파이썬 라이브러리를 활용한 데이터 분석

10장 데이터 집계와 그룹 연산

2020.07.09목 2h

10장 데이터 집계와 그룹 연산

데이터 집계

.5h

데이터 집계 p 396

- 배열에서 스칼라 값을 만들어 내는 모든 데이터 변환 작업

```
In [92]: df
```

Out[92]:

	key1	key2	data1	data2
0	a	one	1.007189	0.886429
1	a	two	-1.296221	-2.001637
2	b	one	0.274992	-0.371843
3	b	two	0.228913	1.669025
4	a	one	1.352917	-0.438570

```
In [93]: grouped = df.groupby('key1')
         grouped['data1'].quantile(0.9)
```

Out[93]:
```
key1
a    1.283771
b    0.270384
Name: data1, dtype: float64
```

```
In [94]: def peak_to_peak(arr):
             return arr.max() - arr.min()
         grouped.agg(peak_to_peak)
```

Out[94]:

	data1	data2
key1		
a	2.649138	2.888067
b	0.046079	2.040868

```
In [98]: grouped.agg(['mean', 'max', 'min'])
```

Out[98]:

	data1			data2		
	mean	max	min	mean	max	min
key1						
a	0.354628	1.352917	-1.296221	-0.517926	0.886429	-2.001637
b	0.251952	0.274992	0.228913	0.648591	1.669025	-0.371843

```
In [95]: grouped.describe()
```

Out[95]:

	data1								data2							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	m
key1																
a	3.0	0.354628	1.440090	-1.296221	-0.144516	1.007189	1.180053	1.352917	3.0	-0.517926	1.445668	-2.001637	-1.220104	-0.438570	0.223930	0.
b	2.0	0.251952	0.032583	0.228913	0.240433	0.251952	0.263472	0.274992	2.0	0.648591	1.443111	-0.371843	0.138374	0.648591	1.158808	1.

그룹을 나눈 후 한 컬럼에 하나의 함수 적용

- 메소드 agg('적용함수명')

```
In [99]: tips = pd.read_csv('examples/tips.csv')
         # Add tip percentage of total bill
         tips['tip_pct'] = tips['tip'] / tips['total_bill']
         tips[:6]
```

Out[99]:

	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

```
In [100]: grouped = tips.groupby(['day', 'smoker'])
```

```
In [101]: grouped_pct = grouped['tip_pct']
          grouped_pct.agg('mean')
```

Out[101]:

```
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
```

그룹핑 후 값인 팁 비율에 여러 함수 적용

- **함수가 열명**
  - def peak_to_peak(arr):
  - return arr.max() - arr.min()

```
In [104]: grouped_pct.agg(['mean', 'std', peak_to_peak])
```

Out[104]:

		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

- **함수의 열명 설정**
  - (name, function), ...

```
In [105]: grouped_pct.agg([('foo', 'mean'), ('bar', np.std)])
```

Out[105]:

		foo	bar
day	smoker		
Fri	No	0.151650	0.028123
	Yes	0.174783	0.051293
Sat	No	0.158048	0.039767
	Yes	0.147906	0.061375
Sun	No	0.160113	0.042347
	Yes	0.187250	0.154134
Thur	No	0.160298	0.038774
	Yes	0.163863	0.039389

여러 컬럼에 동일 함수 적용

- 칼럼 tip_pct와 total_bill에 3 개의 함수 적용
- 컬럼명 바꾸기
  - ('컬럼명', '메소드')

```
In [114]: #ftuples = [('Durchschnitt', 'mean'), ('Abweichung', np.var)]
          ftuples = [('평균', 'mean'), ('분산', np.var)]
          grouped[['tip_pct', 'total_bill']].agg(ftuples)
```

Out[114]:

		tip_pct		total_bill	
		평균	분산	평균	분산
day	smoker				
Fri	No	0.151650	0.000791	18.420000	25.596333
	Yes	0.174783	0.002631	16.813333	82.562438
Sat	No	0.158048	0.001581	19.661778	79.908965
	Yes	0.147906	0.003767	21.276667	101.387535
Sun	No	0.160113	0.001793	20.506667	66.099980
	Yes	0.187250	0.023757	24.120000	109.046044
Thur	No	0.160298	0.001503	17.113111	59.625081
	Yes	0.163863	0.001551	19.190588	69.808518

```
In [108]: functions = ['count', 'mean', 'max']
          result = grouped[['tip_pct', 'total_bill']].agg(functions)
          result
```

Out[108]:

		tip_pct			total_bill		
		count	mean	max	count	mean	max
day	smoker						
Fri	No	4	0.151650	0.187735	4	18.420000	22.75
	Yes	15	0.174783	0.263480	15	16.813333	40.17
Sat	No	45	0.158048	0.291990	45	19.661778	48.33
	Yes	42	0.147906	0.325733	42	21.276667	50.81
Sun	No	57	0.160113	0.252672	57	20.506667	48.17
	Yes	19	0.187250	0.710345	19	24.120000	45.35
Thur	No	45	0.160298	0.266312	45	17.113111	41.19
	Yes	17	0.163863	0.241255	17	19.190588	43.11

```
In [109]: result['tip_pct']
```

Out[109]:

		count	mean	max
day	smoker			
Fri	No	4	0.151650	0.187735
	Yes	15	0.174783	0.263480
Sat	No	45	0.158048	0.291990
	Yes	42	0.147906	0.325733
Sun	No	57	0.160113	0.252672
	Yes	19	0.187250	0.710345
Thur	No	45	0.160298	0.266312
	Yes	17	0.163863	0.241255

칼럼마다 다른 함수 적용

- **사전 형식**
  - { '칼럼명1': [함수1, 함수2],
    '칼럼명2': [함수3, 함수4],
    '칼럼명3': [함수5, 함수6], ...
    }

```
In [115]: grouped.agg({'tip' : np.max, 'size' : 'sum'})
```

Out[115]:

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

Out[116]:

		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

색인되지 않은 형태로 집계된 데이터 변환하기

- 옵션 as_index=False

```
In [119]: tips.groupby(['day', 'smoker'], as_index=False).mean()
```

Out[119]:

	day	smoker	total_bill	tip	size	tip_pct
0	Fri	No	18.420000	2.812500	2.250000	0.151650
1	Fri	Yes	16.813333	2.714000	2.066667	0.174783
2	Sat	No	19.661778	3.102889	2.555556	0.158048
3	Sat	Yes	21.276667	2.875476	2.476190	0.147906
4	Sun	No	20.506667	3.167895	2.929825	0.160113
5	Sun	Yes	24.120000	3.516842	2.578947	0.187250
6	Thur	No	17.113111	2.673778	2.488889	0.160298
7	Thur	Yes	19.190588	3.030000	2.352941	0.163863

```
In [120]: tips.groupby(['day', 'smoker'], as_index=True).mean()
```

Out[120]:

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

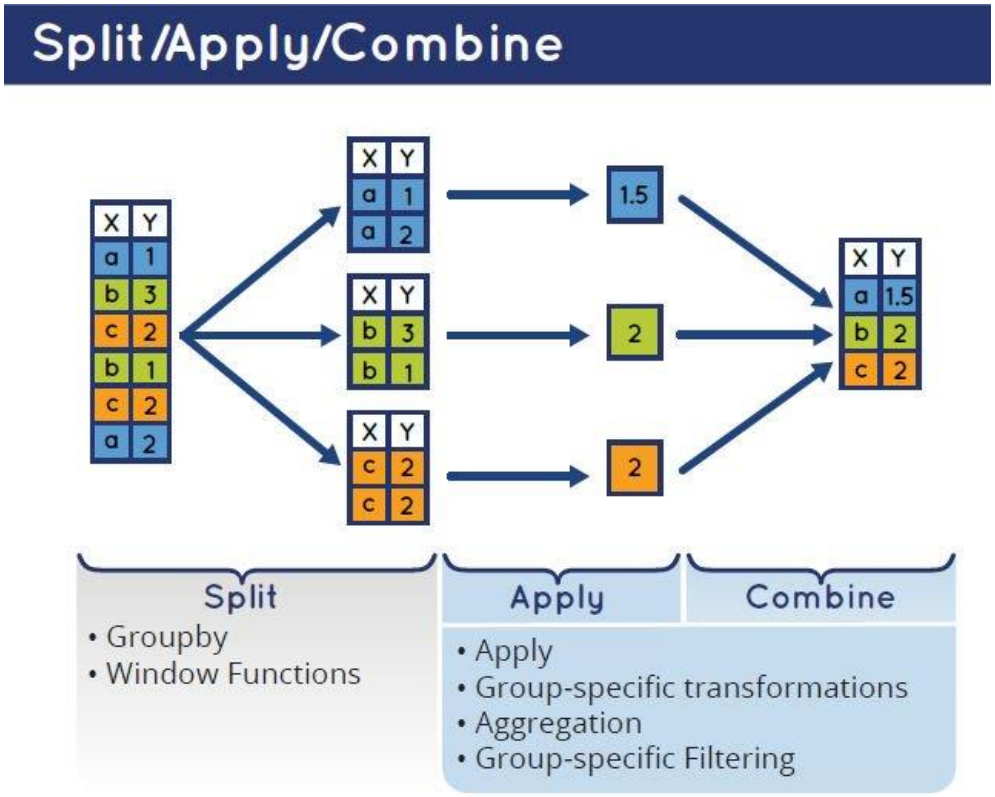
10장 데이터 집계와 그룹 연산

Apply: 일반적인 분리-적용-병합

2h

메소드 apply()

- 객체를 여러 조각인 그룹으로 나누고,
- 전달된 함수를 각 그룹에 일괄 적용한 후 이를 다시 합쳐, 데이터프레임 (또는 시리즈) 반환



그룹별 상위 tip_pct

- 상위 6개
- 흡연 여부에 따른 상위 5개
- 메소드 apply(함수)에서
  - 함수가 추가적인 인자가 필요
    - **함수 이름 뒤에 붙여서 넘김**

```
In [121]: def top(df, n=5, column='tip_pct'):
              return df.sort_values(by=column)[-n:]
          top(tips, n=6)
```

Out[121]:

	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

```
In [122]: tips.groupby('smoker').apply(top)
```

Out[122]:

		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

```
In [123]: tips.groupby(['smoker', 'day']).apply(top, n=1, column='total_bill')
```

Out[123]:

			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

메소드 describe() 활용

```
In [124]: result = tips.groupby('smoker')['tip_pct'].describe()
          result
```

Out[124]:

smoker	count	mean	std	min	25%	50%	75%	max
No	151.0	0.159328	0.039910	0.056797	0.136906	0.155625	0.185014	0.291990
Yes	93.0	0.163196	0.085119	0.035638	0.106771	0.153846	0.195059	0.710345

```
In [125]: result.unstack('smoker')
```

Out[125]:

```
        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
```

그룹 색인 생략하기

- **옵션 group_keys=False**
  - 색인을 하지 않고 열에 남음

```
In [129]: tips.groupby('smoker', group_keys=False).apply(top)
```

NG

Out[129]:

	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

```
In [128]: tips.groupby('smoker').apply(top)
```

Out[128]:

		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

등 간격 버킷 분석

- 난수에서 4등분한 버킷의 수, 최대, 평균, 최소 구하기

```
In [143]: frame = pd.DataFrame({'data1': np.random.randn(1000),
                                'data2': np.random.randn(1000)})
          frame
```

Out[143]:

	data1	data2
0	0.324627	-0.882594
1	1.032549	-1.131414
2	1.054085	0.490516
3	-1.277746	-0.032514
4	0.644594	0.519580
...	...	...
995	0.460863	-0.927341
996	1.694641	1.514676
997	-0.935589	1.588827
998	0.766984	0.712962
999	-2.040757	2.095029

1000 rows × 2 columns

```
In [144]: quartiles = pd.cut(frame.data1, 4)
```

```
In [133]: quartiles[:10]
```

```
Out[133]: 0     (-0.387, 1.133]
          1     (-0.387, 1.133]
          2    (-3.434, -1.908]
          3     (-0.387, 1.133]
          4    (-1.908, -0.387]
          5    (-1.908, -0.387]
          6     (-0.387, 1.133]
          7     (-0.387, 1.133]
          8    (-1.908, -0.387]
          9     (-0.387, 1.133]
          Name: data1, dtype: category
          Categories (4, interval[float64]): [(-3.434, -1.908] < (-1.908, -0.387] < (-0.38
          7, 1.133] < (1.133, 2.654]]
```

```
In [145]: def get_stats(group):
              return {'min': group.min(), 'max': group.max(),
                      'count': group.count(), 'mean': group.mean()}
          grouped = frame.data2.groupby(quartiles)
          grouped
```

```
Out[145]: <pandas.core.groupby.generic.SeriesGroupBy object at 0x00000260B3FB12C8>
```

```
In [146]: grouped.apply(get_stats)
```

```
Out[146]: data1
          (-3.653, -1.853]  min     -1.236664
                            max      2.135719
                            count   36.000000
                            mean     0.098801
          (-1.853, -0.06]   min     -3.018842
                            max      2.903300
                            count  453.000000
                            mean    -0.016572
          (-0.06, 1.733]    min     -3.333767
                            max      2.497837
                            count  479.000000
                            mean    -0.028958
          (1.733, 3.526]    min     -2.494075
                            max      1.676926
                            count   32.000000
                            mean    -0.096919
          Name: data2, dtype: float64
```

```
In [147]: grouped.apply(get_stats).unstack()
```

Out[147]:

	min	max	count	mean
data1				
(-3.653, -1.853]	-1.236664	2.135719	36.0	0.098801
(-1.853, -0.06]	-3.018842	2.903300	453.0	-0.016572
(-0.06, 1.733]	-3.333767	2.497837	479.0	-0.028958
(1.733, 3.526]	-2.494075	1.676926	32.0	-0.096919

수가 같은 버킷 분석

- **표본 변위치 기반**
  - 크기가 같은 버킷
    - **qcut()**
- **옵션 labels=False**
  - 구간이 없이 정수 인덱스로

```
In [148]: # Return quantile numbers
          #grouping = pd.qcut(frame.data1, 10, labels=False)
          grouping = pd.qcut(frame.data1, 10)
          grouped = frame.data2.groupby(grouping)
          grouped.apply(get_stats)
```

```
Out[148]: data1
          (-3.647, -1.272]  min       -3.018842
                            max        2.146716
                            count    100.000000
                            mean      -0.072422
          (-1.272, -0.87]   min       -2.641014
                                          ...
          (0.833, 1.242]    mean      -0.004802
          (1.242, 3.526]    min       -2.494075
                            max        1.716892
                            count    100.000000
                            mean      -0.079520
          Name: data2, Length: 40, dtype: float64
```

```
In [142]: # Return quantile numbers
          grouping = pd.qcut(frame.data1, 10, labels=False)
          grouped = frame.data2.groupby(grouping)
          grouped.apply(get_stats)
```

```
Out[142]: data1
          0      min       -2.434322
                 max        2.465567
                 count    100.000000
                 mean      -0.004318
          1      min       -2.909373
                                ...
          8      mean      -0.054800
          9      min       -1.872121
                 max        2.615416
                 count    100.000000
                 mean       0.127347
          Name: data2, Length: 40, dtype: float64
```

```
In [141]: grouped.apply(get_stats).unstack()
```

Out[141]:

data1	min	max	count	mean
0	-2.434322	2.465567	100.0	-0.004318
1	-2.909373	2.531127	100.0	-0.156638
2	-2.480208	2.275784	100.0	-0.011883
3	-3.548824	2.374374	100.0	0.024064
4	-2.128789	2.419003	100.0	0.039156
5	-1.995456	2.492224	100.0	0.072292
6	-2.372214	3.366626	100.0	0.025465
7	-2.186301	1.861150	100.0	0.002594
8	-2.748685	2.452835	100.0	-0.054800
9	-1.872121	2.615416	100.0	0.127347

결측 치 채우기

- **메소드 fillna()**

```
In [149]: s = pd.Series(np.random.randn(6))
          s[::2] = np.nan
          s

Out[149]: 0         NaN
          1    1.207528
          2         NaN
          3   -0.998061
          4         NaN
          5   -1.356067
          dtype: float64

In [150]: s.fillna(s.mean())

Out[150]: 0   -0.382200
          1    1.207528
          2   -0.382200
          3   -0.998061
          4   -0.382200
          5   -1.356067
          dtype: float64
```

그룹별 평균 구하기

- **결측치를 그대로 그룹별 평균 구하기**
  - 그룹을 외부 배열 사용

```
In [154]: states = ['Ohio', 'New York', 'Vermont', 'Florida',
                    'Oregon', 'Nevada', 'California', 'Idaho']
          data = pd.Series(np.random.randn(8), index=states)

          data[['Vermont', 'Nevada', 'Idaho']] = np.nan
          data

Out[154]: Ohio          -1.765102
          New York       2.575626
          Vermont             NaN
          Florida      -1.772692
          Oregon       -0.408560
          Nevada              NaN
          California   -1.437569
          Idaho               NaN
          dtype: float64

In [155]: group_key = ['East'] * 4 + ['West'] * 4
          data.groupby(group_key).mean()

Out[155]: East   -0.320723
          West   -0.923065
          dtype: float64
```

그룹별로 na 채우기

- 그룹별 평균을 na로 채우기
- 그룹에 특정 값 지정
  - 사전 형식으로
    - fill_func = lambda g: g.fillna(fill_values[g.name])

```
In [156]: fill_mean = lambda g: g.fillna(g.mean())
          data.groupby(group_key).apply(fill_mean)

Out[156]: Ohio          -1.765102
          New York       2.575626
          Vermont       -0.320723
          Florida       -1.772692
          Oregon        -0.408560
          Nevada        -0.923065
          California    -1.437569
          Idaho         -0.923065
          dtype: float64
```

```
Out[154]: Ohio          -1.765102
          New York       2.575626
          Vermont             NaN
          Florida       -1.772692
          Oregon        -0.408560
          Nevada              NaN
          California    -1.437569
          Idaho               NaN
          dtype: float64
```

```
In [158]: fill_values = {'East': 0.5, 'West': -1}
          fill_func = lambda g: g.fillna(fill_values[g.name])
          data.groupby(group_key).apply(fill_func)

Out[158]: Ohio          -1.765102
          New York       2.575626
          Vermont        0.500000
          Florida       -1.772692
          Oregon        -0.408560
          Nevada        -1.000000
          California    -1.437569
          Idaho         -1.000000
          dtype: float64
```

결측치 처리에 따른 결과 비교

- **결측치 처리 후 비교**

```
In [164]: group_key = ['East'] * 4 + ['West'] * 4
          data.groupby(group_key).mean()

Out[164]: East   -0.320723
          West   -0.923065
          dtype: float64

In [165]: fill_values = {'East': 0.5, 'West': -1}
          fill_func = lambda g: g.fillna(fill_values[g.name])
          data2 = data.groupby(group_key).apply(fill_func)
          data2.groupby(group_key).mean()

Out[165]: East   -0.115542
          West   -0.961532
          dtype: float64
```

트럼프 카드 예시

- **카드 덱**
  - 총 13*4 = 52개
  - 모양
    - **4개**

```
In [167]: # Hearts, Spades, Clubs, Diamonds
          suits = ['H', 'S', 'C', 'D']
          card_val = (list(range(1, 11)) + [10] * 3) * 4
          base_names = ['A'] + list(range(2, 11)) + ['J', 'K', 'Q']
          cards = []
          for suit in ['H', 'S', 'C', 'D']:
              cards.extend(str(num) + suit for num in base_names)

          deck = pd.Series(card_val, index=cards)
          deck

Out[167]: AH      1
          2H      2
          3H      3
          4H      4
          5H      5
                  ..
          9D      9
          10D    10
          JD     10
          KD     10
          QD     10
          Length: 52, dtype: int64

In [168]: deck[:13]

Out[168]: 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
```

카드 뽑기

- 카드 5장 랜덤하게 뽑기
- 각 셋 그림별 2장 뽑기
  - deck.groupby(get_suit)
    - 인덱스의 마지막 문자인 카드의 종류(C, D, H, S)에 따라 그룹핑
  - 그림 첫 글자 보이기/안보이기
    - group_keys=False

In [169]:
```
def draw(deck, n=5):
    return deck.sample(n)
draw(deck)
```

Out[169]:
```
JS    10
KC    10
JC    10
AS     1
9H     9
dtype: int64
```

In [180]:
```
[cards[:12], cards[13:25], cards[26:38], cards[-12:]]
```

Out[180]:
```
[['AH', '2H', '3H', '4H', '5H', '6H', '7H', '8H', '9H', '10H', 'JH', 'KH'],
 ['AS', '2S', '3S', '4S', '5S', '6S', '7S', '8S', '9S', '10S', 'JS', 'KS'],
 ['AC', '2C', '3C', '4C', '5C', '6C', '7C', '8C', '9C', '10C', 'JC', 'KC'],
 ['2D', '3D', '4D', '5D', '6D', '7D', '8D', '9D', '10D', 'JD', 'KD', 'QD']]
```

In [181]:
```
get_suit = lambda card: card[-1] # last letter is suit
deck.groupby(get_suit).apply(draw, n=2)
```

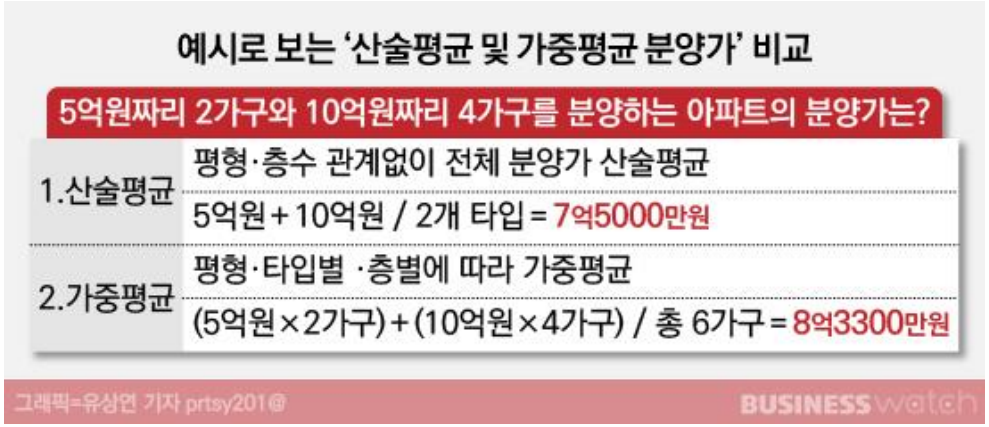
Out[181]:
```
C  7C     7
   3C     3
D  QD    10
   JD    10
H  7H     7
   JH    10
S  KS    10
   7S     7
dtype: int64
```

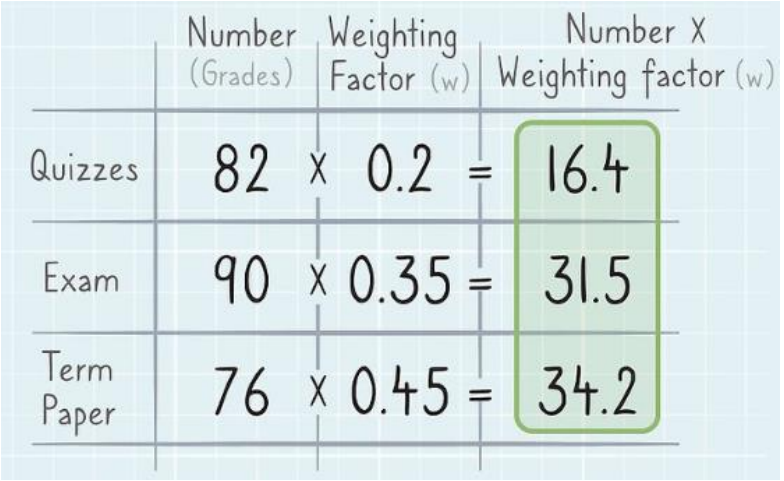
In [182]:
```
deck.groupby(get_suit, group_keys=False).apply(draw, n=2)
```

Out[182]:
```
6C     6
2C     2
10D   10
3D     3
5H     5
2H     2
10S   10
2S     2
dtype: int64
```

가중 평균

- numpy.average(a, axis=None, weights=None, returned=False)





Weighted Mean Formula

$$\text{Weighted mean} = \frac{\sum_{i=1}^{n} \text{weight}_n \times x_n}{\sum_{i=1}^{n} \text{weight}_n}$$

그룹별 가중 평균 구하기

- **열 caterory 별 가중 평균**

In [183]:
```
df = pd.DataFrame({'category': ['a', 'a', 'a', 'a',
                                'b', 'b', 'b', 'b'],
                   'data': np.random.randn(8),
                   'weights': np.random.rand(8)})
df
```

Out[183]:

	category	data	weights
0	a	-1.038668	0.194101
1	a	-0.652685	0.281413
2	a	-1.224352	0.786156
3	a	-0.154844	0.205984
4	b	-0.013619	0.068997
5	b	0.568799	0.217711
6	b	-1.351436	0.542650
7	b	-0.797593	0.974070

In [184]:
```
grouped = df.groupby('category')
get_wavg = lambda g: np.average(g['data'], weights=g['weights'])
grouped.apply(get_wavg)
```

Out[184]:
```
category
a   -0.940077
b   -0.769298
dtype: float64
```

미국 주식 종가 데이터 확인

- **메소드 read_csv(...)**
  - 옵션 parse_dates=True
    - **날짜 정보를 해석하여 DataTime 유형으로 저장**

```
In [12]: !chcp 65001
         Active code page: 65001
```

```
In [15]: !type examples\stock_px_2.csv
         ,AAPL,MSFT,XOM,SPX
         2003-01-02 00:00:00,7.4,21.11,29.22,909.03
         2003-01-03 00:00:00,7.45,21.14,29.24,908.59
         2003-01-06 00:00:00,7.45,21.52,29.96,929.01
         2003-01-07 00:00:00,7.43,21.93,28.95,922.93
         2003-01-08 00:00:00,7.28,21.31,28.83,909.93
         2003-01-09 00:00:00,7.34,21.93,29.44,927.57
         2003-01-10 00:00:00,7.36,21.97,29.03,927.57
         2003-01-13 00:00:00,7.32,22.16,28.91,926.26
         2003-01-14 00:00:00,7.3,22.39,29.17,931.66
         2003-01-15 00:00:00,7.22,22.11,28.77,918.22
         2003-01-16 00:00:00,7.31,21.75,28.9,914.6
         2003-01-17 00:00:00,7.05,20.22,28.6,901.78
         2003-01-21 00:00:00,7.01,20.17,27.94,887.62
         2003-01-22 00:00:00,6.94,20.04,27.58,878.36
         2003-01-23 00:00:00,7.09,20.54,27.52,887.34
         2003-01-24 00:00:00,6.9,19.59,26.93,861.4
         2003-01-27 00:00:00,7.07,19.32,26.21,847.48
         2003-01-28 00:00:00,7.29,19.18,26.9,858.54
```

```
In [8]: close_px = pd.read_csv('examples/stock_px_2.csv', parse_dates=True,
                               index_col=0)
        close_px.shape
Out[8]: (2214, 4)
```

```
In [4]: close_px = pd.read_csv('examples/stock_px_2.csv', index_col=0)
        close_px.shape
Out[4]: (2214, 4)
```

```
In [9]: 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 [10]: close_px.index
Out[10]: DatetimeIndex(['2003-01-02', '2003-01-03', '2003-01-06', '2003-01-07',
                        '2003-01-08', '2003-01-09', '2003-01-10', '2003-01-13',
                        '2003-01-14', '2003-01-15',
                        ...
                        '2011-10-03', '2011-10-04', '2011-10-05', '2011-10-06',
                        '2011-10-07', '2011-10-10', '2011-10-11', '2011-10-12',
                        '2011-10-13', '2011-10-14'],
                       dtype='datetime64[ns]', length=2214, freq=None)
```

미국 주식 종가 데이터프레임 close_px

- 애플, 마소, 엑슨모빌, S&P 500 지수

```
In [16]: close_px[-4:]
```

Out[16]:

	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

SPX와 주식과의 연관관계

- **일일 수익률(일일 변화율)로 상관관계 분석**
  - SPX와 다른 3 회사의 연관관계

```
In [190]: spx_corr = lambda x: x.corrwith(x['SPX'])

In [191]: rets = close_px.pct_change().dropna()

In [192]: get_year = lambda x: x.year
          by_year = rets.groupby(get_year)
          by_year.apply(spx_corr)
```

Out[192]:

	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

  - 애플과 마소의 상관관계

```
In [193]: by_year.apply(lambda g: g['AAPL'].corr(g['MSFT']))
```

Out[193]:

```
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
```

선형회귀

- 딥러닝에서 학습

https://datascienceschool.net/view-notebook/58269d7f52bd49879965cdc4721da42d/

계량경제 라이브러리 **statsmodels**

- **statsmodels 패키지의 OLS 클래스**
  - 선형 회귀분석
    - **model = OLS(y, X)**
  - fit 메서드로 모형 추정
    - **result = model.fit()**
      - **별도의 RegressionResults 클래스 객체로 출력**
  - 메소드 fit()의 결과인 RegressionResults 클래스의 속성
    - **Params: 가중치 벡터**
    - **resid: 잔차 벡터**

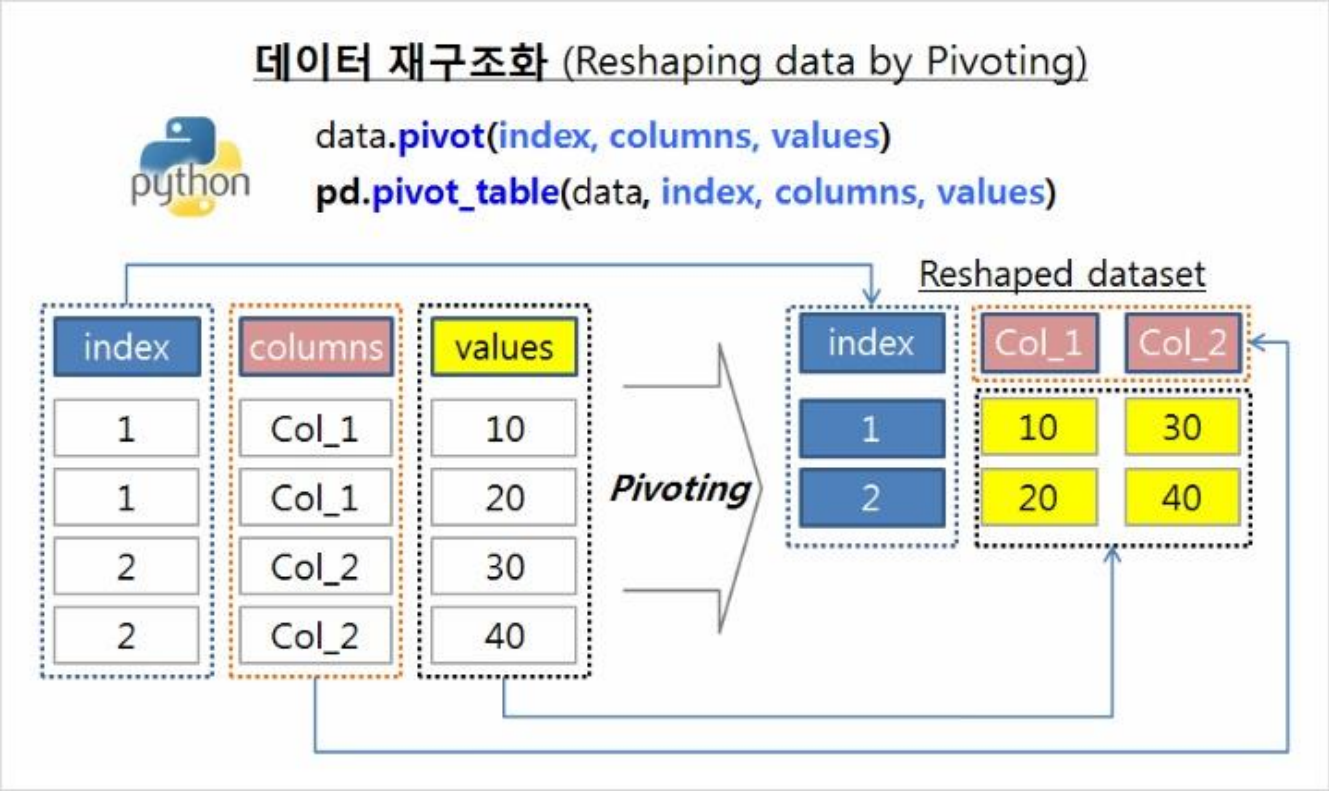
10장 데이터 집계와 그룹 연산

피벗테이블과 교차일람표

2h

피벗 테이블 개요

- **데이터 요약화 도구**
  - 하나 이상의 키로 수집해서, 키를 행과 열에 나눠 데이터를 저장, 기본 값은 평균 값
    - **Groupby()에 의한 계층적 색인을 활용한 재형성 기능**
- **판다스 모듈의 최상위 함수**
  - 데이터프레임 panda.pivot_table()



팁 데이터

- 요일과 흡연자 기준
  - 평균 구하기
- 메소드 pivot_table
- 메소드 groupby

```
In [201]: tips.pivot_table(index=['day', 'smoker'])
```

Out[201]:

day	smoker	size	tip	tip_pct	total_bill
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

```
In [204]: tips.groupby(['day', 'smoker']).mean()
```

Out[204]:

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

= Python

Pivot_table 데이터, 인덱스, 열명 지정

- **목적**
  - 시간(time)과 요일(day)별로 그룹 수(size)와 팁 비율(tip_pct)을 흡연(smoker) 구분(No, Yes)에 따라 평균 구하기
- **다양한 옵션**
  - 옵션 index
  - 옵션 columns
  - 옵션 margins=True
    - **행과 열에 all이 추가**
      - **적절한 부분합 포함되도록**

```
In [205]: tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                           columns='smoker')
```

Out[205]:

		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

```
In [206]: tips.pivot_table(['tip_pct', 'size'], index=['time', 'day'],
                           columns='smoker', margins=True)
```

Out[206]:

		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

흡연과 관계 없는 평균

인덱스인 time, day를 묶은 비 흡연자의 평균

평균이 아닌 다른 집계 함수를 지정

- **옵션 aggfunc=**
  - aggfunc=len
    - **총 개수, 빈도**

```
In [219]: tips.pivot_table('tip_pct', index=['time', 'smoker'], columns='day',
                           aggfunc=len, margins=True)
```

Out[219]:

time	smoker	day Fri	Sat	Sun	Thur	All
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 [220]: tips
```

Out[220]:

	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
...	...	...	...	...	...	...	...
239	29.03	5.92	No	Sat	Dinner	3	0.203927
240	27.18	2.00	Yes	Sat	Dinner	2	0.073584
241	22.67	2.00	Yes	Sat	Dinner	2	0.088222
242	17.82	1.75	No	Sat	Dinner	2	0.098204
243	18.78	3.00	No	Thur	Dinner	2	0.159744

244 rows × 7 columns

비어 있는 값 수정

- **옵션 fill_value=**

```
In [221]: tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
                           columns='day', aggfunc='mean', fill_value=0)
```

Out[221]:

time	size	smoker	day	Fri	Sat	Sun	Thur
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
...	...	...		...	...	...	...
Lunch	3	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

21 rows × 4 columns

```
In [222]: tips.pivot_table('tip_pct', index=['time', 'size', 'smoker'],
                           columns='day', aggfunc='mean')
```

Out[222]:

time	size	smoker	day	Fri	Sat	Sun	Thur
Dinner	1	No		NaN	0.137931	NaN	NaN
		Yes		NaN	0.325733	NaN	NaN
	2	No		0.139622	0.162705	0.168859	0.159744
		Yes		0.171297	0.148668	0.207893	NaN
	3	No		NaN	0.154661	0.152663	NaN
...	...	...		...	...	...	...
Lunch	3	Yes		NaN	NaN	NaN	0.204952
	4	No		NaN	NaN	NaN	0.138919
		Yes		NaN	NaN	NaN	0.155410
	5	No		NaN	NaN	NaN	0.121389
	6	No		NaN	NaN	NaN	0.173706

21 rows × 4 columns

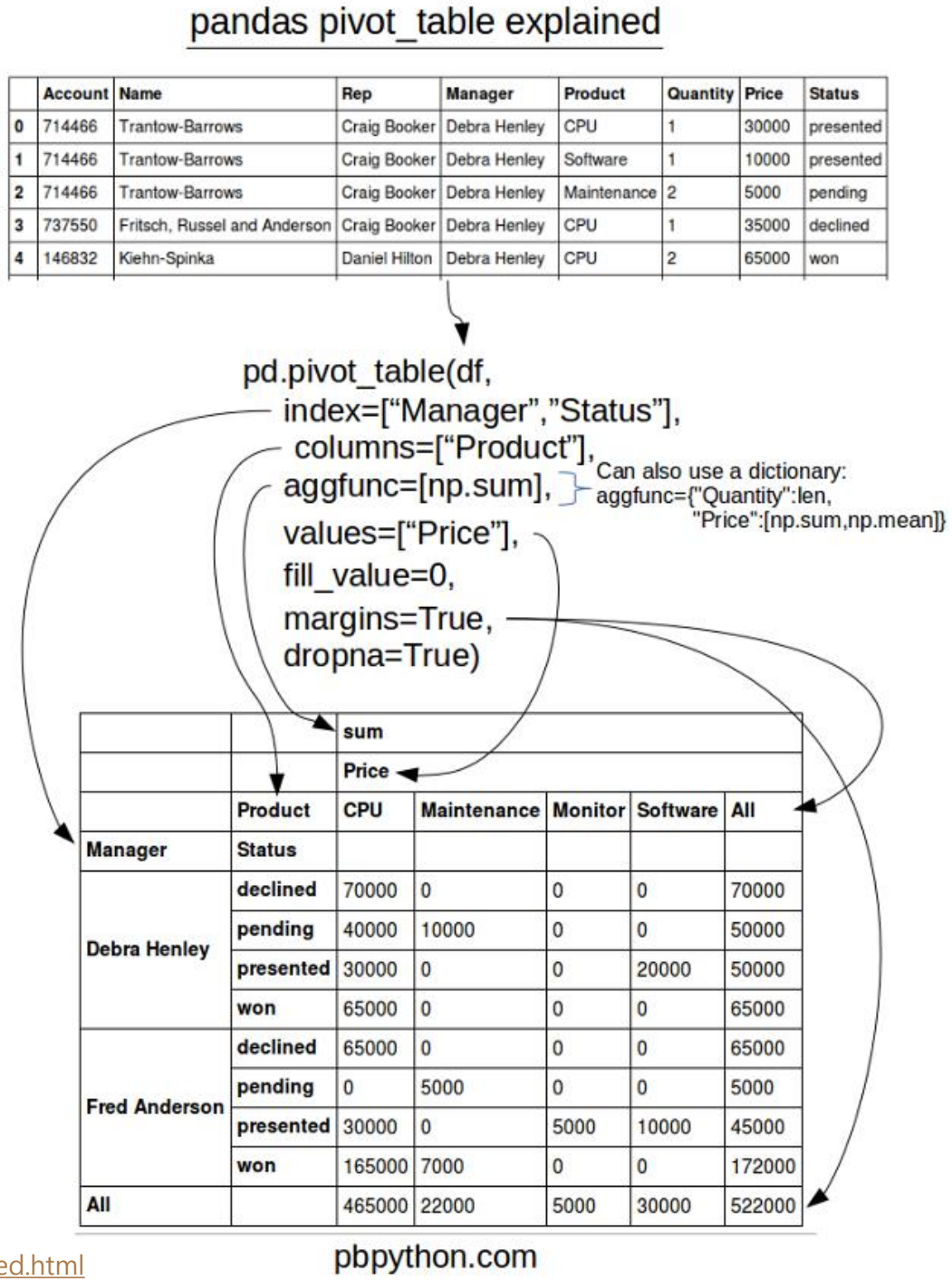
피벗 테이블 옵션

- **DataFrame.pivot_table(self, values=None, index=None, columns=None, aggfunc='mean', fill_value=None, margins=False, dropna=True, margins_name='All', observed=False)**

Table 10-2. pivot_table options

Function name	Description
`values`	Column name or names to aggregate; by default aggregates all numeric columns
`index`	Column names or other group keys to group on the rows of the resulting pivot table
`columns`	Column names or other group keys to group on the columns of the resulting pivot table
`aggfunc`	Aggregation function or list of functions (`'mean'` by default); can be any function valid in a groupby context
`fill_value`	Replace missing values in result table
`dropna`	If `True`, do not include columns whose entries are all NA
`margins`	Add row/column subtotals and grand total (`False` by default)

Pivot_table

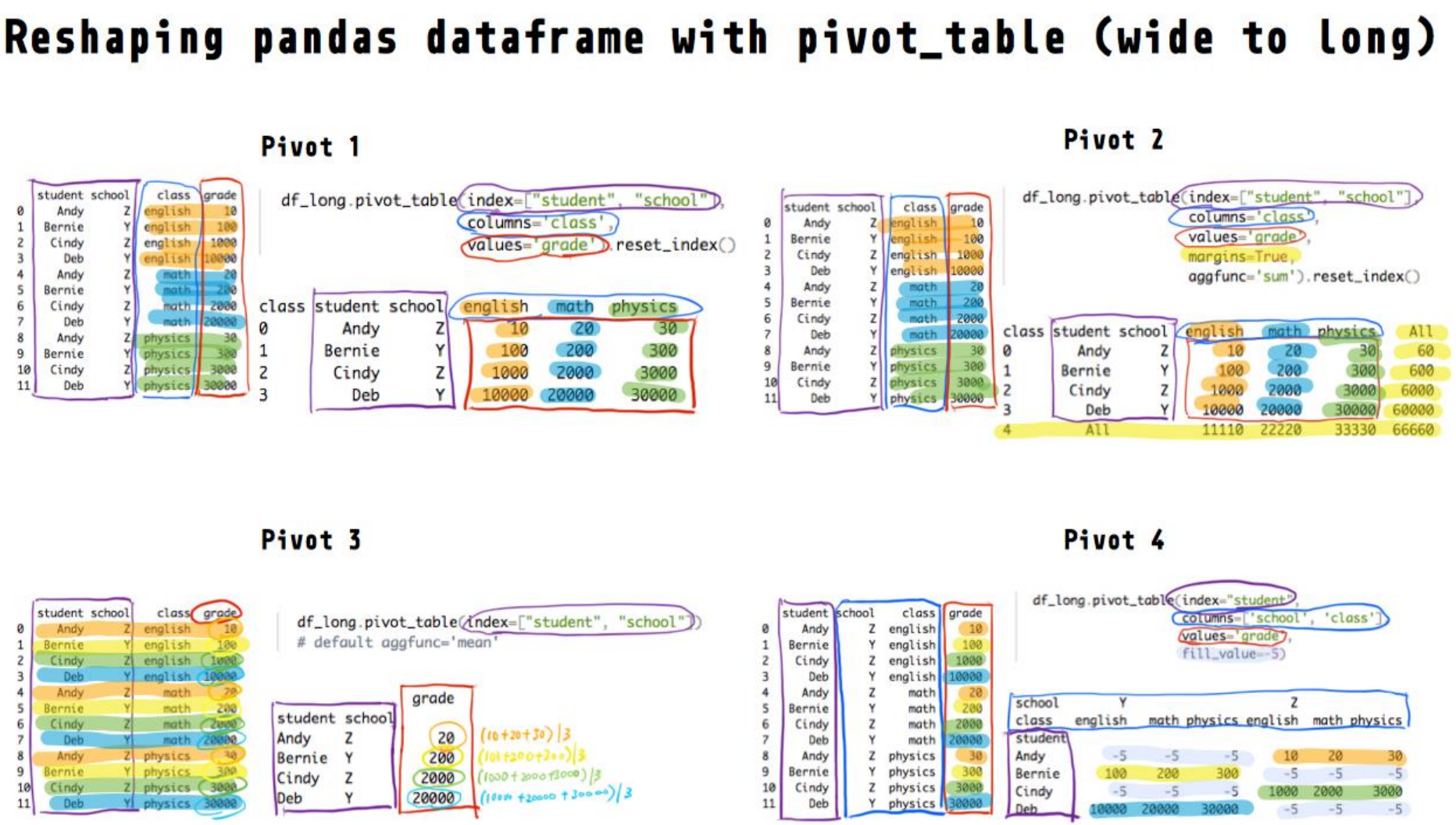


	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

		sum				
		Price				
	Product	CPU	Maintenance	Monitor	Software	All
Manager	Status					
Debra Henley	declined	70000	0	0	0	70000
	pending	40000	10000	0	0	50000
	presented	30000	0	0	20000	50000
	won	65000	0	0	0	65000
Fred Anderson	declined	65000	0	0	0	65000
	pending	0	5000	0	0	5000
	presented	30000	0	5000	10000	45000
	won	165000	7000	0	0	172000
All		465000	22000	5000	30000	522000

https://pbpython.com/pandas-pivot-table-explained.html

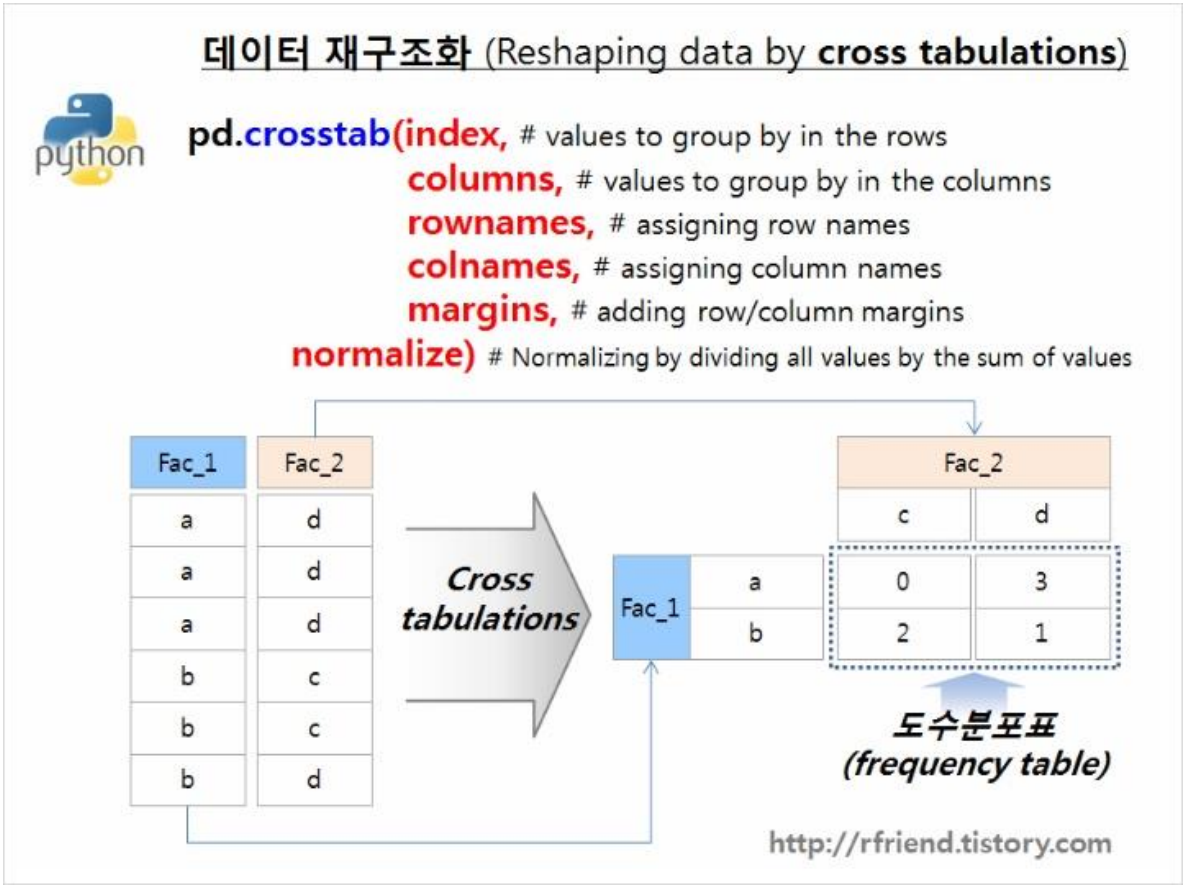
Pivot_table 요약



https://towardsdatascience.com/reshape-pandas-dataframe-with-pivot-table-in-python-tutorial-and-visualization-2248c2012a31

교차 일람표 개요

- **pandas.crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=None, margins=False, margins_name: str = 'All', dropna: bool = True, normalize=False)**
- **그룹의 빈도를 계산하기 위한 피벗테이블의 한 유형**
  - **crosstab의 처음 두 인자는 배열이나 시리즈 혹은 배열의 리스트**
  - 행, 열 요인 기준 별로 빈도를 세어서 도수분포표(frequency table), 교차표(contingency table)를 생성



교차 일람표

- **메소드 crosstab**
  - 첫 인자: index
  - 두 번째 인자: columns
  - 그룹 빈도수를 계산
- **국가 별 손잡이 수**

```
In [226]: data
```

Out[226]:

	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

```
In [233]: pd.crosstab(data.Nationality, data.Handedness, margins=True)
```

Out[233]:

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

```
In [232]: data.pivot_table('Sample', index='Nationality', columns='Handedness',
                           aggfunc=len, margins=True)
```

Out[232]:

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

팁 데이터의 교차 일람표

- **시간과 요일에 따른 흡연 여부의 빈도 수**
  - 다음 결과 동일
    - **crosstab()**
    - **메소드 pivot_table()**
      - **aggfunc=len**

In [249]:
```
tips
```

Out [249]:

	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
...	...	...	...	...	...	...	...
239	29.03	5.92	No	Sat	Dinner	3	0.203927
240	27.18	2.00	Yes	Sat	Dinner	2	0.073584
241	22.67	2.00	Yes	Sat	Dinner	2	0.088222
242	17.82	1.75	No	Sat	Dinner	2	0.098204
243	18.78	3.00	No	Thur	Dinner	2	0.159744

244 rows × 7 columns

In [228]:
```
pd.crosstab([tips.time, tips.day], tips.smoker, margins=True)
```

Out [228]:

	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

열 'smoker', 'day', 'time'은 제외하고 모두 가능

In [247]:
```
tips.pivot_table('size', index=['time', 'day'], columns='smoker',
                 aggfunc=len, margins=True, fill_value=0)
```

Out [247]:

	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