Python 深度解析

之 pandas 进阶篇

前言： 上一期讲解了 pandas 的一些基础用法，包括了切片和索引。切片是我们以后在使用中经常会用到的，必须要熟练掌握。这一节我们将会看到 pandas 的高级用法，这更接近工作中的内容，总体来说就是让我们处理数据更加效率，更加快捷，更加高大上。当然，因为元旦假期，时间充裕，我将会介绍书中几个不常用的技巧。

By 浪ふ渤沙

◆ pandas 数值运算函数

我们在处理数据的时候经常会使用一些函数对表格进行数据处理，由于 pandas 是基于 numpy 而来，我们可以直接使用 numpy 提供的 ufunc 函数进行运算。比如 max,sum 等。这些函数通常都有三个参数：axis（轴，0 表示纵轴，1 表示横轴），level（索引的级别），skipna（是否跳过空值）。我们看下实例。

```
In[2]: import pandas as pd
In[3]: import numpy as np
In[4]: arr = np.random.randint(10,20,(4,4))
In[5]: arr
Out[5]:
array([[14, 13, 10, 14],
       [12, 15, 11, 11],
       [10, 13, 17, 15],
       [18, 13, 11, 10]])
In[6]: index = pd.MultiIndex.from_product([list('AB'),list('ab')],names=['F','S'])
In[7]: df = pd.DataFrame(arr,index=index,columns=list('一二三四'))
In[8]: df
Out[8]:
      一   二   三   四
F S
A a  14  13  10  14
  b  12  15  11  11
B a  10  13  17  15
  b  18  13  11  10
```

我们先来随机生成一个 dataframe。然后分别来演示下这三个参数的用法。

```
In[9]: df.mean(axis=1)
Out[9]:
F  S
A  a    12.75
   b    12.25
B  a    13.75
   b    13.00
dtype: float64
```

```
In[10]: df.mean()
Out[10]:
一    13.50
二    13.50
三    12.25
四    12.50
dtype: float64
```

axis 的默认值是 0，也就是纵轴，如果我们需要按照横向计算，就可以指定 axis=1.

Level 的默认值是 None，如果不给的话，就是对所有的索引作用。

通常情况下，我们没有必要对所有的索引都进行聚合。这里，我们只对一级索引进行聚合，来看看指定和不指定的区别。

```
In[16]: df.sum()
Out[16]:
一    54
二    54
三    49
四    50
dtype: int64
```

```
In[17]: df.sum(level=0)
Out[17]:
   一   二   三   四
F
A  26  28  21  25
B  28  26  28  25
```

```
In[18]: df.sum(level=1)
Out[18]:
   一   二   三   四
S
a  24  26  27  29
b  30  28  22  21
```

如果 level 指定所有的，则返回原来的 dataframe。

```
In[19]: df.sum(level=[0,1])
Out[19]:
      一   二   三   四
F S
A a  14  13  10  14
  b  12  15  11  11
B a  10  13  17  15
  b  18  13  11  10
```

◆ pandas 的 str 系列

由于我们经常对 dataframe 中的某一列进行修改或者计算，pandas 提供了一个空间的函数包，存储在 str 里，当然这个是 series 的方法。因为 dataframe 的一列就是一个 series 对象。来瞅瞅。

```
In[33]: dd = pd.DataFrame(np.random.choice(['a|b','c|d','e|f','g|h'],(4,4)),index=index,columns=list('一二三四'))
In[34]: dd
Out[34]:
      一   二   三   四
F S
A a  a|b  e|f  a|b  c|d
  b  c|d  e|f  c|d  e|f
B a  c|d  c|d  e|f  g|h
  b  c|d  a|b  c|d  a|b
```

- 取 series

```
In[35]: ser_data=dd['二']
In[36]: ser_data
Out[36]:
F  S
A  a    e|f
   b    e|f
B  a    c|d
   b    a|b
Name: 二, dtype: object
```

- Series 分割

```
In[37]: sp_data=ser_data.str.split("|")
In[38]: sp_data
Out[38]:
F  S
A  a    [e, f]
   b    [e, f]
B  a    [c, d]
   b    [a, b]
Name: 二, dtype: object
```

- Series 合并

```
In[39]: jo_data = sp_data.str.join("")
In[40]: jo_data
Out[40]:
F  S
A  a    ef
   b    ef
B  a    cd
   b    ab
Name: 二, dtype: object
```

- Series 更改属性

```
In[41]: up_data = jo_data.str.upper()
In[42]: up_data
Out[42]:
F  S
A  a    EF
   b    EF
B  a    CD
   b    AB
Name: 二, dtype: object
```

- series 长度

如果是纯英文字符看不出什么效果，因为 python3 默认的是 utf-8 编码，所以英文字母永远只是占一个字节。我们来看看中文的长度计算。

```
In[53]: ch_data = pd.Series(['你','不是','个东西'])
In[54]: ch_data
Out[54]:
0      你
1     不是
2    个东西
dtype: object
In[55]: ch_data.str.len()
Out[55]:
0    1
1    2
2    3
dtype: int64
```

Unicode 字符串默认的就是汉字的个数。

```
In[63]: by_data = ch_data.str.encode('utf-8')
In[64]: by_data
Out[64]:
0                              b'\xe4\xbd\xa0'
1                  b'\xe4\xb8\x8d\xe6\x98\xaf'
2    b'\xe4\xb8\xaa\xe4\xb8\x9c\xe8\xa5\xbf'
dtype: object
In[65]: by_data.str.len()
Out[65]:
0    3
1    6
2    9
dtype: int64
```

Bytes 字节码里，一个汉字占三个字节，所以是 3，6，9.

```
In[60]: gb_data = ch_data.str.encode('gbk')
In[61]: gb_data
Out[61]:
0                    b'\xc4\xe3'
1              b'\xb2\xbb\xca\xc7'
2    b'\xb8\xf6\xb6\xab\xce\xf7'
dtype: object
In[62]: gb_data.str.len()
Out[62]:
0    2
1    4
2    6
dtype: int64
```

在常用汉字编码里，一个汉字占两个字节。所以这里是 2，4，6.

◆ Pandas 时间序列

Pandas 提供了三种时间，时间点，时间段，时间间隔

- 时间点

  全球 24 个时区，相邻的两个时区相差一个小时，靠东边的时区永远比它西边的时区快一个小时。而国际日期变更线是位于太平洋中间的一条线，但并不是直线，而是一个折线。北边用北极点绕过白令海峡穿过太平洋连接到南极点。变更线西边的时区比时区东边的时区快一天，这样就保证了时区的连续性，如果你听不懂，可以来群里找我。(^_^)

  我们可以通过 pytz 模块的 common_timezones 查看各个常用城市的名称对应的时区。。

```
In[68]: common_timezones
Out[68]: ['Africa/Abidjan', 'Africa/Accra', 'Africa/Addis_Ababa', 'Africa/Al
In[69]: len(common_timezones)
Out[69]: 439

In[70]:
```

第一个代表大洲，第二个代表城市。全球共 24 个时区，这里必然有些城市是在同一个时区。

我们可以用 tz_localize 来查看在某个时区对应的时间。

```
In[70]: now = pd.Timestamp.now()
In[71]: now
Out[71]: Timestamp('2017-12-31 16:12:23.357792')
In[72]: now.tz_localize("Asia/Shanghai")
Out[72]: Timestamp('2017-12-31 16:12:23.357792+0800', tz='Asia/Shanghai')
In[73]: now.tz_localize("Asia/Hong_Kong")
Out[73]: Timestamp('2017-12-31 16:12:23.357792+0800', tz='Asia/Hong_Kong')
In[74]: now.tz_localize("Asia/Tokyo")
Out[74]: Timestamp('2017-12-31 16:12:23.357792+0900', tz='Asia/Tokyo')
```

可以看出来上海和香港用的都是东八区的时间。而东京用的是东九区的时间。

时区的转换可以用 tz_cconvert。前面我们说了东边的时区比西边的时区快一个小时。我们来看看效果。

```
In[75]: sh_hai = now.tz_localize("Asia/Shanghai")
In[76]: do_jing = sh_hai.tz_convert("Asia/Tokyo")
In[77]: do_jing
Out[77]: Timestamp('2017-12-31 17:12:23.357792+0900', tz='Asia/Tokyo')
In[78]: sh_hai==do_jing
Out[78]: True
```

- 时间段

  Period 表示时间段。从数学上来说或时间段的导数就是时间点。

```
In[80]: today = pd.Period.now(freq='D')
In[81]: today_h = pd.Period.now(freq='H')
In[82]: today
Out[82]: Period('2017-12-31', 'D')
In[83]: today_h
Out[83]: Period('2017-12-31 16:00', 'H')
```

- 时间序列

  可以现象一下 numpy 中的 linspace。一个等差数列。时间序列也是一个等差序列。我们可以用 data_range 获取。

```
In[84]: pd.date_range('2017-10-01','2017-10-10',freq='D')
Out[84]:
DatetimeIndex(['2017-10-01', '2017-10-02', '2017-10-03', '2017-10-04',
               '2017-10-05', '2017-10-06', '2017-10-07', '2017-10-08',
               '2017-10-09', '2017-10-10'],
              dtype='datetime64[ns]', freq='D')
```

  freq 的参数可以是’M’ ‘D’,’H’等。

◆ Pandas 的空值处理

Pandas 使用 NaN 表示空缺值，由于整数列无法使用空值，如果原始数据为整数，筛选出来的 dataframe 有空值，则自动转化为浮点型。

```
In[89]: di = pd.DataFrame(np.random.randint(1,10,(3,3)),columns=list('abc'))
In[90]: di
Out[90]:
   a  b  c
0  6  8  4
1  2  4  1
2  3  8  8
In[91]: di.dtypes
Out[91]:
a    int32
b    int32
c    int32
dtype: object
```

原始数组为整型。筛选之后出现空值，我们再看下类型。

```
In[92]: di.where(di>5)
Out[92]:
     a    b    c
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
In[93]: di.where(di>5).dtypes
Out[93]:
a    float64
b    float64
```

自动转化为 float 类型，这里需要注意，这是个坑。

```
In[97]: dn
Out[97]:
     a    b    c
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
```

```
In[98]: dn.isnull()
Out[98]:
       a      b      c
0  False  False   True
1   True   True   True
2   True  False  False
```

```
In[99]: ~dn.notnull()
Out[99]:
       a      b      c
0  False  False   True
1   True   True   True
2   True  False  False
```

- Isnull 和 notnull 返回一个布尔值的 dataframe，notnull 的效率更高。可以使用~来求反获取对应的效果。
- Pandas 里使用 dropna 来删除含有空值的行或列。

```
In[103]: dn
Out[103]:
     a    b    c
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
```

```
In[107]: dn.dropna(axis=1,thresh=2)
Out[107]:
     b
0  8.0
1  NaN
2  8.0
```

axis 表示轴，默认为 0。

thresh 表示行或列里有几个非空值才不删除。

how 一般使用两个参数‘any’，‘all’，表示有一个空值就删除；所有的都是空值就删除。

```
In[106]: dn.dropna(how='all')
Out[106]:
     a    b    c
0  6.0  8.0  NaN
2  NaN  8.0  8.0
```

```
In[105]: dn.dropna(thresh=1)
Out[105]:
     a    b    c
0  6.0  8.0  NaN
2  NaN  8.0  8.0
```

- pandas 空值的填充

```
In[109]: dn
Out[109]:
     a    b    c
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
```

```
In[110]: dn.ffill()
Out[110]:
     a    b    c
0  6.0  8.0  NaN
1  6.0  8.0  NaN
2  6.0  8.0  8.0
```

```
In[111]: dn.bfill()
Out[111]:
     a    b    c
0  6.0  8.0  8.0
1  NaN  8.0  8.0
2  NaN  8.0  8.0
```

ffill() 表示用空值前边的值来填充。f 代表 front。
bfill() 表示用空值后边的值来填充。b 代表 behind 或者 back 都行吧。
还有一种通过前后值求平均值的方式填充。我们来看下。

```
In[112]: dn.interpolate()
Out[112]:
     a    b    c
0  6.0  8.0  NaN
1  6.0  8.0  NaN
2  6.0  8.0  8.0
```

```
In[113]: dn.interpolate(axis=1)
Out[113]:
     a    b    c
0  6.0  8.0  8.0
1  NaN  NaN  NaN
2  NaN  8.0  8.0
```

默认轴为 0，如果有连续的空值，则默认使用前边的值来填充，否则取前后两个值的平均值。下面来看个更加复杂的计算方式。

```
In[115]: s = pd.Series([3,np.NaN,7],index=[0,8,9])
In[116]: s
Out[116]:
0    3.0
8    NaN
9    7.0
dtype: float64
```

```
In[117]: s.interpolate(method='index')
Out[117]:
0    3.000000
8    6.555556
9    7.000000
dtype: float64
```

Method 取 index 表示按照索引的靠近原则，有点类似 K 近邻的算法。
9-0=9 长度为 9，9-8=1，8-0=8。由于索引为 9 的离空值比较近，采取一个反交叉算法。8/9*7+1/9*3=6.5556.
为了让填充更加灵活，pandas 推出了 fillna 的方法来对某一列的空值填充。

```
In[118]: dn
Out[118]:
     a    b    c
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
```

```
In[119]: dn.fillna({'a':100,'b':200,'c':300})
Out[119]:
       a      b      c
0    6.0    8.0  300.0
1  100.0  200.0  300.0
2  100.0    8.0    8.0
```

注意，这里其实有个坑。不光是能对空值填数字，同样还可以填充字符串，但是 Datafrme 会自动把其它的数字更改为 object 类型。

```
In[120]: dn.fillna({'a':'a','b':'b','c':'c'})
Out[120]:
   a  b  c
0  6  8  c
1  a  b  c
2  a  8  8
```

```
In[121]: dn.fillna({'a':'a','b':'b','c':'c'}).dtypes
Out[121]:
a    object
b    object
c    object
dtype: object
```

- dataframe 的形状更改

可以直接使用列名直接添加一列。Dataframe 有一个 eval 方法可以满足通过其它列的计算来赋值。

```
In[122]: dn['d']=11111
In[123]: dn
Out[123]:
     a    b    c      d
0  6.0  8.0  NaN  11111
1  NaN  NaN  NaN  11111
2  NaN  8.0  8.0  11111
```

```
In[124]: dn['d']=dn.eval('c*1000')
In[125]: dn
Out[125]:
     a    b    c       d
0  6.0  8.0  NaN     NaN
1  NaN  NaN  NaN     NaN
2  NaN  8.0  8.0  8000.0
```

当然如果 dataframe 中有这一列，就会直接替换产生一个行的 dataframe，可以使用 inplace 参数来决定是不是要更改原 dataframe。

Dataframe 还有一个 assign 的方法，来添加列。不过此方法不能更改原来的表格。

```
In[126]: dn.assign(e=dn.a+100)
Out[126]:
     a    b    c       d      e
0  6.0  8.0  NaN     NaN  106.0
1  NaN  NaN  NaN     NaN    NaN
2  NaN  8.0  8.0  8000.0    NaN
```

添加行，我们可以使用 append 方法。

```
In[153]: dn.append(pd.Series([1,1,1],index=list('abc'),name=3))
Out[153]:
     a    b    c
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
3  1.0  1.0  1.0
```

我们使用一个 series 添加成为最后一行。同样还可以使 Dataframe 来添加。

用 series 的方法添加有点古怪，因为 series 是竖着排的，这里 index 就对应着原来 dataframe 里的 columns，name 代表了 dataframe 的 index，ignore_index 代表

是否允许重复索引。我们一般是使用 True，表示不能重复。

使用 dataframe 添加行：

```
In[154]: dn.append(pd.DataFrame([[1,1,1]],columns=list('abc')),ignore_index=True)
Out[154]:
     a    b    c
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
3  1.0  1.0  1.0
```

注意 dataframe 是二维的，所以是两个中括号。

同样，我们还能使用字典的方式添加。

```
In[155]: dn.append({'a':1,'b':1,'c':1},ignore_index=True)
Out[155]:
     a    b    c
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
3  1.0  1.0  1.0
```

- pandas 的行列删除

我们通常使用 drop 来删除 dataframe 的某一行或者列。

```
In[157]: dn.drop(2)
Out[157]:
     a    b   c
0  6.0  8.0 NaN
1  NaN  NaN NaN
```

```
In[158]: dn.drop('c',axis=1)
Out[158]:
     a    b
0  6.0  8.0
1  NaN  NaN
2  NaN  8.0
```

由于 axis 通常默认的是 0，也就是纵向的，我们删行的时候可以不用添加 axis 这个参数，如果是删除列，也就是横向删除，则必须要指定 axis 的轴。

drop 支持多行同时删除，使用切片的方式即可。详情见上一期。

- pandas 索引变换

pandas 支持把索引转化为数据，通常使用 reset_index 把行索引转化为列，使用 set_index 把列转化为行索引。

```
In[162]: dn.index.name='F'
In[163]: dn
Out[163]:
     a    b    c
F
0  6.0  8.0  NaN
1  NaN  NaN  NaN
2  NaN  8.0  8.0
3  1.0  1.0  1.0
```

```
In[164]: dn.reset_index(level='F')
Out[164]:
   F    a    b    c
0  0  6.0  8.0  NaN
1  1  NaN  NaN  NaN
2  2  NaN  8.0  8.0
3  3  1.0  1.0  1.0
```

我们在把行索引转化为列之前，需要给索引一个名称，因为这将是转化为列之后的列索引。如果是想删除原来的索引可以设置 drop 参数为 True，则行索引不会转化为列。

我们再来看看 set_index 的用法。

append 表示是否保留原索引，默认是 false，也就是不保留，如果设置为 True，则原来的索引则会升级为一级索引，而新设置的列将成为二级索引。

```
In[167]: dn.set_index('a')
Out[167]:
        b    c
a
 6.0  8.0  NaN
NaN   NaN  NaN
NaN   8.0  8.0
 1.0  1.0  1.0
```

```
In[168]: dn.set_index('a',append=True)
Out[168]:
          b    c
F a
0 6.0  8.0  NaN
1 NaN  NaN  NaN
2 NaN  8.0  8.0
3 1.0  1.0  1.0
```

其中的 drop 参数表示数据里的列是否保留，默认是 True，就是删除，表示不保留。Fasle 则表示保留。

```
In[169]: dn.set_index('a',drop=False)
Out[169]:
        a    b    c
a
 6.0  6.0  8.0  NaN
NaN   NaN  NaN  NaN
NaN   NaN  8.0  8.0
 1.0  1.0  1.0  1.0
```

Pandas 还支持行索引和列索引相互转换。支持多级索引。Pandas 的 columns 也支持复合索引。我们来重写一个稍微复杂一点的 dataframe。

```
In[170]: index = pd.MultiIndex.from_product([list('AB'),list('abc')],names=['F','S'])
In[171]: columns = pd.MultiIndex.from_product([list('一二'),list('123')],names=['T','U'])
In[172]: dm = pd.DataFrame(np.random.randint(10,20,36).reshape(6,-1),index = index,columns=columns)
In[173]: dm
Out[173]:
T     一        二
U     1   2   3   1   2   3
F S
A a  15  16  14  12  17  19
  b  15  18  12  13  17  14
  c  10  10  19  11  11  19
B a  10  15  12  16  13  15
  b  11  15  13  10  19  13
  c  16  17  16  15  19  17
```

我们得到一个六行六列的 dataframe，行索引和列索引都有两级。
我们使用 stack 的方法来列索引转行索引，反之使用 unstack。

```
In[177]: dm.stack(0)
Out[177]:
U          1   2   3
F S T
A a 一    15  16  14
    二    12  17  19
  b 一    15  18  12
    二    13  17  14
  c 一    10  10  19
    二    11  11  19
B a 一    10  15  12
```

```
In[180]: dm.stack(1).unstack(1)
Out[180]:
T     一          二
S     a   b   c   a   b   c
F U
A 1  15  15  10  12  13  11
  2  16  18  10  17  17  11
  3  14  12  19  19  14  19
B 1  10  11  16  16  10  15
  2  15  15  17  13  19  19
  3  12  13  16  15  13  17
```

不过就我个人感觉来说，这个用处不大，因为单独的转换索引会改变表格的形状。
我们使用可以使用 swaplevel 交换索引的级别，可以用 axis 指定是行索引还是列索引，默认是的 0 也就是行索引。

```
In[182]: dm.swaplevel(0,1)
Out[182]:
T      一             二
U      1   2   3   1   2   3
S F
a A   15  16  14  12  17  19
b A   15  18  12  13  17  14
c A   10  10  19  11  11  19
a B   10  15  12  16  13  15
b B   11  15  13  10  19  13
c B   16  17  16  15  19  17
```

```
In[184]: dm.swaplevel(0,1,axis=1)
Out[184]:
U      1   2   3   1   2   3
T      一   一   一   二   二   二
F S
A a   15  16  14  12  17  19
  b   15  18  12  13  17  14
  c   10  10  19  11  11  19
B a   10  15  12  16  13  15
  b   11  15  13  10  19  13
  c   16  17  16  15  19  17
```

同样我们也可以使用 reorder_levels。

```
In[186]: dm.reorder_levels([1,0])
Out[186]:
T      一             二
U      1   2   3   1   2   3
S F
a A   15  16  14  12  17  19
b A   15  18  12  13  17  14
c A   10  10  19  11  11  19
a B   10  15  12  16  13  15
b B   11  15  13  10  19  13
c B   16  17  16  15  19  17
```

```
In[187]: dm.reorder_levels([1,0],axis=1)
Out[187]:
U      1   2   3   1   2   3
T      一   一   一   二   二   二
F S
A a   15  16  14  12  17  19
  b   15  18  12  13  17  14
  c   10  10  19  11  11  19
B a   10  15  12  16  13  15
  b   11  15  13  10  19  13
  c   16  17  16  15  19  17
```

这两个方法，了解下就行，用处不大，官网上连使用实例都没有给，所以用处不是很大。

转换后的排列顺序比较乱，我们可以使用 sort_index 来按照指定的索引排序。

参数也比较容易理解，这里就不细细讲了。

```
In[188]: dm.reorder_levels([1,0],axis=1).sort_index(axis=1,level=0)
Out[188]:
U      1           2           3
T      一   二      一   二      一   二
F S
A a   15  12  16  17  14  19
  b   15  13  18  17  12  14
  c   10  11  10  11  19  19
B a   10  16  15  13  12  15
  b   11  10  15  19  13  13
  c   16  15  17  19  16  17
```

◆ Pandas 文件读取

pandas 能够读取很多种文件，甚至是我们看见不见的东西，比如 sql 语句和查询。我们来看下几个常用的。

- read_excel 和 read_csv

  看下我们要读取的 excel 文件：

	A	B
1	书籍编号	书籍名称
2	1	python从入门到精通
3	2	java从入门到精通
4	3	c语言从入门到精通
5	4	php从入门到精通
6	5	ruby从入门到精通
7	6	python高级编程
8	7	java高级编程
9	8	c语言高级编程
10	9	php高级编程
11	10	ruby高级编程

书籍表 交易表

  假设我们卖的书的所有种类是这个表，在我们工作中的数据库里，经常会见到这种表。

  下表假设是我们的交易明细表，记录每一次的交易情况。你的老板可能会让你统计一下每种书卖了多少本，每种的交易额又是多少。这种工作是经常要做的。我们来演示一下读取，分组计算。别跟我说一个 sql 就搞定了，这里只是显示下 pandas 的使用技巧。

- read_sql

  可以查看 numpy 的第一期中间的用法，至于 read_json,read_sql_query 我就不再多说，只是一种读取方法。

	A	B	C	D
1	交易时间	销售书籍ID	交易金额	是否包邮
2	2017/10/20 12:34:20	3	66	是
3	2017/10/21 12:34:20	4	91	否
4	2017/10/22 12:34:20	1	86	是
5	2017/10/23 12:34:20	4	82	否
6	2017/10/24 12:34:20	6	85	是
7	2017/10/25 12:34:20	10	66	是
8	2017/10/26 12:34:20	2	65	否
9	2017/10/27 12:34:20	8	58	是
10	2017/10/28 12:34:20	4	58	否
11	2017/10/29 12:34:20	9	73	是
12	2017/10/30 12:34:20	9	64	是
13	2017/10/31 12:34:20	10	100	否
14	2017/11/1 12:34:20	8	59	是
15	2017/11/2 12:34:20	6	63	否
16	2017/11/3 12:34:20	1	70	是
17	2017/11/4 12:34:20	10	50	是
18	2017/11/5 12:34:20	4	99	否
19	2017/11/6 12:34:20	10	61	是
20	2017/11/7 12:34:20	2	98	否

书籍表 交易表

先把两个表读取进来。

```
In[2]: import pandas as pd
In[3]: ex_path = r'C:\Users\wqq\Desktop\pdtest.xlsx'
In[4]: cs_path = r'C:\Users\wqq\Desktop\pdtest.csv'
In[5]: ex_data_book = pd.read_excel(ex_path,sheetname='书籍表')
In[6]: ex_data_book
Out[6]:
   书籍编号           书籍名称
0      1  python从入门到精通
1      2    java从入门到精通
2      3     c语言从入门到精通
3      4     php从入门到精通
4      5    ruby从入门到精通
5      6    python高级编程
6      7      java高级编程
7      8       c语言高级编程
8      9       php高级编程
9     10      ruby高级编程
```

pandas 直接用 read_excel 读取，第一个参数是文件的路径，如果是在同一文件夹

下可以直接写名字，第二个参数是 sheet 的名字或者索引，默认是 0，也就是我们打开 excel 的时候 sheet1 那张表。同样我们读取交易表。

```
In[7]: ex_data_tran = pd.read_excel(ex_path,sheetname='交易表')
In[8]: ex_data_tran
Out[8]:
                 交易时间  销售书籍ID  交易金额 是否包邮
0  2017-10-20 12:34:20       3     66     是
1  2017-10-21 12:34:20       4     91     否
2  2017-10-22 12:34:20       1     86     是
3  2017-10-23 12:34:20       4     82     否
4  2017-10-24 12:34:20       6     85     是
5  2017-10-25 12:34:20      10     66     是
```

在这里我们也粗糙的演示一下读取 csv，我已经创建了一个跟 excel 一样的文件，就是文件类型是 csv。

```
In[13]: cs_data_tran = pd.read_csv(cs_path,sep=',',header=0,index_col=None)
In[14]: cs_data_tran.shape
Out[14]: (19, 4)
In[15]: cs_data_tran.head(5)
Out[15]:
              交易时间  销售书籍ID  交易金额 是否包邮
0  2017/10/20 12:34       3     66     是
1  2017/10/21 12:34       4     91     否
2  2017/10/22 12:34       1     86     是
3  2017/10/23 12:34       4     82     否
4  2017/10/24 12:34       6     85     是
```

Csv 的读法跟 excel 的读法差不多，主要在于 csv 只能读一个 sheet，而且有一个分割的符号，默认是用逗号，header 参数表示列名用那一行，注意这里我写 0 不是指表格的第一行，而是数据的第一行，比如数据是从第二行开始的，0 就代表第二行，index_col 表示指定索引列是哪一列。

- pandas 合并 Dataframe
  - merge

    merge 中文意思是合并，在 excel 里有一个 mergecell 对象，就是表示合并单元格的意思。而我们是要合并表格。经常写 sql 语句的同学一定很熟悉，就跟我们的 left join 一样。

```
In[18]: merge_data =pd.merge(ex_data_tran,ex_data_book,how='left',left_on='销售书籍ID',right_on='书籍编号')
In[19]: merge_data.head(5)
Out[19]:
                 交易时间  销售书籍ID  交易金额 是否包邮  书籍编号          书籍名称
0 2017-10-20 12:34:20       3     66     是     3     c语言从入门到精通
1 2017-10-21 12:34:20       4     91     否     4      php从入门到精通
2 2017-10-22 12:34:20       1     86     是     1  python从入门到精通
3 2017-10-23 12:34:20       4     82     否     4      php从入门到精通
4 2017-10-24 12:34:20       6     85     是     6    python高级编程
```

这里我们就拿到了合并后的数据，合并的方法比较灵活，我这里用了 pandas 的 merge 方法。我们知道在连接表格的时候分左右，按照常理，我们习惯上认为左边是主表，右边是副表，当然你要是个异类完全可以把主表放在右边。但是在连接的时候，附表的连接键不能有重复值，不然整个连接毫无意义。举个例子：比如你的书籍编号有两个都是 1，那么挂在主表上会默认是找到的第一个书籍的名称。当然稍微有点经验的人都不会犯这个错。两本书籍编号一样，那不是乱套了。

我们假设主表在左边，是我们的交易表(ex_data_tran)，附表是我们的书籍表

(ex_data_book)，所以 how 的连接方式我们选择左连接，如果你是异类，主表放右边，你就把交易表和书籍表的名字换个位置，how 的参数给成‘right’。Left_on 表示左边表的用来挂载的键值，right_on 表示右边表挂载的键值。Merge 还提供了一个 on 的参数，如果两个表的键值相同，可以直接使用 on 来指定键值。

现在我们来演示一下 Dataframe 的 merge 挂载方法。

```
In[20]: ex_data_tran.merge(ex_data_book,how='left',left_on='销售书籍ID',right_on='书籍编号')
Out[20]:
                 交易时间  销售书籍ID  交易金额 是否包邮  书籍编号          书籍名称
0  2017-10-20 12:34:20       3    66    是      3     c语言从入门到精通
1  2017-10-21 12:34:20       4    91    否      4      php从入门到精通
2  2017-10-22 12:34:20       1    86    是      1   python从入门到精通
3  2017-10-23 12:34:20       4    82    否      4      php从入门到精通
4  2017-10-24 12:34:20       6    85    是      6    python高级编程
```

可以看见 dataframe 默认是左表，第一个参数代表右表，how 表示谁是主表，left_on 和 right_on 分表代表左右表的挂载键值。

- pandas 的分组技巧 groupby

groupby 是 Dataframe 的方法，得到一个 groupby 对象。我们用 merget_data 来看下。

```
In[21]: merge_data.groupby(by=['是否包邮','书籍名称'],axis=0)
Out[21]: <pandas.core.groupby.DataFrameGroupBy object at 0x03419CD0>
In[22]: group_data = merge_data.groupby(by='书籍名称')
In[23]: group_data
Out[23]: <pandas.core.groupby.DataFrameGroupBy object at 0x060664B0>
```

分组可以按照一个属性分，也可以按照多个属性分，单个属性可以直接字符串的形式传入，多个属性需要按照 list 的方式传递。

有了分组就少不了聚合函数。我们常用的聚合函数有：sum，size，mean，max，min，std。等

这里要说下，在 pandas 里的计数有两种，一种是包含空值的计数 size，一种是只对非空值计数 count。

- agg

我们可以对分组后的对象运用聚合函数。使用最多的也就是 agg 函数了。Agg 全称 Aggregation，也就是聚合。

```
In[31]: group_data.agg({'销售书籍ID':np.size,'交易金额':np.sum})
Out[31]:
              销售书籍ID  交易金额
书籍名称
c语言从入门到精通          1    66
c语言高级编程            2   117
java从入门到精通          2   163
php从入门到精通           4   330
php高级编程             2   137
python从入门到精通        2   156
python高级编程          2   148
ruby高级编程            4   277
```

这里有个很尴尬的地方就是不能像 sql 语句里 as 进行重命名。尝试找了下，也没有看见有好的解决办法，暂时搁置。

agg 还支持直接传函数进行聚合，会多所有的列进行聚合，这种方式我并不推荐，没有通性。所以就不介绍了。下面说下 agg 的 lambad 匿名函数的使用。

Agg 的计算机制是把分组后的列作为一个 series 传递给聚合函数，所以说，series 的所有方法都可以成为 agg 的函数。

```
In[51]: dd
Out[51]:
    A   B   C   D   E
a  13  18  11  12  15
b  13  15  11  17  15
c  10  17  18  16  16
d  13  18  16  17  16
```

```
In[55]: dd.groupby('A').agg({'B':lambda df :df[0]})
Out[55]:
     B
A
10  17
13  18
```

我们知道 series 的 str 系列拥有众多的函数，我们都可以使用。

```
In[60]: dd.groupby('A').agg({'B':lambda df :df[0].astype('str')+'哈哈哈'})
Out[60]:
        B
A
10  17哈哈哈
13  18哈哈哈
```

如果你真的想不明白，就把 df 当成 series，想怎么玩就怎么玩。可以切片，可以组合。

```
In[65]: dd.groupby('A').agg({'B':lambda df :df.tolist()})
Out[65]:
                B
A
10           [17]
13  [18, 15, 18]
```

- transform

  transform 是一个转换运算方法，会把每一个 series 传递给回调函数。

```
In[66]: dd
Out[66]:
    A   B   C   D   E
a  13  18  11  12  15
b  13  15  11  17  15
c  10  17  18  16  16
d  13  18  16  17  16
```

```
In[67]: dd.transform(lambda x:x+10)
Out[67]:
    A   B   C   D   E
a  23  28  21  22  25
b  23  25  21  27  25
c  20  27  28  26  26
d  23  28  26  27  26
```

- filter

  筛选，filter()对每个分组进行条件判断。它将表示每个分组的 dataframe 对象传递给回调函数，该函数返回布尔值，以决定是否保留该分组。

```
In[74]: dd.groupby('B').filter(lambda x:x.C.mean()>13)
Out[74]:
    A   B   C   D   E
a  13  18  11  12  15
c  10  17  18  16  16
d  13  18  16  17  16
```

这里表示保留按 B 分组之后的 C 列平均值大于 13 的行。由于 b 行的 B 列只有 1 个 17，而 C 列的值是 11，所以平均估值就是 11.B 列有两个 18，对应在 C 列上的平均值就是（11+16）/2=13.5。所以保留了。

- apply

apply 可能是 pandas 里最灵活的用法了，可以实现以上 agg，transform,filter 的效果。Apply 的灵活得益与匿名函数。我们可以对 dataframe 直接使用，也可以对分组后的对象使用。

```
In[76]: dd.groupby('A').apply(lambda x:x-x.mean())
Out[76]:
     A    B         C         D         E
a  0.0  1.0 -1.666667 -3.333333 -0.333333
b  0.0 -2.0 -1.666667  1.666667 -0.333333
c  0.0  0.0  0.000000  0.000000  0.000000
d  0.0  1.0  3.333333  1.666667  0.666667
```

```
In[78]: dd.apply(lambda x:x-100,axis=1)
Out[78]:
    A   B   C   D   E
a -87 -82 -89 -88 -85
b -87 -85 -89 -83 -85
c -90 -83 -82 -84 -84
d -87 -82 -84 -83 -84
```

下面说下我的理解，agg 的作用比较固定，可以对指定的列进行聚合，而另外这三个主要还是依靠匿名函数，对 dataframe 里的 series 进行作用，当然可以指定 axis 作为轴，来选取 series 是一行还是一列。由于这种函数比较灵活，所以在使用的时候容易出错，只要能理解匿名函数的参数是一个 series 对象，那么会让你在使用的时候避免很多的麻烦。我们在学习这几个函数的时候一定不要怕出错，多去写，百炼成钢。

- pivot_table

这将是 pandas 的最后一个知识点，如果你懂的 excel 的数据透视表，这个方法你基本上看看就会了。下面我们用 excel 的

顺便说下 dataframe 对象也有一个类似的方法，叫 piovt，我在使用的过程中发现不是太好用，功能没有 pivot_table 的功能多，所以我们着重讲解一下 pivot_table.

我们使用导入的 excel 合并之后的 merge_data 来进行计算。

```
In[80]: merge_data.head(5)
Out[80]:
                 交易时间  销售书籍ID  交易金额 是否包邮  书籍编号          书籍名称
0 2017-10-20 12:34:20       3     66    是      3     c语言从入门到精通
1 2017-10-21 12:34:20       4     91    否      4     php从入门到精通
2 2017-10-22 12:34:20       1     86    是      1  python从入门到精通
3 2017-10-23 12:34:20       4     82    否      4     php从入门到精通
4 2017-10-24 12:34:20       6     85    是      6    python高级编程
```

```
data, values=None, index=None, columns=None, aggfunc='mean',
fill_value=None, margins=False, dropna=True, margins_name='All'

In[81]: pd.pivot_table()
```

我们先来讲解下参数的意思：
Data:你的数据表，是一个 dataframe，这里就是 merge_data
Values:你要汇总的列名，可以传递一列，也可以使用 list 传递多列。
Index:汇总之后的行索引（excel 透视表里的行标签）。
Columns：列索引（excel 里的列标签）。
Aggfunc:按照什么方式聚合。
Fill_value:空值用什么来代替。
Margins:是否需要合计。
Dropna:默认 True,不要全部是空值的列。
Margin_name:默认是 all,也就是我们合计的名字，通常我们使用‘合计’。

```
pd.pivot_table(merge_data,values=['销售书籍ID','交易金额'],index=['书籍名称'],columns='是否包邮',\
               aggfunc={'销售书籍ID':np.size,'交易金额':np.sum},fill_value=0,\
               margins=True,margins_name='合计')
```

```
Out[89]:
                交易金额                    销售书籍ID
是否包邮              否      是      合计       否     是     合计
书籍名称
c语言从入门到精通       0.0    66.0    66.0     0.0    1.0    1.0
c语言高级编程         0.0   117.0   117.0     0.0    2.0    2.0
java从入门到精通    163.0     0.0   163.0     2.0    0.0    2.0
php从入门到精通     330.0     0.0   330.0     4.0    0.0    4.0
php高级编程          0.0   137.0   137.0     0.0    2.0    2.0
python从入门到精通    0.0   156.0   156.0     0.0    2.0    2.0
python高级编程      63.0    85.0   148.0     1.0    1.0    2.0
```

由于语句太长，ipython 里没有办法换行，我粘贴到编辑器里进行了换行。
这里要说一下，aggfunc 里的键必须要在 values 里出现。这很容易理解，你首先需要一列数据，不然你对谁聚合呢。当然我们可以对一个数据进行多个聚合。
aggfunc={'销售书籍 ID':np.size,'交易金额':[np.sum,np.mean]}
我们可以使用这种方式对一个值多种聚合，但是有一个 bug，我在网上也没有找到答案，在对一个值进行多重聚合的时候 margins 需要设置为 false，不然会报错。

小结：pandas 的一些功能到这里也就介绍完了，说实话其中有些功能我都没有用过，只是有个群友逼逼着非要写完整，我就照着教程的大纲写了。今天是 2017 年的最后一天，还有 5 分钟就跨年了，在这里也祝愿 2018 年，能有一个好的开始，只要你努力，全世界都会为你让步。同时也欢迎喜欢编程，数学的同学来群里学习交流。只要我们每人分享出一点知识，那么我们的见识将会成百倍的增长。下一期我们开始画图，matplotlib，你准备好了吗？QQ 交流群：518980304