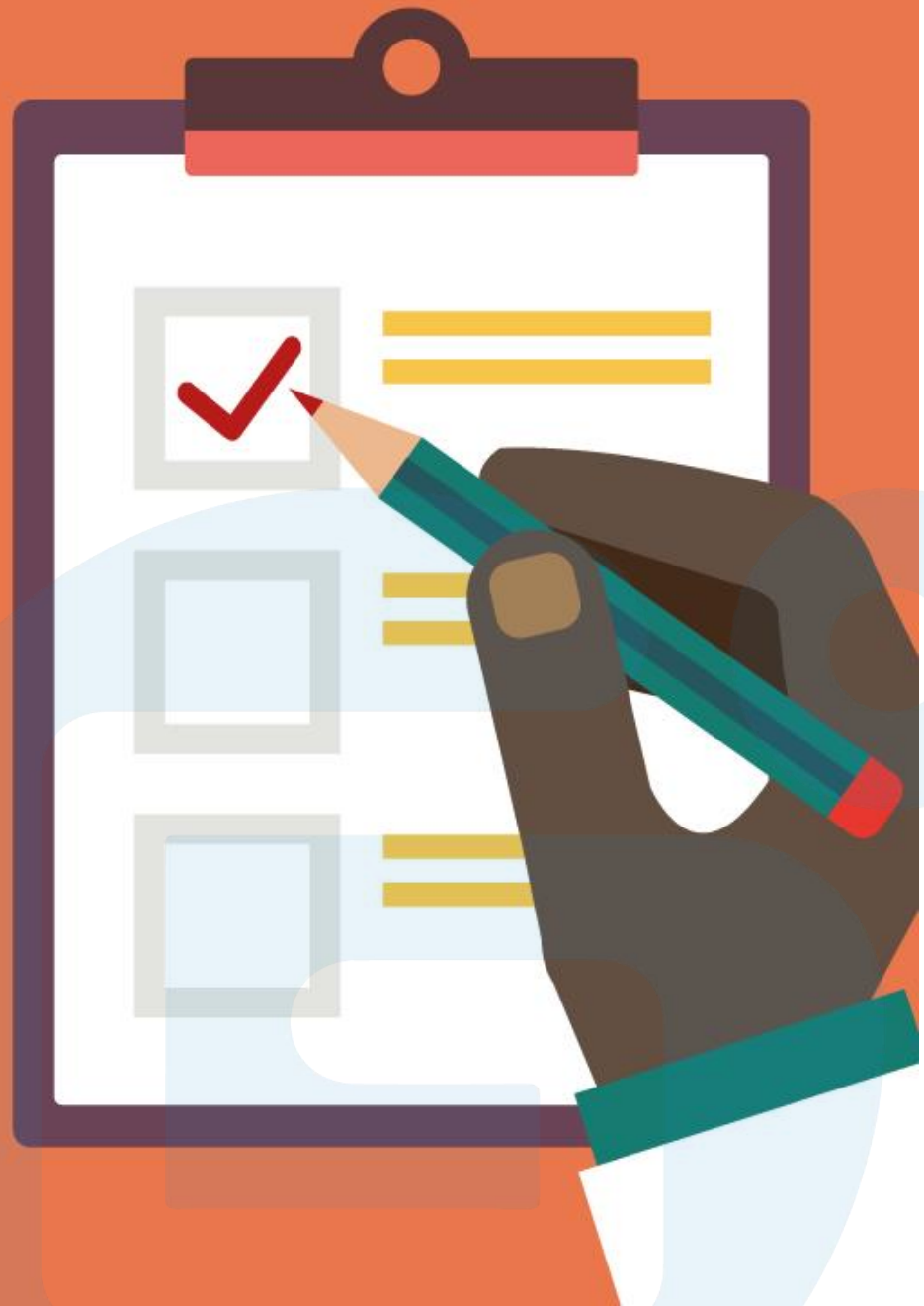
Talk is cheap, show me the code

第五课：pandas基础

Python进阶课程系列



OUTLINE

- Pandas简介
  - 基本结构：Series
  - 基本结构：DataFrame
  - 常用方法

一 Pandas简介

Pandas简介

pandas是基于Numpy数组构建的，特别是基于数组的函数和不使用for循环的数据处理。pandas和Numpy最大的不同在于，pandas是专门用来处理**表格和混杂大数据**设计的，而Numpy更适合处理**统一的数值数组**数据。

pandas这个名字源于panel data(面板数据，这是多维结构化数据集在计量经济学中的术语)以及Python data analysis(Python数据分析)

pandas兼具Numpy高性能的数组计算功能以及电子表格和关系型数据库(如SQL)灵活的数据处理功能。它提供了复杂精细的索引功能，以便更为便捷地完成重塑、切片和切块、聚合以及选取数据子集等操作。因为数据操作、准备、清洗是数据分析最重要的技能，因此pandas在**数据分析**领域得到了很广泛的运用。

引入pandas的两种常用语法：

```
import pandas as pd
from pandas import Series, DataFrame
```

原生的python里不含pandas，anaconda里自带，或者用pip命令安装。和之前的Numpy一样。

Pandas简介

只用一种工具就实现以下所有功能，并使用通用软件开发语言。

- 有标签轴的数据结构，支持自动或清晰的数据对齐。这可以防止由于数据不对齐，和处理来源不同的索引不同的数据，造成的错误。
- 集成时间序列功能。
- 相同的数据结构用于处理时间序列数据和非时间序列数据。
- 保存元数据的算术运算和压缩。
- 灵活处理缺失数据。
- 合并和其它流行数据库（例如基于SQL的数据库）的关系操作。

二 基本结构：Series

基本结构：Series

Pandas中的Series对象是一种带有**标签数据的一维数组**，标签在Pandas中有对应的**数据类型“Index”**，Series类似于**一维数组与字典的结合**。

Series的创建

- 创建Series时，可以通过**index参数指定索引**，未指定索引时，采用**默认索引，从0开始递增**。
- 通过**列表、元组**创建：索引大小必须和容器大小一致。
- 通过**Numpy的ndarray**创建：必须是一维数组，且数组大小要和索引大小一致。
- 通过**字典**创建：**默认索引为字典的关键字**，指定索引时会以索引为key获取值，没有值的话，默认NaN。
- 通过**标量**创建：重复填充标量至每个索引。

一个简单的实例：

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

In [2]: obj=pd.Series([2,-3,5,6])

In [3]: obj
Out[3]:
0    2
1   -3
2    5
3    6
dtype: int64
```

Series的字符串表现形式为：索引在左边，值在右边。

由于这里没有为数据指定索引，于是会自动创建0到N-1的整数型索引。

基本结构：Series

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

In [2]: obj=pd.Series([2,-3,5,6])

In [3]: obj
Out[3]:
0    2
1   -3
2    5
3    6
dtype: int64

In [4]: obj.values
Out[4]: array([ 2, -3,  5,  6], dtype=int64)

In [5]: obj.index
Out[5]: RangeIndex(start=0, stop=4, step=1)
```

Series的字符串表现形式为：索引在左边，值在右边。由于这里没有为数据指定索引，于是会自动创建0到N-1的整数型索引。可以通过values和index属性来获取其数组表示形式和索引对象。

```
In [10]: obj2=pd.Series([2,-3.3,5.5,6.1],index=['d','c','b','a'])

In [11]: obj2.values
Out[11]: array([ 2. , -3.3,  5.5,  6.1])

In [12]: obj2.index
Out[12]: Index(['d', 'c', 'b', 'a'], dtype='object')

In [13]: obj2
Out[13]:
d    2.0
c   -3.3
b    5.5
a    6.1
dtype: float64
```

Series也可以手动为每一个数据点指定索引。

索引名可以是字符串，可以直接用索引名来选中Series里的元素。

```
In [14]: obj2[obj2>0]
Out[14]:
d    2.0
b    5.5
a    6.1
dtype: float64

In [15]: obj2[obj2*2]
Out[15]:
 4.0    NaN
-6.6    NaN
 11.0   NaN
 12.2   NaN
dtype: float64

In [16]: obj2*2
Out[16]:
d     4.0
c    -6.6
b    11.0
a    12.2
dtype: float64

In [17]: obj2['a']
Out[17]: 6.1
```

根据布尔型数组进行过滤、标量乘法等都会保留索引值的链接。但要注意语法格式，如果采用[]的形式，则是对索引值进行处理。

基本结构：Series

Pandas的Index对象可以看成一个**不可变数组**，可以包含重复值。可以直接在pandas里利用Index类来创建Index对象，也可以用Series和DataFrame中的index属性来获取对应的Index对象。Index对象可以在多组数据间共享。

```
In [19]: index1=pd.Index([6,-2,5,3])

In [20]: obj2=pd.Series([2,-3.3,5.5,6.1],index=['d','c','b','a'])

In [21]: index2=obj2.index

In [22]: print(index1);\
    ...: print(index2)
Int64Index([6, -2, 5, 3], dtype='int64')
Index(['d', 'c', 'b', 'a'], dtype='object')
```

```
In [23]: s_2=pd.Series(list("ABCD"),index=index1)

In [24]: s_2
Out[24]:
 6    A
-2    B
 5    C
 3    D
dtype: object
```

Index对象与numpy中的数组类似，支持**索引和切片**，支持常见函数如size，shape，ndim等。不同之处在于**Index对象的值是不可变的**，不像ndarray数组一样可以任意修改。这是为了多个数据之间进行索引共享的时候更加安全。

Index对象同样支持集合操作，如并集，交集，差集等，这些操作也可以通过调用对象方法来实现。不同的是操作结果中可能会存在重复元素。

基本结构：Series

Series和字典很像，也可以直接从字典里创建，字典的键即index，但是有一些区别。

- 字典里的键不能重复，Series里的index可以有重复值。
- 以index为键，可以访问对应的值，如果有重复，则返回结果为Series类型。
- 可以调用字典的一些常用方法，例如keys(),items(),可用于判断是否包含指定索引。
- 也可以用字典类似的语法更新数据，对不存在的索引进行赋值，会添加一个索引。

```
In [26]: sdata={'a':12,'b':2,'c':0.33,'d':23}

In [27]: obj3=pd.Series(sdata)

In [28]: obj3
Out[28]:
a    12.00
b     2.00
c     0.33
d    23.00
dtype: float64
```

```
In [31]: index3=(list("abbc"))

In [32]: obj4=pd.Series(sdata,index=index3)

In [33]: obj4
Out[33]:
a    12.00
b     2.00
b     2.00
c     0.33
dtype: float64
```

```
In [44]: obj3['e']=100

In [45]: obj3
Out[45]:
a     12.00
b      2.00
c      0.33
d     23.00
e    100.00
dtype: float64
```

```
In [38]: print(obj3.index)
Index(['a', 'b', 'c', 'd'], dtype='object')

In [39]: print(obj3.keys())
Index(['a', 'b', 'c', 'd'], dtype='object')

In [40]: print(obj3['b'])
2.0

In [41]: print(obj3.items())
<zip object at 0x000000000A1A19C8>

 In [43]: print(list(obj3.items()))
 [('a', 12.0), ('b', 2.0), ('c', 0.33), ('d', 23.0)]
```

基本结构：Series

Series对象除了可以人为的赋值index(索引)，还有一个默认的整数数组索引，称之为隐式索引。隐式索引从0增大，和ndarray类似。如果index是整数数字，则通过keys键访问使用的是显式索引，通过切片访问采用的是隐式索引（容易引起冲突，不推荐这么做）。其它情况下两者可以互换。

```
In [10]: import pandas as pd
    ...: s_1=pd.Series([2,4,6,8,10])
    ...: index=list('ABCDE')
    ...: s_1=pd.Series([2,4,6,8,10],index)
    ...: s_1
Out[10]:
A     2
B     4
C     6
D     8
E    10
dtype: int64

In [11]: print(s_1[2])
6

In [12]: print(s_1['B':'D'])
B    4
C    6
D    8
dtype: int64

In [13]: print(s_1[1:4])
B    4
C    6
D    8
dtype: int64

In [14]:  #注意区别，显示索引是包含所有元素的，但是隐式索引不包含最后一个元素
```

```
In [17]: s_1=pd.Series(range(2,12,2),index=range(1,6))

In [18]: s_1
Out[18]:
1     2
2     4
3     6
4     8
5    10
dtype: int64
```

如果遇到用整数作为index的情况，最好使用loc和iloc两个关键字区分显式和隐式。

```
In [19]: s_1[2]
Out[19]: 4

In [20]: s_1.iloc[2]
Out[20]: 6

In [21]: s_1[2:4]
Out[21]:
3    6
4    8
dtype: int64

In [22]: s_1.loc[2:4]
Out[22]:
2    4
3    6
4    8
dtype: int64

In [23]: s_1.iloc[2:4]
Out[23]:
3    6
4    8
dtype: int64
```

基本结构：Series

Series对象的常用方法

- **sort_index()**：对Series按照索引进行排序，生成一个新的Series对象；
- **sort_values()**：对Series按照值排序，生成一个新的Series对象；
- **rank()**:对值进行排名，从1开始，对于相同的值默认采用平均排名。
- **reindex()**:重新设置索引，生成一个新的Series对象。新的索引长度和原始索引长度可以不同，如果新的索引不在原始数据中，则对应的值为NaN，如果在原始数据中，则值不变。
- unique():去除值里重复的数据，每个值只保留一个。
- value_counts()：统计各数据出现的次数，并按次数从高到低排序。
- Series对象也可以**执行numpy中的一些运算**，只对值操作，不影响索引和索引与值之间的关系。
- Series对象之间执行运算时，**会自动进行对齐，相同索引之间执行运算**，不同索引之间对应的值为**NaN**。

基本结构：Series

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

In [2]: import pandas as pd

In [3]: s_1=pd.Series([8,9,7.5,3.22],index=list('ADFG'))
   ...: print(s_1);\
   ...: print(s_1.sort_index());\
   ...: print(s_1.sort_values());\
   ...: print(s_1.rank);\
   ...: print(s_1.reindex(list('ABCDEF')));\
   ...: print(s_1.reindex(list('ABCDEF'),fill_value=0));\
   ...: print(np.square(s_1));\
   ...: s_2=pd.Series([1,3,5],index=list('ABC'));\
   ...: print(s_1+s_2)
```

```
A    8.00
D    9.00
F    7.50
G    3.22
dtype: float64
A    8.00
D    9.00
F    7.50
G    3.22
dtype: float64
G    3.22
F    7.50
A    8.00
D    9.00
dtype: float64
<bound method NDFrame.rank of A    8.00
D    9.00
F    7.50
G    3.22
dtype: float64>
A    8.0
B    NaN
C    NaN
D    9.0
E    NaN
F    7.5
dtype: float64
A    8.0
B    0.0
C    0.0
D    9.0
E    0.0
F    7.5
```

```
dtype: float64
A    64.0000
D    81.0000
F    56.2500
G    10.3684
dtype: float64
A    9.0
B    NaN
C    NaN
D    NaN
F    NaN
G    NaN
dtype: float64
```

```
In [4]: s_1=pd.Series([3,6,8,3,2,6,7,6])

In [5]: print(s_1.unique())
[3 6 8 2 7]

In [6]: print(s_1.value_counts())
6    3
3    2
7    1
2    1
8    1
dtype: int64
```

三 基本结构：DataFrame

基本结构：DataFrame

Pandas中的DataFrame对象是一种**表格型的数据结构**，含有一组有序的列，每列可以是不同的值类型(数值、字符串、布尔值等)。它既有**行索引也有列索引**，可以被看成是由多个Series组成的字典（共用一个index）。类似于一张excel的二维表格，是Pandas里**最常用的基本结构**。

DataFrame的创建

- 可以通过值为一维ndarray，list，dict或者Series的字典或列表；二维的ndarray；单个Series，列表，一维数组或者其它DataFrame来创建；
- 创建DataFrame时，可以通过**index和columns**参数指定行索引和列索引，若没有指定索引，则默认为从0开始的连续数字；
- 通过多个Serise创建DataFrame时，多个Series对象会自动对齐。若指定了index，则**丢弃未和index匹配的数据**，如果指定的索引不存在，则对应值为NaN。

基本结构：DataFrame

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

In [2]: import numpy as np

In [3]: names=['lily','lucy','mike']
   ...: ages=[18,17,20]

In [4]: d_1=pd.DataFrame({'name':names,'age':ages})

In [5]: d_1
Out[5]:
   name  age
0  lily   18
1  lucy   17
2  mike   20

In [6]: d_2=pd.DataFrame([names,ages])

In [7]: d_2
Out[7]:
      0     1     2
0  lily  lucy  mike
1    18    17    20
```

```
In [8]: s_names=pd.Series(names,index=['A','B','C'])

In [9]: s_ages=pd.Series(ages,index=['B','C','A'])

In [10]: d_3=pd.DataFrame({'names':s_names,'ages':s_ages})

In [11]: d_3
Out[11]:
  names  ages
A  lily    20
B  lucy    18
C  mike    17

In [13]: d_4=pd.DataFrame(np.random.randint(10,30,(3,4)))

In [14]: d_4
Out[14]:
    0   1   2   3
0  21  21  11  10
1  29  14  21  10
2  26  28  22  13
```

基本结构：DataFrame

DataFrame对象与**二维numpy数组**，**共享索引的多个Series对象构成的字典**类似。

DataFrame视为字典

- 可以以**列的索引为关键字**，获取某一列的所有数据，结果为Series对象，可进一步获得某个具体数据，例如**d_1[列索引][行索引]**，两个中括号不能合并；
- 如果列的索引为字符串，则可以列名为属性名，例如**d_1.属性名**，前提是列名符合标识符的命名规范。如果列名不符合规范或者与DataFrame中属性相同，例如shape、index等，则不能使用属性形式。

```
In [10]: s_names=pd.Series(['alice','bily','coney','dolly','ela','funny'],index=list('abcdef'))

In [11]: s_ages=pd.Series([16,18,20,21,18],index=list('abdef'))

In [12]: d_1=pd.DataFrame({'姓名':s_names,'年龄':s_ages})

In [13]: d_1
Out[13]:
      姓名    年龄
a  alice  16.0
b   bily  18.0
c  coney   NaN
d  dolly  20.0
e    ela  21.0
f  funny  18.0
```

```
In [14]: print(d_1['姓名'])
a    alice
b     bily
c    coney
d    dolly
e      ela
f    funny
Name: 姓名, dtype: object
```

```
In [17]: print(d_1.姓名)
a    alice
b     bily
c    coney
d    dolly
e      ela
f    funny
Name: 姓名, dtype: object
```

```
In [20]: print(d_1['姓名']['b'])
bily
```

基本结构：DataFrame

DataFrame对象与**二维numpy数组**，**共享索引的多个Series对象构成的字典**类似。

DataFrame视为二维数组

与Series类似，也可以利用**显式和隐式索引。**关键字分别是**loc[行索引，列索引]**和**iloc[行索引，列索引]**。

- **支持行列转置，布尔表达式。**
- **支持行切片，不支持列切片。如果想访问多个列，可以将列索引放在列表里。**

```
In [21]: print(d_1.T)
         a     b      c      d    e      f
姓名   alice  bily  coney  dolly  ela  funny
年龄      16    18    NaN     20   21     18

In [22]: print(d_1[d_1.年龄==18])
      姓名    年龄
b    bily  18.0
f   funny  18.0

In [23]: print(d_1[1:3])
      姓名    年龄
b    bily  18.0
c   coney   NaN

In [24]: print(d_1['a':'d'])
      姓名    年龄
a   alice  16.0
b    bily  18.0
c   coney   NaN
d   dolly  20.0

In [25]: print(d_1.iloc[3,1])
20.0

In [26]: print(d_1.loc['d','姓名'])
dolly
```

基本结构：DataFrame

DataFrame常用方法

- shape：获取**形状信息**，结果为一个**元组**；
- dtypes：获取**各字段的数据类型**，结果为**Series**；
- values：获取**数据内容**，结果通常为**二维数组**；
- columns：获取**列索引**，即字段名称，结果为**Index**；
- index: 行索引，即行的标签，结果为**Index**。
- axes：同时获取行和列索引，结果为**Index的列表**；

方法	说明
info()	**显示基本信息**，包括行列索引信息、每列非空元素数量，每列数据的类型，整体所占内存大小等
head(n)	**获取前n行数据，n默认为5**，结果为**DataFrame**
tail(n)	**获取后n行数据，n默认为5**，结果为**DataFrame**
describe()	数据的整体描述信息，包括：非空值数量、平均值、标准差、最小值、最大值等，结果为**DataFrame**
count()	统计各列中非空值的数量，结果为**Series**
sample(n, axis)	随机从数据中**按行或列抽取n行或n列**
apply(fun, axis)	对**每一行或每一列元素执行函数**
applymap(fun)	对**每一个数据执行函数**
to_dict()	**转化为dict类型对象**，可指定字典中值的类型，如list
to_excel(文件名)	将数据**保存到Excel文件**中去

基本结构：DataFrame

DataFrame常用方法

方法	说明
sort_values(by)	**根据值进行排序**，可以指定一列或多列，返回新的对象
sort_index()	根据索引进行排序，**原始索引不一定有序**，返回新的对象
rank()	对每一列的值进行排名，从小到大，从1开始
isna()、isnull()	对每一个元素判断**是否为缺失值**
dropna()	**删除缺失值**，可指定删除行或列、缺失值满足的条件等
fillna(value)	**用value值填充空值**，返回新的对象
rename()	**重命名，通过columns对列索引重命名，index对行索引重命名**
set_index()	**设置索引列**，可以用一个已有列名作为索引，返回新的对象
groupby()	**对数据进行分组**，例如根据某列或多列进行分组
d_1.append(d_2)	将d_2中的行添加到d_1的后面，会**自动对齐**，没有内容的部分**默认为NaN**
sum()、mean()、max()、min()、median()、std()、var()	对每一列数据**求和**、**求平均数**、**最大值**、**最小值**、**中位数**、标准差、方差
nunique()	统计每一列中**不重复的元素个数**

基本结构：DataFrame

DataFrame的合并

DataFrame提供了一个join()方法用于将其他DataFrame中的列合并到当前DataFrame中，类似于数据库中的连接，支持内连接，外连接，左连接和右连接等，默认情况采用左连接。默认根据两个对象的索引进行匹配连接，如果列名相同，则需要指定后缀，也可通过on参数指定关联的列，需要将该列作为其他对象的索引。

```
In [27]: s_names=pd.Series(['alice','bily','coney','dolly','ela','funny'],index=list('abcdef'))
    ...: s_ages=pd.Series([16,18,20,21,18],index=list('abdef'))
    ...: d_1=pd.DataFrame({'姓名':s_names,'年龄':s_ages})

In [28]: s_nums=pd.Series(['001','002','003','008'],index=list('acde'))

In [29]: s_names_2=pd.Series(['alice','bily','coney','xeon'],index=list('adce'))

In [30]: d_2=pd.DataFrame({'姓名':s_names_2,'年龄':s_nums})

In [31]: print(d_1.join(d_2,rsuffix='_r'))
      姓名    年龄   姓名_r 年龄_r
a  alice  16.0  alice  001
b   bily  18.0    NaN  NaN
c  coney   NaN  coney  002
d  dolly  20.0   bily  003
e    ela  21.0   xeon  008
f  funny  18.0    NaN  NaN
```

四 常用方法

常用方法

加载数据的方法（支持大多数文件格式）

- read_excel()：从excel文件中读取数据；
- read_csv()：从csv文件中读取数据；
- read_clipboard()：从剪切板中数据；
- read_html()：从网页中读取数据；
- read_json()：从 json 格式文本中读取数据；
- read_pickle()：从pickle文件中读取数据；

读取Excel文件的核心参数

- io：文件路径，可以是本地文件也可以是网络文件，支持xls、xlsx、xlsm等格式；
- sheet_name：表单序号或名称，可以是一个列表，同时读取多个表单，默认为第一个表单；
- hearder：表头，可以是整数或整数列表；
- names：指定列名；
- index_col：索引列，可以是整数或整数列表；
- usecols：使用到的列；
- dtype：指定每一列的数据类型；
- skiprows：跳过多少行；
- nrows：解析多少行；
- na_values：指定哪些值被看做是缺失值；

常用方法

常用方法

缺失值是指数据集中的某些值为空。常见处理方法有**删除，替换，插补**。

- 删除是指直接将缺失值的记录删除，用于缺失值较少的数据，例如5%以内。
- 替换是指某种值直接替换缺失值，例如连续变量的均值或中位数。
- 插补法是指根据其它值进行预测，例如K近邻、回归法等。

Pandas常用方法

- dropna()：删除包含缺失值的行，可通过axis参数设置删除所在列，通过thresh参数指定阈值，只有非空值大于该阈值的行或列才保留；
- fillna():用指定值填充缺失值，可为不同列指定不同的填充值，可通过method参数指定填充方式，通过limit现定填充数量
- isna()或isnull()：判断元素是否为缺失值。

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

In [2]: import numpy as np

In [3]: s_1=pd.Series(['aa','bb','cc','dd','ee'],index=list('ABCEF'))

In [4]: s_2=pd.Series([20,18,19,20],index=list('ACEG'))

In [5]: s_3=pd.Series(['01','02','03','04','05','06','07'],index=list('ABCDEFG'))

In [6]: d_1=pd.DataFrame({'name':s_1,'age':s_2,'No.':s_3})

In [7]: print(d_1)
  name   age No.
A   aa  20.0  01
B   bb   NaN  02
C   cc  18.0  03
D  NaN   NaN  04
E   dd  19.0  05
F   ee   NaN  06
G  NaN  20.0  07

In [8]: print(d_1.dropna())
  name   age No.
A   aa  20.0  01
C   cc  18.0  03
E   dd  19.0  05

In [9]: print(d_1.dropna(axis=1,thresh=5))
  name No.
A   aa  01
B   bb  02
C   cc  03
D  NaN  04
E   dd  05
F   ee  06
G  NaN  07
```

常用方法

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

In [2]: import numpy as np

In [3]: s_1=pd.Series(['aa','bb','cc','dd','ee'],index=list('ABCEF'))

In [4]: s_2=pd.Series([20,18,19,20],index=list('ACEG'))

In [5]: s_3=pd.Series(['01','02','03','04','05','06','07'],index=list('ABCDEFG'))

In [6]: d_1=pd.DataFrame({'name':s_1,'age':s_2,'No.':s_3})

In [7]: print(d_1)
  name   age No.
A   aa  20.0  01
B   bb   NaN  02
C   cc  18.0  03
D  NaN   NaN  04
E   dd  19.0  05
F   ee   NaN  06
G  NaN  20.0  07

In [8]: print(d_1.dropna())
  name   age No.
A   aa  20.0  01
C   cc  18.0  03
E   dd  19.0  05

In [9]: print(d_1.dropna(axis=1,thresh=5))
  name No.
A   aa  01
B   bb  02
C   cc  03
D  NaN  04
E   dd  05
F   ee  06
G  NaN  07
```

```
In [11]: print(d_1.fillna('unknow',limit=2))
     name     age No.
A      aa      20  01
B      bb  unknow  02
C      cc      18  03
D  unknow  unknow  04
E      dd      19  05
F      ee     NaN  06
G  unknow      20  07

In [12]: print(d_1.fillna(method='ffill'))
  name   age No.
A   aa  20.0  01
B   bb  20.0  02
C   cc  18.0  03
D   cc  18.0  04
E   dd  19.0  05
F   ee  19.0  06
G   ee  20.0  07

In [13]: print(d_1.isnull())
    name    age    No.
A  False  False  False
B  False   True  False
C  False  False  False
D   True   True  False
E  False  False  False
F  False   True  False
G   True  False  False
```

常用方法

Pandas数据合并

主要有merge方法和concat方法。

- 其中concat主要是根据索引进行行或列的拼接，只能取行或列的交集或并集。
- merge主要是根据共同列或者索引进行合并。

merge()方法

- **left**：左边的数据对象；
- **right**：右边的数据对象；
- **how**：连接方式，**默认为inner**，此外还有**left、right、outer**等；
- **on**：连接的**列名称**，必须在两个对象中，默认**以两个对象的列名的交集作为连接键**；
- **left_on**:左边对象中**用于连接的键的列名**；
- **right_on**:右边对象中**用于连接的键的列名**；
- **left_index**:使用**左边的行索引作为连接键**；
- **right_index**:使用**右边的行索引作为连接键**；
- **sort**:是否**将合并的数据排序**，默认为False；
- **suffixes**：列名相同时，指定的后缀；

contact()沿着一条轴，将多个对象堆叠起来。

contact()方法中的参数

- **objs**：需合并的对象序列；
- **axis**：指定合并的轴，0/‘index’，1/‘columns’，默认为0；
- **join**：连接方式，只有inner和outer，**默认为outer**；
- **ignore_index**：是否忽略索引，默认为False；
- **verify_integrity**:验证完整性，较为耗时，默认为False；

常用方法

```
df1 = make_df(list("abc"),[1,2,4])
df1
```

	a	b	c
1	a1	b1	c1
2	a2	b2	c2
4	a4	b4	c4

```
df2 = make_df(list("abcd"),[2,4,6])
df2
```

	a	b	c	d
2	a2	b2	c2	d2
4	a4	b4	c4	d4
6	a6	b6	c6	d6

```
# 按行拼接
pd.concat([df1,df2],sort=False)
```

	a	b	c	d
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
4	a4	b4	c4	NaN
2	a2	b2	c2	d2
5	a5	b5	c5	d5
6	a6	b6	c6	d6

```
# 按列拼接
pd.concat([df1,df2],axis=1)
```

	a	b	c	a	b	c	d
1	a1	b1	c1	NaN	NaN	NaN	NaN
2	a2	b2	c2	a2	b2	c2	d2
4	a4	b4	c4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	a5	b5	c5	d5
6	NaN	NaN	NaN	a6	b6	c6	d6

常用方法

```
df1 = make_df(list("abc"),[1,2,4])
df1
```

	a	b	c
1	a1	b1	c1
2	a2	b2	c2
4	a4	b4	c4

```
df2 = make_df(list("abcd"),[2,4,6])
df2
```

	a	b	c	d
2	a2	b2	c2	d2
4	a4	b4	c4	d4
6	a6	b6	c6	d6

(1) 基于相同列的合并

```
df3 = pd.merge(df1,df2,how='inner',on='a')          # 基于单列的合并
df4 = pd.merge(df1,df2,how='inner',on=['a','b'])    # 基于多列的合并
df5 = pd.merge(df1,df2,how='left',on='a',suffixes=['_1','_2']) # 左连接,且指定后缀
df5
```

	a	b_1	c_1	b_2	c_2	d
0	a1	b1	c1	NaN	NaN	NaN
1	a2	b2	c2	b2	c2	d2
2	a4	b4	c4	b4	c4	d4

(2) 基于不同列名，或者列和索引，或者索引和索引间的合并

```
df6 = pd.merge(df1,df2,how='inner',left_on='a',right_on='b')                # 基于不同列名
df7 = pd.merge(df1,df22,how='inner',left_on='a',right_index=True)           #基于列和索引
df8 = pd.merge(df1,df2,how='inner',left_index=True,right_index=True)       #基于两边都是索引
df8
```

	a_x	b_x	c_x	a_y	b_y	c_y	d
2	a2	b2	c2	a2	b2	c2	d2
4	a4	b4	c4	a4	b4	c4	d4

感谢参与 下堂课见