pandas data cleaning and preparation

In [1]:
```
import pandas as pd
import numpy as np
```

处理缺失数据

> pandas 中缺失值的表示 : ***NaN, None***

obj.isnull( ), obj.isna( ) ¶

In [2]:
```
string_data = pd.Series(['aardvark', 'artichoke', np.nan, 'avocado'])
string_data, string_data.isnull()
```

Out[2]:
```
(0     aardvark
 1    artichoke
 2          NaN
 3      avocado
 dtype: object,
 0    False
 1    False
 2     True
 3    False
 dtype: bool)
```

In [3]:
```
string_data[0] = None
string_data, string_data.isnull()
```

Out[3]:
```
(0         None
 1    artichoke
 2          NaN
 3      avocado
 dtype: object,
 0     True
 1    False
 2     True
 3    False
 dtype: bool)
```

obj.notnull( ), obj.notna( )

In [4]:
```
string_data.notna()
```

Out[4]:
```
0    False
1     True
2    False
3     True
dtype: bool
```

series.dropna( ), frame.dropna(axis, how, thresh)

> *series.dropna( )* 等价于 *series[series.notna( )]*

In [5]:
```
string_data, string_data.dropna()
```

Out[5]:
```
(0         None
 1    artichoke
 2          NaN
 3      avocado
 dtype: object,
 1    artichoke
 3      avocado
 dtype: object)
```

In [6]:

```
string_data[string_data.notna()]
```

Out[6]:

```
1    artichoke
3      avocado
dtype: object
```

how：过滤缺失值的方式

In [7]:

```
data = pd.DataFrame([[1., 6.5, 3.], [1., np.nan, np.nan],
                     [np.nan, np.nan, np.nan], [np.nan, 6.5, 3.]])
data
```

Out[7]:

	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 [8]:

```
data.dropna(how='any') # 去除的行至少有一个缺失值
```

Out[8]:

	0	1	2
0	1.0	6.5	3.0

In [9]:

```
data.dropna(how='all') # 去除的行所有的值都是缺失值
```

Out[9]:

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

thresh = n：过滤缺失值时，对应行或列的剩下的非缺失值的个数大于等于n

In [10]:

```
df = pd.DataFrame(np.random.randn(7,7))
df.iloc[:7, 0] = np.nan
df.iloc[:6, 1] = np.nan
df.iloc[:5, 2] = np.nan
df.iloc[:4, 3] = np.nan
df.iloc[:3, 4] = np.nan
df.iloc[:2, 5] = np.nan
df.iloc[:1, 6] = np.nan
df
```

Out[10]:

	0	1	2	3	4	5	6
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	0.548996
2	NaN	NaN	NaN	NaN	NaN	-0.448875	-0.028554
3	NaN	NaN	NaN	NaN	1.973543	-0.927488	-0.622286
4	NaN	NaN	NaN	1.329444	-0.430934	-0.957348	1.737727
5	NaN	NaN	-0.497778	-0.801027	-0.138910	-0.599382	0.839175
6	NaN	-0.638612	1.437967	0.108482	0.436201	-0.003544	-0.953735

In [11]:

```
df.dropna(thresh=1)
# 保留的行中，至少有一个不是缺失值，即去除的行所有值都是缺失值
```

Out[11]:

	0	1	2	3	4	5	6
1	NaN	NaN	NaN	NaN	NaN	NaN	0.548996
2	NaN	NaN	NaN	NaN	NaN	-0.448875	-0.028554
3	NaN	NaN	NaN	NaN	1.973543	-0.927488	-0.622286
4	NaN	NaN	NaN	1.329444	-0.430934	-0.957348	1.737727
5	NaN	NaN	-0.497778	-0.801027	-0.138910	-0.599382	0.839175
6	NaN	-0.638612	1.437967	0.108482	0.436201	-0.003544	-0.953735

In [12]:

```
df.dropna(thresh=3) # 保留的行中，至少有一个不是缺失值
```

Out[12]:

	0	1	2	3	4	5	6
3	NaN	NaN	NaN	NaN	1.973543	-0.927488	-0.622286
4	NaN	NaN	NaN	1.329444	-0.430934	-0.957348	1.737727
5	NaN	NaN	-0.497778	-0.801027	-0.138910	-0.599382	0.839175
6	NaN	-0.638612	1.437967	0.108482	0.436201	-0.003544	-0.953735

series.fillna(value, method), frame.fillna(value, method, axis)

In [13]:

```
string_data.fillna(-9999)
```

Out[13]:

```
0        -9999
1    artichoke
2        -9999
3      avocado
dtype: object
```

value = dict : 通过传递字典到 fillna 可以实现对不同的列填充不同的值

In [14]:

```
df
```

Out[14]:

	0	1	2	3	4	5	6
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	0.548996
2	NaN	NaN	NaN	NaN	NaN	-0.448875	-0.028554
3	NaN	NaN	NaN	NaN	1.973543	-0.927488	-0.622286
4	NaN	NaN	NaN	1.329444	-0.430934	-0.957348	1.737727
5	NaN	NaN	-0.497778	-0.801027	-0.138910	-0.599382	0.839175
6	NaN	-0.638612	1.437967	0.108482	0.436201	-0.003544	-0.953735

In [15]:

```
df.fillna( {1: 0.5, 2: 0} )
```

Out[15]:

	0	1	2	3	4	5	6
0	NaN	0.500000	0.000000	NaN	NaN	NaN	NaN
1	NaN	0.500000	0.000000	NaN	NaN	NaN	0.548996
2	NaN	0.500000	0.000000	NaN	NaN	-0.448875	-0.028554
3	NaN	0.500000	0.000000	NaN	1.973543	-0.927488	-0.622286
4	NaN	0.500000	0.000000	1.329444	-0.430934	-0.957348	1.737727
5	NaN	0.500000	-0.497778	-0.801027	-0.138910	-0.599382	0.839175
6	NaN	-0.638612	1.437967	0.108482	0.436201	-0.003544	-0.953735

method : 填充方式 , *limit* : 限制填充个数

In [16]:

```
df = pd.DataFrame(np.random.randn(6, 3))
df.iloc[1:5, 1] = np.nan
df.iloc[2:4, 2] = np.nan
df
```

Out[16]:

	0	1	2
0	-0.291262	0.200877	2.589814
1	-0.337112	NaN	0.186565
2	-2.075923	NaN	NaN
3	-0.109867	NaN	NaN
4	0.807568	NaN	-0.173979
5	0.059223	0.818621	0.742860

method = 'ffill' : 用前一个非缺失值去填充该缺失值

In [17]:
```
df.fillna(method='ffill', limit=2)
```

Out[17]:

	0	1	2
0	-0.291262	0.200877	2.589814
1	-0.337112	0.200877	0.186565
2	-2.075923	0.200877	0.186565
3	-0.109867	NaN	0.186565
4	0.807568	NaN	-0.173979
5	0.059223	0.818621	0.742860

method = 'bfill' : 用后一个非缺失值去填充该缺失值

In [18]:
```
df.fillna(method='bfill', limit=2)
```

Out[18]:

	0	1	2
0	-0.291262	0.200877	2.589814
1	-0.337112	NaN	0.186565
2	-2.075923	NaN	-0.173979
3	-0.109867	0.818621	-0.173979
4	0.807568	0.818621	-0.173979
5	0.059223	0.818621	0.742860

处理重复数据

obj.duplicated(columns, keep), obj.drop_duplicates(columns, keep) : 移除重复数据

In [19]:
```
data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                     'k2': [1, 1, 2, 2, 3, 3, 3]})
data
```

Out[19]:

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

In [20]:
```
data.duplicated()
```

Out[20]:
```
0    False
1    False
2    False
3    False
4    False
5    False
6     True
dtype: bool
```

In [21]:
```
data.drop_duplicates()
```

Out[21]:

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

columns : 指定部分列进行重复项判断/过滤

In [22]:
```
data.duplicated(['k1'])
```

Out[22]:
```
0    False
1    False
2     True
3     True
4     True
5     True
6     True
dtype: bool
```

In [23]:
```
data.drop_duplicates(['k2'])
```

Out[23]:

	k1	k2
0	one	1
2	one	2
4	one	3

keep : keep = 'first' / keep = 'last'， 保留项

In [24]:
```
data.duplicated(['k1'], keep='last')
```

Out[24]:
```
0     True
1     True
2     True
3     True
4    False
5     True
6    False
dtype: bool
```

In [25]:
```
data.drop_duplicates(['k2'], keep='last')
```

Out[25]:

	k1	k2
1	two	1
3	two	2
6	two	3

数据映射与替换

series.map(arg) : 利用函数或字典进行数据映射

In [26]:

```
data = pd.DataFrame({'food': ['Bacon', 'pulled pork', 'bacon',
                              'Pastrami', 'corned beef', 'Bacon',
                              'pastrami', 'honey ham','nova lox'],
                     'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
```

Out[26]:

	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

映射数据

In [27]:

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

Out[27]:

```
{'bacon': 'pig',
 'pulled pork': 'pig',
 'pastrami': 'cow',
 'corned beef': 'cow',
 'honey ham': 'pig',
 'nova lox': 'salmon'}
```

series.map(dict)

In [28]:

```
data['animal1'] = data['food'].str.lower().map(meat_to_animal)
data
```

Out[28]:

	food	ounces	animal1
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

series.map(func)

In [29]:
```
data['animal2'] = data['food'].map( lambda x: meat_to_animal[x.lower()] )
data
```

Out[29]:

	food	ounces	animal1	animal2
0	Bacon	4.0	pig	pig
1	pulled pork	3.0	pig	pig
2	bacon	12.0	pig	pig
3	Pastrami	6.0	cow	cow
4	corned beef	7.5	cow	cow
5	Bacon	8.0	pig	pig
6	pastrami	3.0	cow	cow
7	honey ham	5.0	pig	pig
8	nova lox	6.0	salmon	salmon

obj.replace(to_replace, value) : 数据替换

In [30]:
```
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data.replace([-999, -1000], np.nan)
```

Out[30]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    NaN
5    3.0
dtype: float64
```

In [31]:
```
data.replace([-999, -1000], [np.nan, 0])
```

Out[31]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

In [32]:
```
data.replace({-999: np.nan, -1000: 0})
```

Out[32]:
```
0    1.0
1    NaN
2    2.0
3    NaN
4    0.0
5    3.0
dtype: float64
```

obj.rename(index, columns, inplace) : 索引重命名

In [33]:
```
data = pd.DataFrame(np.arange(12).reshape((3, 4)),
                    index=['Ohio', 'Colorado', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
data
```

Out[33]:

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

series.map(func) : map 方法

In [34]:

```
data.index = data.index.map( lambda x: x[:4].upper() )
data
```

Out[34]:

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

series.rename(index, columns) : rename 方法

In [35]:

```
data.rename(index=str.title, columns=str.upper)
```

Out[35]:

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

rename 可以结合字典型对象实现对部分轴标签的更新

In [36]:

```
data.rename(index={'OHIO': 'INDIANA'}, columns={'three': 'peekaboo'})
```

Out[36]:

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

数据划分

pd.cut(x, bins, right, labels) : 划分面元 （ binning ）

> *x* : *The input array to be binned; must be 1-D*
> *bins* : 面元, 可以是确切的面元边界, 也可以是面元数量
> *right* : *True : (,] ; False : [,)*
> *labels* : 设置面元的名称

bins = list

In [37]:

```
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
bins = [18, 25, 35, 60, 100]
cats = pd.cut(ages, bins, right=False)
cats
```

Out[37]:

```
[[18, 25), [18, 25), [25, 35), [25, 35), [18, 25), ..., [25, 35), [60, 100), [35, 60), [35, 60), [25, 35)]
Length: 12
Categories (4, interval[int64, left]): [[18, 25) < [25, 35) < [35, 60) < [60, 100)]
```

In [38]:

```
cats.categories # 展示了划分的面元
```

Out[38]:

```
IntervalIndex([[18, 25), [25, 35), [35, 60), [60, 100)], dtype='interval[int64, left]')
```

In [39]:

```
cats.codes
```

Out[39]:

```
array([0, 0, 1, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
```

In [40]:
```
pd.value_counts(cats)
```

Out[40]:
```
[18, 25)     4
[25, 35)     4
[35, 60)     3
[60, 100)    1
dtype: int64
```

In [41]:
```
group_names = ['Youth', 'YoungAdult', 'MiddleAged', 'Senior']
pd.cut(ages, bins, labels=group_names)
```

Out[41]:
```
['Youth', 'Youth', 'Youth', 'YoungAdult', 'Youth', ..., 'YoungAdult', 'Senior', 'MiddleAged', 'MiddleAged', 'YoungAdul
t']
Length: 12
Categories (4, object): ['Youth' < 'YoungAdult' < 'MiddleAged' < 'Senior']
```

bins = n : 根据样本的 最小值和最大值 计算等长的面元

In [42]:
```
data = np.random.randint(0, 11, (50))
pd.cut(data, 5)
```

Out[42]:
```
[(-0.01, 2.0], (-0.01, 2.0], (-0.01, 2.0], (4.0, 6.0], (-0.01, 2.0], ..., (6.0, 8.0], (8.0, 10.0], (-0.01, 2.0], (4.0,
6.0], (-0.01, 2.0]]
Length: 50
Categories (5, interval[float64, right]): [(-0.01, 2.0] < (2.0, 4.0] < (4.0, 6.0] < (6.0, 8.0] < (8.0, 10.0]]
```

pd.qcut(x, q, labels) : 根据分位数划分面元 （ quantile binning ）

> *x* : *The input array to be binned; must be 1-D*
> *q* : *Number of quantiles,* 分位数
> *labels* : 设置面元的名称

q = n : 根据样本的 分位数 对数据进行面元划分

In [43]:
```
data = np.random.randint(0, 101, (1000))
cats = pd.qcut(data, 4)
cats
```

Out[43]:
```
[(50.5, 76.0], (26.0, 50.5], (-0.001, 26.0], (50.5, 76.0], (26.0, 50.5], ..., (-0.001, 26.0], (26.0, 50.5], (76.0, 10
0.0], (26.0, 50.5], (26.0, 50.5]]
Length: 1000
Categories (4, interval[float64, right]): [(-0.001, 26.0] < (26.0, 50.5] < (50.5, 76.0] < (76.0, 100.0]]
```

In [44]:
```
pd.value_counts(cats)
```

Out[44]:
```
(50.5, 76.0]      260
(-0.001, 26.0]    255
(26.0, 50.5]      245
(76.0, 100.0]     240
dtype: int64
```

q = list : 传递自定义的分位数（0到1之间的数值, 包含端点）

In [45]:
```
pd.qcut(data, [0, 0.1, 0.5, 0.9, 1.])
```

Out[45]:
```
[(50.5, 91.0], (11.0, 50.5], (11.0, 50.5], (50.5, 91.0], (11.0, 50.5], ..., (11.0, 50.5], (11.0, 50.5], (50.5, 91.0],
(11.0, 50.5], (11.0, 50.5]]
Length: 1000
Categories (4, interval[float64, right]): [(-0.001, 11.0] < (11.0, 50.5] < (50.5, 91.0] < (91.0, 100.0]]
```

随机采样

np.random.permutation(x)

In [46]:

```
df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
sampler = np.random.permutation(5)
df.iloc[sampler]
```

Out[46]:

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

obj.sample(n)

In [47]:

```
df = pd.DataFrame(np.arange(5 * 4).reshape((5, 4)))
df.sample(n=5)
```

Out[47]:

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

将 分类变量 转换为 向量变量

pd.get_dummies(series, prefix)

In [48]:

```
df = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                   'data':  range(6)})
df
```

Out[48]:

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

In [49]:

```
pd.get_dummies(df['key'])
```

Out[49]:

	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

prefix **: 给指标 DataFrame 的列加上一个前缀**

In [50]:

```
pd.get_dummies(df['key'], prefix='key')
```

Out[50]:

	key_a	key_b	key_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

In [51]:

```
pd.get_dummies(df['key'], prefix='key').join(df['data'])
```

Out[51]:

	key_a	key_b	key_c	data
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	5

字符串的处理方法

str.split(sep, maxsplit) : 根据 sep 拆分字符串, str → list

In [52]:

```
val = ' a ,b, guido '
val.split(',')
```

Out[52]:

```
[' a ', 'b', ' guido ']
```

str.strip( ) : 去除字符串两边空白符（包括换行符）

str.lstrip( ), str.rstrip( ) : 去除字符串 左或右 的空白符（包括换行符）

In [53]:

```
[x.strip() for x in val.split(',')]
```

Out[53]:

```
['a', 'b', 'guido']
```

sep.join(list) : 去除字符串首尾空白符（包括换行符）

In [54]:

```
'::'.join( val.split(',') )
```

Out[54]:

```
' a ::b:: guido '
```

str.index(sep), str.find(sep) : 返回 sep 在 str 中第一次出现的位置

> 区别 : 如果 sep 在 str 中不存在, *sep.find* 返回 -1 , *sep.index* 会引发异常

In [55]:

```
val.index(',')
```

Out[55]:

```
3
```

In [56]:

```
val.find(':')
```

Out[56]:

```
-1
```

str.rfind(sep) : 返回 sep 在 str 中最后一次出现的位置

```
In [57]: val.rfind(',')
```

```
Out[57]: 5
```

str.count(sep) : 返回 sep 在 str 中出现的次数

```
In [58]: val.count(',')
```

```
Out[58]: 2
```

str.replace(old, new) : 替换

```
In [59]: val.replace(',', '::')
```

```
Out[59]: ' a ::b:: guido '
```

str.endswith(sep), str.startswith(sep) : 判断 str 是否以 sep 结尾或开始

```
In [60]: val.strip().endswith('a')
```

```
Out[60]: False
```

```
In [61]: val.strip().startswith('a')
```

```
Out[61]: True
```

str.lower( ), str.upper( ), str.title( ) : 控制大小写

```
In [62]: val.title(), val.upper(), val.lower()
```

```
Out[62]: (' A ,B, Guido ', ' A ,B, GUIDO ', ' a ,b, guido ')
```

正则表达式

```
In [63]: import re
```

re.split(pattern, str) : 根据 sep 拆分字符串 （ str 中的分隔符 sep 数量不定 ）

```
In [64]: text = "foo    bar\t baz  \tqux"
         re.split('\s+', text)  # 描述一个或多个空白符的正则表达式是'\s+'
```

```
Out[64]: ['foo', 'bar', 'baz', 'qux']
```

re.compile(pattern) : 根据 pattern 返回一个正则表达式类 （ regex ） 的对象

```
In [65]: regex = re.compile('\s+')
         regex.split(text)
```

```
Out[65]: ['foo', 'bar', 'baz', 'qux']
```

regex.split(str) : 根据 regex 拆分字符串

re.findall(pattern, str), regex.findall(str) : 返回字符串中的正则表达式匹配项

```
In [66]: regex.findall(text)
```

```
Out[66]: ['    ', '\t ', ' \t']
```

In [67]:
```
text = """WANG wangbj27@mail2.sysu.edu.cn
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}'
# \.[A-Z]{2,4} : 必须以 '.[A-Z]' 结尾，并且[A-Z]的字符数为2~4个
regex = re.compile(pattern, flags=re.IGNORECASE)
regex.findall(text)
```

Out[67]:
```
['wangbj27@mail2.sysu.edu.cn',
 'dave@google.com',
 'steve@gmail.com',
 'rob@gmail.com',
 'ryan@yahoo.com']
```

re.finditer(pattern, str), regex.finditer(str) : 以迭代器的形式返回字符串中的正则表达式匹配项

In [68]:
```
for x in regex.finditer(text):
    print(x.group())
```

```
wangbj27@mail2.sysu.edu.cn
dave@google.com
steve@gmail.com
rob@gmail.com
ryan@yahoo.com
```

re.sub(pattern, repl, str), regex.sub(repl, str) : 替换字符串中的正则表达式匹配项

In [69]:
```
print(regex.sub(repl='E-mail', string=text))
```

```
WANG E-mail
Dave E-mail
Steve E-mail
Rob E-mail
Ryan E-mail
```

正则表达式的分组模式

pattern : r' ([A-Z0-9._%+-]+) @ ([A-Z0-9.-]+) . ([A-Z]{2,4}) '

In [70]:
```
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
regex = re.compile(pattern, flags=re.IGNORECASE)
regex.findall(text)
```

Out[70]:
```
[('wangbj27', 'mail2.sysu.edu', 'cn'),
 ('dave', 'google', 'com'),
 ('steve', 'gmail', 'com'),
 ('rob', 'gmail', 'com'),
 ('ryan', 'yahoo', 'com')]
```

sub 还能通过 \1、\2 之类的特殊符号访问各匹配项中的分组, 符号 \1 对应第一个匹配的组

In [71]:
```
print(regex.sub(r'Username: \1, Domain: \2, Suffix: \3', text))
```

```
WANG Username: wangbj27, Domain: mail2.sysu.edu, Suffix: cn
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
```

pandas 的矢量化字符串方法 *obj.str.func(...)*

> 将字符串的方法应用于 series 的各个单元里去

series.str.contains(pattern) : 检查各行是否含有字符串 string

In [72]:

```
data = {'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
        'Rob': 'rob@gmail.com', 'Wes': np.nan}
data = pd.Series(data)
data
```

Out[72]:

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

In [73]:

```
data.str.contains('gmail')
```

Out[73]:

```
Dave     False
Steve     True
Rob       True
Wes        NaN
dtype: object
```

series.str.findall(pattern, flags)

In [74]:

```
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
data.str.findall(pattern, flags=re.IGNORECASE)
```

Out[74]:

```
Dave     [(dave, google, com)]
Steve    [(steve, gmail, com)]
Rob        [(rob, gmail, com)]
Wes                        NaN
dtype: object
```

series.str.match(pattern, flags)

In [75]:

```
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
data.str.match(pattern, flags=re.IGNORECASE)
```

Out[75]:

```
Dave     True
Steve    True
Rob      True
Wes       NaN
dtype: object
```

series.str.get(i), series.str.slice(start, stop), series.str[start : stop] : 切片

In [76]:

```
data.str.get(0)
```

Out[76]:

```
Dave       d
Steve      s
Rob        r
Wes      NaN
dtype: object
```

In [77]:

```
data.str[:5]
```

Out[77]:

```
Dave     dave@
Steve    steve
Rob      rob@g
Wes        NaN
dtype: object
```

series.str.len( )

```
In  [78]: data.str.len()
```

```
Out[78]: Dave      15.0
         Steve     15.0
         Rob       13.0
         Wes        NaN
         dtype: float64
```

series1.str.cat(series2, sep) : 根据索引实现元素级字符串连接

```
In  [79]: name = pd.Series({'Dave':'Dave', 'Steve':'Steve', 'Rob':'Rob', 'Wes':'Wes'})
          name.str.cat(data, '----')
```

```
Out[79]: Dave       Dave----dave@google.com
         Steve     Steve----steve@gmail.com
         Rob           Rob----rob@gmail.com
         Wes                            NaN
         dtype: object
```

series.str.len( )
series.str.lower( ), series.str.upper( ), series.str.title( )
series.str.strip( ), series.str.lstrip( ), series.str.rstrip( ) : 去除两边/左/右的空格

series.str.endswith(sep), series.str.startswith(sep)
series.str.find(sep), series.str.rfind(sep) : 返回 sep 在字符串中的位置
series.str.count(sep) : 计数 sep 在字符串中出现的次数
series.str.split(sep) : 根据分隔符 sep 对字符串进行划分, str→list
series.str.join(sep) : 利用分隔符 sep 将 list 连接起来, liat→str
series.str.replace(old, new) : 替换

series1.str.cat(series2, sep) : 根据索引实现元素级字符串连接

... ...