Python_N4_20161008_AaronYu

October 8, 2016

0.1 Summary

Pandas! Pandas! Pandas!

0.1.1 Dealing with 1 DataFrame

- 1.Data Transformation
  - lambda
  - map
  - apply
- 2.Data Aggregation & Group Operations
  - groupby
  - aggregate
  - transform
  - apply
- 3.Pivot Table & Cross Table
  - pivot_table
  - crosstab

0.1.2 Dealing with 2+ DataFrames

- 4.Combining & Merging
  - concat
  - merge

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

0.2 Data Transformation

0.2.1 lambda

- An anonymous function, a fancy way of defining a function.

```
In [7]: l = lambda x:x**2
        l(3)
```

```
Out[7]: 9

In [8]: def fun(x):
            return x**2
        fun(3)

Out[8]: 9
```

0.2.2 map

- map values of Series using input correspondence (which can be a dict, Series, or function)

```
In [10]: ser = pd.Series(['a','b','f'])
         ser

Out[10]: 0    a
         1    b
         2    f
         dtype: object

In [11]: ser.map({'a':'excellent','b':'fair','f':'failed'})

Out[11]: 0    excellent
         1         fair
         2       failed
         dtype: object

In [19]: ser.map(lambda x:x.upper())

Out[19]: 0    A
         1    B
         2    F
         dtype: object

In [16]: ser.map(str.upper)

Out[16]: 0    A
         1    B
         2    F
         dtype: object
```

0.2.3 apply

- Apply a function along any axis of a DataFrame

```
In [23]: frame = pd.DataFrame(np.arange(12).reshape(4,3), columns=list('bde'), inde
         frame

Out[23]:         b   d   e
         Utah    0   1   2
         Ohio    3   4   5
         Texas   6   7   8
         Oregon  9  10  11
```

```
In [24]: f = lambda x: x.max() - x.min()
         frame.apply(f)

Out[24]: b    9
         d    9
         e    9
         dtype: int64

In [25]: frame.apply(f, axis = 1)

Out[25]: Utah      2
         Ohio      2
         Texas     2
         Oregon    2
         dtype: int64

In [26]: frame.apply(lambda x:x.sum())

Out[26]: b    18
         d    22
         e    26
         dtype: int64

In [28]: frame.sum()

Out[28]: b    18
         d    22
         e    26
         dtype: int64
```

0.3 Data Aggregation & Group Operations

- groupby creates grouped objects
- aggregate, transform, apply act on grouped objects

0.3.1 groupby

- Create a groupby object for futher use

```
In [51]: df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'], 'key2' : ['one', 't
                            'data1' : np.arange(5), 'data2' : np.random.randint(2,10,5
         df

Out[51]:    data1  data2 key1 key2
         0      0      5    a  one
         1      1      6    a  two
         2      2      2    b  one
         3      3      5    b  two
         4      4      7    a  one
```

- Group using own columns(s)

```
In [52]: grouped = df.groupby(df['key1'])
         grouped

Out[52]: <pandas.core.groupby.DataFrameGroupBy object at 0x111f08ba8>

In [53]: df.groupby(['key1','key2']).sum()

Out[53]:            data1  data2
         key1 key2
         a    one       4     12
              two       1      6
         b    one       2      2
              two       3      5
```

- Group using equal-length ndarray

```
In [54]: states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
         years = np.array([2005, 2005, 2006, 2005, 2006])
         df.groupby([states,years]).mean()

Out[54]:                  data1  data2
         California 2005    1.0    6.0
                    2006    2.0    2.0
         Ohio       2005    1.5    5.0
                    2006    4.0    7.0
```

- Set arguments as_index = False to keep the columns

```
In [55]: df.groupby(['key1','key2'], as_index = False).sum()

Out[55]:   key1 key2  data1  data2
         0    a  one      4     12
         1    a  two      1      6
         2    b  one      2      2
         3    b  two      3      5
```

0.3.2 aggregate

- Turn array into scalar value

```
In [56]: df

Out[56]:    data1  data2 key1 key2
         0      0      5    a  one
         1      1      6    a  two
         2      2      2    b  one
         3      3      5    b  two
         4      4      7    a  one
```

```
In [70]: df.groupby('key1').max()

Out[70]:       data1  data2 key2
         key1
         a          4      7  two
         b          3      5  two

In [72]: df.groupby('key1').sum()

Out[72]:       data1  data2
         key1
         a          5     18
         b          5      7

In [62]: g = lambda x:x.max() - x.min()
         df.groupby('key1').agg(g)

Out[62]:       data1  data2
         key1
         a          4      2
         b          1      3

In [63]: def peak_bottom(x):
             return x.max() - x.min()
         df.groupby('key1').agg(peak_bottom)

Out[63]:       data1  data2
         key1
         a          4      2
         b          1      3

In [69]: df.groupby('key1').agg([np.max, np.min, peak_bottom])

Out[69]:      data1                   data2
               amax amin peak_bottom  amax amin peak_bottom
         key1
         a        4    0           4     7    5           2
         b        3    2           1     5    2           3

In [74]: df.groupby('key1').agg({'data1': np.max, 'data2': np.min})

Out[74]:       data2  data1
         key1
         a          5      4
         b          2      3
```

0.3.3 transform

- Transform applies a function to each group, then places the result in the approciate locations.

```
In [89]: people = pd.DataFrame(np.random.randint(20, 50, 10).reshape(5,2), columns=
                               index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
         people

Out[89]:         age  income
         Joe      31      30
         Steve    37      41
         Wes      34      41
         Jim      27      40
         Travis   41      22

In [90]: key = ['one', 'two', 'one', 'two', 'one']

In [91]: people.groupby(key).agg(np.min)

Out[91]:      age  income
         one   31      22
         two   27      40

In [92]: people.groupby(key).transform(np.min)

Out[92]:         age  income
         Joe      31      22
         Steve    27      40
         Wes      31      22
         Jim      27      40
         Travis   31      22

In [93]: def minus_min(x):
             return x - x.min()

In [94]: people.groupby(key).transform(minus_min)

Out[94]:         age  income
         Joe       0       8
         Steve    10       1
         Wes       3      19
         Jim       0       0
         Travis   10       0
```

0.3.4 apply

- Split - Apply - Combine
- apply splits the object into pieces, invokes the passed function on each piece, then attempts to concatenate the pieces together.

```
In [110]: tips = pd.read_csv('N3_tips.csv').dropna()
          tips = tips[:10].reset_index(drop = True)
          tips
```

Out[110]:

	total_bill	tip	sex	smoker	day	time	size
0	12.90	1.10	Female	Yes	Sat	Dinner	2.0
1	11.87	1.63	Female	No	Thur	Lunch	2.0
2	3.07	1.00	Female	Yes	Sat	Dinner	1.0
3	24.08	2.92	Female	No	Thur	Lunch	4.0
4	18.71	4.00	Male	Yes	Thur	Lunch	3.0
5	24.55	2.00	Male	No	Sun	Dinner	4.0
6	13.94	3.06	Male	No	Sun	Dinner	2.0
7	10.33	2.00	Female	No	Thur	Lunch	2.0
8	18.29	3.76	Male	Yes	Sat	Dinner	4.0
9	8.77	2.00	Male	No	Sun	Dinner	2.0

```
In [122]: def when_people_get_generous(df, n = 2):
              return df.sort_values(by = 'tip', ascending = False)[:n]

In [123]: tips.groupby('time', as_index = False).apply(when_people_get_generous).re
```

Out[123]:

	total_bill	tip	sex	smoker	day	time	size
0	18.29	3.76	Male	Yes	Sat	Dinner	4.0
1	13.94	3.06	Male	No	Sun	Dinner	2.0
2	18.71	4.00	Male	Yes	Thur	Lunch	3.0
3	24.08	2.92	Female	No	Thur	Lunch	4.0

```
In [124]: def who_are_generous(df, n = 2):
              return df.sort_values(by = 'tip', ascending = False)[:n]
          tips.groupby('sex', as_index = False).apply(who_are_generous).reset_index
```

Out[124]:

	total_bill	tip	sex	smoker	day	time	size
0	24.08	2.92	Female	No	Thur	Lunch	4.0
1	10.33	2.00	Female	No	Thur	Lunch	2.0
2	18.71	4.00	Male	Yes	Thur	Lunch	3.0
3	18.29	3.76	Male	Yes	Sat	Dinner	4.0

0.4 Pivot Table & Cross Table

0.4.1 pivot_table

```
In [128]: pt = pd.pivot_table(tips, values = 'total_bill', index = ['day', 'time'],
          pt
```

Out[128]:

sex		Female	Male
day	time		
Sat	Dinner	15.97	18.29
Sun	Dinner	NaN	47.26
Thur	Lunch	46.28	18.71

```
In [131]: pd.pivot_table(tips, values = 'tip', index = ['time'], columns = ['smoker
```

```
Out[131]: smoker  No  Yes
          time
          Dinner   3    3
          Lunch    3    1
```

0.4.2 crosstab

- A crosstab (cross-tabing) is a special case of a pivot table
- By default, crosstab computes a frequency table of the factors

```
In [138]: pd.crosstab(tips.time, tips.smoker)
```

```
Out[138]: smoker  No  Yes
          time
          Dinner   3    3
          Lunch    3    1
```

```
In [139]: pd.crosstab(tips.time, tips.smoker, margins = True)
```

```
Out[139]: smoker  No  Yes  All
          time
          Dinner   3    3    6
          Lunch    3    1    4
          All      6    4   10
```

0.5 Combining & Merging

0.5.1 concat

- Numpy - concatenate

```
In [141]: arr = np.arange(6).reshape((2,3))
          arr
```

```
Out[141]: array([[0, 1, 2],
                 [3, 4, 5]])
```

```
In [143]: np.concatenate([arr, arr], axis = 1)
```

```
Out[143]: array([[0, 1, 2, 0, 1, 2],
                 [3, 4, 5, 3, 4, 5]])
```

- Pandas - concat

```
In [144]: df = pd.DataFrame(arr)
          df
```

```
Out[144]:    0  1  2
          0  0  1  2
          1  3  4  5
```

```
In [145]: pd.concat([df, df])
```

```
Out[145]:    0  1  2
          0  0  1  2
          1  3  4  5
          0  0  1  2
          1  3  4  5
```

```
In [146]: pd.concat([df, df], axis = 1)
```

```
Out[146]:    0  1  2  0  1  2
          0  0  1  2  0  1  2
          1  3  4  5  3  4  5
```

0.5.2 merge

```
In [147]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a'], 'data1': range(5)})
          df2 = pd.DataFrame({'key': ['a', 'b', 'd'], 'data2': range(3)})
```

```
In [149]: df_merge = pd.merge(df1, df2, on = 'key')
          df_merge
```

```
Out[149]:    data1 key  data2
          0      0   b      1
          1      1   b      1
          2      2   a      0
          3      4   a      0
```

- By default, it's inner join; we can set argument how = 'left' to enable a left join

```
In [152]: df_merge = pd.merge(df1, df2, how = 'left')
          df_merge
```

```
Out[152]:    data1 key  data2
          0      0   b    1.0
          1      1   b    1.0
          2      2   a    0.0
          3      3   c    NaN
          4      4   a    0.0
```

```
In [150]: df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a'],'data1': range(5)})
          df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],'data2': range(3)})
```

```
In [151]: df_merge = pd.merge(df3, df4, left_on='lkey', right_on='rkey')
          df_merge
```

```
Out[151]:    data1 lkey  data2 rkey
          0      0    b      1    b
          1      1    b      1    b
          2      2    a      0    a
          3      4    a      0    a
```