Python data aggregation and packet operations (1) -groupby mechanics

Foreword

Python's PANDAS package is very powerful and flexible.《Python for Data Analysis》This book is detailed in detail in detail, but some details are not common to forget, and intend to summarize this part of this part in the blog to review. According to the chapters in the book, this part of this knowledge includes the following four parts:

1.Groupby Mechanics (GroupBY technology)

2.Data aggregation (data aggregation)

3.Group-Wise Operation and Transformation (Group-level operation and conversion)

4.PIVOT TABLES AND CROSS-Tabulation (Pivotive and Crosslist)

This article is the first part, introducing GROUPBY technology.

First, the packet principle

core:

1. Regardless of the grouping keys, lists, dictionaries, Series, which can be grouped to grouch as long as they are consistent with the shaft length of the group variable.

2. Default AXIS = 0 Packet by line, specify the AXIS = 1 pair of column packets.

The process of packet operations for data can be summarized as: split-appl- combine three steps:

1. Packet data by key (key) or packet variable.

2. For each group to apply our functions, this step is very flexible, which can be a Python's own function, which can be the function we have written by it.

3. Aggregate the result after the function calculates.

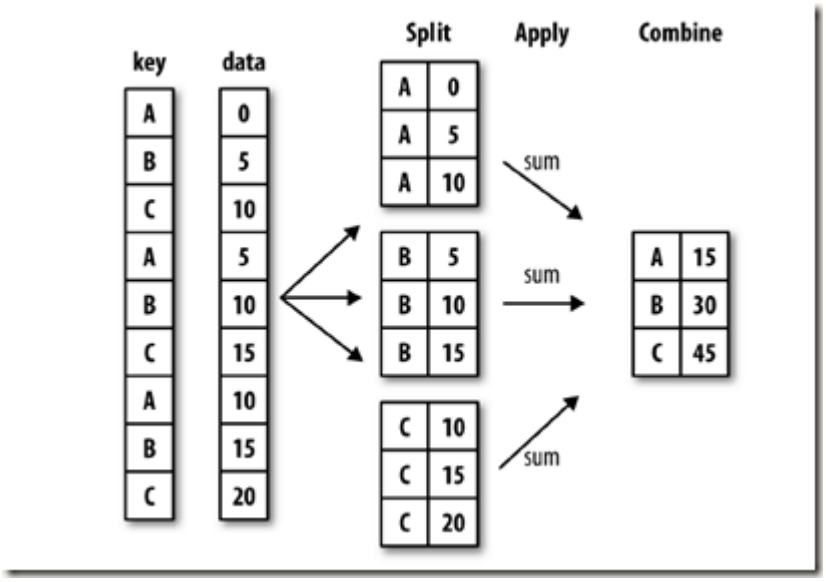

Figure 1: Packet polymerization principle (picture from "Python for Data Analysis" p

```
import pandas as pd
importnumpy as np

df= pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
'key2' : ['one', 'two', 'one', 'two', 'one'],
'data1' :np.random.randn(5),
'data2' :np.random.randn(5)})
```

df

	data1	data2	key1	key2
0	1.338459	-0.188768	a	one
1	-0.158292	0.672239	a	two
2	-0.873731	0.354981	b	one
3	-1.146545	-1.198674	b	two
4	-0.635887	-2.201771	a	one

We use Key1 as our packet key value, packet DATA1, and then ask for average of each group:

```
grouped = df['data1'].groupby(df['key1'])
```

The syntax is very simple, but here you need to pay attention to the data type of GroupD, it is not a data box, but a Groupby object.

```
grouped
```

```
<pandas.core.groupby.SeriesGroupBy object at 0x0000000008883D68>
```

In fact, in this step, we don't have any calculations just create a GroupBy object after creating a key1 group, and any of our follow-up function is based on this object.

Ask average:

```
grouped.mean()
```

Just, we just used key1 to group, we can also use two packet variables, and resolve through the UNSTACK method:

```
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
means
```

```
key1  key2
a     one     0.351286
      two    -0.158292
b     one    -0.873731
      two    -1.146545
Name: data1, dtype: float64
```

```
means.unstack
```

The above our grouping variables are series inside the DF, in fact, as long as the key1 is long, can:

```
states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
years= np.array([2005, 2005, 2006, 2005, 2006])
df['data1'].groupby([states, years]).mean()
```

```
California  2005   -0.158292
            2006   -0.873731
Ohio        2005    0.095957
            2006   -0.635887
Name: data1, dtype: float64
```

Second, iterates the group

The GroupBy object supports iterative operations, which generates a binary group consisting of packet variable names and data blocks:

```
for name, group indf.groupby('key1'):
print name
print group
```

```
a
      data1     data2 key1 key2
0  1.338459 -0.188768    a  one
1 -0.158292  0.672239    a  two
4 -0.635887 -2.201771    a  one
b
      data1     data2 key1 key2
2 -0.873731  0.354981    b  one
3 -1.146545 -1.198674    b  two
```

If the packet variables are two:

```
for (k1,k2), group indf.groupby(['key1','key2']):
print k1,k2
print group
```

```
a one
      data1     data2 key1 key2
0  1.338459 -0.188768    a  one
4 -0.635887 -2.201771    a  one
a two
      data1     data2 key1 key2
1 -0.158292  0.672239    a  two
b one
      data1     data2 key1 key2
2 -0.873731  0.354981    b  one
b two
      data1     data2 key1 key2
3 -1.146545 -1.198674    b  two
```

We can translate the above results to List or Dict to see what the results are:

```
list(df.groupby(['key1','key2']))
```

```
[(('a', 'one'),       data1     data2 key1 key2
  0  1.338459 -0.188768     a  one
  4 -0.635887 -2.201771     a  one),
 (('a', 'two'),       data1     data2 key1 key2
  1 -0.158292  0.672239     a  two),
 (('b', 'one'),       data1     data2 key1 key2
  2 -0.873731  0.354981     b  one),
 (('b', 'two'),       data1     data2 key1 key2
  3 -1.146545 -1.198674     b  two)]
```

I don't know clearly, let's take a look at the first element of this list:

```
list(df.groupby(['key1','key2']))[0]
```

Similarly, we can also convert the results to a DICT (Dictionary):

```
dict(list(df.groupby(['key1','key2'])))
```

```
{('a', 'one'):       data1     data2 key1 key2
 0  1.338459 -0.188768     a  one
 4 -0.635887 -2.201771     a  one, ('a', 'two'):       data1     data2 key1 key2
 1 -0.158292  0.672239     a  two, ('b', 'one'):       data1     data2 key1 key2
 2 -0.873731  0.354981     b  one, ('b', 'two'):       data1     data2 key1 key2
 3 -1.146545 -1.198674     b  two}
```

```
dict(list(df.groupby(['key1','key2'])))[('a','one')]
```

	data1	data2	key1	key2
0	1.338459	-0.188768	a	one
4	-0.635887	-2.201771	a	one

All above is based on row, because the groupBY is grouped by default, we can specify the AXIS

= 1 direction (column direction) to group:

```
grouped=df.groupby(df.dtypes,axis=1)
list(grouped)[0]
```

```
[(dtype('float64'),       data1     data2
  0  1.338459 -0.188768
  1 -0.158292  0.672239
  2 -0.873731  0.354981
  3 -1.146545 -1.198674
  4 -0.635887 -2.201771), (dtype('O'),   key1 key2
  0    a  one
  1    a  two
  2    b  one
  3    b  two
  4    a  one)]
```

```
dict(list(grouped))
```

note,

```
'''The following two statement functions are the same'''
df.groupby('key1')['data1']
df.data1.groupby(df.key1)
```

Third, group through the dictionary

```
people = pd.DataFrame(np.random.randn(5, 5),
columns=['a', 'b', 'c', 'd', 'e'],
index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
people.ix[2:3, ['b', 'c']] = np.nan# Add missing value
people
```

	a	b	c	d	e
Joe	-1.334247	-0.198907	0.739013	0.028262	-0.635999
Steve	-1.601897	0.932179	0.744713	0.048340	-0.695760
Wes	-0.383898	NaN	NaN	-0.841415	-0.599413
Jim	-0.390207	0.454561	-1.263529	-1.209548	1.162338
Travis	-0.072244	0.484924	1.238344	0.071216	-0.063052

If we want to get aggregated by column, what should I do?

Based on the actual situation, we have established a dictionary according to the actual situation, then incorporate this dictionary to GroupBy, remember to specify axis = 1, because we are packet aggregation:

```
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
'd': 'blue', 'e': 'red', 'f' :'orange'}
by_columns=people.groupby(mapping,axis=1)
by_columns.mean()
```

	blue	red
Joe	-0.548280	-1.198571
Steve	-0.157796	0.250417
Wes	0.826476	1.244673
Jim	-0.318242	-0.358831
Travis	-0.500064	-0.775494

Since we can group columns by incoming a dictionary, you can also group columns by passing Series (INDEX in Series is a key in the dictionary):

```
map_series =pd.Series(mapping)
people.groupby(map_series,axis=1).count()
```

	blue	red
Joe	2	3
Steve	2	3
Wes	1	2
Jim	2	3
Travis	2	3

Fourth, group through functions

When we grouped, Dict and Series were used to build a map. For some complex demands, we can directly transfer the function names to the GroupBy function to group, take the PEOPLE data as an example, if we want to group, groups of key is How do you do the letter length of each person? Compare the direct ideas are to find a length relative to each name, establish an array, then incorporate this array to GroupBy, let's test:

```
l=[len(x) for x inpeople.index]
people.groupby(l).count()
```

The program is feasible, then there is a faster and more beautiful way? Of course, we only need to pass the name of the Len to GroupBY:

```
people.groupby(len).count()
```

In addition to transfer functions, we can also use functions and dict, series, array. After all, it will be allocated to arrays:

```
key_list = ['one', 'one', 'one', 'two', 'two']
people.groupby([len, key_list]).min()
```

V, group according to the index level

Just one level, only one level, when the data has multi-level index, you can specify the index we want to group via Level, pay attention to use axis = 1 to repay:

```
columns = pd.MultiIndex.from_arrays([['Asian', 'Asian', 'Asian', 'America', 'America'],
    ['China','Japan','Singapore','UnitedStates','Canada']], names=['continent', 'country'])
hier_df= pd.DataFrame(np.random.randn(4, 5), columns=columns)
hier_df
```