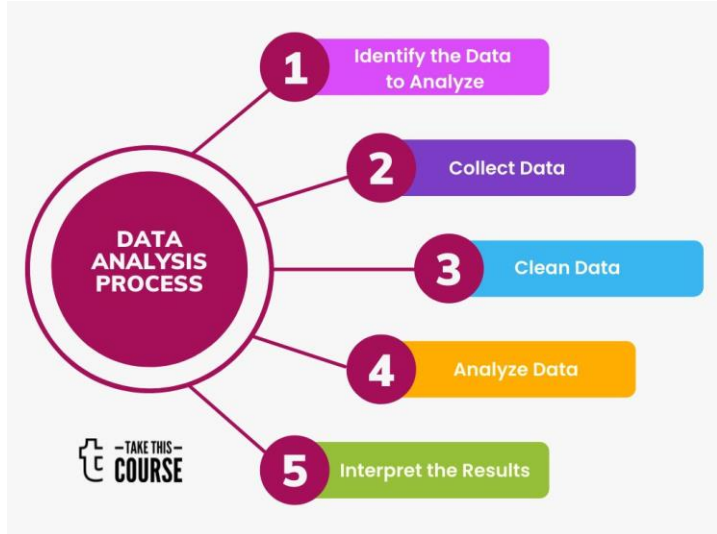


# PHÂN TÍCH DỮ LIỆU

(Data Analysis)



## THE DATA ANALYSIS PROCESS



Lê Văn Hạnh  
levanhhanhvn@gmail.com

# NỘI DUNG MÔN HỌC

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

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

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

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

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

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

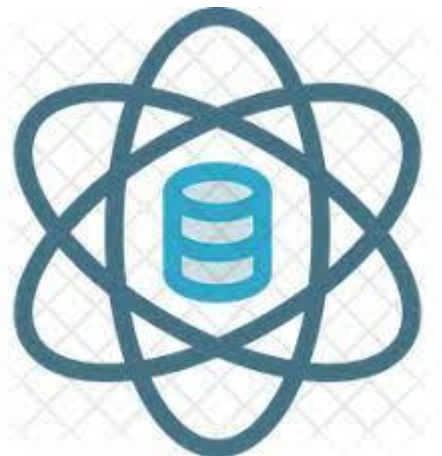
## PHẦN 2

# TIỀN XỬ LÝ DỮ LIỆU (*Data Preprocessing*)

## Chương 7

# LÀM SẠCH & CHUẨN BỊ DỮ LIỆU

## (*Data Cleaning & Preparation*)



Lê Văn Hạnh  
levanhanhvn@gmail.com

# NỘI DUNG CHƯƠNG 7

## 1. Xử lý dữ liệu bị thiếu

- Lọc ra dữ liệu bị thiếu
- Điền dữ liệu còn thiếu

## 2. Chuyển đổi dữ liệu

- Loại bỏ trùng lặp
- Chuyển đổi dữ liệu bằng hàm hoặc ánh xạ
- Thay thế giá trị
- Thay đổi tên cột
- Rời rạc hóa hoặc tách nhóm cho dữ liệu
- Phát hiện và lọc các ngoại lệ
- Hoán vị và lấy mẫu ngẫu nhiên
- Chỉ báo tính toán/Biến giả

## 3. Thao tác với chuỗi

- Các phương thức trên String
- Biểu thức chính quy
- Các hàm chuỗi được vector hóa trong pandas

# 1. XỬ LÝ DỮ LIỆU BỊ THIẾU (*Handling Missing Data*)

## 1.1. Dữ liệu bị thiếu trong Python và Pandas

- Thiếu dữ liệu xảy ra phổ biến trong nhiều ứng dụng phân tích dữ liệu. Một trong những mục tiêu của *pandas* là làm cho việc xử lý dữ liệu bị thiếu trở nên dễ dàng nhất có thể. Ví dụ: theo mặc định, tất cả thống kê mô tả về đối tượng *pandas* loại trừ dữ liệu bị thiếu.
- Đối với dữ liệu số, *pandas* sử dụng giá trị *NaN* (*Not A Number*) để biểu thị dữ liệu bị thiếu.

*Các phương thức xử lý đối với giá trị NA*

<i>Argument</i>	<i>Description</i>
<code>dropna</code>	Lọc các nhãn trục dựa trên việc các giá trị cho mỗi nhãn có bị thiếu dữ liệu hay không.
<code>fillna</code>	Điền vào dữ liệu còn thiếu một số giá trị hoặc sử dụng phương pháp nội suy ( <i>interpolation method</i> ) chẳng hạn như <code>'ffill'</code> hoặc <code>'bfill'</code> .
<code>isnull</code>	Trả về các giá trị boolean cho biết giá trị nào bị thiếu ( <i>missing</i> ) /NA.
<code>notnull</code>	Phủ định của <code>isnull</code> .

## 1. Xử lý dữ liệu bị thiếu (*Handling Missing Data*)

### 1.1. Dữ liệu bị thiếu trong Python và Pandas

- Ví dụ:

```
In [1]: import pandas as pd
.....: string_data = pd.Series(['Huế',
                                'Sài Gòn', np.nan, 'Hà Nội'])
In [2]: string_data
Out[2]:
0    Huế
1    Sài Gòn
2    NaN
3    Hà Nội
dtype: object
In [3]: string_data.isnull()
Out[3]:
0    False
1    False
2     True
3    False
dtype: bool
```

```
In [4]: string_data[0] = None
In [5]: string_data
Out[5]:
0    None
1    Sài Gòn
2    NaN
3    Hà Nội
dtype: object

In [6]: string_data.isnull()
Out[6]:
0     True
1    False
2     True
3    False
dtype: bool
In [7]: string_data.dropna()
Out[7]:
1    Sài Gòn
3    Hà Nội
dtype: object
```

## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.1. Dữ liệu bị thiếu trong Python và Pandas

- Trong *pandas*, gọi dữ liệu bị thiếu là NA (*Not Available* - không có sẵn). Trong các ứng dụng thống kê, dữ liệu **NA** có thể là dữ liệu không tồn tại hoặc tồn tại nhưng không được quan sát (ví dụ: thông qua các vấn đề về thu thập dữ liệu). Giá trị **None** trong *Python* cũng được coi là **NA** trong các mảng đối tượng
- Khi làm sạch dữ liệu để phân tích, điều quan trọng là phải thực hiện phân tích trên chính dữ liệu bị thiếu để xác định các vấn đề về thu thập dữ liệu hoặc các sai lệch tiềm ẩn trong dữ liệu do dữ liệu bị thiếu.

```
string_data
0    aardvark
1    artichoke
2         NaN
3    avocado
```

```
In [13]: string_data[0] = None
In [14]: string_data.isnull()
Out[14]:
0     True
1    False
2     True
3    False
dtype: bool
```

## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.2. Loại ra dữ liệu bị thiếu (*Filtering Out Missing Data*)

**1.2.1. Đối tượng *Series*:** hai phương thức `dropna` và `dropna` sẽ cùng trả về 1 loạt các giá trị dữ liệu khác `null` (kèm theo phía trước là chỉ mục):

```
In [8]: from numpy import nan as NA
In [9]: data = pd.Series([1, NA, 3.5, NA, 7])
In [10]: data.dropna()
Out[10]:
0    1.0
2    3.5
4    7.0
dtype: float64
In [11]: data[data.notnull()]
Out[11]:
0    1.0
2    3.5
4    7.0
dtype: float64
```



## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.2. Loại bỏ dữ liệu bị thiếu (*Filtering Out Missing Data*)

#### 1.2.2. Đối tượng *DataFrame*

- *Đối với các hàng:*

- Theo mặc định, `dropna` loại bỏ bất kỳ hàng nào chứa giá trị bị thiếu (NA):

```
In [12]: data = pd.DataFrame([[1., 6.5, 3.], [1., NA, NA],  
                             [NA, NA, NA], [NA, 6.5, 3.]])  
  
In [13]: cleaned = data.dropna()  
In [14]: data  
Out[14]:
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

```
In [15]: cleaned  
Out[15]:
```

	0	1	2
0	1.0	6.5	3.0

## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.2. Lọc ra dữ liệu bị thiếu (Filtering Out Missing Data)

#### 1.2.2. Đối tượng DataFrame

- Đối với các hàng:

- Theo mặc định, *dropna* loại bỏ bất kỳ hàng nào chứa giá trị bị thiếu (*NA*):

data			
	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0



```
In [16]: data.dropna(how='all')  
Out[16]:
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

- Đối số *thresh*: Một cách liên quan để lọc ra các hàng trong DataFrame có xu hướng liên quan đến dữ liệu chuỗi thời gian. Giả sử chỉ muốn giữ lại các hàng chứa một số quan sát nhất định.

```
In [23]: df.dropna()  
Out[23]:
```

	0	1	2
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437



df			
	0	1	2
0	0.737322	<NA>	<NA>
1	-0.262804	<NA>	<NA>
2	-0.081833	<NA>	2.197865
3	-1.689887	<NA>	-0.277556
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437



```
In [24]: df.dropna(thresh=2)  
Out[24]:
```

	0	1	2
2	-0.081833	<NA>	2.197865
3	-1.689887	<NA>	-0.277556
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437

## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.2. Lọc ra dữ liệu bị thiếu (*Filtering Out Missing Data*)

#### 1.2.2. Đối tượng *DataFrame*

- *Đối với các cột:*

- Tương tự, để xóa các cột chỉ chứa NA, truyền thêm tham số `axis = 1`:

```
In [17]: data[4] = NA
In [18]: data
Out[18]:
```

	0	1	2	4
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

```
In [19]: data.dropna(axis=1, how='all')
Out[19]:
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

1.2. Điền dữ liệu còn thiếu (Filling In Missing Data)

Thay vì lọc bỏ dữ liệu bị thiếu (và có khả năng loại bỏ dữ liệu khác cùng với dữ liệu đó), *pandas* có thể cho người dùng điền vào các “lỗ hổng” NA này bằng 1 giá trị nào đó thông qua phương thức `fillna`.

Các đối số của hàm *fillna*

Argument	Description
value	Điền các giá trị còn thiếu bằng giá trị vô hướng hoặc đối tượng giống dictionary
method	Nội suy giá trị sẽ thay thế cho các NA; giá trị mặc định là 'ffill' (nếu hàm được gọi không có đối số nào khác)
axis	Trục để điền giá trị vào; trục mặc định = 0.
inplace	Sửa đổi trực tiếp trên đối tượng gọi mà không tạo bản sao.
limit	Để điền giá trị theo hướng tiến hoặc lùi, số lượng liên tiếp tối đa cần điền.

## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.2. Điền dữ liệu còn thiếu (Filling In Missing Data)

- Đối với Series, có thể thay thế giá trị NA bằng giá trị trung bình (mean – trung bình số học) hoặc giá trị trung vị (median - là số nằm ở giữa một nhóm các số; có nghĩa là, phân nửa các số có giá trị lớn hơn số trung vị, còn phân nửa các số có giá trị bé hơn số trung vị):

```
In [20]: data = pd.Series([1., NA, 3.5, NA, 7])
In [21]: data.fillna(data.mean())
Out[21]:
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
dtype: float64
```

## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.2. Điền dữ liệu còn thiếu (Filling In Missing Data)

- Gọi `fillna` bằng một hằng số sẽ thay thế các giá trị bị thiếu bằng giá trị đó

df	0	1	2
0	0.737322	<NA>	<NA>
1	-0.262804	<NA>	<NA>
2	-0.081833	<NA>	2.197865
3	-1.689887	<NA>	-0.277556
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437



```
In [20]: df.fillna(0)
Out[20]:
```

	0	1	2
0	0.737322	0.0000	0.0000
1	-0.262804	0.0000	0.0000
2	-0.081833	0.0000	2.197865
3	-1.689887	0.0000	-0.277556
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437

- Gọi `fillna` với tham số là một dictionary, sẽ điền giá trị khác nhau cho mỗi cột.

df	0	1	2
0	0.737322	<NA>	<NA>
1	-0.262804	<NA>	<NA>
2	-0.081833	<NA>	2.197865
3	-1.689887	<NA>	-0.277556
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437



```
In [21]: df.fillna({1: 0.5, 2: 0})
Out[21]:
```

	0	1	2
0	0.737322	0.5000	0.0000
1	-0.262804	0.5000	0.0000
2	-0.081833	0.5000	2.197865
3	-1.689887	0.5000	-0.277556
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437

## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.2. Điền dữ liệu còn thiếu (Filling In Missing Data)

- Mặc định, `fillna` trả về một đối tượng mới, nhưng có thể sửa đổi đối tượng hiện có tại chỗ với tham số **`inplace=True`**

df	0	1	2
0	0.737322	<NA>	<NA>
1	-0.262804	<NA>	<NA>
2	-0.081833	<NA>	2.197865
3	-1.689887	<NA>	-0.277556
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437



```
In [22]: _ = df.fillna(0, inplace=True)
In [23]: df
Out[23]:
```

	0	1	2
0	0.737322	0.0000	0.0000
1	-0.262804	0.0000	0.0000
2	-0.081833	0.0000	2.197865
3	-1.689887	0.0000	-0.277556
4	-0.251734	-1.172459	0.410340
5	1.465266	-0.711997	2.743401
6	-0.596683	0.426551	1.233437

- `ffill` sẽ lấy giá trị khác NA của hàng liền trước và có cùng cột để điền cho giá trị NA hiện tại

df	0	1	2
0	0.429140	-0.747447	1.494652
1	-1.507580	0.309354	0.436922
2	-0.567046	<NA>	0.300925
3	1.316844	<NA>	-0.108739
4	0.695297	<NA>	<NA>
5	0.399554	<NA>	<NA>
6	-1.586771	<NA>	<NA>



```
In [27]: df.fillna(method='ffill')
Out[27]:
```

	0	1	2
0	0.429140	-0.747447	1.494652
1	-1.507580	0.309354	0.436922
2	-0.567046	0.309354	0.300925
3	1.316844	0.309354	-0.108739
4	0.695297	0.309354	-0.108739
5	0.399554	0.309354	-0.108739
6	-1.586771	0.309354	-0.108739

## 1. Xử lý dữ liệu bị thiếu (Handling Missing Data)

### 1.2. Điền dữ liệu còn thiếu (Filling In Missing Data)

- Thêm đối số `limit` để giới hạn số hàng sẽ điền

df	0	1	2
0	0.429140	-0.747447	1.494652
1	-1.507580	0.309354	0.436922
2	-0.567046	<NA>	0.300925
3	1.316844	<NA>	-0.108739
4	0.695297	<NA>	<NA>
5	0.399554	<NA>	<NA>
6	-1.586771	<NA>	<NA>



```
In [28]: df.fillna(method='ffill', limit=2)
Out[28]:
```

	0	1	2
0	0.429140	-0.747447	1.494652
1	-1.507580	0.309354	0.436922
2	-0.567046	0.309354	0.300925
3	1.316844	0.309354	-0.108739
4	0.695297	<NA>	-0.108739
5	0.399554	<NA>	-0.108739
6	-1.586771	<NA>	<NA>




## 2. CHUYỂN ĐỔI DỮ LIỆU (*Data Transformation*)

### 2.1. Loại bỏ trùng lặp (*Removing Duplicates*)

- Phương thức *uplicated* của DataFrame trả về một Series boolean cho biết các hàng có trùng lặp hay không? Lưu ý hàng đầu tiên trong những hàng bị trùng vẫn được xem là False.


data		
	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4



In [47]: data.duplicated()		
Out[47]:		
0	False	
1	False	
2	False	
3	False	
4	False	
5	False	
6	True	
dtype: bool		

- Phương thức *drop\_duplicates* trả về một DataFrame chứa các hàng có giá trị trong mảng *duplicates* là False

data		
	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4
6	two	4



In [48]: data.drop_duplicates()		
Out[48]:		
	k1	k2
0	one	1
1	two	1
2	one	2
3	two	3
4	one	3
5	two	4

## 2. Chuyển đổi dữ liệu (Data Transformation)

### 2.1. Loại bỏ trùng lặp (Removing Duplicates)

- Theo mặc định, cả hai phương thức *drop\_duplicates* và *drop\_duplicates*:
  - Điều xem xét dữ liệu trùng nhau tất cả các cột; Tuy nhiên, có thể chỉ định bất kỳ tập hợp con nào của chúng để phát hiện dữ liệu trùng trên tập con đó.

Trong ví dụ sau, giả sử chỉ muốn lọc các giá trị trùng lặp dựa trên cột 'k2':

- Ưu tiên cho giá trị được quan sát đầu tiên. Nếu thêm tham số *keep='last'* sẽ trả về giá trị cuối cùng

```
In [51]: df1 = data.drop_duplicates(['k1', 'k2'])
.....: df1
Out[51]:
```

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	6



data			
	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
5	two	4	5
6	two	4	6



```
In [52]: df1 = data.drop_duplicates(['k1', 'k2'], keep='last')
.....: df1
Out[52]:
```

	k1	k2	v1
0	one	1	0
1	two	1	1
2	one	2	2
3	two	3	3
4	one	3	4
6	two	4	6

## 2.2. Thêm cột vào DataFrame

- Thêm cột vào DataFrame thông qua Series:

```
# Khai báo 1 dictionary với key là giá trị của 1 field trong DataFrame'''
In [55]:meat_to_animal = { 'bacon': 'pig',
                           'pulled pork': 'pig',
                           'pastrami': 'cow',
                           'corned beef': 'cow',
                           'honey ham': 'pig',
                           'nova lox': 'salmon'
                           }
```

```
''' Xây dựng 1 Series dựa trên 1 field của DataFrame
In [56]: lowercased = data['food'].str.lower()
In [57]: lowercased
Out[57]:
```

	food
0	bacon
1	pulled pork
2	bacon
3	pastrami
4	corned beef
5	bacon
6	pastrami
7	honey ham
8	nova lox

Name: food, dtype: object

data	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	Pastrami	6.0
4	corned beef	7.5
5	Bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

Before



data	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	pastrami	6.0	cow
4	corned beef	7.5	cow
5	bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

After

```
''' Thêm 1 field vào DataFrame thông qua phương thức map của Series với giá trị được lấy từ Dictionary '''
In [58]: data['animal'] = lowercased.map(meat_to_animal)
In [59]: data
Out[59]:
```

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	pastrami	6.0	cow
4	corned beef	7.5	cow
5	bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

## 2. Chuyển đổi dữ liệu (Data Transformation)

### 2.2. Thêm cột vào DataFrame

- Với Dictionary và hàm lambda

''' B1: Khai báo 1 dictionary với key là giá trị của food và value là tên động vật trong DataFrame '''

```
In [55]: meat_to_animal = { 'bacon': 'pig',  
                             'pulled pork': 'pig',  
                             'pastrami': 'cow',  
                             'corned beef': 'cow',  
                             'honey ham': 'pig',  
                             'nova lox': 'salmon'  
                             }
```

data		
	food	ounces
0	bacon	4.0
1	pulled pork	3.0
2	bacon	12.0
3	Pastrami	6.0
4	corned beef	7.5
5	Bacon	8.0
6	pastrami	3.0
7	honey ham	5.0
8	nova lox	6.0

Before



data			
	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	pastrami	6.0	cow
4	corned beef	7.5	cow
5	bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

After

''' B2: Thêm 1 field vào DataFrame thông qua phương thức map và hàm lambda '''

```
In [60]: data['animal'] = data['food'].map(lambda x:  
                                             meat_to_animal[x.lower()])  
Out[60]:
```

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	pastrami	6.0	cow
4	corned beef	7.5	cow
5	bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

## 2.3. Thay thế giá trị với phương thức `replace`

- Mặc dù phương thức `fillna` giúp điền vào dữ liệu còn thiếu; `map` có thể được sử dụng để sửa đổi một tập hợp con các giá trị trong một đối tượng nhưng `replace` cung cấp một cách đơn giản và linh hoạt hơn để thực hiện việc đó.
- Theo mặc định, phương thức `replace`, tạo ra một `Series` mới. Nếu dùng thêm đối số `inplace=True` sẽ tác động trực tiếp lên đối tượng hiện tại.
- Một số ví dụ:
  - Thay `1` giá trị cũ bằng `1` một giá trị mới

```
mySeries
0      1.0
1   -999.0
2      2.0
3   -999.0
4  -1000.0
5      3.0
dtype: float64
```



```
In [62]: mySeries.replace(-999, np.nan)
Out[62]:
0      1.0
1      NaN
2      2.0
3      NaN
4  -1000.0
5      3.0
dtype: float64
```

### 2.3. Thay thế giá trị với phương thức replace

- Một số ví dụ:

- Thay  $n$  giá trị cũ bằng 1 một giá trị mới:

```
In [63]: mySeries.replace([-999, -1000], np.nan)
Out[63]:
```

0	1.0
1	NaN
2	2.0
3	NaN
4	NaN
5	3.0

dtype: float64

*Thay  $n$  giá trị cũ bằng 1 một giá trị mới*

- Thay  $n$  giá trị cũ bằng  $n$  một giá trị mới.

- Sử dụng list:

```
mySeries
```

0	1.0
1	-999.0
2	2.0
3	-999.0
4	-1000.0
5	3.0

dtype: float64

```
In [64]: mySeries.replace([-999, -1000],
                          [np.nan, 0])
Out[64]:
```

0	1.0
1	NaN
2	2.0
3	NaN
4	0.0
5	3.0

dtype: float64

*Sử dụng list:*

- Sử dụng Dictionary

```
In [65]: mySeries.replace({-999: np.nan, -1000: 0})
Out[65]:
```

0	1.0
1	NaN
2	2.0
3	NaN
4	0.0
5	3.0

dtype: float64

*Sử dụng Dictionary:*

**Ghi chú:** Phương thức **data.replace** khác với **data.str.replace**, phương thức này thực hiện thay thế chuỗi theo từng phần tử.

## 2.4. Thay đổi tên chỉ mục và tên cột

- Tên của chỉ mục và tên cột

- *Lấy tên theo mặc định:*

```
#Lấy tên cột và index theo mặc định
```

```
In [66]: data = pd.DataFrame(np.arange(12).reshape((3, 4)))
```

```
.....: df
```

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

```
#Tự chỉ định tên cột và index
```

```
In [67]: data = pd.DataFrame(np.arange(12).reshape((3, 4)),  
                             index=['Ohio', 'Colorado', 'New York'],  
                             columns=['one', 'two', 'three', 'four'])
```

```
.....: df
```

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

- Giống như một *Series*, các chỉ mục trục cũng có phương thức *map*. Có thể gán tên mới cho index của *DataFrame* thông qua hàm lambda:

```
In [68]: transform = lambda x: x[:4].upper()
```

```
In [69]: data.index.map(transform)
```

```
Out[69]: Index(['OHIO', 'COLO', 'NEW'], dtype='object')
```

## 2.4. Thay đổi tên chỉ mục và tên cột

- Giống như một *Series*, các chỉ mục trục cũng có phương thức *map*:

```
In [68]: transform = lambda x: x[:4].upper()  
In [69]: data.index.map(transform)  
Out[69]: Index(['OHIO', 'COLO', 'NEW'], dtype='object')
```

- Có thể gán tên mới cho index của *DataFrame* thông qua hàm lambda:

```
In [70]: data.index = data.index.map(transform)  
In [71]: data  
Out[71]:
```

	one	two	three	four
OHIO	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11



### 2.4. Thay đổi tên chỉ mục và tên cột

#### - Phương thức **rename**:

- Sử dụng khi muốn tạo phiên bản đã chuyển đổi của tập dữ liệu mà không sửa đổi bản gốc

```
In [72]: data.rename(index=str.title, columns=str.upper)
Out[72]:
```

	ONE	TWO	THREE	FOUR
Ohio	0	1	2	3
Colo	4	5	6	7
New	8	9	10	11

- rename** có thể được sử dụng cùng với một đối tượng giống như *dictionary* cung cấp các giá trị mới cho một tập hợp con của các tên chỉ mục và tên cột:

```
In [73]: data.rename(index={'OHIO': 'INDIANA'},
                      columns={'three': 'peekaboo'})
Out[73]:
```

	ONE	TWO	peekaboo	FOUR
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

- Nếu muốn sửa đổi tập dữ liệu tại chỗ, hãy thêm đối số **inplace=True**:

```
In [74]: data.rename(index={'OHIO': 'INDIANA'}, inplace=True)
In [75]: data
Out[75]:
```

	one	two	three	four
INDIANA	0	1	2	3
COLO	4	5	6	7
NEW	8	9	10	11

## 2.5. Rời rạc hóa hoặc tách nhóm cho dữ liệu (*Discretization and Binning*)

- Dữ liệu liên tục thường được rời rạc hóa (discretized) hoặc được tách thành các “nhóm” (bins) để phân tích.
- Các bước để phân nhóm dữ liệu
  - B1: Khai báo list chứa dữ liệu
  - B2: Khai báo list chứa cận dưới của các nhóm
  - B3: Phân chia nhóm cho từng dữ liệu dựa trên phương thức cut

```
''' B1: Khai báo list chứa dữ liệu
In [75]: ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
''' B2: Khai báo list chứa cận dưới của các nhóm từ 18-25, 26-35,
36-60 và 60 trở lên'''
In [76]: bins = [18, 25, 35, 60, 100]
''' B3: Phân chia nhóm cho từng dữ liệu dựa trên phương thức cut
In [77]: cats = pd.cut(ages, bins)
In [78]: cats
Out[78]:
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25,
35], (60, 100], (35, 60], (35, 60], (25, 35]]
Length: 12
Categories (4, interval[int64]): [(18, 25] < (25, 35] < (35,
60] < (60, 100]]
```

## 2. Chuyển đổi dữ liệu (Data Transformation)

### 2.5. Rời rạc hóa hoặc tách nhóm cho dữ liệu (Discretization and Binning)

#### 2.5.1. Phân chia nhóm theo miền giá trị (range) được định trước:

- Ví dụ:

```
# Xem nhóm mà mỗi giá trị thuộc về
In [79]: cats.codes
Out[79]: array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
# Xem danh sách các nhóm
In [80]: cats.categories
Out[80]:
IntervalIndex([(18, 25], (25, 35], (35, 60], (60, 100]],
              dtype='interval[int64, right]')
# Đếm số lượng phần tử có trong mỗi nhóm thuộc về
In [81]: pd.value_counts(cats)
Out[81]:
(18, 25]  5
(35, 60]  3
(25, 35]  3
(60, 100] 1
dtype: int64
```

## 2. Chuyển đổi dữ liệu (Data Transformation)

### 2.5. Rời rạc hóa hoặc tách nhóm cho dữ liệu (Discretization and Binning)

#### 2.5.1. Phân chia nhóm theo miền giá trị (range) được định trước:

- Phù hợp với ký hiệu toán học cho các khoảng, dấu ngoặc đơn có nghĩa là cạnh đang mở (*open* - không bao gồm giá trị đi kèm), trong khi dấu ngoặc vuông có nghĩa là cạnh đó đã đóng (*closed* - bao gồm giá trị đi kèm). Có thể thay đổi ký hiệu đóng mở này bằng cách thêm đối số *right=false* khi gọi phương thức *cut*:

```
In [82]: pd.cut(ages, [18, 26, 36, 61, 100], right=False)
Out[82]:
[[18, 26), [18, 26), [18, 26), [26, 36), [18, 26), ..., [26,
36), [61, 100), [36, 61), [36, 61), [26, 36)]
Length: 12
Categories (4, interval[int64]): [[18, 26) < [26, 36) < [36,
61) < [61, 100)]

In [83]: pd.cut(ages, [18, 26, 36, 61, 100])
Out[83]:
[(18, 26], (18, 26], (18, 26], (26, 36], (18, 26], ..., (26,
36], (61, 100], (36, 61], (36, 61], (26, 36]]
Length: 12
Categories (4, interval[int64]): [[18, 26) < [26, 36) < [36,
61) < [61, 100)]
[(18, 26], (18, 26], (18, 26], (26, 36], (18, 26], ..., (26,
36], (36, 61], (36, 61], (36, 61], (26, 36]]
Length: 12
Categories (4, interval[int64, right]): [(18, 26] < (26, 36]
< (36, 61] < (61, 100]]
```

## 2. Chuyển đổi dữ liệu (*Data Transformation*)

### 2.5. Rời rạc hóa hoặc tách nhóm cho dữ liệu (*Discretization and Binning*)

#### 2.5.1. Phân chia nhóm theo miền giá trị (range) được định trước: của giá trị

- Có thể đặt tên cho các nhóm bằng cách chuyển 1 list hoặc array chứa tên các nhóm:

```
In [84]: group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
In [85]: pd.cut(ages, bins, labels=group_names)
Out[85]:
[Youth, Youth, Youth, YoungAdult, Youth, ..., YoungAdult, Senior,
MiddleAged, MiddleAged, YoungAdult]
Length: 12
Categories (4, object): [Youth < YoungAdult < MiddleAged < Senior]
```

## 2. Chuyển đổi dữ liệu (Data Transformation)

### 2.5. Rời rạc hóa hoặc tách nhóm cho dữ liệu (Discretization and Binning)

#### 2.5.2. Phân chia nhóm có độ dài bằng nhau:

- Cú pháp: `pd.cut (data, NumOfGroup, precision=num)`

Trong đó:

- **data**: dữ liệu
- **NumOfGroup**: số lượng nhóm cần chia
- **num**: số lượng số lẻ khi chia khoảng cho các giá trị

```
In [86]: data = np.random.rand(20)
. . . .: data
Out[86]:
array([0.2421382 , 0.30857392, 0.50532502, 0.94626665, 0.74554511,
        0.20856715, 0.46337254, 0.49201135, 0.17905073, 0.01055192,
        0.6040847 , 0.74642177, 0.81967514, 0.48098384, 0.41006619,
        0.50697759, 0.16292414, 0.24717806, 0.1378857, 0.15032248])

In [87]: pd.cut(data, 4, precision=2)
Out[87]:
[(0.0096, 0.24], (0.24, 0.48], (0.48, 0.71], (0.71, 0.95],
 (0.71, 0.95], ..., (0.48, 0.71], (0.0096, 0.24], (0.24,
 0.48], (0.0096, 0.24], (0.0096, 0.24]]
Length: 20
Categories (4, interval[float64, right]): [(0.0096, 0.24] <
(0.24, 0.48] < (0.48, 0.71] < (0.71, 0.95]]
```

## 2. Chuyển đổi dữ liệu (*Data Transformation*)

### 2.5. Rời rạc hóa hoặc tách nhóm cho dữ liệu (*Discretization and Binning*)

#### 2.5.3. Phân chia nhóm với hàm `qcut`:

- Tùy thuộc vào sự phân bố dữ liệu, việc sử dụng `cut` thường không dẫn đến việc mỗi *bins* sẽ có cùng số điểm dữ liệu.
- Đặc điểm của hàm `qcut`:
  - Sử dụng lượng tử mẫu, nên theo định nghĩa, sẽ thu được các *bins* có kích thước gần bằng nhau.
  - Hàm `qcut` sẽ tạo thành các nhóm có kích thước gần bằng nhau, do đó miền giá trị của các nhóm sẽ không đồng đều.

```
# Normally distributed
In [88]: data = np.random.randn(10)
. . . .: data
Out[88]:
array([-1.27324447,  1.95428508,
        -0.16148546,  0.41486539,
         1.14380234, -1.71812218,
         0.13239404, -0.94740836,
         0.02044638, -0.17033493])
```

```
In [89]: cats = pd.qcut(data, 4) # Cut into quartiles
In [90]: cats
Out[90]:
[(-1.73, -0.75], (0.34, 1.95], (-0.75, -0.071], (0.34, 1.95],
 (0.34, 1.95], (-1.73, -0.75], (-0.071, 0.34], (-1.73, -0.75],
 (-0.071, 0.34], (-0.75, -0.071]]
Categories (4, interval[float64, right]): [(-1.73, -0.75] <
(-0.75, -0.071] < (-0.071, 0.34] < (0.34, 1.95]]
In [91]: pd.value_counts(cats)
Out[91]: |
(-1.73, -0.75]      3
(0.34, 1.95]        3
(-0.75, -0.071]    2
(-0.071, 0.34]     2
dtype: int64
```

## 2. Chuyển đổi dữ liệu (*Data Transformation*)

### 2.5. Rời rạc hóa hoặc tách nhóm cho dữ liệu (*Discretization and Binning*)

#### 2.5.3. Phân chia nhóm với hàm **qcut**:

- Tương tự như phương thức **cut**, có thể chuyển các giá trị của riêng mình (bao gồm các số từ 0 đến 1) cho phương thức **qcut**:

```
In [92]: pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.], precision=2)
Out[92]:
[(-1.32, -0.071], (1.22, 1.95], (-1.32, -0.071], (-0.071, 1.22], (-0.071, 1.22], (-1.73, -1.32], (-0.071, 1.22], (-1.32, -0.071], (-0.071, 1.22], (-1.32, -0.071]]
Categories (4, interval[float64, right]): [(-1.73, -1.32] < (-1.32, -0.071] < (-0.071, 1.22] < (1.22, 1.95]]
```



## 2.6. Phát hiện và lọc các ngoại lệ (Detecting and Filtering Outliers)

- Lọc hoặc chuyển đổi các ngoại lệ phần lớn là vấn đề áp dụng các tác vụ trên mảng.
- Giả sử muốn tìm các giá trị ở cột thứ 2 mà giá trị tuyệt đối của chúng  $> 3$

```
In [92]: data = pd.DataFrame(np.random.randn(1000, 4))
In [93]: data.describe()
Out[93]:
```

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.049091	0.026112	-0.002544	-0.051827
std	0.996947	1.007458	0.995232	0.998311
min	-3.645860	-3.184377	-3.745356	-3.428254
25%	-0.599807	-0.612162	-0.687373	-0.747478
50%	0.047101	-0.013609	-0.022158	-0.088274
75%	0.756646	0.695298	0.699046	0.623331
max	2.653656	3.525865	2.735527	3.366626

```
In [94]: col = data[2]
In [95]: col[np.abs(col) > 3]
Out[95]:
```

41	-3.399312
136	-3.745356

Name: 2, dtype: float64

### 2.6. Phát hiện và lọc các ngoại lệ (Detecting and Filtering Outliers)

- Để chọn tất cả các hàng có chứa giá trị tuyệt đối  $> 3$ , có thể sử dụng phương thức `any(1)` trên `DataFrame`:

```
In [96]: data[(np.abs(data) > 3).any(1)]
```

```
Out[96]:
```

	0	1	2	3
41	0.457246	-0.025907	-3.399312	-0.974657
60	1.951312	3.260383	0.963301	1.201206
136	0.508391	-0.196713	-3.745356	-1.520113
235	-0.242459	-3.056990	1.918403	-0.578828
258	0.682841	0.326045	0.425384	-3.428254
322	1.179227	-3.184377	1.369891	-1.074833
544	-3.548824	1.553205	-2.186301	1.277104
635	-0.578093	0.193299	1.397822	3.366626
782	-0.207434	3.525865	0.283070	0.544635
803	-3.645860	0.255475	-0.549574	-1.907459

## 2.6. Phát hiện và lọc các ngoại lệ

- **`np.sign(data)`** tạo ra các giá trị **1** và **-1** dựa trên việc các giá trị trong **`data`** là dương hay âm:

```
In [99]: np.sign(data).head()
Out[99]:
```

	0	1	2	3
0	-1.0	1.0	-1.0	1.0
1	1.0	-1.0	1.0	-1.0
2	1.0	1.0	1.0	-1.0
3	-1.0	-1.0	1.0	-1.0
4	-1.0	1.0	-1.0	-1.0

- Có thể sử dụng **`np.sign(data)`** để giới hạn các giá trị bên ngoài khoảng **-3** đến **3**

```
In [97]: data[np.abs(data) > 3] = np.sign(data) * 3
In [98]: data.describe()
Out[98]:
```

	0	1	2	3
count	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.050286	0.025567	-0.001399	-0.051765
std	0.992920	1.004214	0.991414	0.995761
min	-3.000000	-3.000000	-3.000000	-3.000000
25%	-0.599807	-0.612162	-0.687373	-0.747478
50%	0.047101	-0.013609	-0.022158	-0.088274
75%	0.756646	0.695298	0.699046	0.623331
max	2.653656	3.000000	2.735527	3.000000

## 2.7. Hoán vị và lấy mẫu ngẫu nhiên (Permutation and Random Sampling)

- Hàm **`numpy.random.permutation`**:
  - Thực hiện hoán vị (sắp xếp lại ngẫu nhiên) một *Series* hoặc các hàng trong *DataFrame* có thể thực hiện bằng cách sử dụng.
  - Gọi **`permutation`** với số lượng giá trị lấy trong *DataFrame* mà ta muốn hoán vị (tính từ trái sang phải và từ trên xuống dưới) . Kết quả sẽ tạo ra một mảng các số nguyên biểu thị thứ tự mới:
- Hàm **`take`** (tương đương **`iloc`**): dùng để lập chỉ mục. Có thể sử dụng hàm này để lập chỉ mục dựa trên mảng vừa thu được

```
df
```

	0	1	2	3
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15
4	16	17	18	19

```
In [102]: sampler = np.random.permutation(5)
In [103]: sampler
Out[103]: array([3, 1, 4, 2, 0])
```

```
In [104]: df.take(sampler)
Out[104]:
```

3	12	13	14	15
1	4	5	6	7
4	16	17	18	19
2	8	9	10	11
0	0	1	2	3

## 2.7. Hoán vị và lấy mẫu ngẫu nhiên (Permutation and Random Sampling)

- Phương thức **sample**:

- Dùng để chọn một tập hợp con ngẫu nhiên trên *Series* và *DataFrame* mà không cần thay thế:

```
In [105]: df.sample(n=3)
Out[105]:
```

	0	1	2	3
3	12	13	14	15
4	16	17	18	19
2	8	9	10	11

```
In [104]: df.take(sampler)
Out[104]:
```

	0	1	2	3
3	12	13	14	15
1	4	5	6	7
4	16	17	18	19
2	8	9	10	11
0	0	1	2	3

- Chuyển thêm đối số **replace=True** cho **sample** để tạo một mẫu cho phép các giá trị được lặp lại:

```
In [106]: choices = pd.Series([5, 7, -1, 6, 4])
In [107]: draws = choices.sample(n=10, replace=True)
In [108]: draws
Out[108]:
```

4	4
1	7
4	4
2	-1
0	5
3	6
4	4
0	5
4	4

dtype: int64

## 2.8. Chỉ báo tính toán/Biến giả (Computing Indicator/Dummy Variables)

- Một loại chuyển đổi khác cho các ứng dụng mô hình thống kê hoặc học máy là chuyển đổi một biến phân loại thành ma trận “giả” (“*dummy*” matrix) hoặc ma trận “chỉ báo” (“*indicator*” matrix).
- Hàm ***get\_dummies***:
  - Được sử dụng khi một cột trong *DataFrame* có k giá trị riêng biệt, ta cần lấy được một ma trận hoặc một *DataFrame* có k cột chỉ chứa giá trị **1** hoặc **0**.

```
In [109]: df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                             'data1': range(6)})
In [110]: df
Out[110]:
```

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	b	5

```
In [110]: pd.get_dummies(df['key'])
Out[110]:
```

	a	b	c
0	0	1	0
1	0	1	0
2	1	0	0
3	0	0	1
4	1	0	0
5	0	1	0

## 2.8. Chỉ báo tính toán/Biến giả (Computing Indicator/Dummy Variables)

### - Hàm `get_dummies`:

- Đối số **`prefix`**: được dùng khi muốn thêm tiền tố (*prefix*) vào các cột chỉ báo trong *DataFrame*, tiền tố này sau đó có thể được hợp nhất với dữ liệu khác:

```
In [111]: dummies = pd.get_dummies(df['key'], prefix='Key')
In [112]: df_with_dummy = df[['data1']].join(dummies)
In [113]: df_with_dummy
Out[113]:
```

	data1	Key_a	Key_b	Key_c
0	0	0	1	0
0	1	0	1	0
1	2	1	0	0
2	3	0	0	1
3	4	1	0	0
4	5	0	1	0

### 2.8. Chỉ báo tính toán/Biến giả (Computing Indicator/Dummy Variables)

- Khi một giá trị trên hàng trong 1 cột của *DataFrame* chứa nhiều giá trị:

```
# dữ liệu trong movies gồm rất nhiều dòng
```

```
In [116]: movies[:10]
```

```
Out[116]:
```

	movie_id		title	genres
0	1	Toy Story (1995)	Animation Children's Comedy	
1	2	Jumanji (1995)	Adventure Children's Fantasy	
2	3	Grumpier Old Men (1995)	Comedy Romance	
3	4	Waiting to Exhale (1995)	Comedy Drama	
4	5	Father of the Bride Part II (1995)	Comedy	
5	6	Heat (1995)	Action Crime Thriller	
6	7	Sabrina (1995)	Comedy Romance	
7	8	Tom and Huck (1995)	Adventure Children's	
8	9	Sudden Death (1995)	Action	
9	10	GoldenEye (1995)	Action Adventure Thriller	

```
# B1: trích danh sách các thể loại
```

```
In [117]: all_genres = []
```

```
In [118]: for x in movies.genres:
```

```
.....:     all_genres.extend(x.split('|'))
```

```
In [119]: genres = pd.unique(all_genres)
```

```
In [120]: genres # genres là 1 array
```

```
Out[120]: array(['Animation', 'Children's', 'Comedy', 'Adventure',  
                'Fantasy', 'Romance', 'Drama', 'Action', 'Crime',  
                'Thriller', 'Horror', 'Sci-Fi', 'Documentary', 'War',  
                'Musical', 'Mystery', 'Film-Noir', 'Western'],  
              dtype=object)
```



### 2.8. Chỉ báo tính toán/Biến giả

- Khi một giá trị trên hàng trong 1 cột của *DataFrame* chứa nhiều giá trị:

**Ghi chú:** Đối với dữ liệu lớn hơn, phương pháp xây dựng các biến chỉ báo (*indicator variables*) với nhiều thành viên này sẽ không thực hiện được nhanh. Sẽ tốt hơn nếu viết một hàm cấp thấp hơn để ghi trực tiếp vào một mảng NumPy, sau đó đưa kết quả vào một *DataFrame*.

```
# B2:tạo DataFrame chỉ chứa toàn giá trị zero
In [121]: zero_matrix = np.zeros((len(movies), len(genres)))
In [122]: dummies = pd.DataFrame(zero_matrix, columns=genres)
# B3:tính toán chỉ số cột cho từng thể loại phim
In [123]: gen = movies.genres[0]
In [124]: gen.split('|')
Out[124]: ['Animation', 'Children's', 'Comedy']
In [125]: dummies.columns.get_indexer(gen.split('|'))
Out[125]: array([0, 1, 2])
# B4:sử dụng iloc để đặt giá trị dựa trên các chỉ số
In [126]: for i, gen in enumerate(movies.genres):
            indices = dummies.columns.get_indexer(gen.split('|'))
            dummies.iloc[i, indices] = 1
# B5: thêm tiền tố 'Genres_' cho các thể loại
In [127]: movies_windic = movies.join(dummies.add_prefix('Genre_'))
In [128]: movies_windic.iloc[0]
Out[128]:
movie_id      1
title      Toy Story (1995)
genres      Animation|Children's|Comedy
Genre_Animation      1
Genre_Children's      1
Genre_Comedy      1
Genre_Adventure      0
Genre_Fantasy      0
Genre_Romance      0
Genre_Drama      0
...
```

2.8. Chỉ báo tính toán/Biến giả

- Kết hợp *get\_dummies* với một hàm rời rạc như *cut*

myArray
0.9296
0.3164
0.1839
0.2046
0.5677
0.5955
0.9645
0.6532
0.7489
0.6536

```
myArray
array([0.9296, 0.3164, 0.1839, 0.2046, 0.5677, 0.5955,
       0.9645, 0.6532, 0.7489, 0.6536])
In [132]: bins = [0, 0.2, 0.4, 0.6, 0.8, 1]
In [133]: pd.get_dummies(pd.cut(myArray, bins))
Out[133]:
```

	(0.0, 0.2]	(0.2, 0.4]	(0.4, 0.6]	(0.6, 0.8]	(0.8, 1.0]
0	0	0	0	0	1
1	0	1	0	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	1	0	0
6	0	0	0	0	1
7	0	0	0	1	0
8	0	0	0	1	0
9	0	0	0	1	0

# 3. THAO TÁC VỚI DỮ LIỆU KIỂU CHUỖI (String Manipulation)

## 3.1. Các phương thức trên String (String Object Methods)

Argument	Description
count	Trả về số lần xuất hiện chuỗi con trong chuỗi.
endswith	Trả về True nếu chuỗi kết thúc bằng hậu tố được chỉ ra.
startswith	Trả về True nếu chuỗi bắt đầu bằng tiền tố được chỉ ra.
join	Sử dụng chuỗi làm dấu phân cách để nối một loạt các chuỗi khác.
index	Trả về vị trí của ký tự đầu tiên trong chuỗi con nếu tìm thấy trong chuỗi; phát sinh ValueError nếu không tìm thấy.
find	Trả về vị trí đầu tiên của chuỗi con trong chuỗi; giống như index, nhưng trả về -1 nếu không tìm thấy.
rfind	Trả về vị trí ký tự đầu tiên của chuỗi con xuất hiện cuối cùng trong chuỗi; trả về -1 nếu không tìm thấy.
replace	Thay thế lần xuất hiện đầu tiên của chuỗi bằng một chuỗi khác.
strip, rstrip, lstrip	Cắt bớt khoảng trắng, bao gồm cả dòng mới (newlines); tương đương với x.strip() (và rstrip, lstrip tương ứng) cho từng phần tử.
split	Chia chuỗi thành danh sách các chuỗi con bằng cách sử dụng dấu phân cách đã truyền.
lower	Chuyển đổi các ký tự hoa thành chữ thường.
upper	Chuyển đổi các ký tự thường thành chữ hoa.
casefold	Chuyển đổi các ký tự thành chữ thường và chuyển đổi bất kỳ tổ hợp ký tự biến đổi theo vùng cụ thể nào thành dạng có thể so sánh thông dụng.
ljust, rjust	Căn trái hoặc căn phải chuỗi tương ứng; đệm phía đối diện của chuỗi bằng dấu cách (hoặc một số ký tự được chỉ định khác) để trả về một chuỗi có chiều dài được chỉ ra (trong danh sách đối số).

### 3. Thao tác với dữ liệu kiểu chuỗi (String Manipulation)

#### 3.1. Các phương thức trên String (String Object Methods)

```
# Dùng split để phân tách chuỗi ngăn cách nhau bởi dấu phẩy
In [134]: val = 'a,b, guido'
In [135]: val.split(',')
Out[135]: ['a', 'b', ' guido']
''' split thường được kết hợp với strip để cắt khoảng trắng thừa ở
2 đầu gồm cả ký tự ngắt dòng'''
In [136]: pieces = [x.strip() for x in val.split(',')]
In [137]: pieces
Out[137]: ['a', 'b', 'guido']
# Nối chuỗi bằng toán tử cộng (+)
In [138]: first, second, third = pieces
In [139]: first + '::' + second + '::' + third
Out[139]: 'a::b::guido'
# Nối chuỗi bằng phương thức join
In [140]: '::'.join(pieces)
Out[140]: 'a::b::guido'
''' Sử dụng toán tử in (hoặc not in) để tìm kiếm 1 chuỗi con trong
chuỗi lớn'''
In [141]: 'guido' in val
Out[141]: True
```

### 3. Thao tác với dữ liệu kiểu chuỗi (String Manipulation)

#### 3.1. Các phương thức trên String (String Object Methods)

```
''' Sử dụng phương thức index để tìm kiếm 1 chuỗi con trong chuỗi lớn. Sẽ phát sinh exception nếu không tìm thấy chuỗi con trong chuỗi lớn'''
In [142]: val.index(',')
Out[142]: 1
In [143]: val.find(':')
Out[143]: -1
In [144]: val.index(':')
-----
ValueError Traceback (most recent call last)
<ipython-input-144-280f8b2856ce> in <module>()
----> 1 val.index(':')
ValueError: substring not found
''' Phương thức count: dùng để đếm số lần xuất hiện của chuỗi con trong chuỗi lớn'''
In [145]: val.count(',')
Out[145]: 2
''' Phương thức replace: dùng để thay thế mẫu này bằng mẫu khác'''
In [146]: val.replace(',', ' :: ')
Out[146]: 'a :: b :: guido'
''' Có thể dùng replace để xóa các mẫu bằng một chuỗi rỗng'''
In [147]: val.replace(',', '')
Out[147]: 'ab guido'
```

## 3.2. Biểu thức chính quy (Regular Expressions)

- Một số phương thức dùng trong biểu thức chính quy

<i>Argument</i>	<i>Description</i>
<code>findall</code>	Trả về tất cả các mẫu khớp trong một chuỗi dưới dạng 1 list
<code>finditer</code>	Giống như <code>findall</code> , nhưng trả về một iterator
<code>match</code>	Khớp mẫu ở đầu chuỗi và tùy ý phân đoạn các thành phần mẫu thành các nhóm; nếu mẫu khớp, trả về một đối tượng khớp và nếu không sẽ trả về <code>None</code>
<code>search</code>	Quét chuỗi để so khớp với mẫu; trả về một đối tượng phù hợp nếu có; không giống như <code>match</code> , <code>match</code> có thể ở bất kỳ đâu trong chuỗi thay vì chỉ ở đầu
<code>split</code>	Chia chuỗi thành từng mảnh ở mỗi lần xuất hiện mẫu
<code>sub</code> , <code>subn</code>	Thay thế tất cả ( <code>sub</code> ) hoặc n lần xuất hiện ( <code>subn</code> ) đầu tiên của mẫu trong chuỗi bằng biểu thức thay thế; sử dụng các ký hiệu <code>\1</code> , <code>\2</code> , ... để chỉ các phần tử group khớp trong chuỗi thay thế



### 3. Thao tác với dữ liệu kiểu chuỗi (String Manipulation)

## 3.2. Biểu thức chính quy (Regular Expressions)

''' tách chuỗi có số ký tự khoảng trắng thay đổi (gồm tab, dấu cách và ký tự ngắt dòng). Biểu thức chính quy mô tả một hoặc nhiều ký tự khoảng trắng là `\s+`'''

```
In [148]: import re
```

```
In [149]: text = "VN    UK \t US \n FR"
```

```
In [150]: re.split('\s+', text)
```

```
Out[150]: ['VN', 'UK', 'US', 'FR']
```

''' có thể dùng `re.compile` để biên dịch biểu thức chính quy thành 1 đối tượng biểu thức chính quy để tái sử dụng sau này'''

```
In [151]: regex = re.compile('\s+')
```

```
In [152]: regex.split(text)
```

```
Out[152]: ['VN', 'UK', 'US', 'FR']
```

''' Sử dụng phương thức `findall` lấy danh sách tất cả các mẫu khớp với biểu thức chính quy, có thể sử dụng'''

```
In [153]: regex.findall(text)
```

```
Out[153]: [' ', ' \t ', ' \n ']
```

## 3.2. Biểu thức chính quy (Regular Expressions)

```
''' tách chuỗi có số ký tự khoảng trắng thay đổi (gồm tab, dấu  
cách và ký tự ngắt dòng). Biểu thức chính quy mô tả một hoặc nhiều  
ký tự khoảng trắng là \s+'''  
In [148]: import re  
In [149]: text = "VN    UK \t  US  \n FR"  
In [150]: re.split('\s+', text)  
Out[150]: ['VN', 'UK', 'US', 'FR']  
''' có thể dùng re.compile để biên dịch biểu thức chính quy thành  
1 đối tượng biểu thức chính quy để tái sử dụng sau này'''  
In [151]: regex = re.compile('\s+')  
In [152]: regex.split(text)  
Out[152]: ['VN', 'UK', 'US', 'FR']  
''' Sử dụng phương thức findall lấy danh sách tất cả các mẫu khớp  
với biểu thức chính quy, có thể sử dụng'''  
In [153]: regex.findall(text)  
Out[153]: [' ', ' ', '\t ', ' \n ']
```

- **Ghi chú**: Nên thay ký tự thoát \ trong biểu thức chính quy, hãy sử dụng các chuỗi ký tự thô (raw) như r 'C:\x' thay vì dùng 'C:\\x'.



### 3. Thao tác với dữ liệu kiểu chuỗi (String Manipulation)

#### 3.2. Biểu thức chính quy

- So sánh 3 phương thức ***match***, ***search*** và ***findall***:
  - ***findall*** trả về tất cả các kết quả khớp trong một chuỗi
  - ***search*** chỉ trả về kết quả khớp đầu tiên
  - ***match*** chỉ khớp ở đầu chuỗi.

```
text = """Dave dave@google.com
Steve steve@gmail.com
Rob rob@gmail.com
Ryan ryan@yahoo.com
"""

pattern = r'[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}'
# re.IGNORECASE makes the regex case-insensitive
regex = re.compile(pattern, flags=re.IGNORECASE)
# findall trả về tất cả các kết quả khớp
In [155]: regex.findall(text)
Out[155]:
['dave@google.com',
 'steve@gmail.com',
 'rob@gmail.com',
 'ryan@yahoo.com']

# search trả về kết quả khớp đầu tiên
In [156]: m = regex.search(text)
In [157]: m
Out[157]: <_sre.SRE_Match object; span=(5, 20),
          match='dave@google.com'>

In [158]: text[m.start():m.end()]
Out[158]: 'dave@google.com'
# match chỉ so khớp với đầu chuỗi
In [159]: print(regex.match(text))
None
```

### 3. Thao tác với dữ liệu kiểu chuỗi (*String Manipulation*)

#### 3.2. Biểu thức chính quy

- **sub** cũng có quyền truy cập vào các groups trong mỗi đối tượng match bằng các ký hiệu đặc biệt như \1 và \2. Ký hiệu \1 tương ứng với nhóm khớp đầu tiên, \2 tương ứng với nhóm thứ hai, v.v.

```
In [166]: print(regex.sub(r'Username: \1, Domain: \2, Suffix: \3', text))
Dave Username: dave, Domain: google, Suffix: com
Steve Username: steve, Domain: gmail, Suffix: com
Rob Username: rob, Domain: gmail, Suffix: com
Ryan Username: ryan, Domain: yahoo, Suffix: com
```

3. Thao tác với dữ liệu kiểu chuỗi (String Manipulation)

3.3. Các hàm chuỗi được vector hóa trong pandas (Vectorized String Functions in pandas)

Method	Description
cat	Nối các chuỗi theo từng phần tử với dấu phân cách tùy chọn
contains	Trả về mảng boolean nếu mỗi chuỗi có chứa mẫu (pattern)/regex
count	Đếm số lần xuất hiện của mẫu (pattern)
extract	Sử dụng biểu thức chính quy với các nhóm để trích xuất một hoặc nhiều chuỗi từ Series của chuỗi (strings); kết quả sẽ là DataFrame với một cột cho mỗi nhóm
endswith	Tương đương với x.endswith(pattern) cho mỗi phần tử
startswith	Tương đương với x.startswith(pattern) cho mỗi phần tử
findall	Danh sách tính toán tất cả các lần xuất hiện của mẫu/regex cho mỗi chuỗi
get	Lập chỉ mục vào từng phần tử (truy xuất phần tử thứ i)
isalnum	Tương đương với str.alnum
isalpha	Tương đương với str.alpha
isdecimal	Tương đương với str.isdecimal
isdigit	Tương đương với str.isdigit tích hợp
islower	Tương đương với str.islower
isnumeric	Tương đương với str.isnumeric
isupper	Tương đương với str.isupper
join	Nối các chuỗi trong mỗi phần tử của Series bằng dấu phân cách (separator) đã truyền

3. Thao tác với dữ liệu kiểu chuỗi (String Manipulation)

3.3. Các hàm chuỗi được vector hóa trong pandas (Vectorized String Functions in pandas)

Method	Description
len	Trả về chiều dài chuỗi
lower, upper	Chuyển đổi chữ hoa sang chữ thường, tương đương với <code>x.lower()</code> . Hoặc Chuyển đổi chữ thường sang chữ hoa, tương đương với <code>x.upper()</code>
match	Sử dụng <code>re.match</code> với biểu thức chính quy được truyền trên mỗi phần tử, trả về các nhóm khớp dưới dạng list
pad	Thêm khoảng trắng vào bên trái, bên phải hoặc cả hai bên của chuỗi
center	Tương đương với <code>pad(side='both')</code>
repeat	Lặp lại 1 chuỗi với số lần được chỉ định. ví dụ: <code>s.str.repeat(3)</code> tương đương với <code>x * 3</code> cho mỗi chuỗi
replace	Thay thế các lần xuất hiện của pattern/regex bằng một số chuỗi khác
slice	Cắt từng chuỗi trong Series
split	Tách chuỗi dựa trên dấu phân cách hoặc biểu thức chính quy
strip	Cắt bớt khoảng trắng ở đầu và cuối chuỗi, bao gồm cả ký tự xuống dòng ( <code>\n</code> )
rstrip	Cắt bớt khoảng trắng bên phải
lstrip	Cắt bớt khoảng trắng bên trái

### 3. Thao tác với dữ liệu kiểu chuỗi (*String Manipulation*)

#### 3.3. Các hàm chuỗi được vector hóa trong pandas (*Vectorized String Functions in pandas*)

- Kiểm tra giá trị null bằng hàm isnull

```
#hàm isnull: xác định giá trị null
In [167]: data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
.....: 'Rob': 'rob@gmail.com', 'Wes': np.nan}
In [168]: data = pd.Series(data)
In [169]: data
Out[169]:
Dave dave@google.com
Rob rob@gmail.com
Steve steve@gmail.com
Wes NaN
dtype: object
In [170]: data.isnull()
Out[170]:
Dave False
Rob False
Steve False
Wes True
dtype: bool
```

### 3. Thao tác với dữ liệu kiểu chuỗi (String Manipulation)

#### 3.3. Các hàm chuỗi được vector hóa trong pandas

- ***str.contains***: là 1 trong những phương thức hướng mảng của Series .

```
data
Dave dave@google.com
Rob rob@gmail.com
Steve steve@gmail.com
Wes NaN
dtype: object
```

```
In [171]: data.str.contains('gmail')
Out[171]:
Dave False
Rob True
Steve True
Wes NaN
dtype: object
```

- Cũng có thể sử dụng các biểu thức chính quy, với đối số ***flags*** được gán bởi bất kỳ tùy chọn nào của ***re*** như ***IGNORECASE***:

```
In [172]: pattern
Out[172]: '([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\\.([A-Z]{2,4})'
In [173]: data.str.findall(pattern, flags=re.IGNORECASE)
Out[173]:
Dave [(dave, google, com)]
Rob [(rob, gmail, com)]
Steve [(steve, gmail, com)]
Wes NaN
dtype: object
```

### 3. Thao tác với dữ liệu kiểu chuỗi (String Manipulation)

#### 3.3. Các hàm chuỗi được vector hóa trong pandas

- Có một số cách để thực hiện truy xuất phần tử được vector hóa. Sử dụng ***str.get*** hoặc ***index*** vào thuộc tính ***str***:

```
In [174]: matches = data.str.match(pattern,  
                                     flags=re.IGNORECASE)  
  
In [175]: matches  
Out[175]:  
      Dave      True  
      Rob      True  
      Steve    True  
      Wes      NaN  
      dtype: object
```

- Có thể cắt chuỗi có độ dài tương tự bằng cú pháp sau:

```
data  
Dave dave@google.com  
Rob  rob@gmail.com  
Steve steve@gmail.com  
Wes  NaN  
dtype: object
```

```
In [178]: data.str[:5]  
Out[178]:  
      Dave  dave@  
      Rob   rob@g  
      Steve steve  
      Wes   NaN  
      dtype: object
```



