấu trúc dữ liệu trong pandas

- Dữ liệu của pandas có 3 thành phần chính:
  - Series (dãy): cấu trúc 1 chiều, mảng dữ liệu đồng nhất.
  - Dataframe (khung): cấu trúc 2 chiều, dữ liệu trên các cột là đồng nhất (có phần giống như table trong SQL, nhưng với các dòng được đặt tên)
  - Panel (bảng): cấu trúc 3 chiều, có thể xem như một tập các dataframe với thông tin bổ sung
- Dữ liệu series gần giống kiểu array trong numpy, nhưng có 2 điểm khác biệt quan trọng:
  - Chấp nhận dữ liệu thiếu (NaN –không xác định)
  - Hệ thống chỉ mục phong phú

# Ví dụ: Series

- Dữ liệu một chiều
- Có thể coi như một dạng kết hợp giữa List và Dictionary.
- Mọi dữ liệu được lưu trữ theo thứ tự và có label.
- Cột đầu tiên là Index, nó giống như Keys trong Dictionary. Cột thứ 2 mới là dữ liệu.
- Cột dữ liệu có label riêng của nó và có thể gọi bằng thuộc tính .name

The Series		
	Animals	← Name
0	Dog	
1	Bear	
2	Tiger	
3	Moose	← Values
4	Giraffe	
5	Hippopotamus	
6	Mouse	

# Ví dụ: DataFrame

- Dữ liệu 2 chiều
- Các cột có tên
- Dữ liệu trên cột là đồng nhất
- Các dòng có thể có tên
- Có thể có ô thiếu dữ liệu

**The DataFrame**

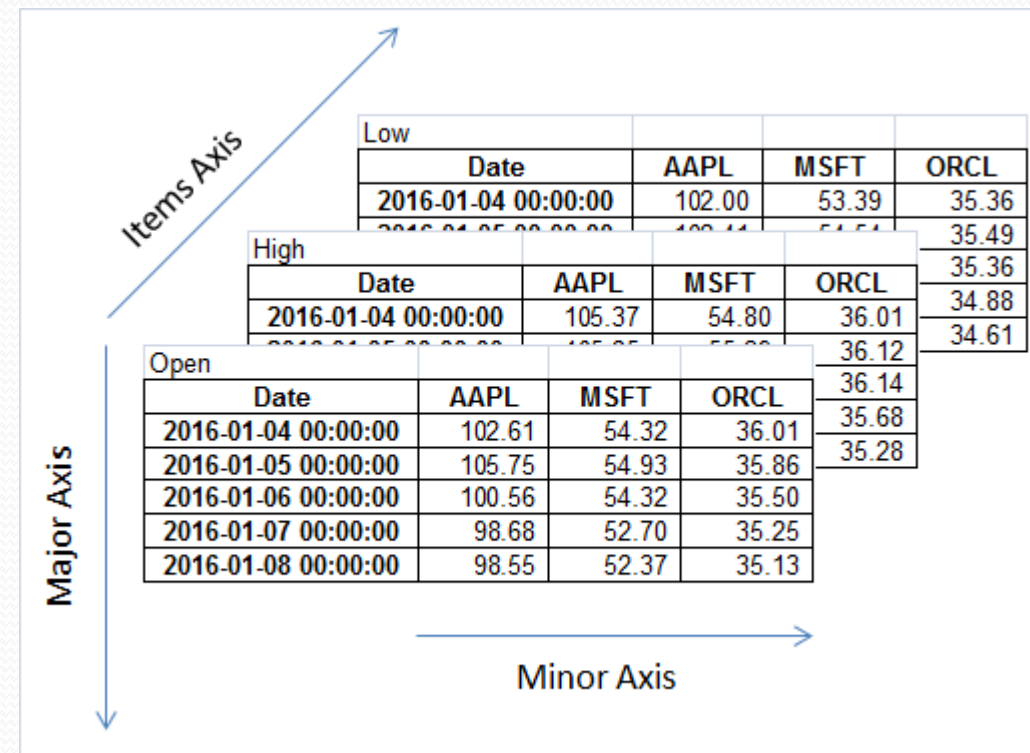
The diagram illustrates a DataFrame with two columns: 'Animals' and 'Owners'. The rows are indexed from 0 to 6. Annotations include: 'Index .loc(), .iloc()' pointing to the index column; 'Axis 1 (columns)' pointing to the column headers; 'Axis 0 (rows)' pointing to the row indices; 'df.iloc(2)' pointing to the row with index 2; 'df.iloc(5) ["Animals"]' pointing to the 'Animals' cell in row 5; 'df["Owners"]' pointing to the 'Owners' column; '.columns()' pointing to the column headers; and 'Values' pointing to the data cells.

	Animals	Owners
0	Dog	Chris
1	Bear	Kevyn
2	Tiger	Bob
3	Moose	Vinod
4	Giraffe	Daniel
5	Hippopotamus	Fil
6	Mouse	Stephanie



# Panel

- Dữ liệu 3 chiều
- Một tập các dataframe
- Các dataframe có cấu trúc tương đồng
- Có thể có các thông tin bổ sung cho từng dataframe



# Series

- `pandas.Series(data=None, index=None, dtype=None, name=None, copy=False, fastpath=False)` [\[source\]](#)

## Parameters:

**data:** *array-like, Iterable, dict, or scalar value*

Contains data stored in Series. If data is a dict, argument order is maintained.

**index:** *array-like or Index (1d)*

Values must be hashable and have the same length as *data*. Non-unique index values are allowed. Will default to RangeIndex (0, 1, 2, ..., n) if not provided. If data is dict-like and index is None, then the keys in the data are used as the index. If the index is not None, the resulting Series is reindexed with the index values.

**dtype:** *str, numpy.dtype, or ExtensionDtype, optional*

Data type for the output Series. If not specified, this will be inferred from *data*. See the [user guide](#) for more usages.

**name:** *str, optional*

The name to give to the Series.

**copy:** *bool, default False*

Copy input data. Only affects Series or 1d ndarray input.

# Series

- Tạo dữ liệu series

```
import pandas as pd
import numpy as np
S = pd.Series(np.random.randint(10, size = 5))
# tạo ngẫu nhiên 5 số trong khoảng [0,9]
print(S)
print(S.index)
print(S.values)
```

```
0    2
1    7
2    4
3    5
4    5
dtype: int32
RangeIndex(start=0, stop=5, step=1)
[2 7 4 5 5]
```

# Series

- Tạo dữ liệu series

```
import pandas as pd
import numpy as np
chiso = ["Ke toan", "Moi Truong", "CNTT", "Toan"]
giatri = [200, 160, 800, 100]
S = pd.Series(giatri, index=chiso)
print(S)
print(S.index)
print(S.values)
```

```
Ke toan      200
Moi Truong   160
CNTT         800
Toan         100
```

```
dtype: int64
```

```
Index(['Ke toan', 'Moi Truong', 'CNTT', 'Toan'], dtype='object')
[200 160 800 100]
```

# Series

```
import pandas as pd
import numpy as np
chiso = ["Ke toan", "Ke toan", "CNTT", "Co khi"] # trùng nhau
giatri = [200, 160, 800, 100]
S = pd.Series(giatri, index=chiso)
print(S)
print(S.index)
print(S.values)
```

```
Ke toan    200
Ke toan    160
CNTT       800
Co khi     100
```

```
dtype: int64
```

```
Index(['Ke toan', 'Ke toan', 'CNTT', 'Co khi'], dtype='object')
[200 160 800 100]
```

# Series

- Truy vấn dữ liệu thông qua chỉ số

```
import pandas as pd
import numpy as np
chiso = ["Ke toan", "Ke toan", "CNTT", "Co khi"] # trùng nhau
giatri = [200, 160, 800, 100]
S = pd.Series(giatri, index=chiso)
print(S['CNTT'])
print(S['Co khi'])
print(S['Ke toan'])
print(S.CNTT) # không có khoảng trắng
```

```
800
100
Ke toan      200
Ke toan      160
dtype: int64
800
```

# Series

- Nguyên tắc chung của việc thực hiện phép toán trên series như sau:
  - Nếu là phép toán giữa 2 series, thì các giá trị cùng chỉ số sẽ thực hiện phép toán với nhau, trường hợp không có giá trị ở cả 2 series thì trả về NaN
  - Nếu là phép toán giữa series và 1 số, thì thực hiện phép toán trên số đó với tất cả các giá trị trong series

# Phép toán trên series

- Phép cộng (+) hai series

```
import pandas as pd
import numpy as np
chiso = ["Ke toan", "Moi Truong", "CNTT", "Toan"]
giatri = [200, 160, 800, 100]
S = pd.Series(giatri, index=chiso)
# chỉ số giống nhau thì tính gộp, nếu không thì NaN
P = pd.Series([300, 400], ['CNTT', 'VatLy'])
Y = S + P
print(Y)
```

```
CNTT      1100.0
Ke toan    NaN
Moi Truong NaN
Toan       NaN
VatLy      NaN
dtype: float64
```

Tương tự cho các phép toán -, \*, / hai series



# Phép toán trên series

- Phép cộng (+) của Serie và số nguyên

```
import pandas as pd
import numpy as np
chiso = ["Ke toan", "Moi Truong", "CNTT", "Toan"]
giatri = [200, 160, 800, 100]
S = pd.Series(giatri, index=chiso)
Y = S + 200
print(Y)
```

```
Ke toan      400
Moi Truong   360
CNTT         1000
Toan         300
dtype: int64
```

Tương tự cho các phép toán -, \*, /

# Series

- Một vài phương thức phổ biến của Series:
  - `S.axes`: trả về danh sách các chỉ mục của `S`
  - `S.dtype`: trả về kiểu dữ liệu các phần tử của `S`
  - `S.empty`: trả về `True` nếu `S` rỗng
  - `S.ndim`: trả về số chiều của `S`
  - `S.size`: trả về số phần tử của `S`
  - `S.values`: trả về list các phần tử của `S`
  - `S.head(n)`: trả về `n` phần tử đầu tiên của `S`
  - `S.tail(n)`: trả về `n` phần tử cuối cùng của `S`

# Series.axes

- Trả về danh sách các chỉ mục của S

```
# importing pandas as pd
import pandas as pd
# Creating the Series
sr = pd.Series(['New York', 'Chicago', 'Toronto', 'Lisbon'])
# Creating the row axis labels
sr.index = ['City 1', 'City 2', 'City 3', 'City 4']
# Print the series
print(sr)
# return the element at the first position
sr.axes
```

```
City 1    New York
City 2     Chicago
City 3     Toronto
City 4      Lisbon
dtype: object
```

```
[Index(['City 1', 'City 2', 'City 3', 'City 4'], dtype='object')]
```

```
# importing pandas as pd
import pandas as pd
# Creating the Series
sr = pd.Series([1000, 5000, 1500, 8222])
# Print the series
print(sr)
# return the data type
print(sr.dtype)
# check if empty
print(sr.empty)
# return the dimension
print(sr.ndim)
# return the number of elements
print(sr.size)
# return the values in the given series object
# as an ndarray.
print(sr.values)
# return top n (5 by default) rows of a data
print(sr.head(2))
# return bottom n (5 by default) rows of a data
print(sr.tail(2))
```

```
0    1000
1    5000
2    1500
3    8222
dtype: int64
int64
False
1
4
[1000  5000  1500  8222]
0    1000
1    5000
dtype: int64
2    1500
3    8222
dtype: int64
```

# apply()

- Phương thức apply()

```
import pandas as pd
import numpy as np
def Tang(x):
    return x if x > 500 else x + 1000
chi_so = ["Ke toan", "KT", "CNTT", "Co khi"]
gia_tri = [310, 360, 580, 340]
S = pd.Series(gia_tri, chi_so)
# áp dụng Tang trên S (không thay đổi S)
print(S.apply(Tang))
print(S)
```

```
Ke toan    1310
KT         1360
CNTT        580
Co khi     1340
dtype: int64
Ke toan     310
KT          360
CNTT        580
Co khi      340
dtype: int64
```

# Làm việc với Dataframe

- Cú pháp chung:

`pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=None)`

- Trong đó:

- ‘data’ sẽ nhận giá trị từ nhiều kiểu khác nhau như list, dictionary, ndarray, series, ... và cả các DataFrame khác
- ‘index’ là nhãn chỉ mục hàng của dataframe
- ‘columns’ là nhãn chỉ mục cột của dataframe
- ‘dtype’ là kiểu dữ liệu cho mỗi cột
- ‘copy’ nhận giá trị True/False để chỉ rõ dữ liệu có được copy sang vùng nhớ mới không, mặc định là False

# Tạo dataframe từ list

```
import pandas as pd
names = ['MIT', "Stanford"]
df = pd.DataFrame(names)
print(df)
```

	0
0	MIT
1	Stanford

```
names_rank =
[['MIT',1],["Stanford",2]]
df = pd.DataFrame(names_rank)
print(df)
```

	0	1
0	MIT	1
1	Stanford	2

# Tạo dataframe từ dictionary các list

```
import pandas as pd
d = {'col1': [1, 2], 'col2': [3, 4]}
df = pd.DataFrame(data=d)
print(df)
# check data type
print(df.dtypes)
# To enforce a single dtype
df = pd.DataFrame(d, dtype=np.int8)
print(df.dtypes)
```

```
   col1  col2
0      1     3
1      2     4
col1    int64
col2    int64
dtype: object
col1    int8
col2    int8
dtype: object
```



# Tạo dataframe từ dictionary các list

```
crimes_rates = {  
    "Year": [1960, 1961, 1962, 1963, 1964],  
    "Population": [179323175, 182992000, 185771000, 188483000, 191141000],  
    "Total": [3384200, 3488000, 3752200, 4109500, 4564600],  
    "Violent": [288460, 289390, 301510, 316970, 364220]  
}  
crimes_dataframe = pd.DataFrame(crimes_rates)  
print(crimes_dataframe)
```

	Year	Population	Total	Violent
0	1960	179323175	3384200	288460
1	1961	182992000	3488000	289390
2	1962	185771000	3752200	301510
3	1963	188483000	4109500	316970
4	1964	191141000	4564600	364220

# Tạo dataframe từ list các dictionary

```
data = [  
    {'MIT': 5000, 'Stanford': 4500},  
    {'MIT': 1, 'Stanford': 2}  
]  
df = pd.DataFrame(data, index=['NumOfStudents', "ranking"])  
print(df)  
print(df.MIT.dtype)
```

	MIT	Stanford
NumOfStudents	5000	4500
ranking	1	2
int64		

# Tạo dataframe từ dictionary series

```
data = {  
    "one": pd.Series([1,23,45], index = [1,2,3]),  
    "two": pd.Series([1000,2400,1132,3434], index = [1,2,3,4])  
}  
df = pd.DataFrame(data)  
print(df)
```

	one	two
1	1.0	1000
2	23.0	2400
3	45.0	1132
4	NaN	3434

# Tạo dataframe từ dictionary series

```
d = {'col1': [0, 1, 2, 3], 'col2': pd.Series([2, 3], index=[2, 3])}  
df = pd.DataFrame(data=d, index=[0, 1, 2, 3])  
df
```

	col1	col2
0	0	NaN
1	1	NaN
2	2	2.0
3	3	3.0

# Tạo dataframe từ numpy ndarray

```
df2 = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]),  
                    columns=['a', 'b', 'c'])  
df2
```

	a	b	c
0	1	2	3
1	4	5	6
2	7	8	9

# Các thao tác cơ bản trên dataframe

- Display the index, columns, and data
- Column selection, addition, deletion
- Assigning new columns in method chains
- Indexing / selection
- Data alignment and arithmetic

## Display the index, columns, and data


```
import pandas as pd
d = {
    "col1": [5.1, 4.9, 4.7, 4.6, 5.0],
    "col2": [3.5, 3.0, 3.2, 3.1, 3.6],
}
index = ["a", "b", "c", "d", "e"]
df = pd.DataFrame(data=d, index=index)
print(df)
print(df.index)
print(df.columns)
print(df.values)
```

```
      col1  col2
a      5.1    3.5
b      4.9    3.0
c      4.7    3.2
d      4.6    3.1
e      5.0    3.6
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
Index(['col1', 'col2'], dtype='object')
[[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 [4.6 3.1]
 [5.  3.6]]
```

# Sorting

- Sorting: sort by the index (i.e., reorder columns or rows), not by the data in the table

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(10, 4), columns=["A", "B", "C", "D"])
print(df)
df.sort_index(axis=1, ascending=False)
```



column

Out[2]:

	A	B	C	D
0	-0.070897	1.288179	-0.503842	-0.480920
1	1.312666	1.897197	1.391129	1.414491
2	-0.400311	-0.720584	1.041182	1.098383
3	-0.454919	-0.486415	-0.549674	1.899689
4	0.425034	-1.271647	-0.083970	-0.012258
5	-0.662657	-0.629345	0.519322	0.675120
6	-0.120715	1.797798	0.989356	0.132498
7	0.228090	-0.104063	0.215566	-0.616781
8	-1.397870	-1.386189	-0.941996	-0.650729
9	0.381559	0.537504	-0.539080	-1.333322

	D	C	B	A
0	-0.480920	-0.503842	1.288179	-0.070897
1	1.414491	1.391129	1.897197	1.312666
2	1.098383	1.041182	-0.720584	-0.400311
3	1.899689	-0.549674	-0.486415	-0.454919
4	-0.012258	-0.083970	-1.271647	0.425034
5	0.675120	0.519322	-0.629345	-0.662657
6	0.132498	0.989356	1.797798	-0.120715
7	-0.616781	0.215566	-0.104063	0.228090
8	-0.650729	-0.941996	-1.386189	-1.397870
9	-1.333322	-0.539080	0.537504	0.381559



# Sorting

- Sorting: sort by the data values

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(10, 4), columns=["A", "B", "C", "D"])
print(df)
df.sort_values(by='C')
```

	A	B	C	D
0	1.244627	0.538063	-0.268220	1.551755
1	0.652621	0.084949	0.793983	-0.076990
2	1.129478	0.678506	-1.072695	-1.678143
3	0.133166	-0.545814	-1.075790	-0.740002
4	-0.914219	0.014593	-0.884466	1.491016
5	-0.214583	1.312629	-0.822387	-1.239508
6	0.824118	-0.028464	0.029642	0.609071
7	0.305566	0.418756	-1.098640	-1.186511
8	-0.454051	1.002630	-0.186442	-0.298726
9	-0.668789	-0.778334	0.024634	0.042592

	A	B	C	D
7	0.305566	0.418756	-1.098640	-1.186511
3	0.133166	-0.545814	-1.075790	-0.740002
2	1.129478	0.678506	-1.072695	-1.678143
4	-0.914219	0.014593	-0.884466	1.491016
5	-0.214583	1.312629	-0.822387	-1.239508
0	1.244627	0.538063	-0.268220	1.551755
8	-0.454051	1.002630	-0.186442	-0.298726
9	-0.668789	-0.778334	0.024634	0.042592
6	0.824118	-0.028464	0.029642	0.609071
1	0.652621	0.084949	0.793983	-0.076990

# Column selection, addition, deletion

```
import pandas as pd
d = {'col1': [1, 2, 3], 'col2': [10, 20, 30]}
df = pd.DataFrame(data=d)
df
```

	col1	col2
0	1	10
1	2	20
2	3	30

```
# Column selection
print(df['col1'])
```

```
0    1
1    2
2    3
Name: col1, dtype: int64
```

```
df["col3"] = df["col1"] * df["col2"]
df["flag"] = df["col1"] > 2
df
```

	col1	col2	col3	flag
0	1	10	10	False
1	2	20	40	False
2	3	30	90	True

# Column selection, addition, deletion

- Columns can be deleted or popped like with a dict:

	col1	col2	col3	flag
0	1	10	10	False
1	2	20	40	False
2	3	30	90	True

```
del df["col2"]  
three = df.pop("col3")  
print(three)  
print(df)
```

```
0    10  
1    40  
2    90  
Name: col3, dtype: int64  
   col1  flag  
0     1  False  
1     2  False  
2     3   True
```

# Column selection, addition, deletion

- Inserting a scalar value

```
df["foo"] = "bar"  
df
```

	col1	flag	foo
0	1	False	bar
1	2	False	bar
2	3	True	bar

- Inserting a Series that does not have the same index as the DataFrame, it will be conformed to the DataFrame's index:

```
df["one_trunc"] = df["col1"][:2]  
df
```

	col1	flag	foo	one_trunc
0	1	False	bar	1.0
1	2	False	bar	2.0
2	3	True	bar	NaN

# Column selection, addition, deletion

- By default, columns get inserted at the end. The insert function is available to insert at a particular location in the columns

```
df.insert(1, "bar", [10,30,40])  
df
```

	col1	bar	flag	foo	one_trunc
0	1	10	False	bar	1.0
1	2	30	False	bar	2.0
2	3	40	True	bar	NaN

# Assigning new columns in method chains

- `assign()` method that allows you to easily create new columns that are potentially derived from existing columns.

```
d = {  
    "col1": [5.1, 4.9, 4.7, 4.6, 5.0],  
    "col2": [3.5, 3.0, 3.2, 3.1, 3.6],  
}  
df = pd.DataFrame(data=d)  
df
```

	col1	col2
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2
3	4.6	3.1
4	5.0	3.6

```
df.assign(col3 = df["col1"] / df["col2"])
```

or

```
df.assign(col3 = lambda x: (x["col1"] / x["col2"]))
```

	col1	col2	col3
0	5.1	3.5	1.457143
1	4.9	3.0	1.633333
2	4.7	3.2	1.468750
3	4.6	3.1	1.483871
4	5.0	3.6	1.388889

`assign` **always** returns a copy of the data, leaving the original DataFrame untouched.

# Assigning new columns in method chains

- Ví dụ

```
dfa = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})  
dfa.assign(C=lambda x: x["A"] + x["B"], D=lambda x: x["A"] + x["C"])
```

	A	B	C	D
0	1	4	5	6
1	2	5	7	9
2	3	6	9	12

# Indexing / selection

- The basics of indexing are as follows:

Operation	Syntax	Result
Select column	<code>df[col]</code>	Series
Select row by label	<code>df.loc[label]</code>	Series
Select row by integer location	<code>df.iloc[loc]</code>	Series
Slice rows	<code>df[5:10]</code>	DataFrame
Select rows by boolean vector	<code>df[bool_vec]</code>	DataFrame



# Ví dụ:

```
d = {  
    "col1": [5.1, 4.9, 4.7, 4.6, 5.0],  
    "col2": [3.5, 3.0, 3.2, 3.1, 3.6],  
}  
index = ["a", "b", "c", "d", "e"]  
df = pd.DataFrame(data=d, index=index)  
df
```

	col1	col2
a	5.1	3.5
b	4.9	3.0
c	4.7	3.2
d	4.6	3.1
e	5.0	3.6

```
df["col1"]
```

```
a    5.1  
b    4.9  
c    4.7  
d    4.6  
e    5.0  
Name: col1, dtype: float64
```

```
df[[True, False, False, True, True]]
```

	col1	col2
a	5.1	3.5
d	4.6	3.1
e	5.0	3.6

```
df.loc["c"]
```

```
col1    4.7  
col2    3.2  
Name: c, dtype: float64
```

```
df.iloc[1]
```

```
col1    4.9  
col2    3.0  
Name: b, dtype: float64
```

```
df[0:3]
```

	col1	col2
a	5.1	3.5
b	4.9	3.0
c	4.7	3.2

# Ví dụ:

## Indexing both axes

```
df.iloc[[1]]
```

	col1	col2
<b>b</b>	4.9	3.0

```
df.iloc[0, 1]
```

```
Out[11]: 3.5
```

```
df.iloc[[2, 1]]
```

	col1	col2
<b>c</b>	4.7	3.2
<b>b</b>	4.9	3.0

```
df.iloc[[0, 3], [0, 1]]
```

	col1	col2
<b>a</b>	5.1	3.5
<b>d</b>	4.6	3.1

```
df.iloc[:3]
```

	col1	col2
<b>a</b>	5.1	3.5
<b>b</b>	4.9	3.0
<b>c</b>	4.7	3.2

```
df.iloc[2:4, 0:]
```

	col1	col2
<b>c</b>	4.7	3.2
<b>d</b>	4.6	3.1

# Data alignment and arithmetic

- Data alignment between DataFrame objects automatically align on **both the columns and the index (row labels)**.
- The resulting object will have the union of the column and row labels.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(10, 4), columns=["A", "B", "C", "D"])
df
```

	A	B	C	D
0	-0.797449	0.527920	-2.306775	1.740996
1	0.921609	1.784563	0.871792	-0.116722
2	1.121019	1.885745	-0.009241	-0.570125
3	0.389236	-0.564230	0.200664	0.080426
4	-0.313274	1.157791	-1.334049	-1.498198
5	0.906112	0.935350	0.133427	-0.961729
6	1.282177	0.008660	0.065187	-0.320325
7	-0.047931	0.179807	-1.028409	0.786557
8	0.606488	-0.829325	-0.231664	-0.829068
9	0.321457	0.953504	0.155475	-0.598032

```
df2 = pd.DataFrame(np.random.randn(7, 3), columns=["A", "B", "C"])
df2
```

	A	B	C
0	-0.263586	-0.194690	-0.341931
1	0.235225	-0.469747	2.135652
2	0.297726	-0.020956	0.599596
3	-1.001315	-2.050571	-1.213695
4	-0.778979	0.970515	-1.358900
5	-0.308514	0.293263	0.947084
6	1.701105	-0.799807	0.190508

	A	B	C	D
0	-0.797449	0.527920	-2.306775	1.740996
1	0.921609	1.784563	0.871792	-0.116722
2	1.121019	1.885745	-0.009241	-0.570125
3	0.389236	-0.564230	0.200664	0.080426
4	-0.313274	1.157791	-1.334049	-1.498198
5	0.906112	0.935350	0.133427	-0.961729
6	1.282177	0.008660	0.065187	-0.320325
7	-0.047931	0.179807	-1.028409	0.786557
8	0.606488	-0.829325	-0.231664	-0.829068
9	0.321457	0.953504	0.155475	-0.598032

df

	A	B	C
0	-0.263586	-0.194690	-0.341931
1	0.235225	-0.469747	2.135652
2	0.297726	-0.020956	0.599596
3	-1.001315	-2.050571	-1.213695
4	-0.778979	0.970515	-1.358900
5	-0.308514	0.293263	0.947084
6	1.701105	-0.799807	0.190508

df2

$$dfc = df + df2$$

	A	B	C	D
0	-1.061035	0.333230	-2.648705	NaN
1	1.156833	1.314816	3.007444	NaN
2	1.418745	1.864789	0.590355	NaN
3	-0.612079	-2.614801	-1.013031	NaN
4	-1.092253	2.128306	-2.692949	NaN
5	0.597597	1.228613	1.080511	NaN
6	2.983283	-0.791147	0.255695	NaN
7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN

# Data alignment and arithmetic

- When doing an operation between DataFrame and Series, the default behavior is to align the Series **index** on the DataFrame **columns**, thus **broadcasting** row-wise.

```
d = {  
    "col1": [1, 2, 3],  
    "col2": [4, 5, 6],  
}  
index = ["a", "b", "c"]  
df = pd.DataFrame(data=d, index=index)  
df
```

	col1	col2
a	1	4
b	2	5
c	3	6

```
df - df.iloc[0]
```

	col1	col2
a	0	0
b	1	1
c	2	2

# Data alignment and arithmetic

`df * 5 + 2`

	col1	col2
<b>a</b>	7	22
<b>b</b>	12	27
<b>c</b>	17	32

`1 / df`

	col1	col2
<b>a</b>	1.000000	0.250000
<b>b</b>	0.500000	0.200000
<b>c</b>	0.333333	0.166667

`df ** 4`

	col1	col2
<b>a</b>	1	256
<b>b</b>	16	625
<b>c</b>	81	1296

# Data alignment and arithmetic

- Boolean operators

```
df1 = pd.DataFrame({"a": [1, 0, 1], "b": [0, 1, 1]}, dtype=bool)  
df1
```

	a	b
0	True	False
1	False	True
2	True	True

```
df2 = pd.DataFrame({"a": [0, 1, 1], "b": [1, 1, 0]}, dtype=bool)  
df2
```

	a	b
0	False	True
1	True	True
2	True	False

# Data alignment and arithmetic

- Boolean operators

```
print(df1 & df2)  
print(df1 | df2)  
print(df1 ^ df2)  
print(~df1)
```

	a	b
0	False	False
1	False	True
2	True	False

	a	b
0	True	True
1	True	True
2	True	True

	a	b
0	True	True
1	True	False
2	False	True

	a	b
0	False	True
1	True	False
2	False	False



# Transposing

```
d = {  
    "col1": [1, 2, 3],  
    "col2": [4, 5, 6],  
}  
index = ["a", "b", "c"]  
df = pd.DataFrame(data=d, index=index)  
df
```

```
df.T
```

	col1	col2
a	1	4
b	2	5
c	3	6

	a	b	c
col1	1	2	3
col2	4	5	6

# groupby() method

- Group DataFrame using a mapper or by a Series of columns

```
df2 = pd.DataFrame({'X' : ['B', 'B', 'A', 'A'], 'Y' : [1, 2, 3, 4]})
print(df2)
df2.groupby(['X']).sum()
```

	X	Y
0	B	1
1	B	2
2	A	3
3	A	4

	Y
X	
A	7
B	3

```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
                              'Parrot', 'Parrot'],
                  'Max Speed': [380., 370., 24., 26.]})
print(df)
df.groupby(['Animal']).mean()
```

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0
2	Parrot	24.0
3	Parrot	26.0

	Max Speed
Animal	
Falcon	375.0
Parrot	25.0

# groupby() method

```
df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',  
                             'Parrot', 'Parrot'],  
                  'Max Speed': [380., 370., 24., 26.]})  
print(df)  
df.groupby(['Animal']).get_group('Falcon')
```

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0
2	Parrot	24.0
3	Parrot	26.0

	Animal	Max Speed
0	Falcon	380.0
1	Falcon	370.0

# Tìm hiểu thêm

- Join, Merge và Concatenate

# FILE CSV/Excel

- Read file csv file: `pandas.read_csv()`
- [https://pandas.pydata.org/docs/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html)

`pandas.read_csv(filepath_or_buffer, sep=NoDefault.no_default, delimiter=None, header='infer', names=NoDefault.no_default, index_col=None, usecols=None, squeeze=None, prefix=NoDefault.no_default, mangle_dupe_cols=True, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skipinitialspace=False, skiprows=None, skipfooter=0, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, skip_blank_lines=True, parse_dates=None, infer_datetime_format=False, keep_date_col=False, date_parser=None, dayfirst=False, cache_dates=True, iterator=False, chunksize=None, compression='infer', thousands=None, decimal='.', line_terminator=None, quotechar='"', quoting=0, doublequote=True, escapechar=None, comment=None, encoding=None, encoding_errors='strict', dialect=None, error_bad_lines=None, warn_bad_lines=None, on_bad_lines=None, delim_whitespace=False, low_memory=True, memory_map=False, float_precision=None, storage_options=None)`[\[source\]](#)

# FILE CSV/Excel

- Read file csv: `pandas.read_csv()`

```
import pandas as pd  
df = pd.read_csv("file2.csv")  
print(df)
```

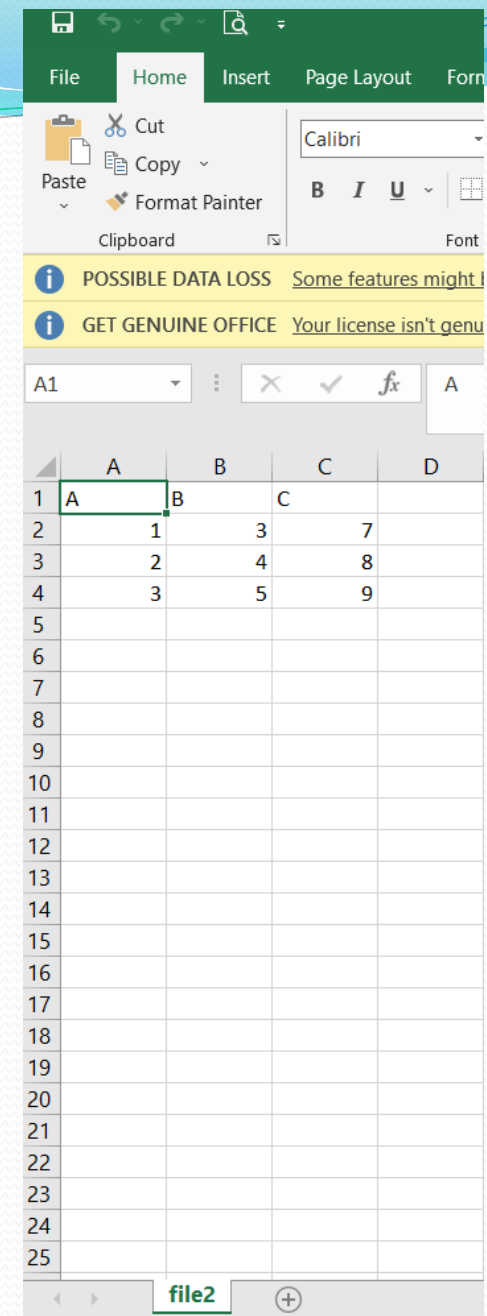
	A	B	C
0	1	3	7
1	2	4	8
2	3	5	9

	A	B	C	D
1	1	3	7	
2	2	4	8	
3	3	5	9	
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				

# FILE CSV/Excel

```
import pandas as pd
df = pd.read_csv("file2.csv",header=None)
df
```

	0	1	2
0	A	B	C
1	1	3	7
2	2	4	8
3	3	5	9



# FILE CSV/Excel

- Read file excel: `pandas.read_excel()`
- [https://pandas.pydata.org/docs/reference/api/pandas.read\\_excel.html](https://pandas.pydata.org/docs/reference/api/pandas.read_excel.html)

`pandas.read_excel(io, sheet_name=0, header=0, names=None, index_col=None, usecols=None, squeeze=None, dtype=None, engine=None, converters=None, true_values=None, false_values=None, skiprows=None, nrows=None, na_values=None, keep_default_na=True, na_filter=True, verbose=False, parse_dates=False, date_parser=None, thousands=None, decimal='.', comment=None, skipfooter=0, convert_float=None, mangle_dupe_cols=True, storage_options=None)`[\[source\]](#)

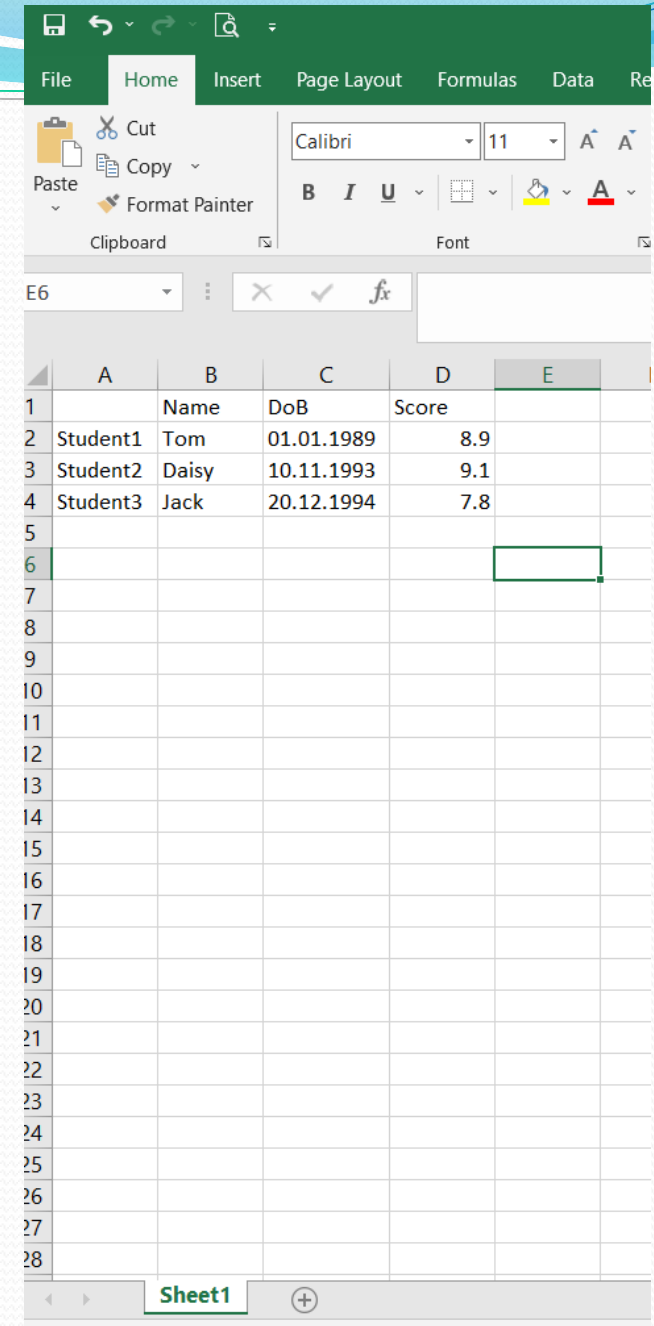


# FILE CSV/Excel

- Read file excel: `pandas.read_excel()`

```
import pandas as pd
df = pd.read_excel("file1.xlsx", "Sheet1")
df
```

Unnamed: 0	Name	DoB	Score	
0	Student1	Tom	01.01.1989	8.9
1	Student2	Daisy	10.11.1993	9.1
2	Student3	Jack	20.12.1994	7.8



The screenshot shows the Microsoft Excel interface. The 'Home' tab is selected in the ribbon. The font is set to Calibri, size 11. The spreadsheet contains data in columns A, B, C, and D, with row 1 as the header. The data is as follows:

	A	B	C	D	E
1		Name	DoB	Score	
2	Student1	Tom	01.01.1989	8.9	
3	Student2	Daisy	10.11.1993	9.1	
4	Student3	Jack	20.12.1994	7.8	
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					

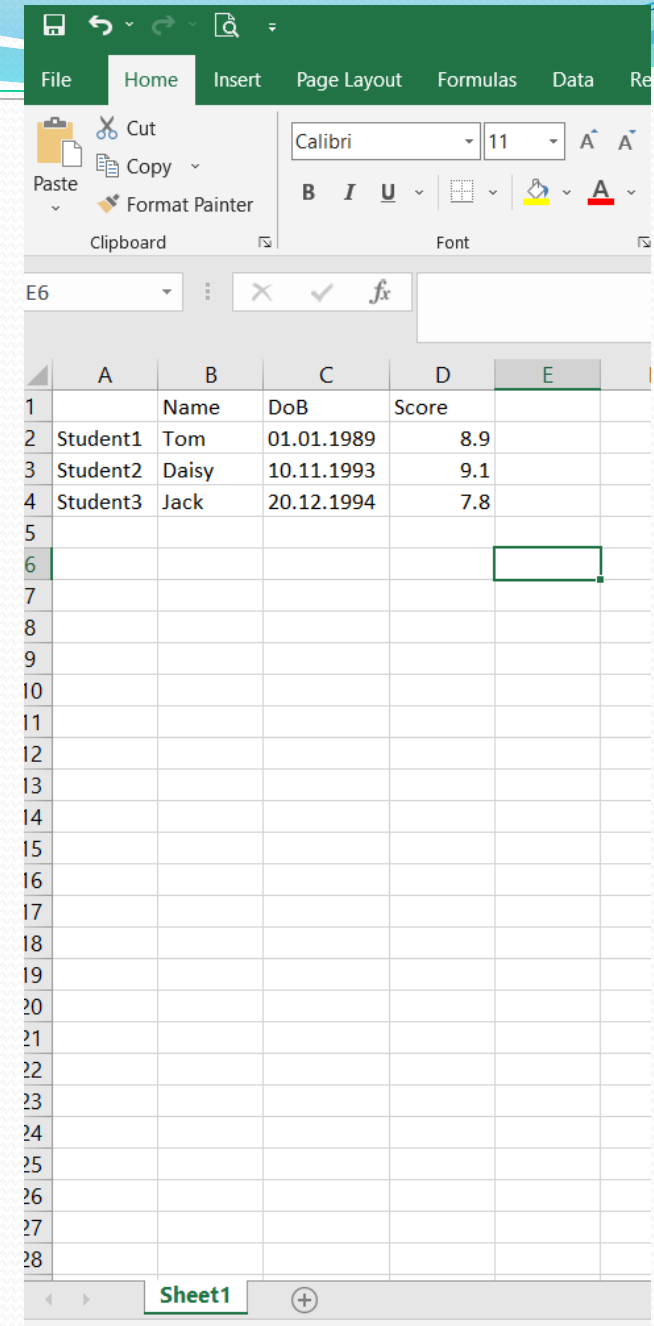
The sheet is named 'Sheet1' and is the only sheet visible in the bottom pane.

# FILE CSV/Excel

- Read file excel: `pandas.read_excel()`

```
import pandas as pd
df = pd.read_excel("file1.xlsx", "Sheet1", index_col=0)
df
```

	Name	DoB	Score
Student1	Tom	01.01.1989	8.9
Student2	Daisy	10.11.1993	9.1
Student3	Jack	20.12.1994	7.8



The screenshot shows the Microsoft Excel interface. The 'Home' tab is selected in the ribbon. The font is set to Calibri, size 11. The spreadsheet contains the following data:

	A	B	C	D	E
1		Name	DoB	Score	
2	Student1	Tom	01.01.1989	8.9	
3	Student2	Daisy	10.11.1993	9.1	
4	Student3	Jack	20.12.1994	7.8	
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					
23					
24					
25					
26					
27					
28					

The sheet is named 'Sheet1'.

```
import pandas as pd
df = pd.read_excel("file1.xlsx", "Sheet1", header=None)
print(df)
```

	0	1	2	3
0	NaN	Name	DoB	Score
1	Student1	Tom	01.01.1989	8.9
2	Student2	Daisy	10.11.1993	9.1
3	Student3	Jack	20.12.1994	7.8

```
import pandas as pd
df = pd.read_excel("file1.xlsx", "Sheet1", index_col=0, header=None)
print(df)
```

	1	2	3	
0	NaN	Name	DoB	Score
Student1	Tom	01.01.1989	8.9	
Student2	Daisy	10.11.1993	9.1	
Student3	Jack	20.12.1994	7.8	

# Làm việc với panel

- Panel được sử dụng nhiều trong kinh tế lượng
  - Dữ liệu có 3 trục:
    - Items (trục 0): mỗi item là một dataframe bên trong
    - Major axis (trục 1 –trục chính): các dòng
    - Minor axis (trục 2 –trục phụ): các cột
- Không được phát triển tiếp (thay bởi MultiIndex)
- Tìm hiểu thêm: <https://pandas.pydata.org/pandas-docs/version/0.24.0/reference/panel.html>



# Scipy

- SciPy chứa nhiều loại gói phụ giúp giải quyết vấn đề phổ biến nhất liên quan đến tính toán khoa học.
- Dễ sử dụng và hiểu cũng như sức mạnh tính toán nhanh.
- Có thể hoạt động trên mảng (array) của thư viện NumPy.
- Tên “SciPy” là viết tắt từ “Scientific Python”
- Để cài đặt module SciPy dùng lệnh:

`pip install scipy`

<https://scipy.github.io/devdocs/index.html>

# SciPy

- SciPy bao gồm nhiều gói khác nhau để thực hiện một loạt các chức năng. SciPy có các gói cho các yêu cầu cụ thể.
- SciPy có một gói dành riêng cho các hàm thống kê, đại số tuyến tính, phân cụm dữ liệu, xử lý hình ảnh và tín hiệu, cho ma trận, để tích hợp và phân biệt, v.v

Gói con	Miêu tả
cluster	Thuật toán phân cụm (Clustering Algorithms)
constants	Các hằng số toán học và vật lý
fftpack	Hàm biến đổi Fourier nhanh (Fast Fourier Transform)
integrate	Giải phương trình vi phân và tích phân
interpolate	Nội suy và làm mịn spline
io	Đầu vào và đầu ra
linalg	Đại số tuyến tính
ndimage	Xử lý ảnh N chiều
odr	Hồi quy khoảng cách trực giao
optimize	Tối ưu hóa và chương trình root-finding
signal	Xử lý tín hiệu
sparse	Ma trận sparse và các đoạn chương trình liên quan
spatial	Các cấu trúc dữ liệu không gian và thuật toán
special	Các hàm toán học đặc biệt
stats	Các hàm và phân phối thống kê





# Scikit-learn

- Scikit-learn xuất phát là một dự án trong một cuộc thi lập trình của Google vào năm 2007, người khởi xướng dự án là David Cournapeau
- Sau đó nhiều viện nghiên cứu và các nhóm ra nhập, đến năm 2010 mới có bản đầu tiên (v0.1 beta)
- Scikit-learn cung cấp gần như tất cả các loại thuật toán học máy cơ bản (khoảng vài chục) và vài trăm biến thể của chúng, cùng với đó là các kỹ thuật xử lý dữ liệu đã được chuẩn hóa
- Cài đặt: `pip install scikit-learn`