DataAggregation

October 1, 2020

1 데이터 집계와 그룹연산

데이터 집계에서 다루어질 내용 > 하나이상의 키(key)를 이용해서 pandas객체를 여러 조각으로 나누는 방법
> 합계, 평균, 표준편자, 사용자 정의 함수 같은 그룹 요약통계를 계산하는 방법
> 정규화, 선형 회귀, 동급 또는 부분집합 선택 같은 집단 내 변형이나 다른 조작을 적용하는 방법
> 피벗테이블과 교차 알람표를 구하는 방법
> 변위치 분석과 다른 통계집단을 분석 수행하는 방법

1.1 GroupBy 메카닉

그룹 연산의 첫번째 단계에서는 Series, DataFrame 같은 pandas객체나 아니면 다른 객체에 들어 있는 데이터를 하나 이상의 키를 기준으로 분리

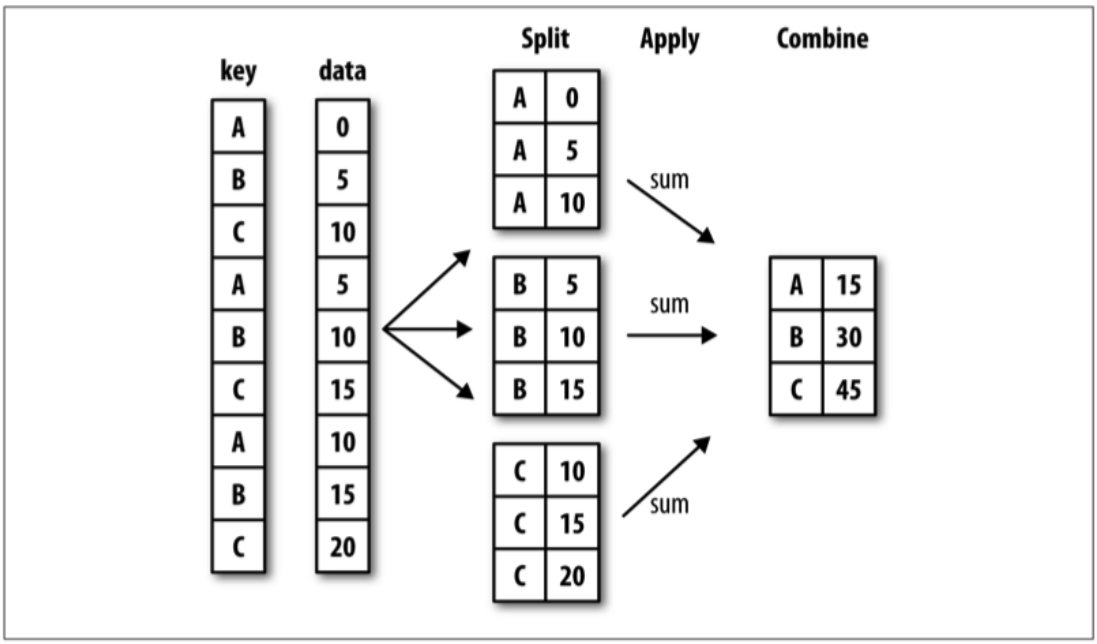

Figure 10-1. Illustration of a group aggregation

각 그룹의 색인은 다음과 같이 다양한 형태가 될 수 있으며 모두 같은 타입일 필요도 없다. - 그룹으로 묶을 축과 동일한 길이의 리스트나 배열 - DataFrame의 컬럼 이름을 지칭하는 값 - 그룹으로 묶을 값과 그룹 이름에 대응하는 사전이나 Series객체 - 축 색인 혹인 색인내의 개별 이름에 대해 실행되는 함수

[1]:
```
import pandas as pd
import numpy as np
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                   'key2' : ['one', 'two', 'one', 'two', 'one'],
                   'data1' : np.random.randint(10,size=(5)),
                   'data2' : np.random.randint(10,size=(5))})
df
```

[1]:

	key1	key2	data1	data2
0	a	one	0	0
1	a	two	9	4
2	b	one	1	1
3	b	two	5	1
4	a	one	2	5

[2]:
```
grouped = df['data1'].groupby(df['key1'])
grouped
```

[2]:
```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x107810880>
```

데이터를 key1 으로 묶고 각 그룹에서 data1의 평균구하기
groupBy객체의 mean메서드를 사용

[3]:
```
grouped.mean()
```

[3]:
```
key1
a    3.666667
b    3.000000
Name: data1, dtype: float64
```

여러개의 배열을 리스트로 넘기면 다음과 같은 결과로 두개의 색인으로 묶이고, 계층적인 색인을 가지는 Series를 얻을 수 있음

[4]:
```
means = df['data1'].groupby([df['key1'], df['key2']]).mean()
means
```

[4]:
```
key1  key2
a     one     1
      two     9
b     one     1
      two     5
Name: data1, dtype: int64
```

[5]:
```
means.unstack()
```

[5]:

key2	one	two
key1		
a	1	9

```
b       1    5
```

[6]:
```
df['data1']
```

[6]:
```
0    0
1    9
2    1
3    5
4    2
Name: data1, dtype: int64
```

[7]:
```
states = np.array(['Ohio', 'California', 'California', 'Ohio', 'Ohio'])
years = np.array([2004, 2005, 2006, 2005, 2006])
df['data1'].groupby([states, years]).mean()
```

[7]:
```
California  2005    9
            2006    1
Ohio        2004    0
            2005    5
            2006    2
Name: data1, dtype: int64
```

[8]:
```
df.groupby('key1').mean()
```

[8]:
```
          data1  data2
key1
a     3.666667    3.0
b     3.000000    1.0
```

[9]:
```
df
```

[9]:
```
  key1 key2  data1  data2
0    a  one      0      0
1    a  two      9      4
2    b  one      1      1
3    b  two      5      1
4    a  one      2      5
```

[10]:
```
df.groupby(['key1', 'key2']).mean()
```

[10]:
```
            data1  data2
key1 key2
a    one      1.0    2.5
     two      9.0    4.0
b    one      1.0    1.0
     two      5.0    1.0
```

[11]:
```
df.groupby(['key1', 'key2']).size()
```

[11]:
```
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

1.1.1 그룹간 순회하기

groupby 객체는 이터레이션을 지원하는데, 그룹 이름과 그에 따른 데이터 묶음을 튜플로 반환

[12]:
```
df
```

[12]:
```
   key1 key2  data1  data2
0     a  one      0      0
1     a  two      9      4
2     b  one      1      1
3     b  two      5      1
4     a  one      2      5
```

[13]:
```
for name, group in df.groupby('key1'):
    print(name)
    print(group)
```

```
a
  key1 key2  data1  data2
0    a  one      0      0
1    a  two      9      4
4    a  one      2      5
b
  key1 key2  data1  data2
2    b  one      1      1
3    b  two      5      1
```

[14]:
```
for (k1, k2), group in df.groupby(['key1', 'key2']):
    print((k1, k2))
    print(group)
```

```
('a', 'one')
  key1 key2  data1  data2
0    a  one      0      0
4    a  one      2      5
('a', 'two')
  key1 key2  data1  data2
1    a  two      9      4
('b', 'one')
```

```
  key1 key2  data1  data2
2    b  one      1      1
('b', 'two')
  key1 key2  data1  data2
3    b  two      5      1
```

원하는 데이터만 고르기 위해서 그룹별 데이터를 사전형으로 쉽게 바꾸어 사용 가능

[15]:
```
pieces = dict(list(df.groupby('key1')))
pieces['b']
```

[15]:
```
  key1 key2  data1  data2
2    b  one      1      1
3    b  two      5      1
```

axis = 0 에 대해서 그룹을 만드는데 다른 축으로 그룹을 만드는 것도 가능 아래의 예제는 df의 컬럼을 dtype에 따라 그룹으로 묶기 가능

[16]:
```
df
```

[16]:
```
  key1 key2  data1  data2
0    a  one      0      0
1    a  two      9      4
2    b  one      1      1
3    b  two      5      1
4    a  one      2      5
```

[17]:
```
df.dtypes
grouped = df.groupby(df.dtypes, axis=1)
```

[18]:
```
for dtype, group in grouped:
    print(dtype)
    print(group)
```

```
int64
   data1  data2
0      0      0
1      9      4
2      1      1
3      5      1
4      2      5
object
  key1 key2
0    a  one
1    a  two
2    b  one
3    b  two
4    a  one
```

1.1.2 컬럼이나 컬럼의 일부만 선택하기

[19]:
```
df
```

[19]:
```
  key1 key2  data1  data2
0    a  one      0      0
1    a  two      9      4
2    b  one      1      1
3    b  two      5      1
4    a  one      2      5
```

[20]:
```
df.groupby('key1')['data1']
df.groupby('key1')[['data2']]
```

[20]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x115f039d0>
```

[21]:
```
df['data1'].groupby(df['key1'])
a= df[['data2']].groupby(df['key1'])
for index, data in a:
    print(index)
    print(data)
```

```
a
   data2
0      0
1      4
4      5
b
   data2
2      1
3      1
```

아래의 예는 데이터에서 data2컬럼에 대해서만 평균을 구하고 결과를 DataFrame으로 받고 싶다면 아래와 같이 작성

[22]:
```
df.groupby(['key1', 'key2'])[['data2']].mean()
```

[22]:
```
           data2
key1 key2
a    one     2.5
     two     4.0
b    one     1.0
     two     1.0
```

[23]:
```
s_grouped = df.groupby(['key1', 'key2'])['data2']
s_grouped
s_grouped.mean()
```

[23]:
```
key1  key2
a     one     2.5
      two     4.0
b     one     1.0
      two     1.0
Name: data2, dtype: float64
```

1.1.3 사전과 Series에서 그룹핑하기

각 컬럼을 나타낼 그룹 목록이 있고, 그룹별로 컬럼의 값을 모두 더한다고 할 경우

[24]:
```
people = pd.DataFrame(np.random.randint(10, size=(5,5)),
                      columns=['a', 'b', 'c', 'd', 'e'],
                      index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])
people
```

[24]:

	a	b	c	d	e
Joe	4	4	0	1	0
Steve	3	0	3	3	1
Wes	9	4	8	7	3
Jim	6	4	8	4	6
Travis	9	2	4	7	7

[25]:
```
people.iloc[2:3, [1, 2]] = np.nan # Add a few NA values
people
```

[25]:

	a	b	c	d	e
Joe	4	4.0	0.0	1	0
Steve	3	0.0	3.0	3	1
Wes	9	NaN	NaN	7	3
Jim	6	4.0	8.0	4	6
Travis	9	2.0	4.0	7	7

[26]:
```
mapping = {'a': 'red', 'b': 'red', 'c': 'blue',
           'd': 'blue', 'e': 'red', 'f' : 'orange'}
```

[27]:
```
by_column = people.groupby(mapping, axis=1)
by_column.sum()
```

[27]:

	blue	red
Joe	1.0	8.0
Steve	6.0	4.0
Wes	7.0	12.0
Jim	12.0	16.0
Travis	11.0	18.0

[28]:
```
map_series = pd.Series(mapping)
map_series
```

[28]:
```
a       red
b       red
c      blue
d      blue
e       red
f    orange
dtype: object
```

[29]:
```
people
```

[29]:

	a	b	c	d	e
Joe	4	4.0	0.0	1	0
Steve	3	0.0	3.0	3	1
Wes	9	NaN	NaN	7	3
Jim	6	4.0	8.0	4	6
Travis	9	2.0	4.0	7	7

[30]:
```
people.groupby(map_series, axis=1).count()
```

[30]:

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

1.1.4 함수로 그룹핑하기

[31]:
```
people
```

[31]:

	a	b	c	d	e
Joe	4	4.0	0.0	1	0
Steve	3	0.0	3.0	3	1
Wes	9	NaN	NaN	7	3
Jim	6	4.0	8.0	4	6
Travis	9	2.0	4.0	7	7

위의 people은 DataFrame은 사람의 이름을 색인값으로 사용.
만약 사람의 이름의 길이 별로 그룹을 묶고 싶다면 길이가 담긴 배열을 만들어 넘기는 대신 len 함수 사용 가능.

[32]:
```
people.groupby(len).sum()
```

[32]:
```
    a    b    c   d  e
3  19  8.0  8.0  12  9
5   3  0.0  3.0   3  1
6   9  2.0  4.0   7  7
```

내부적으로는 모두 배열로 변환되므로 함수를 배열, 사전 또는 Series와 함께 섞어 쓰더라도 전혀 문제가 되지 않음

[33]:
```
key_list = ['one', 'one', 'one', 'two', 'two']
people.groupby([len, key_list]).min()
```

[33]:
```
       a    b    c  d  e
3 one  4  4.0  0.0  1  0
  two  6  4.0  8.0  4  6
5 one  3  0.0  3.0  3  1
6 two  9  2.0  4.0  7  7
```

1.1.5 색인 단계로 그룹핑하기

계층적으로 색인된 데이터는 축 색인의 단계중 하나를 사용해서 편리하게 집계 가능

[34]:
```
columns = pd.MultiIndex.from_arrays([['US', 'US', 'US', 'JP', 'JP'],
                                     [1, 3, 5, 1, 3]],
                                     names=['city', 'tenor'])
hier_df = pd.DataFrame(np.random.randn(4, 5), columns=columns)
hier_df
```

[34]:
```
city         US                            JP
tenor         1         3         5         1         3
0      0.818702 -0.310974  0.608758 -0.384082  1.353732
1     -0.081016  0.161245  0.812204 -0.367812 -1.692647
2      1.003298  1.142377  0.321394  0.761760 -1.115075
3     -0.526310 -1.526414 -1.420809  0.596818 -1.033814
```

level 예약어를 사용해서 레벨 번호나 이름을 넘기면 가능

[35]:
```
hier_df.groupby(level='city', axis=1).sum()
```

[35]:
```
city        JP        US
0     0.969650  1.116486
1    -2.060459  0.892434
2    -0.353315  2.467068
3    -0.436996 -3.473533
```

1.2 데이터 집계

[36]:
```
df
```

[36]:
```
  key1 key2  data1  data2
0    a  one      0      0
1    a  two      9      4
2    b  one      1      1
3    b  two      5      1
4    a  one      2      5
```

[37]:
```
grouped = df.groupby('key1')
grouped['data1'].quantile(0.5)
```

[37]:
```
key1
a    2.0
b    3.0
Name: data1, dtype: float64
```

자신만의 데이터 집계함수를 사용하려면 배열의 agg메서드에 해당 함수를 넣으면 됨

[38]:
```
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

[38]:
```
      data1  data2
key1
a         9      5
b         4      0
```

describe같은 메서드는 데이터를 집계하지 않는데도 잘 작동

[39]:
```
grouped.describe()
```

[39]:
```
     data1                                                    data2                \
     count      mean       std  min  25%  50%  75%  max count mean       std
key1
a      3.0  3.666667  4.725816  0.0  1.0  2.0  5.5  9.0   3.0  3.0  2.645751
b      2.0  3.000000  2.828427  1.0  2.0  3.0  4.0  5.0   2.0  1.0  0.000000


      min  25%  50%  75%  max
key1
a     0.0  2.0  4.0  4.5  5.0
b     1.0  1.0  1.0  1.0  1.0
```

1.2.1 컬럼에 여러가지 함수 적용하기

[40]:
```
tips = pd.read_csv('examples/tips.csv')
# Add tip percentage of total bill
tips['tip_pct'] = tips['tip'] / tips['total_bill']
tips[:6]
```

[40]:

	total_bill	tip	smoker	day	time	size	tip_pct
0	16.99	1.01	No	Sun	Dinner	2	0.059447
1	10.34	1.66	No	Sun	Dinner	3	0.160542
2	21.01	3.50	No	Sun	Dinner	3	0.166587
3	23.68	3.31	No	Sun	Dinner	2	0.139780
4	24.59	3.61	No	Sun	Dinner	4	0.146808
5	25.29	4.71	No	Sun	Dinner	4	0.186240

컬럼에 따라 다른 함수를 사용해서 집계를 수행 하거나 열개의 함수를 한번에 적용하기 원한다면 쉽고 간단하게 사용가능

[41]:
```
grouped = tips.groupby(['day', 'smoker'])
```

[42]:
```
grouped_pct = grouped['tip_pct']
grouped_pct.agg('mean')
```

[42]:
```
day   smoker
Fri   No        0.151650
      Yes       0.174783
Sat   No        0.158048
      Yes       0.147906
Sun   No        0.160113
      Yes       0.187250
Thur  No        0.160298
      Yes       0.163863
Name: tip_pct, dtype: float64
```

함수 목록이나 함수 이름을 넘기면 함수 이름을 컬럼으로 하는 DataFrame을 얻을 수 있음

[43]:
```
grouped_pct.agg(['mean', 'std', peak_to_peak])
```

[43]:

		mean	std	peak_to_peak
day	smoker			
Fri	No	0.151650	0.028123	0.067349
	Yes	0.174783	0.051293	0.159925
Sat	No	0.158048	0.039767	0.235193
	Yes	0.147906	0.061375	0.290095
Sun	No	0.160113	0.042347	0.193226
	Yes	0.187250	0.154134	0.644685
Thur	No	0.160298	0.038774	0.193350
	Yes	0.163863	0.039389	0.151240

[44]:
```
grouped.agg({'tip' : np.max, 'size' : 'sum'})
grouped.agg({'tip_pct' : ['min', 'max', 'mean', 'std'],
             'size' : 'sum'})
```

[44]:

		tip_pct				size
		min	max	mean	std	sum
day	smoker					
Fri	No	0.120385	0.187735	0.151650	0.028123	9
	Yes	0.103555	0.263480	0.174783	0.051293	31
Sat	No	0.056797	0.291990	0.158048	0.039767	115
	Yes	0.035638	0.325733	0.147906	0.061375	104
Sun	No	0.059447	0.252672	0.160113	0.042347	167
	Yes	0.065660	0.710345	0.187250	0.154134	49
Thur	No	0.072961	0.266312	0.160298	0.038774	112
	Yes	0.090014	0.241255	0.163863	0.039389	40

1.3 Apply: 일반적인 분리-적용-병합

상위 5개의 tip_pct 값을 고르기
특정 칼럼에서 가장 큰 값을 갖는 로우를 선택하는 함수 필요

[45]:
```
def top(df, n=5, column='tip_pct'):
    return df.sort_values(by=column)[-n:]
top(tips, n=6)
```

[45]:

	total_bill	tip	smoker	day	time	size	tip_pct
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
232	11.61	3.39	No	Sat	Dinner	2	0.291990
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

smoker그룹에 대해서 이 함수를 적용하면 다음과 가은 결과

[46]:
```
tips.groupby('smoker').apply(top)
```

[46]:

		total_bill	tip	smoker	day	time	size	tip_pct
smoker								
No	88	24.71	5.85	No	Thur	Lunch	2	0.236746
	185	20.69	5.00	No	Sun	Dinner	5	0.241663
	51	10.29	2.60	No	Sun	Dinner	2	0.252672
	149	7.51	2.00	No	Thur	Lunch	2	0.266312
	232	11.61	3.39	No	Sat	Dinner	2	0.291990
Yes	109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
	183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
	67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
	178	9.60	4.00	Yes	Sun	Dinner	2	0.416667

```
        172        7.25  5.15    Yes    Sun  Dinner     2  0.710345
```

apply 메서드를 넘길 함수가 추가적인 인자를 받는 다면 이 함수 이름 뒤에 붙여서 넘겨주면 가능

[47]:
```
tips.groupby(['smoker', 'day']).apply(top, n=1, column='total_bill')
```

[47]:
```
                 total_bill    tip smoker   day    time  size   tip_pct
smoker day
No     Fri  94        22.75   3.25     No   Fri  Dinner     2  0.142857
       Sat  212       48.33   9.00     No   Sat  Dinner     4  0.186220
       Sun  156       48.17   5.00     No   Sun  Dinner     6  0.103799
       Thur 142       41.19   5.00     No  Thur   Lunch     5  0.121389
Yes    Fri  95        40.17   4.73    Yes   Fri  Dinner     4  0.117750
       Sat  170       50.81  10.00    Yes   Sat  Dinner     3  0.196812
       Sun  182       45.35   3.50    Yes   Sun  Dinner     3  0.077178
       Thur 197       43.11   5.00    Yes  Thur   Lunch     4  0.115982
```

[48]:
```
result = tips.groupby('smoker')['tip_pct'].describe()
result
```

[48]:
```
        count      mean       std       min       25%       50%       75%  \
smoker
No      151.0  0.159328  0.039910  0.056797  0.136906  0.155625  0.185014
Yes      93.0  0.163196  0.085119  0.035638  0.106771  0.153846  0.195059

             max
smoker
No      0.291990
Yes     0.710345
```

[49]:
```
result.unstack('smoker')
```

[49]:
```
        smoker
count   No        151.000000
        Yes        93.000000
mean    No         0.159328
        Yes        0.163196
std     No         0.039910
        Yes        0.085119
min     No         0.056797
        Yes        0.035638
25%     No         0.136906
        Yes        0.106771
50%     No         0.155625
        Yes        0.153846
75%     No         0.185014
        Yes        0.195059
```

```
max     No          0.291990
        Yes         0.710345
dtype: float64
```

f = lambda x: x.describe() grouped.apply(f)

1.3.1 그룹 색인 생략하기

[50]:
```
tips.groupby('smoker', group_keys=False).apply(top)
```

[50]:

	total_bill	tip	smoker	day	time	size	tip_pct
88	24.71	5.85	No	Thur	Lunch	2	0.236746
185	20.69	5.00	No	Sun	Dinner	5	0.241663
51	10.29	2.60	No	Sun	Dinner	2	0.252672
149	7.51	2.00	No	Thur	Lunch	2	0.266312
232	11.61	3.39	No	Sat	Dinner	2	0.291990
109	14.31	4.00	Yes	Sat	Dinner	2	0.279525
183	23.17	6.50	Yes	Sun	Dinner	4	0.280535
67	3.07	1.00	Yes	Sat	Dinner	1	0.325733
178	9.60	4.00	Yes	Sun	Dinner	2	0.416667
172	7.25	5.15	Yes	Sun	Dinner	2	0.710345

1.3.2 변위치 분석과 버킷분석

pandas의 cut과 qcut메서드를 사용하여 선택한 크기만큼 혹은 표분 변위치에 따라 데이터를 나눌 수 있음
cut을 이용해서 등간격 구간으로 나누기

[51]:
```
frame = pd.DataFrame({'data1': np.random.randn(1000),
                      'data2': np.random.randn(1000)})
quartiles = pd.cut(frame.data1, 4)
quartiles[:10]
```

[51]:
```
0    (-1.47, 0.339]
1    (-1.47, 0.339]
2    (-1.47, 0.339]
3    (-1.47, 0.339]
4    (-1.47, 0.339]
5    (-1.47, 0.339]
6    (0.339, 2.148]
7    (-1.47, 0.339]
8    (0.339, 2.148]
9    (-1.47, 0.339]
Name: data1, dtype: category
Categories (4, interval[float64]): [(-3.286, -1.47] < (-1.47, 0.339] < (0.339,
2.148] < (2.148, 3.957]]
```

cut에서 반환된 categorical객체는 바로 groupby로 넘기기 가능

[52]:
```
def get_stats(group):
    return {'min': group.min(), 'max': group.max(),
            'count': group.count(), 'mean': group.mean()}
grouped = frame.data2.groupby(quartiles)
grouped.apply(get_stats).unstack()
```

[52]:

data1	min	max	count	mean
(-3.286, -1.47]	-2.368330	2.244605	73.0	0.118499
(-1.47, 0.339]	-3.024442	2.999532	528.0	-0.055516
(0.339, 2.148]	-2.709583	2.990369	387.0	-0.062723
(2.148, 3.957]	-3.033057	1.492883	12.0	-0.036099

표본 변위치에 기반하여 크기각 같은 버킷을 계산하기 위해서는 qcut을 사용

[53]:
```
# Return quantile numbers
grouping = pd.qcut(frame.data1, 10, labels=False)
grouped = frame.data2.groupby(grouping)
grouped.apply(get_stats).unstack()
```

[53]:

data1	min	max	count	mean
0	-2.368330	2.244685	100.0	0.065246
1	-1.988513	2.959305	100.0	-0.039974
2	-3.024442	2.999532	100.0	-0.055866
3	-1.854205	2.891143	100.0	-0.094500
4	-2.522434	2.356145	100.0	-0.144006
5	-2.470763	2.786807	100.0	0.060909
6	-2.709583	2.990369	100.0	0.078960
7	-2.508011	2.055839	100.0	0.057988
8	-2.345976	2.533089	100.0	-0.156430
9	-3.033057	1.992618	100.0	-0.226014

1.3.3 Example: 그룹에 따른 값으로 결측치 채우기

누락된 데이터를 정리할때 어떤 경우에는 dropna를 사용해서 데이터를 살펴보고 걸러내기가능
어떤 경우에는 누락된 값을 고정된 값이나 데이터로부터 도출된 어떤 값으로 채우고 싶을때에는 fillna 메서드를 사용 누락된 값을 평균값으로 대체

[54]:
```
s = pd.Series(np.random.randn(6))
s[::2] = np.nan
s
```

[54]:
```
0         NaN
1   -0.663811
2         NaN
3    0.606571
4         NaN
```

```
5    2.140735
dtype: float64
```

[55]:
```
s.fillna(s.mean())
```

[55]:
```
0    0.694498
1   -0.663811
2    0.694498
3    0.606571
4    0.694498
5    2.140735
dtype: float64
```

[56]:
```
states = ['Ohio', 'New York', 'Vermont', 'Florida',
          'Oregon', 'Nevada', 'California', 'Idaho']
group_key = ['East'] * 4 + ['West'] * 4
data = pd.Series(np.random.randint(10,size=(8)), index=states)
data
```

[56]:
```
Ohio          3
New York      9
Vermont       8
Florida       4
Oregon        6
Nevada        0
California    4
Idaho         0
dtype: int64
```

[57]:
```
data[['Vermont', 'Nevada', 'Idaho']] = np.nan
data
data.groupby(group_key).mean()
```

[57]:
```
East    5.333333
West    5.000000
dtype: float64
```

[58]:
```
fill_mean = lambda g: g.fillna(g.mean())
data
```

[58]:
```
Ohio          3.0
New York      9.0
Vermont       NaN
Florida       4.0
Oregon        6.0
Nevada        NaN
California    4.0
```

```
Idaho          NaN
dtype: float64
```

[59]:
```
data.groupby(group_key).apply(fill_mean)
```

[59]:
```
Ohio          3.000000
New York      9.000000
Vermont       5.333333
Florida       4.000000
Oregon        6.000000
Nevada        5.000000
California    4.000000
Idaho         5.000000
dtype: float64
```

[60]:
```
fill_values = {'East': 0.5, 'West': -1}
fill_func = lambda g: g.fillna(fill_values[g.name])
data.groupby(group_key).apply(fill_func)
```

[60]:
```
Ohio          3.0
New York      9.0
Vermont       0.5
Florida       4.0
Oregon        6.0
Nevada       -1.0
California    4.0
Idaho        -1.0
dtype: float64
```

1.3.4 Example: 그룹의 가중 평균과 상관관계

[61]:
```
df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
                                'b', 'b', 'b', 'b'],
                   'data': np.random.randn(8),
                   'weights': np.random.rand(8)})
df
```

[61]:

	category	data	weights
0	a	0.292888	0.838569
1	a	0.674027	0.456900
2	a	0.220865	0.651325
3	a	-0.762143	0.494381
4	b	-0.169229	0.248822
5	b	-0.853779	0.638314
6	b	-0.994059	0.871774
7	b	2.389166	0.425014

[62]:
```
grouped = df.groupby('category')
get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
grouped.apply(get_wavg)
```

[62]:
```
category
a    0.131345
b   -0.200672
dtype: float64
```

야후의 파이낸스에서 가져온 몇몇 주식과 s&p 500 지수(종목코드 SPX)의 종가 데이터를 살펴보자

[63]:
```
close_px = pd.read_csv('examples/stock_px_2.csv', parse_dates=True,
                       index_col=0)
close_px.info()
close_px[-4:]
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2214 entries, 2003-01-02 to 2011-10-14
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   AAPL    2214 non-null   float64
 1   MSFT    2214 non-null   float64
 2   XOM     2214 non-null   float64
 3   SPX     2214 non-null   float64
dtypes: float64(4)
memory usage: 86.5 KB
```

[63]:

	AAPL	MSFT	XOM	SPX
2011-10-11	400.29	27.00	76.27	1195.54
2011-10-12	402.19	26.96	77.16	1207.25
2011-10-13	408.43	27.18	76.37	1203.66
2011-10-14	422.00	27.27	78.11	1224.58

퍼센트의 변화율로 일일 수익률을 계산하여 연간 SPX 지수와의 상관 관계를 알아보기

[64]:
```
spx_corr = lambda x: x.corrwith(x['SPX'])
```

pct_change함수를 이용해서 close_px의 페센트 변화율을 계산

[65]:
```
rets = close_px.pct_change().dropna()
```

datetime에서 연도 속성만 반환하는 한줄짜리 함수를 이용하여 연도별 퍼센트 변화율

[66]:
```
get_year = lambda x: x.year
by_year = rets.groupby(get_year)
by_year.apply(spx_corr)
```

[66]:
```
          AAPL      MSFT       XOM  SPX
2003  0.541124  0.745174  0.661265  1.0
2004  0.374283  0.588531  0.557742  1.0
2005  0.467540  0.562374  0.631010  1.0
2006  0.428267  0.406126  0.518514  1.0
2007  0.508118  0.658770  0.786264  1.0
2008  0.681434  0.804626  0.828303  1.0
2009  0.707103  0.654902  0.797921  1.0
2010  0.710105  0.730118  0.839057  1.0
2011  0.691931  0.800996  0.859975  1.0
```

아래는 애플과 마이크로 소프트의 주가의 연간 상관관계

[67]:
```
by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
```

[67]:
```
2003    0.480868
2004    0.259024
2005    0.300093
2006    0.161735
2007    0.417738
2008    0.611901
2009    0.432738
2010    0.571946
2011    0.581987
dtype: float64
```

1.4 피벗테이블과 교차 일람표

피벗테이블은 앞에서 설명한 groupby 기능을 사용해서 계측정 색인 활용한 재형성 연산가능하고. DataFrame에는 pivot_table 메서드가 있어서

[68]:
```
tips.pivot_table(index=['day', 'smoker'])
```

[68]:
```
                size       tip   tip_pct  total_bill
day  smoker
Fri  No      2.250000  2.812500  0.151650   18.420000
     Yes     2.066667  2.714000  0.174783   16.813333
Sat  No      2.555556  3.102889  0.158048   19.661778
     Yes     2.476190  2.875476  0.147906   21.276667
Sun  No      2.929825  3.167895  0.160113   20.506667
     Yes     2.578947  3.516842  0.187250   24.120000
Thur No      2.488889  2.673778  0.160298   17.113111
     Yes     2.352941  3.030000  0.163863   19.190588
```

[69]:
```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker')
```

[69]:
```
                size             tip_pct
smoker            No       Yes        No       Yes
time   day
Dinner Fri    2.000000  2.222222  0.139622  0.165347
       Sat    2.555556  2.476190  0.158048  0.147906
       Sun    2.929825  2.578947  0.160113  0.187250
       Thur   2.000000       NaN  0.159744       NaN
Lunch  Fri    3.000000  1.833333  0.187735  0.188937
       Thur   2.500000  2.352941  0.160311  0.163863
```

[70]:
```
tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                 columns='smoker', margins=True)
```

[70]:
```
                size                     tip_pct
smoker            No       Yes       All        No       Yes       All
time   day
Dinner Fri    2.000000  2.222222  2.166667  0.139622  0.165347  0.158916
       Sat    2.555556  2.476190  2.517241  0.158048  0.147906  0.153152
       Sun    2.929825  2.578947  2.842105  0.160113  0.187250  0.166897
       Thur   2.000000       NaN  2.000000  0.159744       NaN  0.159744
Lunch  Fri    3.000000  1.833333  2.000000  0.187735  0.188937  0.188765
       Thur   2.500000  2.352941  2.459016  0.160311  0.163863  0.161301
All           2.668874  2.408602  2.569672  0.159328  0.163196  0.160803
```

[71]:
```
tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
                 aggfunc=len, margins=True)
```

[71]:
```
day             Fri   Sat   Sun  Thur    All
time   smoker
Dinner No       3.0  45.0  57.0   1.0  106.0
       Yes      9.0  42.0  19.0   NaN   70.0
Lunch  No       1.0   NaN   NaN  44.0   45.0
       Yes      6.0   NaN   NaN  17.0   23.0
All            19.0  87.0  76.0  62.0  244.0
```

[72]:
```
tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
                 columns='day', aggfunc='mean', fill_value=0)
```

[72]:
```
day                     Fri       Sat       Sun      Thur
time   size smoker
Dinner 1    No      0.000000  0.137931  0.000000  0.000000
            Yes     0.000000  0.325733  0.000000  0.000000
       2    No      0.139622  0.162705  0.168859  0.159744
            Yes     0.171297  0.148668  0.207893  0.000000
       3    No      0.000000  0.154661  0.152663  0.000000
            Yes     0.000000  0.144995  0.152660  0.000000
       4    No      0.000000  0.150096  0.148143  0.000000
```

```
                Yes      0.117750  0.124515  0.193370  0.000000
        5       No       0.000000  0.000000  0.206928  0.000000
                Yes      0.000000  0.106572  0.065660  0.000000
        6       No       0.000000  0.000000  0.103799  0.000000
Lunch   1       No       0.000000  0.000000  0.000000  0.181728
                Yes      0.223776  0.000000  0.000000  0.000000
        2       No       0.000000  0.000000  0.000000  0.166005
                Yes      0.181969  0.000000  0.000000  0.158843
        3       No       0.187735  0.000000  0.000000  0.084246
                Yes      0.000000  0.000000  0.000000  0.204952
        4       No       0.000000  0.000000  0.000000  0.138919
                Yes      0.000000  0.000000  0.000000  0.155410
        5       No       0.000000  0.000000  0.000000  0.121389
        6       No       0.000000  0.000000  0.000000  0.173706
```

1.4.1 Cross-Tabulations: Crosstab

[73]:
```
from io import StringIO
data = """\
Sample  Nationality  Handedness
1   USA  Right-handed
2   Japan    Left-handed
3   USA  Right-handed
4   Japan    Right-handed
5   Japan    Left-handed
6   Japan    Right-handed
7   USA  Right-handed
8   USA  Left-handed
9   Japan    Right-handed
10  USA  Right-handed"""
data = pd.read_table(StringIO(data), sep='\s+')
```

[74]:
```
data
```

[74]:
```
   Sample Nationality    Handedness
0       1         USA  Right-handed
1       2       Japan   Left-handed
2       3         USA  Right-handed
3       4       Japan  Right-handed
4       5       Japan   Left-handed
5       6       Japan  Right-handed
6       7         USA  Right-handed
7       8         USA   Left-handed
8       9       Japan  Right-handed
9      10         USA  Right-handed
```

[75]:
```
pd.crosstab(data.Nationality, data.Handedness, margins=True)
```

[75]:
Handedness	Left-handed	Right-handed	All
Nationality			
Japan	2	3	5
USA	1	4	5
All	3	7	10

[76]:
```
pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
```

[76]:
smoker		No	Yes	All
time	day			
Dinner	Fri	3	9	12
	Sat	45	42	87
	Sun	57	19	76
	Thur	1	0	1
Lunch	Fri	1	6	7
	Thur	44	17	61
All		151	93	244