10.데이터 집계와 그룹 연산

데이터셋을 분류하고 각 그룹에 집계나 변형 같은 함수를 적용하는 건 데이터 분석 과정에서 무척 중요한 일이다. 데이터를 불러오고 취합해서 하나의 데이터 집합을 준비하고 나면 그룹 통계를 구하거나 가능하다면 **피벗 테이블**을 구해서 보고서를 만들거나 시각화한다. pandas는 데이터 집합을 자연스럽게 나누고 요약할 수 있는 groupby 라는 유연한 방법을 제공한다.

관계형 DB와 SQL이 인기 있는 이유 중 하나는 데이터를 쉽게 합치고 걸러내고 변형하고 집계할 수 있기 때문이다. 하지만 SQL 같은 쿼리문은 그룹 연산에 제약이 있다. 앞으로 살펴보겠지만 파이썬과 pandas의 강력한 표현력을 잘 이용하면 아주 복잡한 그룹 연산도 pandas 객체나 Numpy 배열을 받는 함수의 조합으로 해결할 수 있다. 이 장에서는 다음 내용을 배운다.

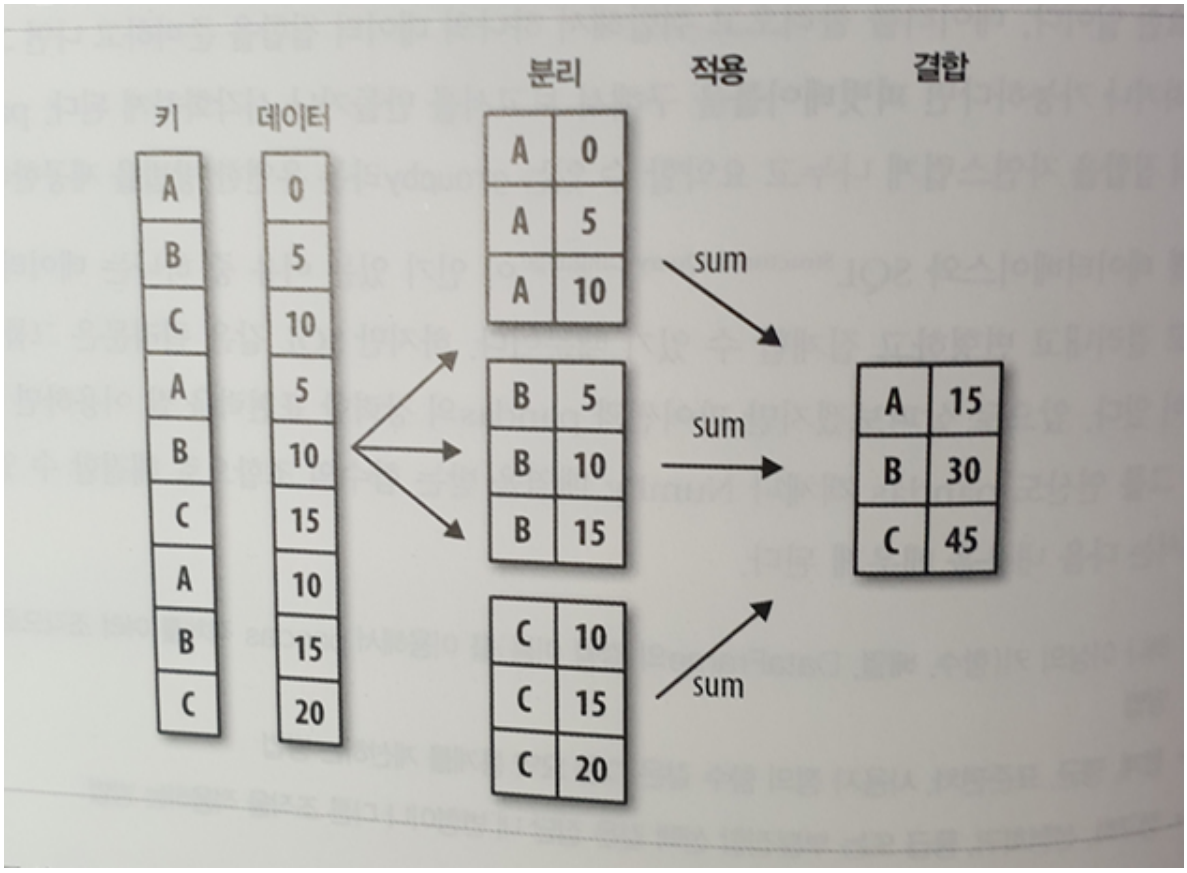
- **하나 이상의 키(함수, 배열, DataFrame의 컬럼 이름)를 이용해서 pandas 객체를 여러 조각으로 나누는 방법**
- **합계, 평균, 표준편차, 사용자 정의 함수 같은 그룹 요약 통계를 계산하는 방법**
- **정규화, 선형회귀, 등급 또는 부분집합 선택 같은 집단 내 변형이나 다른 조작을 적용하는 방법**
- **피벗테이블과 교차일람표를 구하는 방법**
- **변위치 분석과 다른 통계 집단 분석을 수행하는 방법**

NOTE_ 시계열 데이터의 집계 같은 특수한 groupby 사용 방법을 리샘플링 이라고 한다. 이는 다음장에서 학습한다.

10.1 GroupBy 메카닉

다수의 인기있는 R프로그래밍 패키지의 저자인 해들리 위컴은 분리-적용-결합 이라는 그룹 연산에 대한 새로운 용어를 만들었는데, 나는 이 말이 그룹 연산에 대한 훌륭한 설명이라고 생각한다.

그룹 연산의 첫 번째 단계에서는 Series, DataFrame 같은 pandas 객체나 아니면 다른 객체에 들어있는 데이터를 하나 이상의 **키**를 기준으로 **분리**한다. 객체는 하나의 축을 기준으로 분리하는데, 예를 들어 DataFrame은 로우 (axis = 0)로 분리하거나 컬럼(axis = 1)으로 분리할 수 있다. 분리하고 나서는 함수를 각 그룹에 **적용**시켜 새로운 값을 얻어낸다. 마지막으로 함수를 적용한 결과를 하나의 객체로 **결합**한다. 결과를 담는 객체는 보통 데이터에 어떤 연산을 했는지에 따라 결정된다. 간단한 연산의 예시를 살펴보자.



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 Series 객체**
- **축 색인 혹은 색인 내의 개별 이름에 대해 실행되는 함수**

앞 목록에서 마지막 세 방법은 객체를 나눌 때 사용할 배열을 생성하기 위한 방법이라는 것을 기억하자. 먼저 다음과 같이 DataFrame으로 표현되는 간단한 표 형식의 데이터가 있다고 하자.

```
In [196]: 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)})

In [197]: df
Out[197]:
  key1 key2     data1     data2
0    a  one  0.137201  0.002452
1    a  two -0.291850 -2.263151
2    b  one -1.006514  0.042143
3    b  two -1.266896  0.331270
4    a  one  1.005932  0.892894
```

이 데이터를 key1으로 묶고 각 그룹에서 data1의 평균을 구해보자. 여러 가지 방법이 있지만 그 중 하나는 data1에 대해 groupby 메서드를 호출하고 key1 컬럼을 넘기는 것이다.

```
In [199]: grouped = df['data1'].groupby(df['key1'])

In [200]: grouped
Out[200]: <pandas.core.groupby.generic.SeriesGroupBy object at
0x000001D9C5E82A88>
```

이 groupby 변수는 Groupby 객체다. df['key1']로 참조되는 중간값에 대한 것 외에는 아무것도 계산되지 않은 객체다. 이 객체는 그룹 연산을 위해 필요한 모든 정보를 가지고 있어서 각 그룹에 어떤 연산을 적용할 수 있게 해준다. 예를 들어 그룹별 평균을 구하려면 Groupby 객체의 mean 메서드를 사용하면 된다.

```
In [202]: grouped.mean()
Out[202]:
key1
a    0.283761
b   -1.136705
Name: data1, dtype: float64
```

mean() 메서드를 호출했을 때의 자세한 내용은 나중에 설명하기로 한다. 이 예제에서 중요한 점은 데이터(Series 객체)가 그룹 색인에 따라 수집되고 key1 컬럼에 있는 유일한 값으로 색인되는 새로운 Series 객체가 생성된다는 것이다. 새롭게 생성된 Series객체의 색인은 'key1' 인데, 그 이유는 DataFrame 컬럼인 df['key1'] 때문이다.

만약 여러 개의 배열을 리스트로 넘겼다면 조금 다른 결과를 얻었을 것이다.

```
In [7]: means = df['data1'].groupby([df['key1'],df['key2']]).mean()

In [8]: means
Out[8]:
key1  key2
a     one    -0.029811
      two    -1.774734
b     one    -0.849316
      two     1.052916
Name: data1, dtype: float64

In [9]: df
Out[9]:
  key1 key2     data1     data2
0    a  one -0.658756  1.175920
1    a  two -1.774734 -0.405946
2    b  one -0.849316  1.099892
3    b  two  1.052916 -0.380402
4    a  one  0.599135 -0.551196
```

여기서는 데이터를 두 개의 색인으로 묶었고, 그 결과 계층적 색인을 가지는 Series를 얻을 수 있었다.

```
In [10]: means.unstack()
Out[10]:
key2       one       two
key1
a    -0.029811 -1.774734
b    -0.849316  1.052916
```

이 예제에서는 그룹의 색인 모두 Series 객체인데, 길이만 같다면 어떤 배열이라도 상관없다.

```
In [11]: states = np.array(['Ohio','California','California','Ohio','Ohio'])

In [12]: years = np.array([2005,2005,2006,2005,2006])

In [13]: df['data1'].groupby([states,years]).mean()
Out[13]:
California  2005   -1.774734
            2006   -0.849316
Ohio        2005    0.197080
            2006    0.599135
Name: data1, dtype: float64
```

한 그룹으로 묶을 정보는 주로 같은 DataFrame 안에서 찾게 되는데, 이 경우 컬럼 이름(문자열, 숫자 혹은 다른 파이썬 객체)을 넘겨서 그룹의 색인으로 사용할 수 있다.

```
In [14]: df.groupby('key1').mean()
Out[14]:
         data1     data2
key1
a    -0.611452  0.072926
b     0.101800  0.359745
```

위에서 df.groupby('key1').mean() 코드를 보면 key2 컬럼이 결과에서 빠져 있는 것을 확인할 수 있다. 그 이유는 df['key2']는 숫자 데이터가 아니기에, 이런 컬럼을 **성가진 컬럼**이라고 부르며 결과에서 제외시킨다. 기본적으로 모든 숫자 컬럼이 수집되지만 곧 살펴보듯이 원하는 부분만 따로 걸러내는 것도 가능하다.

groupby를 쓰는 목적과 별개로, 일반적으로 유용한 GroupBy 메서드는 그룹의 크기를 담고 있는 Series를 반환하는 Size 메서드다.

```
In [16]: df.groupby(['key1','key2']).size()
Out[16]:
key1  key2
a     one     2
      two     1
b     one     1
      two     1
dtype: int64
```

그룹 색인에서 누락된 값은 결과에서 제외된다는 것을 기억하자.

10.1.1 그룹 간 순회하기

Groupby 객체는 이터레이션을 지원하는데, 그룹 이름과 그에 따른 데이터 묶음을 튜플로 반환한다. 다음 예제를 살펴보자.

```
In [17]: for name, group in df.groupby('key1'):
    ...:     print(name)
    ...:     print(group)
    ...:
a
  key1 key2     data1     data2
0    a  one -0.658756  1.175920
1    a  two -1.774734 -0.405946
4    a  one  0.599135 -0.551196
b
  key1 key2     data1     data2
2    b  one -0.849316  1.099892
3    b  two  1.052916 -0.380402
```

이처럼 색인이 여럿 존재하는 경우 튜플의 첫 번째 원소가 색인값이 된다.

```
In [21]: for(k1,k2), group in df.groupby(['key1','key2']):
    ...:     print((k1,k2))
    ...:     print(group)
    ...:
('a', 'one')
  key1 key2     data1     data2
0    a  one -0.658756  1.175920
4    a  one  0.599135 -0.551196
('a', 'two')
  key1 key2     data1     data2
1    a  two -1.774734 -0.405946
('b', 'one')
  key1 key2     data1     data2
2    b  one -0.849316  1.099892
('b', 'two')
  key1 key2     data1     data2
3    b  two  1.052916 -0.380402
```

당연히 이 안에서 원하는 데이터만 골라낼 수 있다. 한 줄이면 그룹별 데이터를 사전형으로 쉽게 바꿔서 유용하게 사용할 수 있다.

```
In [23]: pieces
Out[23]:
{'a':   key1 key2     data1     data2
 0    a  one -0.658756  1.175920
 1    a  two -1.774734 -0.405946
 4    a  one  0.599135 -0.551196,
 'b':   key1 key2     data1     data2
 2    b  one -0.849316  1.099892
 3    b  two  1.052916 -0.380402}

In [25]: pieces['b']
Out[25]:
  key1 key2     data1     data2
2    b  one -0.849316  1.099892
3    b  two  1.052916 -0.380402
```

groupby 메서드는 기본적으로 axis = 0에 대해 그룹을 만드는데, 다른 축으로 그룹을 만드는 것도 가능하다. 예를 들어 예제로 살펴본 df의 컬럼을 dtype에 따라 그룹으로 묶을 수도 있다.

```
In [26]: df.dtypes
Out[26]:
key1      object
key2      object
data1    float64
data2    float64
dtype: object

In [27]: grouped = df.groupby(df.dtypes , axis = 1)

In [28]: grouped
Out[28]: <pandas.core.groupby.generic.DataFrameGroupBy
        object at 0x000001D501706FC8>
```

그룹을 아래처럼 출력해볼 수 있다.

```
In [29]: for dtype, group in grouped:
    ...:     print(dtype)
    ...:     print(group)
    ...:
float64
      data1     data2
0 -0.658756  1.175920
1 -1.774734 -0.405946
2 -0.849316  1.099892
3  1.052916 -0.380402
4  0.599135 -0.551196
object
  key1 key2
0    a  one
1    a  two
2    b  one
3    b  two
4    a  one
```

10.4.2 컬럼이나 컬럼의 일부만 선택하기

DataFrame에서 만든 GroupBy 객체를 컬럼 이름이나 컬럼이름이 담긴 배열로 색인하면 수집을 위해 해당 컬럼을 선택하게 된다.

```
In [30]: df.groupby('key1')['data1']
Out[30]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x000001D5015953C8>

In [31]: df.groupby('key1')[['data2']]
Out[31]: <pandas.core.groupby.generic.DataFrameGroupBy object at
0x000001D501587548>
```

위 코드는 아래 코드에 대한 신택틱 슈거로 같은 결과를 반환한다.

```
In [32]: df['data1'].groupby(df['key1'])
Out[32]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x000001D501719108>

In [33]: df['data2'].groupby(df['key1'])
Out[33]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x000001D5015D2348>
```

특히 대용량 데이터를 다룰 경우 소수의 컬럼만 집계하고 싶을 때가 종종 있는데, 예를 들어 위 데이터에서 data2 컬럼에 대해서만 평균을 구하고 결과를 DataFrame으로 받고 싶다면 아래와 같이 작성한다.

```
In [34]: df.groupby(['key1','key2'])[['data2']].mean()
Out[34]:
              data2
key1 key2
a    one   0.312362
     two  -0.405946
b    one   1.099892
     two  -0.380402
```

색인으로 얻은 객체는 groupby 메서드에 리스트나 배열을 넘겼을 경우, DataFrameGroupBy 객체가 되고, 단일 값으로 하나의 컬럼 이름만 넘겼을 경우 SeriesGroupBy 객체가 된다.

```
In [35]: s_grouped = df.groupby(['key1','key2'])['data2']

In [36]: s_grouped
Out[36]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x000001D5017435C8>

In [37]: s_grouped.mean()
Out[37]:
key1  key2
a     one     0.312362
      two    -0.405946
b     one     1.099892
      two    -0.380402
Name: data2, dtype: float64
```

10.1.3 사전과 Series에서 그룹핑하기

그룹 정보는 배열이 아닌 형태로 존재하기도 한다. 다른 DataFrame 예제를 살펴보자.

```
In [38]: people = pd.DataFrame(np.random.randn(5,5),
    ...:                       columns = ['a','b','c','d','e'],
    ...:                       index = ['Joe','Steve','Wes','Jim','Travis'])

In [39]: people.iloc[2:3, [1,2]] = np.nan

In [40]: people
Out[40]:
               a         b         c         d         e
Joe     1.324058  0.627396 -0.672775 -0.232616 -1.252971
Steve  -0.461243 -0.629434 -1.110294  0.353904  0.664200
Wes    -0.614544       NaN       NaN -0.350625  1.570130
Jim     0.407202 -0.852681  0.515682  0.382633  1.073183
Travis -1.387862 -0.768294 -0.180329 -1.739463  1.573623
```

이제 각 컬럼을 나타낼 그룹 목록이 있고, 그룹별로 컬럼의 값을 모두 더한다고 해보자.

```
In [41]: mapping = {'a':'red', 'b':'red','c':'blue','d':'blue','e':'red',
    ...:            'f':'orange'}
```

이 사전에서 groupby 메서드로 넘길 배열을 뽑아낼 수 있지만 그냥 이 사전을 groupby 메서드로 넘기자 (사용하지 않는 그룹 키도 문제없다는 것을 보이기 위해 'f'도 포함시켰다).

```
In [41]: mapping = {'a':'red', 'b':'red','c':'blue','d':'blue','e':'red',
    ...:             'f':'orange'}

In [42]: by_column = people.groupby(mapping, axis =1)

In [44]: by_column.sum()
Out[44]:
            blue       red
Joe    -0.905392  0.698484
Steve  -0.756391 -0.426476
Wes    -0.350625  0.955586
Jim     0.898315  0.627704
Travis -1.919792 -0.582533
```

Series에 대해서도 같은 기능을 수행할 수 있는데, 고정된 크기의 맵이라고 보면된다.

```
In [45]: map_series = pd.Series(mapping)

In [46]: map_series
Out[46]:
a       red
b       red
c      blue
d      blue
e       red
f    orange
dtype: object

In [47]: people.groupby(map_series, axis = 1).count()
Out[47]:
        blue  red
Joe        2    3
Steve      2    3
Wes        1    2
Jim        2    3
Travis     2    3
```

10.1.4 함수로 그룹핑하기

파이썬 함수를 사용하는 것은 사전이나 Series를 사용해서 그룹을 매핑하는 것보다 좀 더 일반적이다. 그룹 색인으로 넘긴 함수는 색인값 ㅁ하나마다 한 번씩 호출되며, 반환값은 그 그룹의 이름으로 사용된다. 좀 더 구체적으로 말하자면 좀 전에 살펴본 예제에서 people DataFrame은 사람의 이름을 색인값으로 사용했다. 만약 이름의 길이별로 그룹을 묶고 싶다면 이름의 길이가 담긴 배열을 만들어 넘기는 대신 len함수를 넘기면 된다.

```
In [51]: people.groupby(len).sum()
Out[51]:
          a         b         c         d         e
3  1.116716 -0.225285 -0.157093 -0.200609  1.390342
5 -0.461243 -0.629434 -1.110294  0.353904  0.664200
6 -1.387862 -0.768294 -0.180329 -1.739463  1.573623

In [52]: people
Out[52]:
              a         b         c         d         e
```

```
Joe     1.324058  0.627396 -0.672775 -0.232616 -1.252971
Steve  -0.461243 -0.629434 -1.110294  0.353904  0.664200
Wes    -0.614544       NaN       NaN -0.350625  1.570130
Jim     0.407202 -0.852681  0.515682  0.382633  1.073183
Travis -1.387862 -0.768294 -0.180329 -1.739463  1.573623
```

내부적으로는 모두 배열로 변환되므로 함수를 배열, 사전 또는 Series와 섞어 쓰더라도 전혀 문제가 되지 않는다.

```
In [53]: key_list = ['one','one','one','two','two']

In [54]: people.groupby([len, key_list]).min()
Out[54]:
              a         b         c         d         e
3 one -0.614544  0.627396 -0.672775 -0.350625 -1.252971
  two  0.407202 -0.852681  0.515682  0.382633  1.073183
5 one -0.461243 -0.629434 -1.110294  0.353904  0.664200
6 two -1.387862 -0.768294 -0.180329 -1.739463  1.573623
```

10.1.5 색인 단계로 그룹핑하기

계층적으로 색인된 데이터는 축 색인의 단계 중 하나를 사용해서 편리하게 집계할 수 있는 기능을 제공한다. 다음 예제를 보자.

```
In [55]: columns = pd.MultiIndex.from_arrays([['US','US','US','JP','JP'],
    ...:                                      [1,3,5,1,3]],
    ...:                                      names = ['cty','tenor'])

In [56]: hier_df = pd.DataFrame(np.random.randn(4,5),columns=columns)

In [57]: hier_df
Out[57]:
cty          US                            JP
tenor         1         3         5         1         3
0     -1.843890  0.495627  0.548369 -0.013118  2.407938
1     -0.130768  0.132752 -0.843204 -0.704021  0.105731
2     -0.334726 -0.422182  1.055342 -0.087038  0.600932
3     -0.293233  0.478383 -0.715492  1.301283  1.577202
```

이 기능을 사용하려면 level 예약어를 사용해서 레벨 번호나 이름을 넘기면 된다.

```
In [65]: hier_df.groupby(level='tenor',axis=1).count()
Out[65]:
tenor  1  3  5
0      2  2  1
1      2  2  1
2      2  2  1
3      2  2  1

In [66]: hier_df.groupby(level='cty',axis=1).count()
Out[66]:
cty  JP  US
0     2   3
1     2   3
2     2   3
```

```
3     2   3
```

10.2 데이터 집계

데이터 집계는 배열로부터 스칼라값을 만들어내는 모든 데이터 변환 작업을 말한다. 위 예제에서 나는 mean, count, min, sum을 이용해서 스칼라값을 구했다. GroupBy 객체에 대해 mean()을 수행하면 어떤 일이 생기는지 궁금할 것이다. 다음 표에 있는 것과 같은 많은 일반적인 데이터 집계는 데이터 묶음에 대한 준비된 통계를 계산해내는 최적화된 구현을 가지고 있다. 하지만 이 메서드만 사용해야 하는 건 아니다.

최적화된 groupby 메서드

함수	설명
median	NA가 아닌 값들의 산술 중간값을 구한다.
std, var	편향되지 않은(n-1을 분모로 하는)표준편차와 분산
min, max	NA가 아닌 값들 중 최솟값과 최댓값
prod	NA가 아닌 값들의 곱
first, last	NA가 아닌 값들 중 첫째 값과 마지막 값

직접 고안한 집계함수를 사용하고 추가적으로 그룹 객체에 이미 정의된 메서드를 연결해서 사용하는 것도 가능하다. 예를 들어 quantile 메서드가 Series나 DataFrame의 컬럼의 변위치를 계산한다는 점을 생각해봇.

quantile 메서드는 GroupBy만을 위해 구현되지 않았지만 Series 메서드이기 때문에 여기서 사용할 수 있다. 내부적으로 GroupBy는 Series를 효과적으로 잘게 자르고 각 조각에 대해 piece.quantile(0.9)를 호출한다. 그리고 이 결과들을 모두 하나의 객체로 합쳐서 반환한다.

```
In [73]: df
Out[73]:
  key1 key2     data1     data2
0    a  one -0.658756  1.175920
1    a  two -1.774734 -0.405946
2    b  one -0.849316  1.099892
3    b  two  1.052916 -0.380402
4    a  one  0.599135 -0.551196

In [74]: grouped = df.groupby('key1')

In [75]: for a,b in grouped:
    ...:     print(a)
    ...:     print(b)
    ...:
a
  key1 key2     data1     data2
0    a  one -0.658756  1.175920
1    a  two -1.774734 -0.405946
4    a  one  0.599135 -0.551196
b
  key1 key2     data1     data2
2    b  one -0.849316  1.099892
3    b  two  1.052916 -0.380402
```

```
In [76]: grouped['data1'].quantile(0.9)
Out[76]:
key1
a    0.347556
b    0.862693
Name: data1, dtype: float64
```

자신만의 데이터 집계함수를 사용하려면 배열의 aggregate나 agg 메서드에 해당 함수를 넘기면 된다.

```
In [78]: def peak_to_peak(arr):
    ...:     return arr.max() - arr.min()
    ...:

In [79]: grouped.agg(peak_to_peak)
Out[79]:
         data1     data2
key1
a     2.373868  1.727117
b     1.902232  1.480294
```

describe 같은 메서드는 데이터를 집계하지 않는데도 잘 작동함을 확인할 수 있다.

```
In [9]: grouped.describe()
Out[9]:
     data1                          ...     data2
     count      mean       std     ...       50%       75%       max
key1                                ...
a      3.0 -0.174644  0.855267     ... -0.541044  0.025019  0.591081
b      2.0 -0.510650  0.344844     ...  0.111365  1.009707  1.908050

[2 rows x 16 columns]
```

10.2.1 컬럼에 여러 가지 함수 적용하기

여기서는 read_csv 함수로 데이터를 불러온 다음 팁의 비율을 담기 위한 컬럼인 tip_pct를 추가했다.

```
In [11]: tips = pd.read_csv('tips.csv')

In [13]: tips['tip_pct'] = tips['tip']/(tips['total_bill']-tips['ti
    ...: p'])

In [14]: tips
Out[14]:
     total_bill   tip smoker   day    time  size   tip_pct
0         16.99  1.01     No   Sun  Dinner     2  0.063204
1         10.34  1.66     No   Sun  Dinner     3  0.191244
2         21.01  3.50     No   Sun  Dinner     3  0.199886
3         23.68  3.31     No   Sun  Dinner     2  0.162494
4         24.59  3.61     No   Sun  Dinner     4  0.172069
..          ...   ...    ...   ...     ...   ...       ...
239       29.03  5.92     No   Sat  Dinner     3  0.256166
240       27.18  2.00    Yes   Sat  Dinner     2  0.079428
241       22.67  2.00    Yes   Sat  Dinner     2  0.096759
242       17.82  1.75     No   Sat  Dinner     2  0.108899
243       18.78  3.00     No  Thur  Dinner     2  0.190114
```

```
[244 rows x 7 columns]
```

이미 봤듯, Series나 DataFrame의 모든 컬럼을 집계하는 것은 mean이나 std 같은 메서드를 호출하거나 원하는 함수에 aggregate를 사용하는 것이다. 하지만 컬럼에 따라 다른 함수를 사용해서 집계를 수행하거나 여러 개의 함수를 한 번에 적용하기 원한다면 이를 쉽고 간단하게 수행할 수 있다.

먼저 tips를 day와 smoker 별로 묶어보자.

```
In [15]: grouped = tips.groupby(['day','smoker'])

In [17]: grouped_pct = grouped['tip_pct']

In [18]: grouped_pct.agg('mean')
Out[18]:
day   smoker
Fri   No        0.179740
      Yes       0.216293
Sat   No        0.190412
      Yes       0.179833
Sun   No        0.193617
      Yes       0.322021
Thur  No        0.193424
      Yes       0.198508
Name: tip_pct, dtype: float64
```

만일 함수 목록이나 함수 이름을 넘기면 함수 이름을 컬럼 이름으로 하는 DataFrame을 얻게 된다.

```
In [29]: grouped_pct.agg(['mean','std',peak_to_peak])
Out[29]:
                mean       std  peak_to_peak
day  smoker
Fri  No      0.179740  0.039458      0.094263
     Yes     0.216293  0.077530      0.242219
Sat  No      0.190412  0.058626      0.352192
     Yes     0.179833  0.089496      0.446137
Sun  No      0.193617  0.060302      0.274897
     Yes     0.322021  0.538061      2.382107
Thur No      0.193424  0.056065      0.284273
     Yes     0.198508  0.057170      0.219047
```

여기서는 데이터 그룹에 대해 독립적으로 적용하기 위해 agg에 집계함수들의 리스트를 넘겼다.

GroupBy 객체에서 자동으로 지정하는 컬럼 이름을 그대로 쓰지 않아도 된다. lambda 함수는 이름(함수 이름은 _name_ 속성으로 확인 가능하다) 이 '<lambda>' 인데, 이를 그대로 쓸 경우 알아보기 힘들어진다. 이때 이름과 함수가 담긴 (name, function) 튜플의 리스트를 넘기면 각 튜플에서 첫 번째 원소가 DataFrame에서 컬럼 이름으로 사용된다. (2개의 튜플을 가지는 리스트가 순서대로 매핑된다).

```
In [28]: grouped_pct.agg([('foo','mean'),('bar',np.std)])
Out[28]:
                  foo       bar
day  smoker
Fri  No      0.179740  0.039458
     Yes     0.216293  0.077530
Sat  No      0.190412  0.058626
     Yes     0.179833  0.089496
Sun  No      0.193617  0.060302
     Yes     0.322021  0.538061
Thur No      0.193424  0.056065
     Yes     0.198508  0.057170
```

DataFrame은 컬럼마다 다른 함수를 적용하거나 여러 개의 함수를 모든 컬럼에 적용할 수 있다.

tip_pct와 total_bill 컬럼에 대해 동일한 세 가지 통계를 계산한다고 가정하자.

```
In [30]: functions = ['count','mean','max']

In [31]: result = grouped['tip_pct','total_bill'].agg(functions)

In [32]: result
Out[32]:
            tip_pct                     total_bill
              count      mean       max      count       mean    max
day  smoker
Fri  No           4  0.179740  0.231125          4  18.420000  22.75
     Yes         15  0.216293  0.357737         15  16.813333  40.17
Sat  No          45  0.190412  0.412409         45  19.661778  48.33
     Yes         42  0.179833  0.483092         42  21.276667  50.81
Sun  No          57  0.193617  0.338101         57  20.506667  48.17
     Yes         19  0.322021  2.452381         19  24.120000  45.35
Thur No          45  0.193424  0.362976         45  17.113111  41.19
     Yes         17  0.198508  0.317965         17  19.190588  43.11
```

위에서 확인할 수 있듯이 반환된 DataFrame은 계층적인 컬럼을 가지고 있으며 이는 각 컬럼을 따로 계산한 다음 concat 메서드를 이용해서 keys 인자로 컬럼 이름을 넘겨서 이어붙인 것과 동일하다.

```
In [33]: result['tip_pct']
Out[33]:
             count      mean       max
day  smoker
Fri  No          4  0.179740  0.231125
     Yes        15  0.216293  0.357737
Sat  No          45  0.190412  0.412409
     Yes        42  0.179833  0.483092
Sun  No          57  0.193617  0.338101
     Yes        19  0.322021  2.452381
Thur No          45  0.193424  0.362976
     Yes        17  0.198508  0.317965
```

위에서처럼 컬럼 이름과 메서드가 담긴 튜플의 리스트를 넘기는 것도 가능하다.

```
In [34]: ftuples = [('Durchschniff','mean'),('Abweichung',np.var)]
```

```
In [35]: grouped['tip_pct','total_bill'].agg(ftuples)
Out[35]:
                tip_pct              total_bill
            Durchschniff Abweichung Durchschniff  Abweichung
day  smoker
Fri  No         0.179740   0.001557    18.420000   25.596333
     Yes        0.216293   0.006011    16.813333   82.562438
Sat  No         0.190412   0.003437    19.661778   79.908965
     Yes        0.179833   0.008010    21.276667  101.387535
Sun  No         0.193617   0.003636    20.506667   66.099980
     Yes        0.322021   0.289509    24.120000  109.046044
Thur No         0.193424   0.003143    17.113111   59.625081
     Yes        0.198508   0.003268    19.190588   69.808518
```

컬럼마다 다른 함수를 적용하고 싶다면 agg메서드에 컬럼 이름에 대응하는 함수가 들어 있는 사전을 넘기면 된다.

```
In [36]: grouped.agg({'tip':np.max,'size':'sum'})
Out[36]:
               tip  size
day  smoker
Fri  No       3.50     9
     Yes      4.73    31
Sat  No       9.00   115
     Yes     10.00   104
Sun  No       6.00   167
     Yes      6.50    49
Thur No       6.70   112
     Yes      5.00    40

In [37]: grouped.agg({'tip_pct':['min','max','mean','std'],'size':'sum'})
Out[37]:
             tip_pct                               size
                 min       max      mean       std  sum
day  smoker
Fri  No      0.136861  0.231125  0.179740  0.039458    9
     Yes     0.115518  0.357737  0.216293  0.077530   31
Sat  No      0.060217  0.412409  0.190412  0.058626  115
     Yes     0.036955  0.483092  0.179833  0.089496  104
Sun  No      0.063204  0.338101  0.193617  0.060302  167
     Yes     0.070274  2.452381  0.322021  0.538061   49
Thur No      0.078704  0.362976  0.193424  0.056065  112
     Yes     0.098918  0.317965  0.198508  0.057170   40
```

단 가지 컬럼에라도 여러 개의 함수가 적용됐다면 DataFrame은 계층적인 컬럼을 가진다.

10.2.2 색인되지 않은 형태로 집계된 데이터 반환하기

지금까지 살펴본 모든 예제에서 집계된 데이터는 유일한 그룹키 조합으로 색인(어떤 경우에는 계층적 색인)되어 반환되었다. 하지만 항상 이런 동작을 기대한느 것은 아니므로, groupby 메서드에서 as_index = False를 넘겨서 색인되지 않도록 할 수 있다.

```
In [38]: tips.groupby(['day','smoker'],as_index=False).mean()
Out[38]:
    day smoker  total_bill       tip      size   tip_pct
0   Fri     No   18.420000  2.812500  2.250000  0.179740
1   Fri    Yes   16.813333  2.714000  2.066667  0.216293
2   Sat     No   19.661778  3.102889  2.555556  0.190412
3   Sat    Yes   21.276667  2.875476  2.476190  0.179833
4   Sun     No   20.506667  3.167895  2.929825  0.193617
5   Sun    Yes   24.120000  3.516842  2.578947  0.322021
6  Thur     No   17.113111  2.673778  2.488889  0.193424
7  Thur    Yes   19.190588  3.030000  2.352941  0.198508
```

물론 이렇게 하지 않고 색인된 결과에 대해 reset_index 메서드를 호출해서 같은 결과를 얻을 수 있다. as_index = False 옵션을 사용하면 불필요한 계산을 피할 수 있다.

10.3 Apply : 일반적인 분리-적용-병합

가장 일반적인 GroupBy 메서드의 목적은 apply인데 지금부터 다루게 될 주제다. apply 메서드는 객체를 여러 조각으로 나누고, 전달된 함수를 각 조각에 일괄 적용한후 이를 다시 합친다.

앞서 살펴봤던 팁 데이터에서 그룹별 상위 5개의 tip_pct값을 골라보자. 우선 특정 컬럼에서 가장 큰 값을 가지는 로우를 선택하는 함수를 바로 작성해보자.

```
In [39]: def top(df,n=5,column='tip_pct'):
    ...:     return df.sort_values(by=column)[-n:]
    ...:

In [40]: top(tips)
Out[40]:
     total_bill   tip smoker  day    time  size   tip_pct
183       23.17  6.50    Yes  Sun  Dinner     4  0.389922
232       11.61  3.39     No  Sat  Dinner     2  0.412409
67         3.07  1.00    Yes  Sat  Dinner     1  0.483092
178        9.60  4.00    Yes  Sun  Dinner     2  0.714286
172        7.25  5.15    Yes  Sun  Dinner     2  2.452381
```

이제 흡연자(smoker) 그룹에 대해 이 함수(top)를 apply 하면 다음과 같은 결과가 나온다.

```
In [43]: tips.groupby('smoker').apply(top)
Out[43]:
            total_bill   tip smoker   day    time  size   tip_pct
smoker
No     88         24.71  5.85     No  Thur   Lunch     2  0.310180
       185        20.69  5.00     No   Sun  Dinner     5  0.318674
       51         10.29  2.60     No   Sun  Dinner     2  0.338101
       149         7.51  2.00     No  Thur   Lunch     2  0.362976
       232        11.61  3.39     No   Sat  Dinner     2  0.412409
Yes    109        14.31  4.00    Yes   Sat  Dinner     2  0.387973
       183        23.17  6.50    Yes   Sun  Dinner     4  0.389922
       67          3.07  1.00    Yes   Sat  Dinner     1  0.483092
       178         9.60  4.00    Yes   Sun  Dinner     2  0.714286
       172         7.25  5.15    Yes   Sun  Dinner     2  2.452381
```

위 결과를 보면 top 함수가 나뉘어진 DataFrame의 각 부분에 모두 적용됐고, pandas.concat을 이용해서 하나로 합쳐진 다음 그룹 이름표가 붙었다. 이에 결과는 계층적 색인을 갖게 되고 내부 색인은 원본 DataFrame의 색인값을 가진다.

만일 apply 메서드로 넘길 함수가 추가적인 인자를 받는다면 함수 이름 뒤에 붙여서 넘겨주면 된다.

```
In [47]: tips.groupby(['smoker','day']).apply(top, n=1,column = 'total_bill')
    ...:
Out[47]:
                 total_bill    tip smoker   day    time  size   tip_pct
smoker day
No     Fri  94        22.75   3.25     No   Fri  Dinner     2  0.166667
       Sat  212       48.33   9.00     No   Sat  Dinner     4  0.228833
       Sun  156       48.17   5.00     No   Sun  Dinner     6  0.115821
       Thur 142       41.19   5.00     No  Thur   Lunch     5  0.138160
Yes    Fri  95        40.17   4.73    Yes   Fri  Dinner     4  0.133465
       Sat  170       50.81  10.00    Yes   Sat  Dinner     3  0.245038
       Sun  182       45.35   3.50    Yes   Sun  Dinner     3  0.083632
       Thur 197       43.11   5.00    Yes  Thur   Lunch     4  0.131199
```

이 책의 앞부분에서 GroupBy 객체에 describe 메서드를 호출했던 적이 있다.

```
In [50]: result = tips.groupby('smoker')['tip_pct'].describe()

In [51]: result
Out[51]:
        count      mean       std  ...       50%       75%       max
smoker                             ...
No      151.0  0.192237  0.057665  ...  0.184308  0.227015  0.412409
Yes      93.0  0.218176  0.254295  ...  0.181818  0.242326  2.452381

[2 rows x 8 columns]
```

```
In [52]: result.unstack('smoker')
Out[52]:
       smoker
count  No        151.000000
       Yes        93.000000
mean   No          0.192237
       Yes         0.218176
std    No          0.057665
       Yes         0.254295
min    No          0.060217
       Yes         0.036955
25%    No          0.158622
       Yes         0.119534
50%    No          0.184308
       Yes         0.181818
75%    No          0.227015
       Yes         0.242326
max    No          0.412409
       Yes         2.452381
dtype: float64
```

describe 같은 메서드를 호출하면 GroupBy 내부적으로 다음과 같은 단계를 수행한다.

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

10.3.1 그룹 색인 생략하기

앞서 살펴본 예제들에서 반환된 객체는 원본 객체의 각 조각에 대한 색인과 그룹 키가 계층적 색인으로 사용됨을 볼 수 있었다. 이런 결과는 group 메서드에 group_keys = False 를 넘겨서 막을 수 있다.

```
In [54]: tips.groupby('smoker',group_keys = False).apply(top)
Out[54]:
     total_bill   tip smoker   day    time  size   tip_pct
88        24.71  5.85     No  Thur   Lunch     2  0.310180
185       20.69  5.00     No   Sun  Dinner     5  0.318674
51        10.29  2.60     No   Sun  Dinner     2  0.338101
149        7.51  2.00     No  Thur   Lunch     2  0.362976
232       11.61  3.39     No   Sat  Dinner     2  0.412409
109       14.31  4.00    Yes   Sat  Dinner     2  0.387973
183       23.17  6.50    Yes   Sun  Dinner     4  0.389922
67         3.07  1.00    Yes   Sat  Dinner     1  0.483092
178        9.60  4.00    Yes   Sun  Dinner     2  0.714286
172        7.25  5.15    Yes   Sun  Dinner     2  2.452381

In [55]: tips.groupby('smoker',group_keys = True).apply(top)
Out[55]:
            total_bill   tip smoker   day    time  size   tip_pct
smoker
No     88        24.71  5.85     No  Thur   Lunch     2  0.310180
       185       20.69  5.00     No   Sun  Dinner     5  0.318674
       51        10.29  2.60     No   Sun  Dinner     2  0.338101
       149        7.51  2.00     No  Thur   Lunch     2  0.362976
       232       11.61  3.39     No   Sat  Dinner     2  0.412409
Yes    109       14.31  4.00    Yes   Sat  Dinner     2  0.387973
       183       23.17  6.50    Yes   Sun  Dinner     4  0.389922
       67         3.07  1.00    Yes   Sat  Dinner     1  0.483092
       178        9.60  4.00    Yes   Sun  Dinner     2  0.714286
       172        7.25  5.15    Yes   Sun  Dinner     2  2.452381
```

10.3.2 변위치 분석과 버킷 분석

8장에서 본 내용을 떠올려보면 pandas의 cut과 qcut 메서드를 사용해서 선택한 크기만큼 혹은 표본 변위치에 따라 데이터를 나눌 수 있었다. 이 함수들은 groupby와 조합하면 데이터 묶음에 대해 변위치 분석이나 버킷 분석을 매우 쉽게 수행할 수 있다. 임의의 데이터 묶음을 cut을 이용해서 등간격 구간으로 나눠보자,

```
In [56]: frame = pd.DataFrame({'data1':np.random.randn(1000),
    ...:                       'data2':np.random.randn(1000)})

In [57]: quartiles = pd.cut(frame.data1,4)

In [58]: quartiles[:10]
Out[58]:
0     (0.736, 2.405]
1    (-0.934, 0.736]
2    (-0.934, 0.736]
3     (0.736, 2.405]
```

```
4    (-0.934, 0.736]
5    (-2.61, -0.934]
6    (-0.934, 0.736]
7    (-0.934, 0.736]
8    (-0.934, 0.736]
9    (-2.61, -0.934]
Name: data1, dtype: category
Categories (4, interval[float64]): [(-2.61, -0.934] < (-0.934, 0.736]
                        < (0.736, 2.405] <
                                    (2.405, 4.075]]
#각 해당 값이 어느 범위에 속하는지 출력해줌
```

cut에서 반환된 Categorical 객체는 바로 groupby로 넘길 수 있다. 그러므로 data2 컬럼에 대한 몇 가지 통계를 다음과 같이 계산할 수 있다.

```
In [67]: def get_stats(group):
    ...:     return {'min':group.min(),'max':group.max(),
    ...:             'count':group.count(),'mean':group.mean()}
    ...:

In [68]: grouped = frame.data2.groupby(quartiles)

In [71]: grouped.apply(get_stats).unstack()
Out[71]:
                      min       max  count      mean
data1
(-2.61, -0.934] -2.528053  2.288859  171.0 -0.146386
(-0.934, 0.736] -3.393811  2.987879  620.0 -0.014126
(0.736, 2.405]  -2.577783  2.701606  201.0  0.026972
(2.405, 4.075]  -0.930021  1.689458    8.0  0.133599

In [72]: grouped.apply(get_stats)
Out[72]:
data1
(-2.61, -0.934]  min       -2.528053
                 max        2.288859
                 count    171.000000
                 mean      -0.146386
(-0.934, 0.736]  min       -3.393811
                 max        2.987879
                 count    620.000000
                 mean      -0.014126
(0.736, 2.405]   min       -2.577783
                 max        2.701606
                 count    201.000000
                 mean       0.026972
(2.405, 4.075]   min       -0.930021
                 max        1.689458
                 count      8.000000
                 mean       0.133599
Name: data2, dtype: float64
```

이는 등간격 버킷이었고, 표본 변위치에 기반하여 크기가 같은 버킷을 계산하려면 qcut을 사용한다.

나는 labels = False를 넘겨서 변위치 숫자를 구했다.

```
In [73]: grouping = pd.qcut(frame.data1,10,labels = False)
#frame.data1을 10개 단위로 표본에 나와있는 순서대로 짤라라 (qcut 사용)
# label 표시하지말고

In [74]: grouped = frame.data2.groupby(grouping)

In [75]: grouped.apply(get_stats).unstack()
Out[75]:
            min       max  count      mean
data1
0     -2.528053  2.081114  100.0 -0.257376
1     -2.518411  2.571830  100.0  0.037853
2     -2.525451  2.225495  100.0 -0.129485
3     -3.393811  2.987879  100.0 -0.059516
4     -1.887710  2.165063  100.0 -0.035861
5     -2.752683  2.915985  100.0 -0.004277
6     -2.756956  2.278931  100.0  0.176686
7     -2.577783  2.696690  100.0 -0.080511
8     -1.983989  2.701606  100.0  0.030828
9     -2.009257  2.269476  100.0  0.048657

#cf) In [77]: grouping = pd.qcut(frame.data1,10) 시

In [79]: grouped.apply(get_stats).unstack()
Out[79]:
                        min       max  count      mean
data1
(-2.605, -1.203]  -2.528053  2.081114  100.0 -0.257376
(-1.203, -0.863]  -2.518411  2.571830  100.0  0.037853
(-0.863, -0.505]  -2.525451  2.225495  100.0 -0.129485
(-0.505, -0.271]  -3.393811  2.987879  100.0 -0.059516
(-0.271, -0.0309] -1.887710  2.165063  100.0 -0.035861
(-0.0309, 0.236]  -2.752683  2.915985  100.0 -0.004277
(0.236, 0.468]    -2.756956  2.278931  100.0  0.176686
(0.468, 0.78]     -2.577783  2.696690  100.0 -0.080511
(0.78, 1.245]     -1.983989  2.701606  100.0  0.030828
```

10.3.3 예제 : 그룹에 따른 값으로 결측치 채우기

누락된 데이터를 정리할 때면 어떤 경우에는 dropna를 사용해서 데이터를 살펴보고 걸러내기도 한다. 하지만 어떤 경우에는 누락된 값을 고정된 값이나 혹은 데이터로부터 도출된 어떤 값으로 채우고 싶을 때도 있다. 이런 경우 fillna 메서드를 사용하는데, 누락된 값을 평균값으로 대체하는 예제를 살펴보자.

```
In [80]: s = pd.Series(np.random.randn(6))

In [81]: s[::2]=np.nan

In [82]: s
Out[82]:
0         NaN
1   -1.067548
2         NaN
3    0.253869
4         NaN
5    0.015893
dtype: float64
```

```
In [83]: s.fillna(s.mean())
Out[83]:
0   -0.265929
1   -1.067548
2   -0.265929
3    0.253869
4   -0.265929
5    0.015893
dtype: float64
```

그룹별로 채워 넣고 싶은 값이 다르다고 가정해보자. 아마도 추측했듯이 데이터를 그룹으로 나누고 apply 함수를 사용해서 각 그룹에 대해 fillna를 적용하면 된다. 여기서 사용된 데이터는 동부와 서부로 나눈 미국의 지역에 대한 데이터다.

```
In [84]: states = ['Ohio','New York', 'Vermont', 'Florida', 'Oregon',
    ...:           'Nevada','California','Idaho']

In [85]: group_key = ['East'] * 4 + ['West'] * 4

In [86]: data = pd.Series(np.random.randn(8),index = states)

In [87]: data
Out[87]:
Ohio         -1.413404
New York     -1.199768
Vermont      -1.007669
Florida      -1.201286
Oregon       -1.332035
Nevada        0.550274
California    0.125880
Idaho        -1.062496
dtype: float64
```

데이터에서 몇몇 값을 결측치로 만들어보자.

```
In [88]: data[['Vermont','Nevada','Idaho']] = np.nan

In [89]: data
Out[89]:
Ohio         -1.413404
New York     -1.199768
Vermont            NaN
Florida      -1.201286
Oregon       -1.332035
Nevada             NaN
California    0.125880
Idaho              NaN
dtype: float64

In [90]: data.groupby(group_key).mean()
Out[90]:
East   -1.271486
West   -0.603078
dtype: float64
```

다음과 같이 누락된 값을 그룹의 평균값으로 채울 수 있다.

```
In [91]: fill_mean = lambda g: g.fillna(g.mean())

In [93]: data.groupby(group_key).apply(fill_mean)
Out[93]:
Ohio         -1.413404
New York     -1.199768
Vermont      -1.271486
Florida      -1.201286
Oregon       -1.332035
Nevada       -0.603078
California    0.125880
Idaho        -0.603078
dtype: float64
```

아니면 그룹에 따라 미리 정의된 다른 값을 채워 넣어야 할 경우도 있다. 각 그룹은 내부적으로 name 이라는 속성을 가지고 있으므로 이를 이용하자.

```
In [94]: fill_values = {'East':0.5,'West':-1}

In [95]: fill_func = lambda g: g.fillna(fill_values[g.name])

In [96]: data.groupby(group_key).apply(fill_func)
Out[96]:
Ohio         -1.413404
New York     -1.199768
Vermont       0.500000
Florida      -1.201286
Oregon       -1.332035
Nevada       -1.000000
California    0.125880
Idaho        -1.000000
dtype: float64
```

10.3.4 예제 : 랜덤 표본과 순열

대용량의 데이터를 몬테카를로 시뮬레이션이나 다른 애플리케이션에서 사용하기 위해 랜덤 표본을 뽑아 낸다고 해보자. 뽑아내는 방법은 여러 가지가 있는데 여기서는 Series의 sample 메서드를 사용하자.

예시를 위해 트럼프 카드 덱을 한번 만들어보자.

```
In [103]: suits = ['H','S','C','D']

In [104]: card_val = (list(range(1,11))+[10]*3)*4

In [105]: base_names = ['A']+list(range(2,11))+['J','K','Q']

In [106]: cards = []

In [107]: for suit in ['H','S','C','D']:
     ...:     cards.extend(str(num)+suit for num in base_names)
     ...:

In [109]: deck = pd.Series(card_val,index = cards)
```

이렇게 해서 블랙잭 값은 게임에서 사용하는 카드 이름과 값을 색인으로 하는 52장의 카드가 Series 객체로 준비됐다.

```
In [112]: deck[:13]
Out[112]:
AH      1
2H      2
3H      3
4H      4
5H      5
6H      6
7H      7
8H      8
9H      9
10H    10
JH     10
KH     10
QH     10
dtype: int64
```

이제 앞에서 얘기한 것 처럼 5장의 카드를 뽑기 위해 다음 코드를 작성한다.

```
In [113]: def draw(deck, n=5):
     ...:     return deck.sample(n)
     ...:

In [114]: draw(deck)
Out[114]:
AD    1
8S    8
8D    8
AC    1
7C    7
dtype: int64
```

각 세트(하트, 스페이드, 클로버, 다이아몬드) 별로 2장의 카드를 무작위로 뽑고 싶다고 가정하자. 세트는 각 카드 이름의 마지막 글자이므로 이를 이용해서 그룹을 나누고 apply를 사용하자.

```
In [115]: get_suit = lambda card: card[-1]

In [116]: deck.groupby(get_suit).apply(draw, n=2)
Out[116]:
C  QC     10
   3C      3
D  10D    10
   5D      5
H  5H      5
   4H      4
S  10S    10
   4S      4
dtype: int64
```

아래와 같은 방법으로 각 세트별 2장의 카드를 무작위로 뽑을 수 있다.

```
In [118]: deck.groupby(get_suit, group_keys=False).apply(draw,n=2)
Out[118]:
3C     3
AC     1
2D     2
3D     3
3H     3
7H     7
QS    10
7S     7
dtype: int64
```

10.3.5 예제: 그룹 가중 평균과 상관관계

groupby의 나누고 적용하고 합치는 패러다임에서 (그룹 가중 평균과 같은) DataFrame의 컬럼 간 연산이나 두 Series 간의 연산은 일상적인 일이다. 예를 들어 그룹 키와 값 그리고 어떤 가중치를 갖는 다음 데이터 묶음을 살펴보자.

```
In [120]: df = pd.DataFrame({'catagory':['a','a','a','a',
     ...:                                 'b','b','b','b'],
     ...:                    'data':np.random.randn(8),
     ...:                    'weights':np.random.rand(8)})

In [121]: df
Out[121]:
  catagory      data   weights
0        a -2.419846  0.679589
1        a  1.452334  0.155272
2        a -0.386101  0.879118
3        a -0.298851  0.596061
4        b -0.035605  0.141617
5        b  0.355341  0.121727
6        b  0.070556  0.337134
7        b  0.217770  0.698021
```

category 별 그룹 가중 평균을 보면 다음과 같다.

```
In [122]: grouped = df.groupby('catagory')

In [125]: get_wavg = lambda g: np.average(g['data'],weights = g['weights'])

In [126]: grouped.apply(get_wavg)
Out[126]:
catagory
a   -0.838321
b    0.164812
dtype: float64
```

좀 더 복잡한 예제로 야후! 파이낸스에서 가져온 몇몇 주식과 S&P 500 지수 (종목 코드 SPX의 종가 데이터를 살펴보자.)

```
In [127]: close_px = pd.read_csv('stock_px_2.csv', parse_dates = True,
```

```
     ...:                           index_col=0)

In [128]: close_px.info()
<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

In [130]: close_px[-4:]
Out[130]:
              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
```

퍼센트 변화율로 일일 수익률을 계산하여 연간 SPX 지수와의 상관관계를 살펴보는 일은 흥미로울 수 있는데, 다음과 같이 구할 수 있다. 우선 'SPX' 컬럼과 다른 컬럼의 **상관관계**를 계산하는 함수를 마만든다.

```
In [9]: spx_corr = lambda x:x.corrwith(x['SPX'])
```

그리고 pct_change 함수를 이용해서 close_px의 퍼센트 변화율을 계산한다.

```
In [35]: rets = close_px.pct_change().dropna()

In [36]: rets
Out[36]:
                AAPL      MSFT       XOM       SPX
2003-01-03  0.006757  0.001421  0.000684 -0.000484
2003-01-06  0.000000  0.017975  0.024624  0.022474
2003-01-07 -0.002685  0.019052 -0.033712 -0.006545
2003-01-08 -0.020188 -0.028272 -0.004145 -0.014086
2003-01-09  0.008242  0.029094  0.021159  0.019386
...              ...       ...       ...       ...
2011-10-10  0.051406  0.026286  0.036977  0.034125
2011-10-11  0.029526  0.002227 -0.000131  0.000544
2011-10-12  0.004747 -0.001481  0.011669  0.009795
2011-10-13  0.015515  0.008160 -0.010238 -0.002974
2011-10-14  0.033225  0.003311  0.022784  0.017380

[2213 rows x 4 columns]
```

마지막으로 각 datetime에서 연도 속성만 한줄짜리 함수를 이용해서 연도별 퍼센트 변화율을 구한다.

```
In [48]: get_year = lambda x : x.year

In [49]: by_year = rets.groupby(get_year)

In [50]: by_year.apply(spx_corr)
```

```
Out[50]:
          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
```

물론 두 컬럼 간의 상관관계를 계산하는 것도 가능하다. 다음은 애플과 마이크로소프트 주가의 연관 상관관계다.

```
In [51]: by_year.apply(lambda g:g['AAPL'].corr(g['SPX']))
Out[51]:
2003    0.541124
2004    0.374283
2005    0.467540
2006    0.428267
2007    0.508118
2008    0.681434
2009    0.707103
2010    0.710105
2011    0.691931
dtype: float64
```

10.3.6 예제 : 그룹상의 선형회귀

이전 예제와 같은 맥락으로, pandas 객체나 스칼라값을 반환하기만 한다면 groupby를 좀 더 복잡한 그룹상의 통계분석을 위해 사용할 수 있다. 예를 들어 계량 경제 라이브러리 인 statmodel 를 사용해서 regress 라는 함수를 작성하고 각 데이터 묶음 마다 최소제곱으로 회귀를 수행할 수 있다.

```
In [52]: import statsmodels.api as sm

In [53]: def regress(data, yvar, xvars):
    ...:     Y = data[yvar]
    ...:     X = data[xvars]
    ...:     X['intercept']=1.
    ...:     result = sm.OLS(Y,X).fit()
    ...:     return result.params
```

이제 SPX 수익률에 대한 애플(APPL) 주식의 연관 선형회귀는 다음과 같다.

```
In [55]: by_year.apply(regress, 'AAPL',['SPX'])
Out[55]:
           SPX  intercept
2003  1.195406   0.000710
2004  1.363463   0.004201
2005  1.766415   0.003246
2006  1.645496   0.000080
2007  1.198761   0.003438
2008  0.968016  -0.001110
2009  0.879103   0.002954
2010  1.052608   0.001261
2011  0.806605   0.001514
```

10.4 피벗테이블과 교차알림표

피벗 테이블은 스프레드시트 프로그램과 그 외 다른 데이터 분석 SW에서 흔히 볼 수 있는 데이터 요약 도구다. 피벗 테이블은 데이터를 하나 이상의 키로 수집해서 어떤 키는 로우에, 어떤 키는 컬럼에 나열해서 데이터를 정렬한다. pandas에서 피벗테이블은 이 장에서 설명했던 groupby기능을 사용해서 계층적 색인을 활용한 재형성 연산을 가능하게 해준다. DataFrame에는 pivot_table 메서드가 있는데 이는 pandas 모듈의 최상위 함수로도 존재한다.

(pandas.pivot_table).groupby를 위한 편리한 인터페이스를 제공하기 위해 pivot_table은 **마진**

이라고 하는 부분합을 추가할 수 있는 기능을 제공한다.

팁 데이터로 돌아가서 요일(day)과 흡연자(smoker) 집단에서 평균 (pivot_table의 기본 연산을 구해보자.)

```
In [64]: tips.pivot_table(index = ['day', 'smoker'])
Out[64]:
                 size       tip  total_bill
day  smoker
Fri  No      2.250000  2.812500   18.420000
     Yes     2.066667  2.714000   16.813333
Sat  No      2.555556  3.102889   19.661778
     Yes     2.476190  2.875476   21.276667
Sun  No      2.929825  3.167895   20.506667
     Yes     2.578947  3.516842   24.120000
Thur No      2.488889  2.673778   17.113111
     Yes     2.352941  3.030000   19.190588
```

이는 groupby를 사용해서 쉽게 구할 수 있는데, 이제 tip_pct와 size에 대해서만 집계를 하고 날짜(time) 별로 그룹지어보자. 이를 위해 day 로우와 smoker 컬럼을 추가했다.

```
In [69]: tips.pivot_table(['tip_pct','size'],index = ['time','day'],
    ...:                                   columns = 'smoker')
Out[69]:
                size             tip_pct
smoker            No       Yes        No       Yes
time   day
Dinner Fri   2.000000  2.222222  0.162612  0.202545
       Sat   2.555556  2.476190  0.190412  0.179833
       Sun   2.929825  2.578947  0.193617  0.322021
       Thur  2.000000       NaN  0.190114       NaN
Lunch  Fri   3.000000  1.833333  0.231125  0.236915
       Thur  2.500000  2.352941  0.193499  0.198508
```

이 테이블은 margins = True를 넘겨서 부분합을 포함하도록 확장할 수 있는데, 그렇게 하면 All 컬럼과 All 로우가 추가되어 단일 줄 안에서 그룹 통계를 얻을 수 있다.

```
In [70]: tips.pivot_table(['tip_pct','size'],index = ['time','day'],
    ...:                                   columns = 'smoker',margins=True)
    ...:
Out[70]:
                size                      tip_pct
smoker            No       Yes       All        No       Yes       All
time   day
Dinner Fri   2.000000  2.222222  2.166667  0.162612  0.202545  0.192562
       Sat   2.555556  2.476190  2.517241  0.190412  0.179833  0.185305
       Sun   2.929825  2.578947  2.842105  0.193617  0.322021  0.225718
       Thur  2.000000       NaN  2.000000  0.190114       NaN  0.190114
Lunch  Fri   3.000000  1.833333  2.000000  0.231125  0.236915  0.236088
       Thur  2.500000  2.352941  2.459016  0.193499  0.198508  0.194895
All          2.668874  2.408602  2.569672  0.192237  0.218176  0.202123
```

여기서 All 값은 흡연자와 비흡연자를 구분하지 않은 평균값(All 컬럼)이거나 로우에서 두 단계를 묶은 그룹의 평균값(All 로우)이다.

다른 집계함수를 사용하려면 그냥 aggfunc로 넘기면 되는데, 예를 들어 'count'나 len 함수는 그룹 크기의 교차일람표 (총 개수나 빈도)를 반환한다.

```
In [72]: tips.pivot_table('tip_pct',index = ['time','smoker'],
    ...:                                   columns = 'day',aggfunc = len,
  ...:                                  margins=True)
Out[72]:
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

In [73]: tips.pivot_table('tip_pct',index = ['time','smoker'],
    ...:                                   columns = 'day',aggfunc = len,
    ...:                                   margins=False)
Out[73]:
day             Fri   Sat   Sun  Thur
time   smoker
Dinner No       3.0  45.0  57.0   1.0
```

```
       Yes     9.0  42.0  19.0   NaN
Lunch  No      1.0   NaN   NaN  44.0
       Yes     6.0   NaN   NaN  17.0
```

만약 어떤 조합이 비어 있다면 (혹은 NA값) fill_value를 넘길 수도 있다.

```
In [74]: tips.pivot_table('tip_pct',index = ['time','size', 'smoker'],
    ...:                  columns='day', aggfunc = 'mean', fill_value =0)
Out[74]:
day                       Fri       Sat       Sun      Thur
time   size smoker
Dinner 1    No      0.000000  0.160000  0.000000  0.000000
            Yes     0.000000  0.483092  0.000000  0.000000
       2    No      0.162612  0.198319  0.206535  0.190114
            Yes     0.211180  0.178877  0.400522  0.000000
       3    No      0.000000  0.183870  0.182962  0.000000
            Yes     0.000000  0.176599  0.183278  0.000000
       4    No      0.000000  0.177734  0.175289  0.000000
            Yes     0.133465  0.147074  0.254373  0.000000
       5    No      0.000000  0.000000  0.263344  0.000000
            Yes     0.000000  0.119284  0.070274  0.000000
       6    No      0.000000  0.000000  0.115821  0.000000
Lunch  1    No      0.000000  0.000000  0.000000  0.222087
            Yes     0.288288  0.000000  0.000000  0.000000
       2    No      0.000000  0.000000  0.000000  0.201503
            Yes     0.226641  0.000000  0.000000  0.191197
       3    No      0.231125  0.000000  0.000000  0.092162
            Yes     0.000000  0.000000  0.000000  0.257941
       4    No      0.000000  0.000000  0.000000  0.161573
            Yes     0.000000  0.000000  0.000000  0.186592
       5    No      0.000000  0.000000  0.000000  0.138160
       6    No      0.000000  0.000000  0.000000  0.211191
```

pivot_table 메서드를 요약해 두었다.

pivot_table 옵션

함수	설명
values	집계하려는 컬럼 이름 혹은 이름의 리스트, 기본적으로 모든 숫자 컬럼을 집계한다.
index	만들어지는 피벗테이블의 로우를 그룹으로 묶을 컬럼 이름이나 그룹 키
columns	만들어지는 피벗테이블의 컬럼을 그룹으로 묶을 컬럼 이름이나 그룹 키
aggfunc	집계함수나 함수 리스트, 기본값은 'mean'이 사용된다. groupby 컨텍스트 안에서 유효한 어떤 함수라도 가능하다.
fill_value	결과 테이블에서 누락된 값을 대체하기 위한 값
dropna	True인 경우 모든 항목이 NA인 컬럼은 포함하지 않는다.
margins	부분합이나 총계를 담기 위한 로우/컬럼을 추가할지 여부. 기본값은 False

10.5 마치며

pandas의 데이터 그룹핑 도구를 마스터한다면 데이터 정제 뿐만 아니라 모델링이나 통계 분석 작업에도 도움이 될 것이다.