8장. 데이터 준비하기 : 조인 병합 변형

2017010715 허지혜

데이터 준비하기 : 조인 병합 변형

데이터 준비하기 : 조인, 병합, 변형

In [1]:
```
import numpy as np
import pandas as pd
pd.options.display.max_rows = 20
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4, suppress=True)
```

1. 계층적 색인

In [4]:
```
data = pd.Series(np.random.randn(9),
                 index=[['a', 'a', 'a', 'b', 'b', 'c', 'c', 'd', 'd'],
                        [1, 2, 3, 1, 3, 1, 2, 2, 3]])
data
```

Out[4]:
```
a  1    1.669025
   2   -0.438570
   3   -0.539741
b  1    0.476985
   3    3.248944
c  1   -1.021228
   2   -0.577087
d  2    0.124121
   3    0.302614
dtype: float64
```

In [5]:
```
data.index
```

Out[5]:
```
MultiIndex([('a', 1),
            ('a', 2),
            ('a', 3),
            ('b', 1),
            ('b', 3),
            ('c', 1),
            ('c', 2),
            ('d', 2),
            ('d', 3)],
           )
```

In [8]:
```
data['b']
```

Out[8]:
```
1    0.476985
3    3.248944
dtype: float64
```

In [7]:
```
data['b':'c']
```

Out[7]:
```
b  1    0.476985
   3    3.248944
c  1   -1.021228
   2   -0.577087
dtype: float64
```

In [9]:
```
data.loc[['b', 'd']]
```

Out[9]:
```
b  1    0.476985
   3    3.248944
d  2    0.124121
   3    0.302614
dtype: float64
```

In [10]:
```
#하위 계층 객체 선택
data.loc[:, 2]
```

Out[10]:
```
a   -0.438570
c   -0.577087
d    0.124121
dtype: float64
```

In [11]:
```
# 피벗테이블 생성 같은 그룹 기반의 작업
data.unstack()
```

Out[11]:

	1	2	3
a	1.669025	-0.438570	-0.539741
b	0.476985	NaN	3.248944
c	-1.021228	-0.577087	NaN
d	NaN	0.124121	0.302614

In [13]:
```
# unstack()의 반대 = stack()
data.unstack().stack()
```

Out[13]:
```
a  1    1.669025
   2   -0.438570
   3   -0.539741
b  1    0.476985
   3    3.248944
c  1   -1.021228
   2   -0.577087
d  2    0.124121
   3    0.302614
dtype: float64
```

In [14]:
```
frame = pd.DataFrame(np.arange(12).reshape((4, 3)),
                     index=[['a', 'a', 'b', 'b'], [1, 2, 1, 2]],
                     columns=[['Ohio', 'Ohio', 'Colorado'],
                              ['Green', 'Red', 'Green']])
frame
```

Out[14]:

		Ohio		Colorado
		Green	Red	Green
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

In [15]:
```
frame.index.names = ['key1', 'key2']
frame.columns.names = ['state', 'color'] # 색인 이름임 로우 라벨이랑 혼돈 ㄴㄴ
frame
```

Out[15]:

	state	Ohio		Colorado
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

In [16]:
```
frame['Ohio']
```

Out[16]:

	color	Green	Red
key1	key2		
a	1	0	1
	2	3	4
b	1	6	7
	2	9	10

8.1.1 계층 순서를 바꾸고 정렬

	state	Ohio		Colorado
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

In [18]:
```
# swaplevel = 넘겨받은 두 개의 계층번호나 이름이 뒤바뀐 새로운 객체를 반환한다.
# 단 데이터 변경은 안됨~!
frame.swaplevel('key1', 'key2')
```

Out[18]:

	state	Ohio		Colorado
	color	Green	Red	Green
key2	key1			
1	a	0	1	2
2	a	3	4	5
1	b	6	7	8
2	b	9	10	11

In [23]:
```
frame.sort_index(level=0) #단일 계층 데이터 정렬
```

Out[23]:

	state	Ohio		Colorado
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
	2	3	4	5
b	1	6	7	8
	2	9	10	11

In [28]:
```
frame.sort_index(level=1) #단일 계층 데이터 정렬
```

Out[28]:

	state	Ohio		Colorado
	color	Green	Red	Green
key1	key2			
a	1	0	1	2
b	1	6	7	8
a	2	3	4	5
b	2	9	10	11

In [26]:
```
frame.swaplevel(0, 1).sort_index(level=0)
```

Out[26]:

	state	Ohio		Colorado
	color	Green	Red	Green
key2	key1			
1	a	0	1	2
	b	6	7	8
2	a	3	4	5
	b	9	10	11

객체가 계층적 색인으로 상위 계층부터 사전적으로 정렬되어 있다면 데이터를 선택하는 성능이 훨씬 좋아진다.

8.1.2 계층별 요약 통계

In [31]:
```
frame.sum(level='key2')
```

Out[31]:

state	Ohio		Colorado
color	Green	Red	Green
key2			
1	6	8	10
2	12	14	16

In [32]:
```
# 행의 모든 열에 대해 동작
# green 은 두 개 니까 두 개의 합
frame.sum(level='color', axis=1)
```

Out[32]:

	color	Green	Red
key1	key2		
a	1	2	1
	2	8	4
b	1	14	7
	2	20	10

8.1.3 DataFrame 컬럼 사용

In [48]:
```
frame = pd.DataFrame({'a': range(7), 'b': range(7, 0, -1),
                      'c': ['one', 'one', 'one', 'two', 'two',
                            'two', 'two'],
                      'd': [0, 1, 2, 0, 1, 2, 3]})
frame
```

Out[48]:

	a	b	c	d
0	0	7	one	0
1	1	6	one	1
2	2	5	one	2
3	3	4	two	0
4	4	3	two	1
5	5	2	two	2
6	6	1	two	3

In [37]:
```
frame.set_index(['c', 'd'], drop=False)
```

Out[37]:

		a	b	c	d
c	**d**				
one	**0**	0	7	one	0
	1	1	6	one	1
	2	2	5	one	2
two	**0**	3	4	two	0
	1	4	3	two	1
	2	5	2	two	2
	3	6	1	two	3

In [36]:
```
# set_index 함수는 하나 이상의 컬럼을 색인으로 하는 새로운 DataFrame 생성
frame2 = frame.set_index(['c', 'd'])
frame2
```

Out[36]:

		a	b
c	**d**		
one	**0**	0	7
	1	1	6
	2	2	5
two	**0**	3	4
	1	4	3
	2	5	2
	3	6	1

In [38]:
```
#set_index와 반대되는 개념, 계층적 색인 단계가 컬럼으로 이동한다.
frame2.reset_index()
```

Out[38]:

	c	d	a	b
0	one	0	0	7
1	one	1	1	6
2	one	2	2	5
3	two	0	3	4
4	two	1	4	3
5	two	2	5	2
6	two	3	6	1

8.2 데이터 합치기

df1

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

df2

	key	data2
0	a	0
1	b	1
2	d	2

```
pd.merge(df1, df2)
```

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

```
# merge함수는 중복된 컬럼 이름을 키로 사용
pd.merge(df1, df2, on='key')
```

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

```
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
                    'data2': range(3)})
# 중복되는 이름이 없으면  따로 지정
pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

	lkey	data1	rkey	data2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0

```
# 외부 조인 = 양쪽 테이블에 존재하는 모든 키 조합을 사용한다.(합집합)
pd.merge(df1, df2, how='outer')
```

```
#왼쪽 조인 = 왼쪽 테이블에 존재하는 모든 키 조합을 사용한다.
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                    'data1': range(6)})
df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                    'data2': range(5)})
pd.merge(df1, df2, on='key', how='left')
```

	key
0	b
1	b
2	b
3	a
4	a
5	a
6	c
7	d

	key	data1	data2
0	b	0	1.0
1	b	0	3.0
2	b	1	1.0
3	b	1	3.0
4	a	2	0.0
5	a	2	2.0
6	c	3	NaN
7	a	4	0.0
8	a	4	2.0
9	b	5	1.0
10	b	5	3.0

```
#내부조인 = 양쪽 테이블 모두 존재하는 키 조합을 사용한다.(교집합)
pd.merge(df1, df2, how='inner')
```

	key	data1	data2
0	b	0	1
1	b	0	3
2	b	1	1
3	b	1	3
4	b	5	1
5	b	5	3
6	a	2	0
7	a	2	2
8	a	4	0
9	a	4	2

```
left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                     'key2': ['one', 'two', 'one'],
                     'lval': [1, 2, 3]})
right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                      'key2': ['one', 'one', 'one', 'two'],
                      'rval': [4, 5, 6, 7]})
# 여러 개의 키 병합시 컬럼 이름이 담긴 리스트를 넘기면 된다
pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

:

	key1	key2	lval	rval
0	foo	one	1.0	4.0
1	foo	one	1.0	5.0
2	foo	two	2.0	NaN
3	bar	one	3.0	6.0

```
pd.merge(left, right, on='key1')
```

	key1	key2_x	lval	key2_y	rval
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

```
pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
```

	key1	key2_left	lval	key2_right	rval
0	foo	one	1	one	4
1	foo	one	1	one	5
2	foo	two	2	one	4
3	foo	two	2	one	5
4	bar	one	3	one	6
5	bar	one	3	two	7

8.2.2 색인 병합하기

```
left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
                      'value': range(6)})
right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
left1
right1
pd.merge(left1, right1, left_on='key', right_index=True)
```

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0

```
pd.merge(left1, right1, left_on='key', right_index=True, how='outer')
```

	key	value	group_val
0	a	0	3.5
2	a	2	3.5
3	a	3	3.5
1	b	1	7.0
4	b	4	7.0
5	c	5	NaN

```
righth
```

		event1	event2
Nevada	2001	0	1
	2000	2	3
Ohio	2000	4	5
	2000	6	7
	2001	8	9
	2002	10	11

```
lefth
```

	key1	key2	data
0	Ohio	2000	0.0
1	Ohio	2001	1.0
2	Ohio	2002	2.0
3	Nevada	2001	3.0
4	Nevada	2002	4.0

```
pd.merge(lefth, righth, left_on=['k
```

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4	5
0	Ohio	2000	0.0	6	7
1	Ohio	2001	1.0	8	9
2	Ohio	2002	2.0	10	11
3	Nevada	2001	3.0	0	1

```
#중복되는 색인값을 다룰 때는 how='outer' 옵션을 사용해야한다.
pd.merge(lefth, righth, left_on=['key1', 'key2'],
         right_index=True, how='outer')
```

	key1	key2	data	event1	event2
0	Ohio	2000	0.0	4.0	5.0
0	Ohio	2000	0.0	6.0	7.0
1	Ohio	2001	1.0	8.0	9.0
2	Ohio	2002	2.0	10.0	11.0
3	Nevada	2001	3.0	0.0	1.0
4	Nevada	2002	4.0	NaN	NaN
4	Nevada	2000	NaN	2.0	3.0

```
left2 = pd.DataFrame([[1., 2.], [3., 4.], [5., 6.]],
                     index=['a', 'c', 'e'],
                     columns=['Ohio', 'Nevada'])
right2 = pd.DataFrame([[7., 8.], [9., 10.], [11., 12.], [13, 14]],
                      index=['b', 'c', 'd', 'e'],
                      columns=['Missouri', 'Alabama'])
left2
right2
pd.merge(left2, right2, how='outer', left_index=True, right_index=True)
```

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

```
#색인으로 병합할 때 join 메서드를 사용하면 편리함.
left2.join(right2, how='outer')
```

	Ohio	Nevada	Missouri	Alabama
a	1.0	2.0	NaN	NaN
b	NaN	NaN	7.0	8.0
c	3.0	4.0	9.0	10.0
d	NaN	NaN	11.0	12.0
e	5.0	6.0	13.0	14.0

8.2.2 축 따라 이어붙이기

```
arr = np.arange(12).reshape((3, 4))
arr
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

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

```
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

```
s1 = pd.Series([0, 1], index=['a', 'b'])
s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
s3 = pd.Series([5, 6], index=['f', 'g'])
```

```
pd.concat([s1, s2, s3])
```

```
a    0
b    1
c    2
d    3
e    4
f    5
g    6
dtype: int64
```

```
s4 = pd.concat([s1, s3])
s4
```

```
a    0
b    1
f    5
g    6
dtype: int64
```

```
pd.concat([s1, s2, s3], axis=1)
```

	0	1	2
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

```
pd.concat([s1, s4], axis=1, join='inner')
```

	0	1
a	0	0
b	1	1

```
pd.concat([s1, s4], axis=1)
```

	0	1
a	0.0	0
b	1.0	1
f	NaN	5
g	NaN	6

```
result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])
result
```

```
one    a    0
       b    1
two    a    0
       b    1
three  f    5
       g    6
dtype: int64
```

```
result.unstack()
```

	a	b	f	g
one	0.0	1.0	NaN	NaN
two	0.0	1.0	NaN	NaN
three	NaN	NaN	5.0	6.0

```
#series를 axis=1 으로 병합할 경우 keys는 DataFrame의 컬럼 제목이 된다.
pd.concat([s1, s2, s3], axis=1, keys=['one', 'two', 'three'])
```

	one	two	three
a	0.0	NaN	NaN
b	1.0	NaN	NaN
c	NaN	2.0	NaN
d	NaN	3.0	NaN
e	NaN	4.0	NaN
f	NaN	NaN	5.0
g	NaN	NaN	6.0

df1

	one	two
a	0	1
b	2	3
c	4	5

df2

	key	data2
0	a	0
1	b	1
2	a	2
3	b	3
4	d	4

```
pd.concat([df1, df2], axis=1, keys=['level1', 'level2'])
```

	level1		level2	
	key	data1	key	data2
0	b	0	a	0.0
1	b	1	b	1.0
2	a	2	a	2.0
3	c	3	b	3.0
4	a	4	d	4.0
5	b	5	NaN	NaN

```
pd.concat({'level1': df1, 'level2': df2}, axis=1)
```

	level1		level2	
	key	data1	key	data2
0	b	0	a	0.0
1	b	1	b	1.0
2	a	2	a	2.0
3	c	3	b	3.0
4	a	4	d	4.0
5	b	5	NaN	NaN

```
pd.concat([df1, df2], axis=1, keys=['level1', 'level2'],
          names=['upper', 'lower'])
```

upper	level1		level2	
lower	key	data1	key	data2
0	b	0	a	0.0
1	b	1	b	1.0
2	a	2	a	2.0
3	c	3	b	3.0
4	a	4	d	4.0
5	b	5	NaN	NaN

```
df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])
df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
df1
```

	a	b	c	d
0	0.286350	0.377984	-0.753887	0.331286
1	1.349742	0.069877	0.246674	-0.011862
2	1.004812	1.327195	-0.919262	-1.549106

```
df2
```

	b	d	a
0	0.022185	0.758363	-0.660524
1	0.862580	-0.010032	0.050009

```
#DataFrame의 로우 색인이 분석에 필요한 데이터를 포함하고 있지 않은 경우
#ignore_index=True 옵션을 주면 된다.
pd.concat([df1, df2], ignore_index=True)
```

	a	b	c	d
0	0.286350	0.377984	-0.753887	0.331286
1	1.349742	0.069877	0.246674	-0.011862
2	1.004812	1.327195	-0.919262	-1.549106
3	-0.660524	0.022185	NaN	0.758363
4	0.050009	0.862580	NaN	-0.010032

8.2.4 겹치는 데이터 합치기

```
a = pd.Series([np.nan, 2.5, np.nan, 3.5, 4.5, np.nan],
              index=['f', 'e', 'd', 'c', 'b', 'a'])
b = pd.Series(np.arange(len(a), dtype=np.float64),
              index=['f', 'e', 'd', 'c', 'b', 'a'])
b[-1] = np.nan
```

```
a
```

```
f    NaN
e    2.5
d    NaN
c    3.5
b    4.5
a    NaN
dtype: float64
```

```
b
```

```
f    0.0
e    1.0
d    2.0
c    3.0
b    4.0
a    NaN
dtype: float64
```

```
np.where(pd.isnull(a), b, a)
```

```
array([0. , 2.5, 2. , 3.5, 4.5, nan])
```

```
#combine_first 메서드는 위와 동일한 연산을 제공하며 정렬 기능까지 제공
b[:-2].combine_first(a[2:])
```

```
a    NaN
b    4.5
c    3.0
d    2.0
e    1.0
f    0.0
dtype: float64
```

```
df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],
                    'b': [np.nan, 2., np.nan, 6.],
                    'c': range(2, 18, 4)})
df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],
                    'b': [np.nan, 3., 4., 6., 8.]})
df1
```

	a	b	c
0	1.0	NaN	2
1	NaN	2.0	6
2	5.0	NaN	10
3	NaN	6.0	14

```
df2
```

	a	b
0	5.0	NaN
1	4.0	3.0
2	NaN	4.0
3	3.0	6.0
4	7.0	8.0

```
df1.combine_first(df2)
```

	a	b	c
0	1.0	NaN	2.0
1	4.0	2.0	6.0
2	5.0	4.0	10.0
3	3.0	6.0	14.0
4	7.0	8.0	NaN

8.3 재형성과 피벗

표 형식의 데이터를 재배치하는 기본 연산이 존재 ==> 재형성 또는 피벗 연산

stack
데이터의 컬럼을 로우로 회전

unstack
로우를 컬럼으로 피벗시킨다.

```
data = pd.DataFrame(np.arange(6).reshape((2, 3)),
                    index=pd.Index(['Ohio', 'Colorado'], name='state'),
                    columns=pd.Index(['one', 'two', 'three'],
                    name='number'))
data
```

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

```
result = data.stack()
result
```

```
state     number
Ohio      one       0
          two       1
          three     2
Colorado  one       3
          two       4
          three     5
dtype: int32
```

```
result.unstack()
```

number	one	two	three
state			
Ohio	0	1	2
Colorado	3	4	5

```
result.unstack(0)
```

state	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

```
result.unstack('state')
```

state	Ohio	Colorado
number		
one	0	3
two	1	4
three	2	5

```
s1 = pd.Series([0, 1, 2, 3], index=['a', 'b', 'c', 'd'])
s2 = pd.Series([4, 5, 6], index=['c', 'd', 'e'])
data2 = pd.concat([s1, s2], keys=['one', 'two'])
data2
```

```
one  a    0
     b    1
     c    2
     d    3
two  c    4
     d    5
     e    6
dtype: int64
```

```
data2.unstack()
```

	a	b	c	d	e
one	0.0	1.0	2.0	3.0	NaN
two	NaN	NaN	4.0	5.0	6.0

```
data2.unstack().stack()
```

```
one  a    0.0
     b    1.0
     c    2.0
     d    3.0
two  c    4.0
     d    5.0
     e    6.0
dtype: float64
```

```
data2.unstack().stack(dropna=False)
```

```
one  a    0.0
     b    1.0
     c    2.0
     d    3.0
     e    NaN
two  a    NaN
     b    NaN
     c    4.0
     d    5.0
     e    6.0
dtype: float64
```

```
df
```

	side	left	right
state	number		
Ohio	one	0	5
	two	1	6
	three	2	7
Colorado	one	3	8
	two	4	9
	three	5	10

```
df.unstack('state')
```

side	left		right	
state	Ohio	Colorado	Ohio	Colorado
number				
one	0	3	5	8
two	1	4	6	9
three	2	5	7	10

```
df.unstack('state').stack('side')
```

	state	Colorado	Ohio
number	side		
one	left	3	0
	right	8	5
two	left	4	1
	right	9	6
three	left	5	2
	right	10	7

8.3.2 긴 형식 => 넓은 형식 피벗

```
data = pd.read_csv('C:/Users/HOME/Desktop/수DA쟁이/Python_for_data_analysis/8장/m
```

```
data.head()
```

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unen
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5

```
# 시간 간격을 나타내기 위한 자료형 PeriodIndex
periods = pd.PeriodIndex(year=data.year, quarter=data.quarter, name='data')
```

```
columns = pd.Index(['realgdp','infl','unemp'],name='item')
```

```
data = data.reindex(columns=columns)
```

```
data.index = periods.to_timestamp('D','end')
```

```
ldata = data.stack().reset_index().rename(columns={0:'value'})
```

```
ldata[:10]
```

	data	item	value
0	1959-03-31 23:59:59.999999999	realgdp	2710.349
1	1959-03-31 23:59:59.999999999	unemp	5.800
2	1959-06-30 23:59:59.999999999	realgdp	2778.801
3	1959-06-30 23:59:59.999999999	unemp	5.100
4	1959-09-30 23:59:59.999999999	realgdp	2775.488
5	1959-09-30 23:59:59.999999999	unemp	5.300
6	1959-12-31 23:59:59.999999999	realgdp	2785.204
7	1959-12-31 23:59:59.999999999	unemp	5.600
8	1960-03-31 23:59:59.999999999	realgdp	2847.699
9	1960-03-31 23:59:59.999999999	unemp	5.200

```
pivoted = ldata.pivot('data', 'item', 'value')
```

```
ldata['value2'] = np.random.randn(len(ldata))
ldata[:10]
```

	data	item	value	value2
0	1959-03-31 23:59:59.999999999	realgdp	2710.349	0.567106
1	1959-03-31 23:59:59.999999999	unemp	5.800	0.081577
2	1959-06-30 23:59:59.999999999	realgdp	2778.801	-0.302335
3	1959-06-30 23:59:59.999999999	unemp	5.100	-0.726916
4	1959-09-30 23:59:59.999999999	realgdp	2775.488	0.180335
5	1959-09-30 23:59:59.999999999	unemp	5.300	-0.520209
6	1959-12-31 23:59:59.999999999	realgdp	27	
7	1959-12-31 23:59:59.999999999	unemp		
8	1960-03-31 23:59:59.999999999	realgdp	28	
9	1960-03-31 23:59:59.999999999	unemp		

```
pivoted = ldata.pivot('data', 'item')
pivoted[:5]
```

	value		value2	
item	**realgdp**	**unemp**	**realgdp**	**unemp**
data				
1959-03-31 23:59:59.999999999	2710.349	5.8	0.567106	0.081577
1959-06-30 23:59:59.999999999	2778.801	5.1	-0.302335	-0.726916
1959-09-30 23:59:59.999999999	2775.488	5.3	0.180335	-0.520209
1959-12-31 23:59:59.999999999	2785.204	5.6	0.398092	-0.916935
1960-03-31 23:59:59.999999999	2847.699	5.2	-0.082650	-1.939691

:
```
unstacked = ldata.set_index(['data', 'item']).unstack('item')
unstacked[:7]
```

:

	value		value2	
item	realgdp	unemp	realgdp	unemp
data				
1959-03-31 23:59:59.999999999	2710.349	5.8	0.567106	0.081577
1959-06-30 23:59:59.999999999	2778.801	5.1	-0.302335	-0.726916
1959-09-30 23:59:59.999999999	2775.488	5.3	0.180335	-0.520209
1959-12-31 23:59:59.999999999	2785.204	5.6	0.398092	-0.916935
1960-03-31 23:59:59.999999999	2847.699	5.2	-0.082650	-1.939691
1960-06-30 23:59:59.999999999	2834.390	5.2	1.407994	1.512406
1960-09-30 23:59:59.999999999	2839.022	5.6	0.526493	-0.266931

8.3.3 넓은 형식 => 긴 형식 피벗

pivot와 반대되는 연

```
df = pd.DataFram
```

```
df
```

	key	A	B	C
0	foo	1	4	7
1	bar	2	5	8
2	baz	3	6	9

```
# pivot를 사용해서 원래 모양으로 되돌릴 수 있다.                ])
reshaped = melted.pivot('key', 'variable', 'value')
reshaped
```

variable	A	B	C
key			
bar	2	5	8
baz	3	6	9
foo	1	4	7

```
#pivot의 결과는 로우 라벨로 사용하던 컬럼에서 색인을 생성하므로
#reset_index를 이용해서 데이터를 다시 컬럼으로 돌려놓음
reshaped.reset_index()
```

variable	key	A	B	C
0	bar	2	5	8
1	baz	3	6	9
2	foo	1	4	7

```
pd.melt(df, id_vars=['key'], value_vars=['A', 'B'])
```

	key	variable	value
0	foo	A	1
1	bar	A	2
2	baz	A	3
3	foo	B	4
4	bar	B	5
5	baz	B	6

```
pd.melt(df, value_vars=['A', 'B', 'C'])
```

	variable	value
0	A	1
1	A	2
2	A	3
3	B	4
4	B	5
5	B	6
6	C	7
7	C	8
8	C	9

```
pd.melt(df, value_vars=['key', 'A', 'B'])
```

	variable	value
0	key	foo
1	key	bar
2	key	baz
3	A	1
4	A	2
5	A	3
6	B	4
7	B	5
8	B	6

끝~!