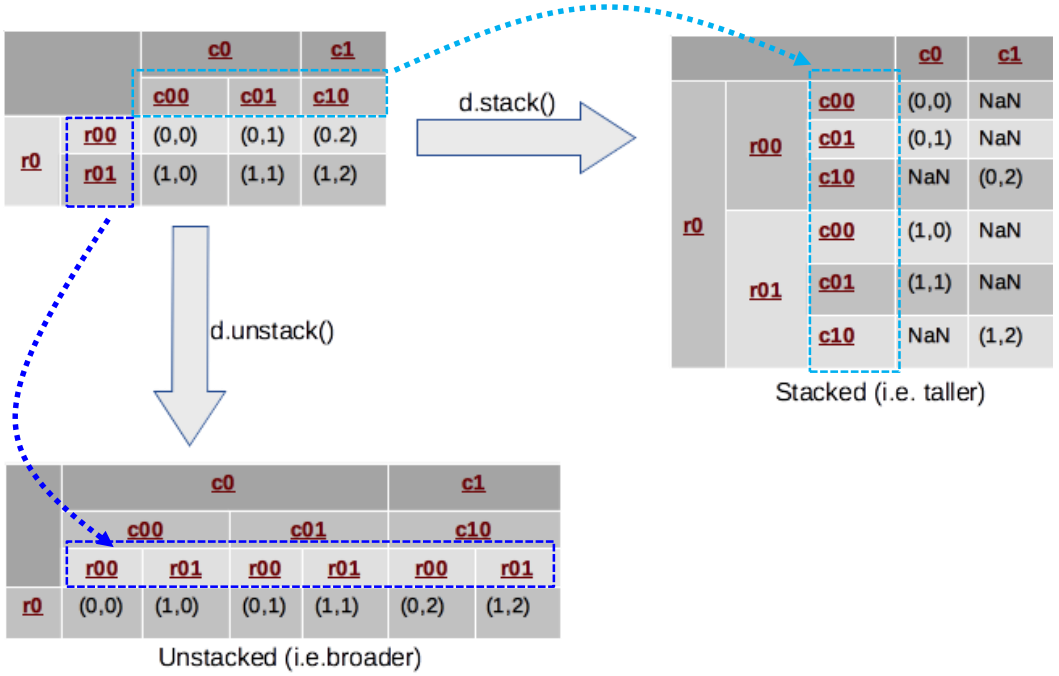
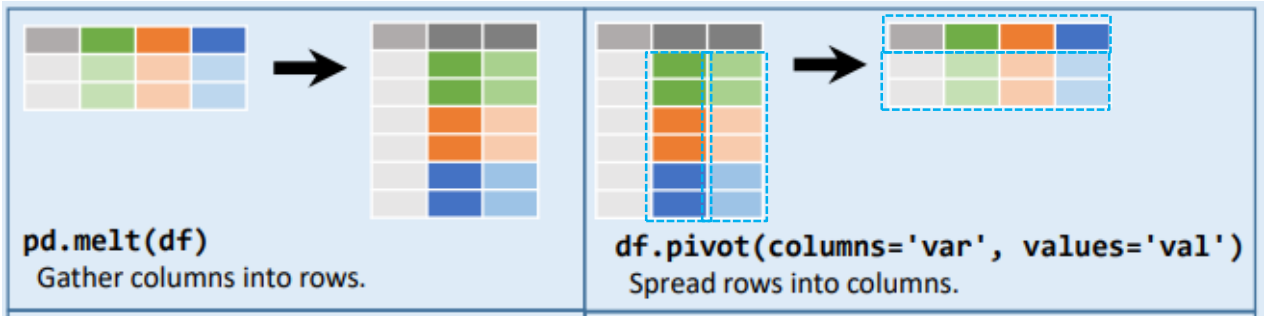
파이썬 라이브러리를 활용한 데이터 분석

8장 데이터 준비하기: 조인, 병합, 변형

8.3 재형성과 피벗 p334

- **계층적 색인으로 재형성**
  - index와 columns 사이의 이동
  - stack
    - **index <= columns**
  - unstack
    - **index => columns**
- **긴 형식에서 넓은 형식으로**
  - pivot
- **넓은 형식에서 긴 형식으로**
  - melt





8장 데이터 준비하기: 조인, 병합, 변형

stack unstack

stack과 unstack의 사전적 의미

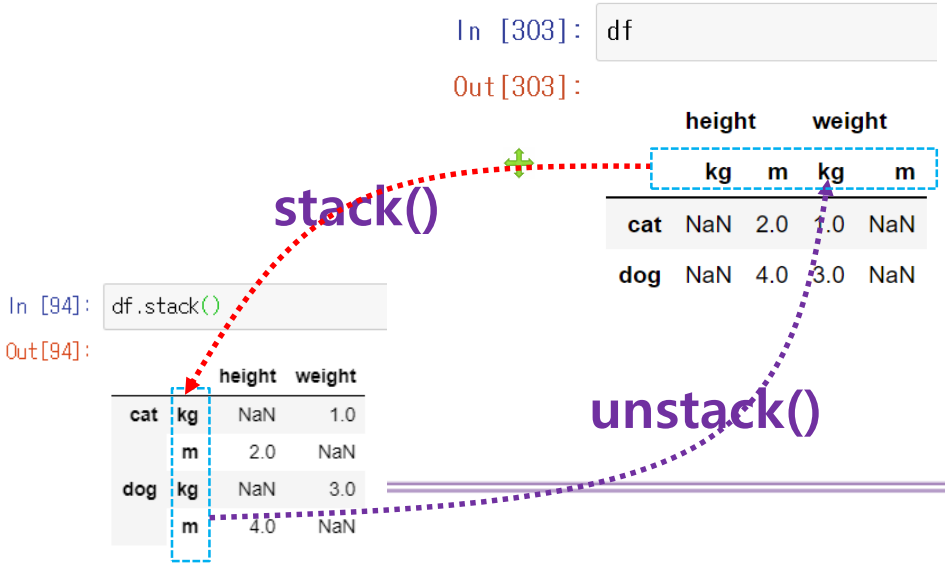
- **stack을 영어사전에서 찾아보면 뜻**
  - stack[stæk]
    - **~ (sth) (up) (깔끔하게 정돈하여) 쌓다[포개다]; 쌓이다, 포개지다**
    - **~ sth (with sth) (어떤 곳에 물건을 쌓아서) 채우다**
- **stack**
  - (위에서 아래로 길게, 높게) 쌓는 것이면
- **unstack**
  - 옆으로 늘어 놓는 것(왼쪽에서 오른쪽으로 넓게)라고 생각

- **용어 정리**
  - index == 인덱스 == 행 색인(인덱스)
  - columns == 열 == 열 색인(인덱스)

출처: https://rfriend.tistory.com/276 [R, Python 분석과 프로그래밍의 친구 (by R Friend)]

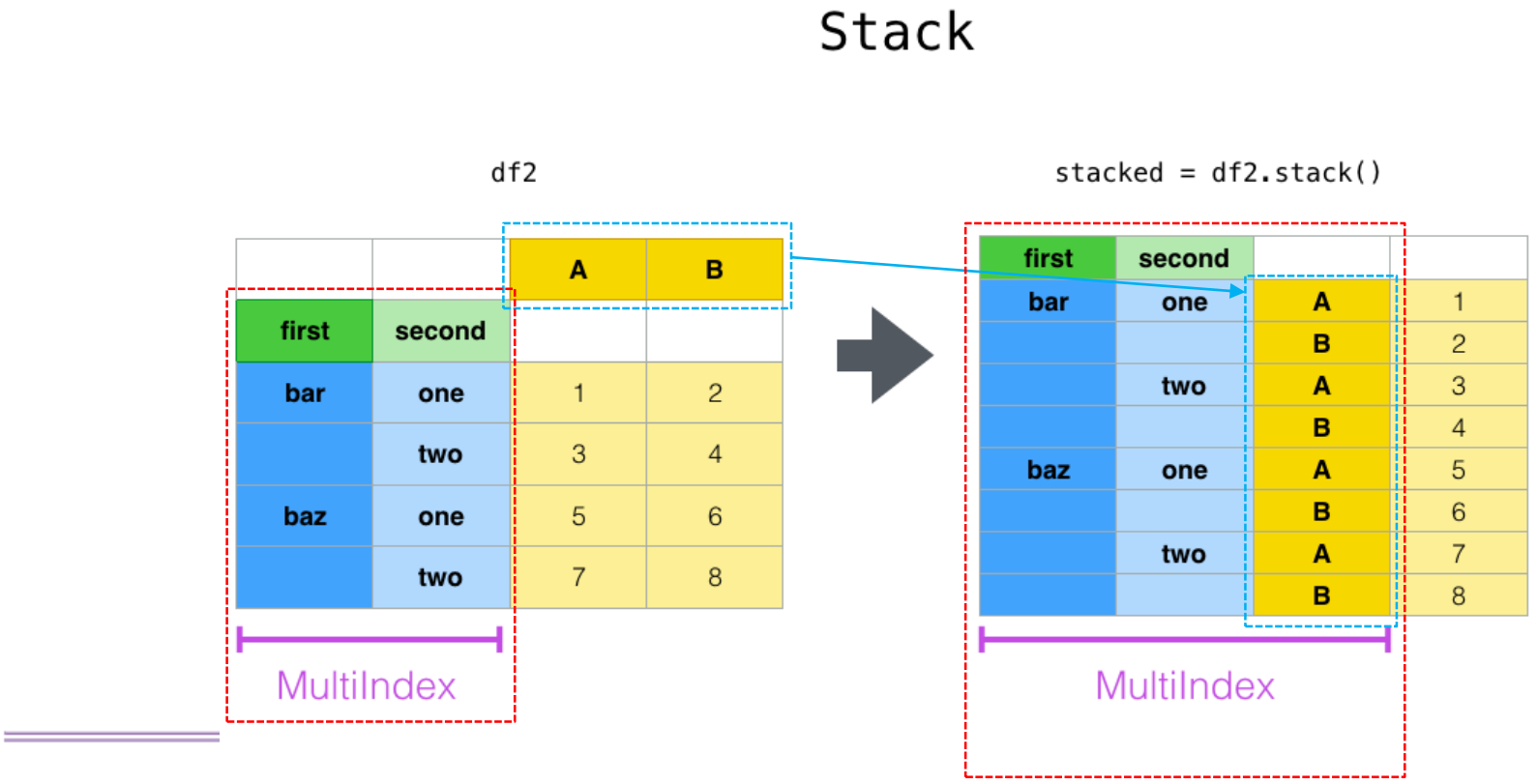
stack과 unstack

- **열 인덱스를 행 인덱스로 바꾸거나 반대로 행 인덱스를 열 인덱스로 바꾸는 작업**
  - stack(): 데이터의 칼럼을 로우로 회전
    - **열 인덱스 -> 행 인덱스로 변환**
    - **열 인덱스가 반시계 방향으로 90도 회전한 것과 비슷한 모양**
  - unstack(): 로우를 칼럼으로 회전
    - **행 인덱스 -> 열 인덱스로 변환**
    - **마찬가지로 실행하면 행 인덱스가 반시계 방향으로 90도 회전한 것과 비슷**
  - 인덱스를 지정 방법
    - **문자열 이름과 순서를 표시하는 숫자 인덱스를 모두 사용 가능**



Reshaping by stacking

- **stack: 열 레이블 => 행 인덱스로 이동**
  - "pivot" a level of the (possibly hierarchical) column labels,
  - returning a DataFrame with an index with a new inner-most level of row labels.
    - **(계층적) 열 레이블 수준을 "피벗"(회전)하여 가장 안쪽의 새로운 행 레이블의 인덱스 형태의 DataFrame을 반환**



스택: 열 레이블 => 행의 인덱스로 이동

- **열의 레이블이 하나 줄어 행의 인덱스로 이동**
  - "rotates" or pivots from the columns in the data to the rows

```
In [120]: data = pd.DataFrame(np.arange(6).reshape((2, 3)),
   .....:                     index=pd.Index(['Ohio', 'Colorado'], name='state'),
   .....:                     columns=pd.Index(['one', 'two', 'three'],
   .....:                     name='number'))

In [121]: data
Out[121]:
number    one  two  three
state
Ohio        0    1      2
Colorado    3    4      5
```

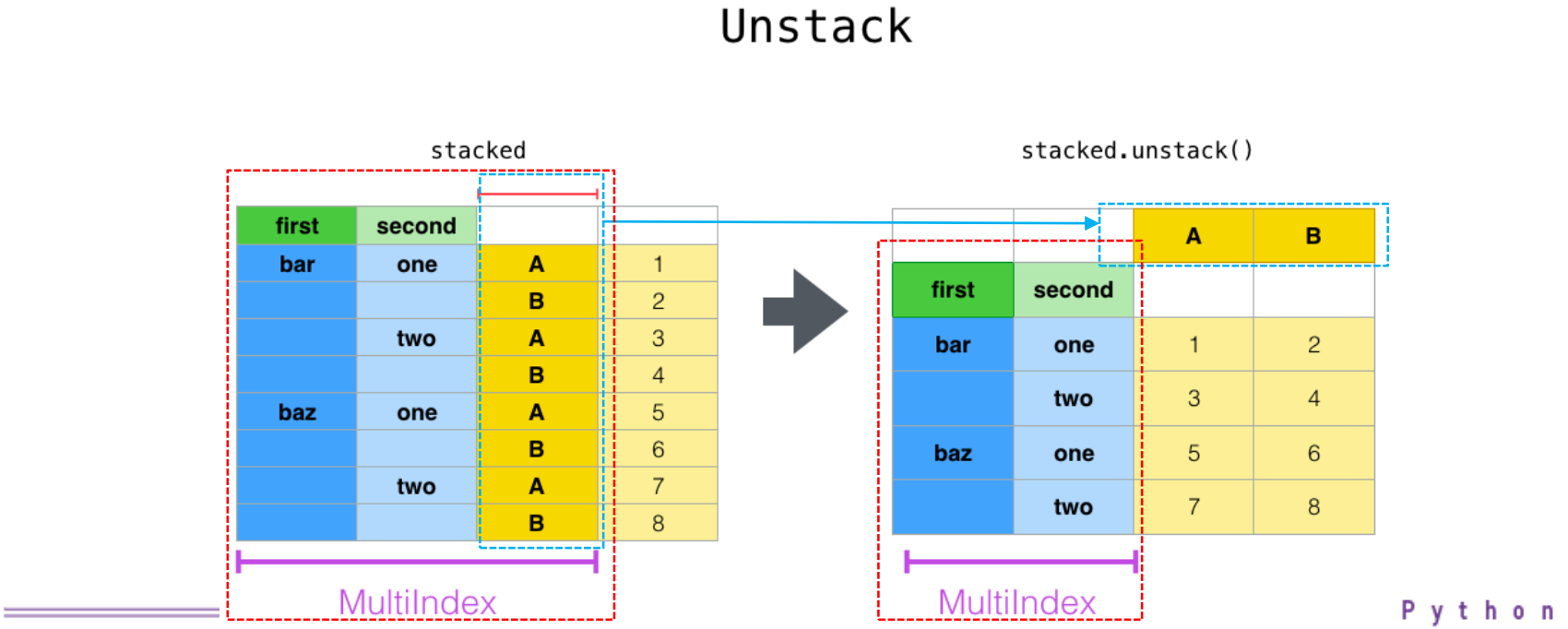
Using the stack method on this data pivots the columns into the rows, producing a Series:

```
In [122]: result = data.stack()

In [123]: result
Out[123]:
state     number
Ohio      one       0
          two       1
          three     2
Colorado  one       3
          two       4
          three     5
dtype: int64
```

Reshaping by unstacking

- **unstack: 행 인덱스 => 열 레이블로 이동**
  - (inverse operation of stack) "pivot" a level of the (possibly hierarchical) row index to the column axis,
  - producing a reshaped DataFrame with a new inner-most level of column labels.
    - **unstack: (stack 역연산) (가급적 계층적) 행 인덱스의 레벨을 컬럼 축으로 "피벗"(회전)해 새로운 가장 안쪽 레벨의 컬럼 레이블로 재구성된 DataFrame을 반환**



unstack(색인 첨자 번호)

- 다중 색인에서 첨자로 색인을 지정

Unstack(0)

stacked

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

0 1 2

stacked.unstack(0)
or
stacked.unstack('first')

	first	bar	baz
second			
one	A	1	5
	B	2	6
two	A	3	7
	B	4	8

MultiIndex

Unstack(1)

stacked

first	second		
bar	one	A	1
		B	2
	two	A	3
		B	4
baz	one	A	5
		B	6
	two	A	7
		B	8

MultiIndex

stacked.unstack(1)
or
stacked.unstack('second')

	second	one	two
first			
bar	A	1	3
	B	2	4
baz	A	5	7
	B	6	8

MultiIndex

unstack p336

- **계층적 색인의 시리즈에서 다시 DataFrame을 반환(가로로 넓어짐)**
  - 가장 안쪽부터(가장 큰 첨자 숫자) 열로 지정

```
In [123]: result
Out[123]:
state     number
Ohio      one      0
          two      1
          three    2
Colorado  one      3
          two      4
          three    5
dtype: int64
```

```
In [124]: result.unstack()
Out[124]:
number    one  two  three
state
Ohio        0    1      2
Colorado    3    4      5
```

  - 레벨 숫자나 이름으로 단계를 지정 가능

0 1

```
In [125]: result.unstack(0)
Out[125]:
state   Ohio  Colorado
number
one        0         3
two        1         4
three      2         5

In [126]: result.unstack('state')
Out[126]:
state   Ohio  Colorado
number
one        0         3
two        1         4
three      2         5
```

unstack, stack

- **unstack: 지정한 인덱스 state를 열로 지정**

```
In [135]: df = pd.DataFrame({'left': result, 'right': result + 5},
   .....:                   columns=pd.Index(['left', 'right'], name='side'))

In [136]: df
Out[136]:
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

In [137]: df.unstack('state')
Out[137]:
side   left          right
state  Ohio Colorado  Ohio Colorado
number
one       0        3     5        8
two       1        4     6        9
three     2        5     7       10

In [138]: df.unstack('state').stack('side')
Out[138]:
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
```


- **stack: 지정한 열 side를 인덱스로**

색인과 칼럼이 모두 계층적 색인

- **행과 열이 모두 2층인 계층적 색인**

In [8]:

```
np.random.seed(0)
df4 = pd.DataFrame(np.round(np.random.randn(6, 4), 2),
                   columns=[["A", "A", "B", "B"],
                            ["C", "D", "C", "D"]],
                   index=[["M", "M", "M", "F", "F", "F"],
                          ["id_" + str(i + 1) for i in range(3)] * 2])
df4.columns.names = ["Cidx1", "Cidx2"]
df4.index.names = ["Ridx1", "Ridx2"]
df4
```

	Cidx1	A		B	
	Cidx2	C	D	C	D
Ridx1	Ridx2				
M	id_1	1.76	0.40	0.98	2.24
	id_2	1.87	-0.98	0.95	-0.15
	id_3	-0.10	0.41	0.14	1.45
F	id_1	0.76	0.12	0.44	0.33
	id_2	1.49	-0.21	0.31	-0.85
	id_3	-2.55	0.65	0.86	-0.74

DataFrame.stack('열이름')

- 열 => 색인으로

	Cidx1	A		B	
	Cidx2	C	D	C	D
Ridx1	Ridx2				
M	id_1	1.76	0.40	0.98	2.24
	id_2	1.87	-0.98	0.95	-0.15
	id_3	-0.10	0.41	0.14	1.45
F	id_1	0.76	0.12	0.44	0.33
	id_2	1.49	-0.21	0.31	-0.85
	id_3	-2.55	0.65	0.86	-0.74

```
In [9]:
```

```
df4.stack("Cidx1")
```

		Cidx2	C	D
Ridx1	Ridx2	Cidx1		
M	id_1	A	1.76	0.40
		B	0.98	2.24
	id_2	A	1.87	-0.98
		B	0.95	-0.15
	id_3	A	-0.10	0.41
		B	0.14	1.45
F	id_1	A	0.76	0.12
		B	0.44	0.33
	id_2	A	1.49	-0.21
		B	0.31	-0.85
	id_3	A	-2.55	0.65
		B	0.86	-0.74

DataFrame.stack(열첨자)

- 열 => 색인으로

	Cidx1	A		B	
	Cidx2	C	D	C	D
Ridx1	Ridx2				
M	id_1	1.76	0.40	0.98	2.24
	id_2	1.87	-0.98	0.95	-0.15
	id_3	-0.10	0.41	0.14	1.45
F	id_1	0.76	0.12	0.44	0.33
	id_2	1.49	-0.21	0.31	-0.85
	id_3	-2.55	0.65	0.86	-0.74

```
In [10]:
```

```
df4.stack(1)
```

		Cidx1	A	B
Ridx1	Ridx2	Cidx2		
M	id_1	C	1.76	0.98
		D	0.40	2.24
	id_2	C	1.87	0.95
		D	-0.98	-0.15
	id_3	C	-0.10	0.14
		D	0.41	1.45
F	id_1	C	0.76	0.44
		D	0.12	0.33
	id_2	C	1.49	0.31
		D	-0.21	-0.85
	id_3	C	-2.55	0.86
		D	0.65	-0.74

DataFrame.unstack('색인이름')

- 색인 => 열로

```
In [11]:
df4.unstack("Ridx2")
```

Cidx1	A						B					
Cidx2	C			D			C			D		
Ridx2	id_1	id_2	id_3	id_1	id_2	id_3	id_1	id_2	id_3	id_1	id_2	id_3
Ridx1												
F	0.76	1.49	-2.55	0.12	-0.21	0.65	0.44	0.31	0.86	0.33	-0.85	-0.74
M	1.76	1.87	-0.10	0.40	-0.98	0.41	0.98	0.95	0.14	2.24	-0.15	1.45

	Cidx1	A		B	
	Cidx2	C	D	C	D
Ridx1	Ridx2				
M	id_1	1.76	0.40	0.98	2.24
	id_2	1.87	-0.98	0.95	-0.15
	id_3	-0.10	0.41	0.14	1.45
F	id_1	0.76	0.12	0.44	0.33
	id_2	1.49	-0.21	0.31	-0.85
	id_3	-2.55	0.65	0.86	-0.74

DataFrame.unstack(열첨자)

- 색인 => 열로

```
In [12]:
    df4.unstack(0)
```

	Cidx1	A		B	
	Cidx2	C	D	C	D
Ridx1	Ridx2				
M	id_1	1.76	0.40	0.98	2.24
	id_2	1.87	-0.98	0.95	-0.15
	id_3	-0.10	0.41	0.14	1.45
F	id_1	0.76	0.12	0.44	0.33
	id_2	1.49	-0.21	0.31	-0.85
	id_3	-2.55	0.65	0.86	-0.74

Cidx1	A				B			
Cidx2	C		D		C		D	
Ridx1	F	M	F	M	F	M	F	M
Ridx2								
id_1	0.76	1.76	0.12	0.40	0.44	0.98	0.33	2.24
id_2	1.49	1.87	-0.21	-0.98	0.31	0.95	-0.85	-0.15
id_3	-2.55	-0.10	0.65	0.41	0.86	0.14	-0.74	1.45

요약 정리 stack unstack

Stack / Unstack

```
>>> stacked = df5.stack()
>>> stacked.unstack()
```

Pivot a level of column labels
Pivot a level of index labels

		0	1
1	5	0.233482	0.390959
2	4	0.184713	0.237102
3	3	0.433522	0.429401

Unstacked

1	5	0	0.233482
		1	0.390959
2	4	0	0.184713
		1	0.237102
3	3	0	0.433522
		1	0.429401

Stacked

데이터 재구조화 (Reshaping data by stack, unstack)

pd.DataFrame.stack() , pd.DataFrame.unstack()

Stacked (long)

index		
1	Col_1	10
1	Col_1	20
2	Col_2	30
2	Col_2	40

unstack / *stack*

Un-stacked (wide)

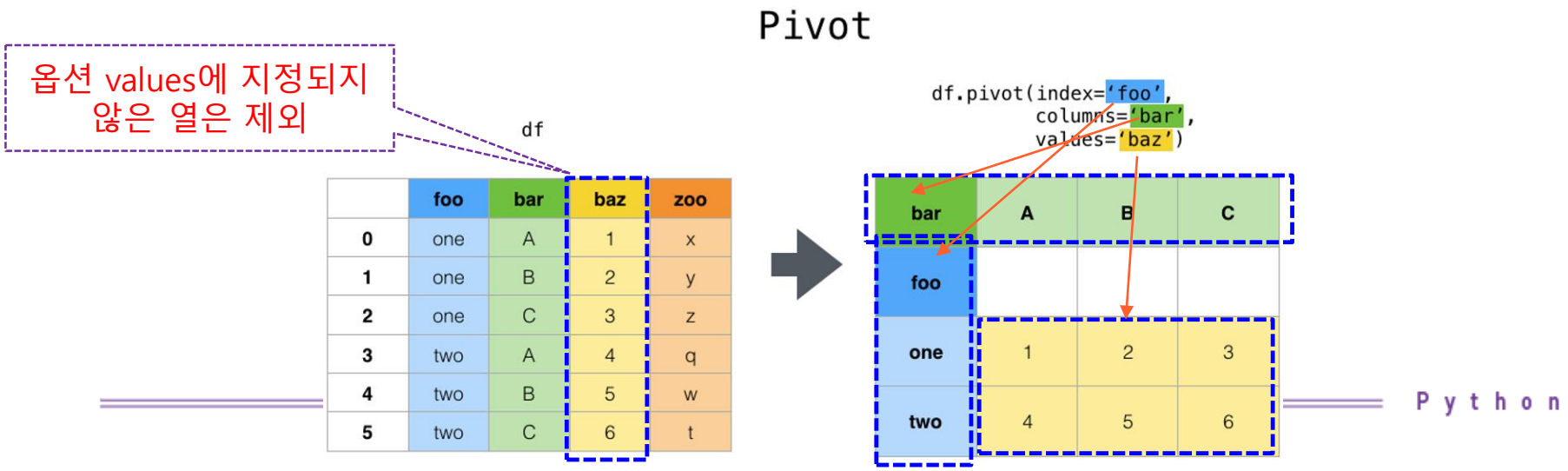
index	Col_1	Col_2
1	10	30
2	20	40

8장 데이터 준비하기: 조인, 병합, 변형

pivot melt

피봇 개요

- **피봇 테이블(pivot table)**
  - 데이터 열 중에서 두 개의 열을 각각 (행) 인덱스, 열 인덱스로 사용
    - **맞는 데이터를 저장하여 펼쳐놓은 것**
- **df1.pivot(index, columns, values)**
  - index, columns: 각각 (행) 인덱스, 열 인덱스로 사용할 이름
  - values: 데이터로 사용할 열 이름을 지정
    - **values가 아예 없으면 행과 열을 제외한 모든 열을 value로**
    - **값 대입 방법**
      - **행 인덱스와 열 인덱스의 라벨 값이 같은 데이터를 찾아서 해당 칸에 저장**
      - **만약 주어진 데이터가 존재하지 않으면 해당 칸에 NaN 값 저장**

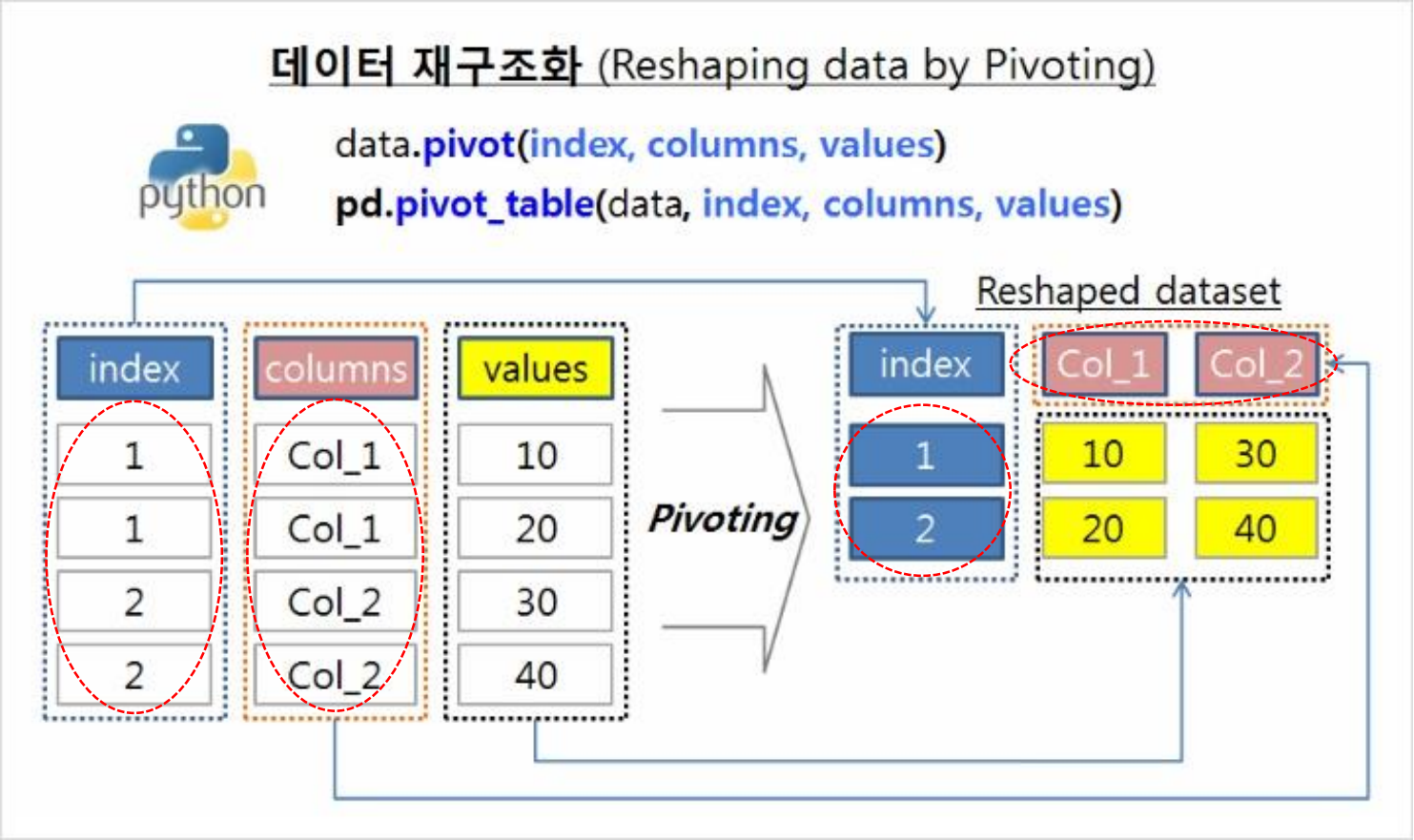


	foo	bar	baz	zoo
0	one	A	1	x
1	one	B	2	y
2	one	C	3	z
3	two	A	4	q
4	two	B	5	w
5	two	C	6	t

bar	A	B	C
foo			
one	1	2	3
two	4	5	6

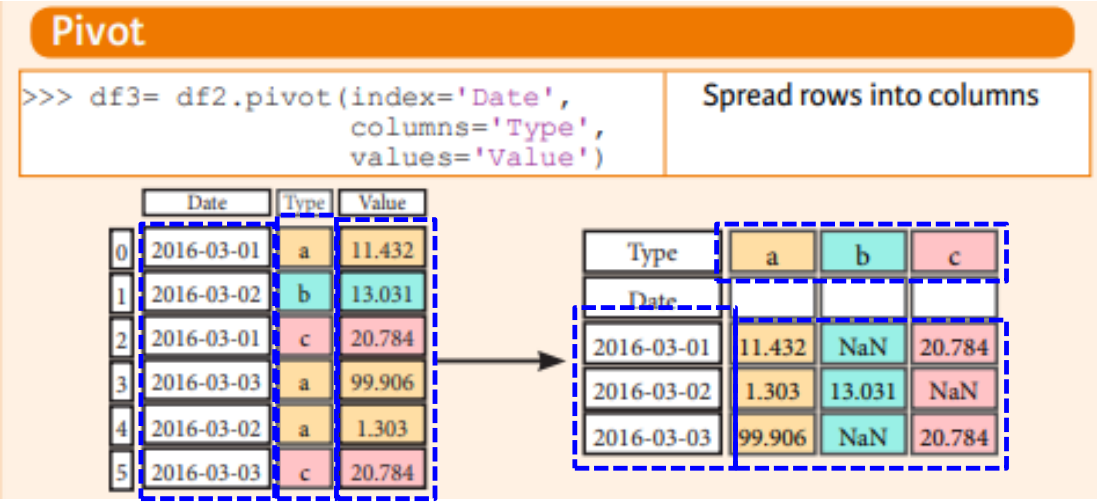
3개의 인자

- **index='cust_id', columns='prod_cd', values**



pivot 옵션

- **옵션 index**
  - 인덱스 지정
    - **지정하지 않으면**
      - **원래 인덱스 사용**
- **옵션 columns**
  - 열로 지정
- **옵션 values**
  - 값 지정
    - **아예 지정하지 않으면**
      - **남아 있는 모든 열이 상위의 다중 색인으로 열 이름이 들어 감**
    - **지정에서 빠진 열**
      - **제거**
      - **예에서 '지역'은 제거**



	도시	연도	인구	지역
0	서울	2015	9904312	수도권
1	서울	2010	9631482	수도권
2	서울	2005	9762546	수도권
3	부산	2015	3448737	경상권
4	부산	2010	3393191	경상권
5	부산	2005	3512547	경상권
6	인천	2015	2890451	수도권
7	인천	2010	263203	수도권

```
df1.pivot("도시", "연도", "인구")
```

연도	2005	2010	2015
도시			
부산	3512547.0	3393191.0	3448737.0
서울	9762546.0	9631482.0	9904312.0
인천	NaN	263203.0	2890451.0

pivot == set_index().unstack()

- **피봇테이블**
  - set_index 명령과 unstack 명령을 사용해서 생성도 가능
    - **df.pivot(index, columns, values)**
    - **df.set_index([index, columns])[values].unstack()**
  - set_index()
    - **지정한 열을 행 인텍스로 지정**

	도시	연도	인구	지역
0	서울	2015	9904312	수도권
1	서울	2010	9631482	수도권
2	서울	2005	9762546	수도권
3	부산	2015	3448737	경상권
4	부산	2010	3393191	경상권
5	부산	2005	3512547	경상권
6	인천	2015	2890451	수도권
7	인천	2010	263203	수도권

```
df1.pivot("도시", "연도", "인구")
```

연도	2005	2010	2015
도시			
부산	3512547.0	3393191.0	3448737.0
서울	9762546.0	9631482.0	9904312.0
인천	NaN	263203.0	2890451.0

```
df1.set_index(["도시", "연도"])['인구'].unstack()
```

	인구		
연도	2005	2010	2015
도시			
부산	3512547.0	3393191.0	3448737.0
서울	9762546.0	9631482.0	9904312.0
인천	NaN	263203.0	2890451.0

시계열 자료 p338

- **열이 14개인 시계열 자료**

In [2]:
```
data = pd.read_csv('examples/macrodata.csv')
data.head()
```

Out[2]:

	year	quarter	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unemp	pop	infl	realint
0	1959.0	1.0	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5.8	177.146	0.00	0.00
1	1959.0	2.0	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5.1	177.830	2.34	0.74
2	1959.0	3.0	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5.3	178.657	2.74	1.09
3	1959.0	4.0	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5.6	179.386	0.27	4.06
4	1960.0	1.0	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5.2	180.007	2.31	1.19

In [4]:
```
data.shape
```

Out[4]:
```
(203, 14)
```

In [5]:
```
# 시간간격을 나타내기 위한 자료형, 년도와 분기 칼럼을 합치는 작업
periods = pd.PeriodIndex(year=data.year, quarter=data.quarter, name='date')
periods
```

Out[5]:
```
PeriodIndex(['1959Q1', '1959Q2', '1959Q3', '1959Q4', '1960Q1', '1960Q2',
             '1960Q3', '1960Q4', '1961Q1', '1961Q2',
             ...
             '2007Q2', '2007Q3', '2007Q4', '2008Q1', '2008Q2', '2008Q3',
             '2008Q4', '2009Q1', '2009Q2', '2009Q3'],
            dtype='period[Q-DEC]', name='date', length=203, freq='Q-DEC')
```

In [6]:
```
columns = pd.Index(['realgdp', 'infl', 'unemp'], name='item')
columns
```

Out[6]:
```
Index(['realgdp', 'infl', 'unemp'], dtype='object', name='item')
```

pivot() 적용

In [6]:
```
columns = pd.Index(['realgdp', 'infl', 'unemp'], name='item')
columns
```

Out[6]:
```
Index(['realgdp', 'infl', 'unemp'], dtype='object', name='item')
```

In [9]:
```
data = data.reindex(columns=columns) # 지정된 열에 없는 것은 제외
```

In [10]:
```
data.index = periods.to_timestamp('D', 'end')
data.head()
```

Out[10]:

date	item	realgdp	infl	unemp
1959-03-31 23:59:59.999999999		2710.349	0.00	5.8
1959-06-30 23:59:59.999999999		2778.801	2.34	5.1
1959-09-30 23:59:59.999999999		2775.488	2.74	5.3
1959-12-31 23:59:59.999999999		2785.204	0.27	5.6
1960-03-31 23:59:59.999999999		2847.699	2.31	5.2

In [12]:
```
data2 = data.stack()
data2.head(6)
```

Out[12]:
```
date                          item
1959-03-31 23:59:59.999999999 realgdp    2710.349
                              infl          0.000
                              unemp         5.800
1959-06-30 23:59:59.999999999 realgdp    2778.801
                              infl          2.340
                              unemp         5.100
dtype: float64
```

In [28]:
```
ldata = data.stack().reset_index()
ldata.head()
```

Out[28]:

	date	item	0
0	1959-03-31 23:59:59.999999999	realgdp	2710.349
1	1959-03-31 23:59:59.999999999	infl	0.000
2	1959-03-31 23:59:59.999999999	unemp	5.800
3	1959-06-30 23:59:59.999999999	realgdp	2778.801
4	1959-06-30 23:59:59.999999999	infl	2.340

In [30]:
```
ldata = data.stack().reset_index().rename(columns={0: 'value'})
ldata.head()
```

Out[30]:

	date	item	value
0	1959-03-31 23:59:59.999999999	realgdp	2710.349
1	1959-03-31 23:59:59.999999999	infl	0.000
2	1959-03-31 23:59:59.999999999	unemp	5.800
3	1959-06-30 23:59:59.999999999	realgdp	2778.801
4	1959-06-30 23:59:59.999999999	infl	2.340

In [31]:
```
pivoted = ldata.pivot('date', 'item', 'value')
pivoted.head()
```

Out[31]:

date	item	infl	realgdp	unemp
1959-03-31 23:59:59.999999999		0.00	2710.349	5.8
1959-06-30 23:59:59.999999999		2.34	2778.801	5.1
1959-09-30 23:59:59.999999999		2.74	2775.488	5.3
1959-12-31 23:59:59.999999999		0.27	2785.204	5.6
1960-03-31 23:59:59.999999999		2.31	2847.699	5.2

열 추가 후

- 다시 pivot()
  - 옵션 values를 아예 지정하지 않으면
    - **남아 있는 모든 열: value, value2**
      - **(상위의) 다중 색인으로 열 이름이 들어 감**

In [33]:
```
ldata['value2'] = np.random.randn(len(ldata)) # 열 value2 추가
ldata.head()
```

Out[33]:

	date	item	value	value2
0	1959-03-31 23:59:59.999999999	realgdp	2710.349	0.981007
1	1959-03-31 23:59:59.999999999	infl	0.000	-0.873717
2	1959-03-31 23:59:59.999999999	unemp	5.800	-1.015634
3	1959-06-30 23:59:59.999999999	realgdp	2778.801	-0.411244
4	1959-06-30 23:59:59.999999999	infl	2.340	1.465621

In [ ]:
```
ldata.pivot?
```

In [34]:
```
pivoted = ldata.pivot('date', 'item') #지정하지 않은 열은 자동으로 열이 되어 계층적 열이 됨
pivoted[:5]
```

Out[34]:

	value			value2		
item	infl	realgdp	unemp	infl	realgdp	unemp
date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-0.873717	0.981007	-1.015634
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	1.465621	-0.411244	-1.006219
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	0.752769	-0.902148	-0.490509
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-0.699196	-0.524672	0.352361
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	-0.930342	0.068103	0.845400

In [35]:
```
pivoted['value'][:5]
```

Out[35]:

item	infl	realgdp	unemp
date			
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2

ldata.pivot('date', 'item')

- 위 pivot()은 다음과 같음
  - ldata.set_index(['date', 'item']).unstack('item')

```
In [36]: ldata.set_index(['date', 'item'])
```

Out[36]:

date	item	value	value2
1959-03-31 23:59:59.999999999	realgdp	2710.349	0.981007
	infl	0.000	-0.873717
	unemp	5.800	-1.015634
1959-06-30 23:59:59.999999999	realgdp	2778.801	-0.411244
	infl	2.340	1.465621
...	...	...	...
2009-06-30 23:59:59.999999999	infl	3.370	-0.544736
	unemp	9.200	0.977949
2009-09-30 23:59:59.999999999	realgdp	12990.341	-0.256382
	infl	3.560	0.510783
	unemp	9.600	0.720283

609 rows × 2 columns

```
In [37]: # 피벗은 set_index()로 계층적 색인을 만들고, 다시 unstack()을 사용하는 것과 동일
unstacked = ldata.set_index(['date', 'item']).unstack('item')
unstacked[:7]
```

Out[37]:

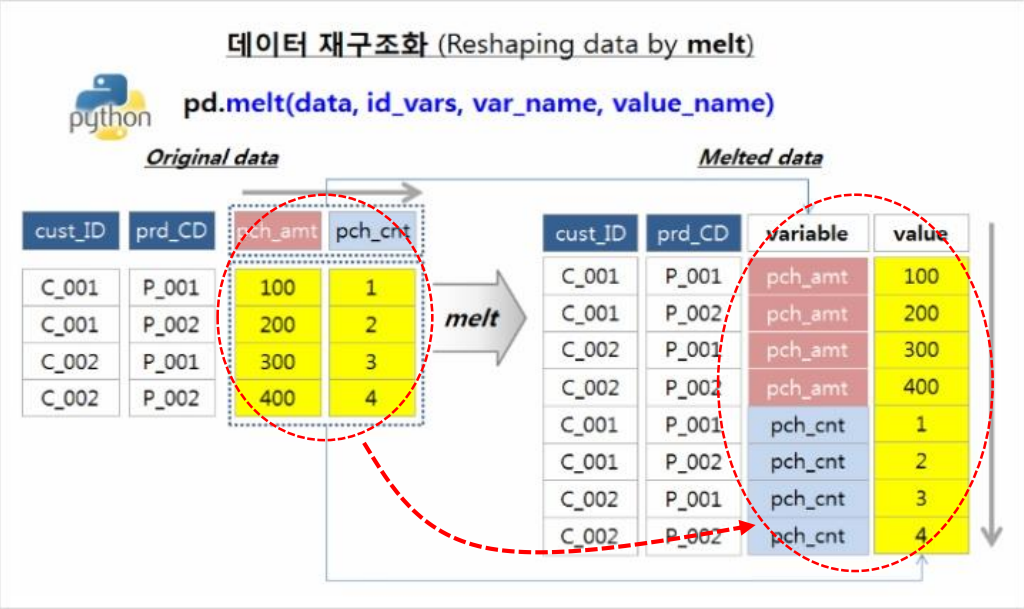
	value			value2		
item	infl	realgdp	unemp	infl	realgdp	unemp
date						
1959-03-31 23:59:59.999999999	0.00	2710.349	5.8	-0.873717	0.981007	-1.015634
1959-06-30 23:59:59.999999999	2.34	2778.801	5.1	1.465621	-0.411244	-1.006219
1959-09-30 23:59:59.999999999	2.74	2775.488	5.3	0.752769	-0.902148	-0.490509
1959-12-31 23:59:59.999999999	0.27	2785.204	5.6	-0.699196	-0.524672	0.352361
1960-03-31 23:59:59.999999999	2.31	2847.699	5.2	-0.930342	0.068103	0.845400
1960-06-30 23:59:59.999999999	0.14	2834.390	5.2	0.844963	0.016472	1.850834
1960-09-30 23:59:59.999999999	2.70	2839.022	5.6	-1.369179	0.022074	0.887204

8장 데이터 준비하기: 조인, 병합, 변형

melt pivot

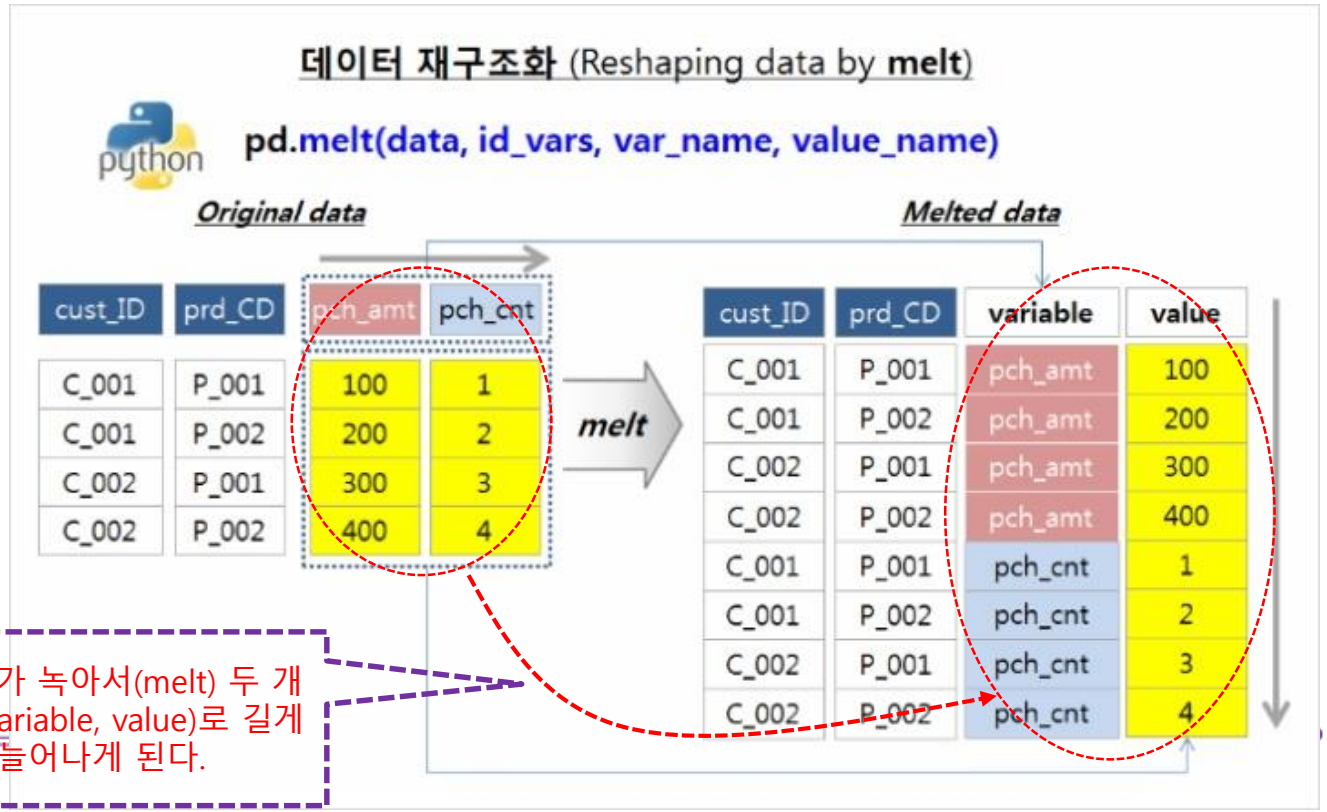
녹이는 melt() 개요 (1)

- **칼럼명이 2개: variable, value**
  - 데이터프레임의 컬럼 이름 자체를 한 컬럼 varibale에 모두 내리고
  - 해당하는 값을 다른 칼럼 value에 따로 빼서 재형성
    - **인자 id_vars를 기준으로**
      - **그대로 열이 남음**
    - **나머지 칼럼 이름을 'variable' 칼럼에 위에서 아래로 길게 쌓아놓고**
    - **'value' 칼럼에**
      - **id_vars와 variable에 해당하는 값을 넣어주는 식으로 데이터를 다시 생성**
    - **기존 index는 상관 없이 반환 값의 인덱스는 기본 정수 인덱스 RangeIndex**



녹이는 melt() 개요 (2)

- **인자 id_vars**
  - 남기는 열 지정
- **열 ' variable '**
  - 원래 데이터프레임에 있던 여러 개의 칼럼 이름을 위에서 아래로 길게 쌓아놓고
- **열 ' value '**
  - id_vars와 variable에 해당하는 값을 입력



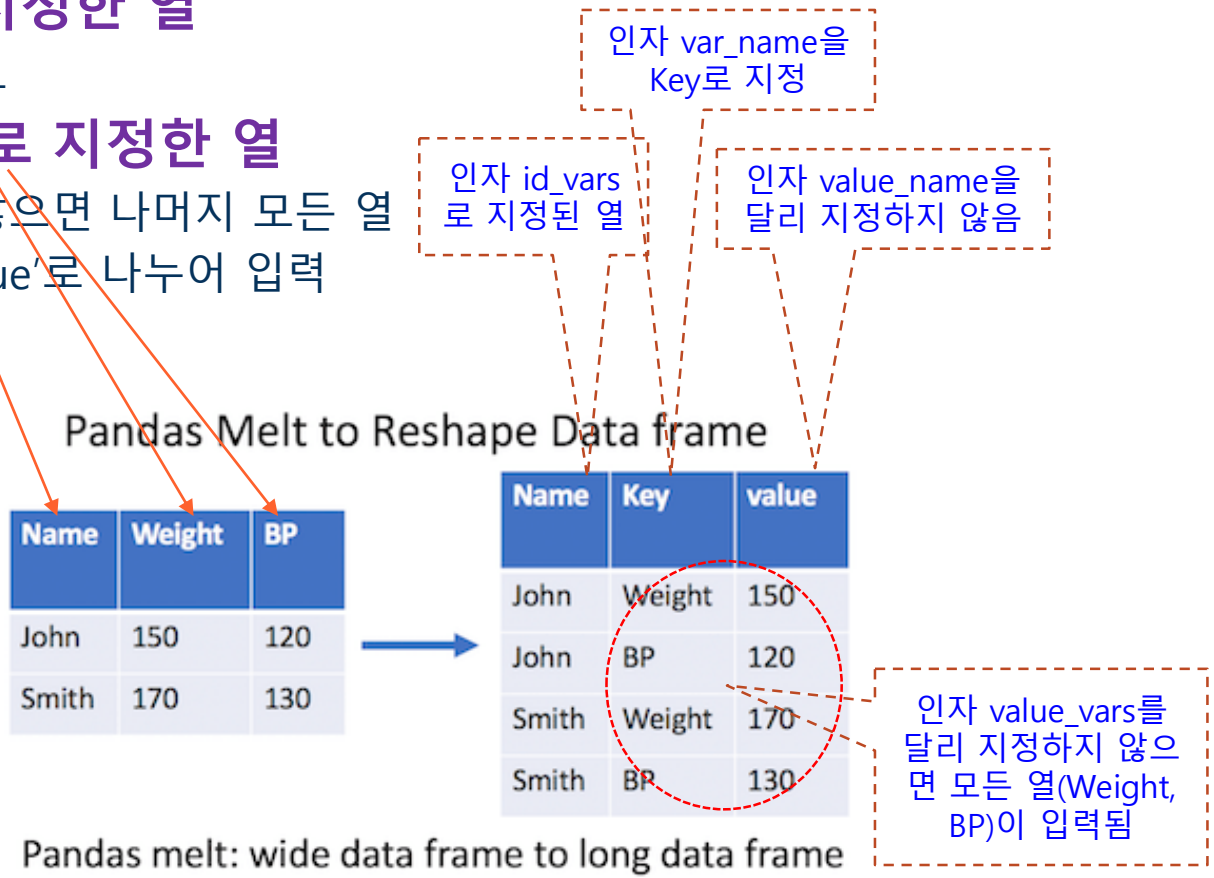
메소드 melt 인자

- **DataFrame.melt(self, id_vars=None,**
  value_vars=None,
  var_name=None, value_name='value',
  col_level=None)
  - DataFrame를 와이드 형식에서 긴 형식으로 unpivot
    - **모양이 길어짐**
- **주요 인자**
  - 선택적으로 식별자 집합(identifiers set)을 남김
    - **열을 그대로 유지하는 칼럼**
    - **Unpivot a DataFrame from wide to long format**
    - **optionally leaving identifiers set.**
  - 식별자 변수(id_vars)에 지정된 여러 열을 유지하는 형태의 DataFrame으로 변환
    - **기본적으로 식별자 변수 id_vars 외의 모든 열**
      - **두 개의 열인 'variable'과 'value ' 로 " unpivoluted " 됨**
    - **옵션으로 value_vars에 지정하면 그 열만 'variable'과 'value ' 로 이동**
    - **두 개의 열(non-identifier columns)인 'variable'과 'value'만 남음**
  - 인자 var_name, value_name
    - **두 열, 'variable'과 'value' 의 이름을 각각 var_name, value_name으로 수정 가능**

인자 value_vars가 지정되지 않으면 id_vars에 지정되지 않은 나머지 모든 열이 value_vars가 되어 variable, value 열로 이동

메소드 **melt** 간단 예

- **인자 id_vars로 지정한 열**
  - 그대로 열로 두고
- **인자 value_vars로 지정한 열**
  - 달리 지정하지 않으면 나머지 모든 열
  - 'variable' 과 'value'로 나누어 입력



두 개의 열로, 세로로 길게: melt()

- **열들을 간단히 두 개 열**
  - variable, value로 병합, 긴 형태로 재구성
  - 아예 인자가 없으면
    - **모든 열을 variable과 value로 구성**
  - 인자 id_vars
  - 인자 value_vars
  - 인자 var_name, value_name

```
In [3]: mydf = pd.DataFrame({'이름': ['홍길동', '임걱정', '아무개'],
                             '영어': [99, 79, 55],
                             '수학': [92, 82, 72]})
        mydf
```

Out[3]:

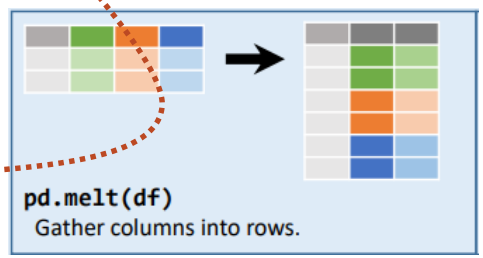
	이름	영어	수학
0	홍길동	99	92
1	임걱정	79	82
2	아무개	55	72

모든 인자가 지정되지 않으면 나머지 모든 열이 variable, value 열로 이동

```
In [5]: mydf.melt()
```

Out[5]:

	variable	value
0	이름	홍길동
1	이름	임걱정
2	이름	아무개
3	영어	99
4	영어	79
5	영어	55
6	수학	92
7	수학	82
8	수학	72




```
In [7]: mydf.melt(id_vars='이름', value_vars='영어')
```

Out[7]:

	이름	variable	value
0	홍길동	영어	99
1	임걱정	영어	79
2	아무개	영어	55

인자 id_vars에 지정한 열은 그대로 남고

인자 value_vars에 지정한 열만 variable, value 열로 이동

```
In [6]: mydf.melt(id_vars='이름')
```

Out[6]:

	이름	variable	value
0	홍길동	영어	99
1	임걱정	영어	79
2	아무개	영어	55
3	홍길동	수학	92
4	임걱정	수학	82
5	아무개	수학	72

인자 id_vars에 지정한 열만 남고 나머지 모든 열은 variable, value 열로 이동

```
In [9]: mydf.melt(id_vars='이름', value_vars='영어',
                  var_name='과목', value_name='점수')
```

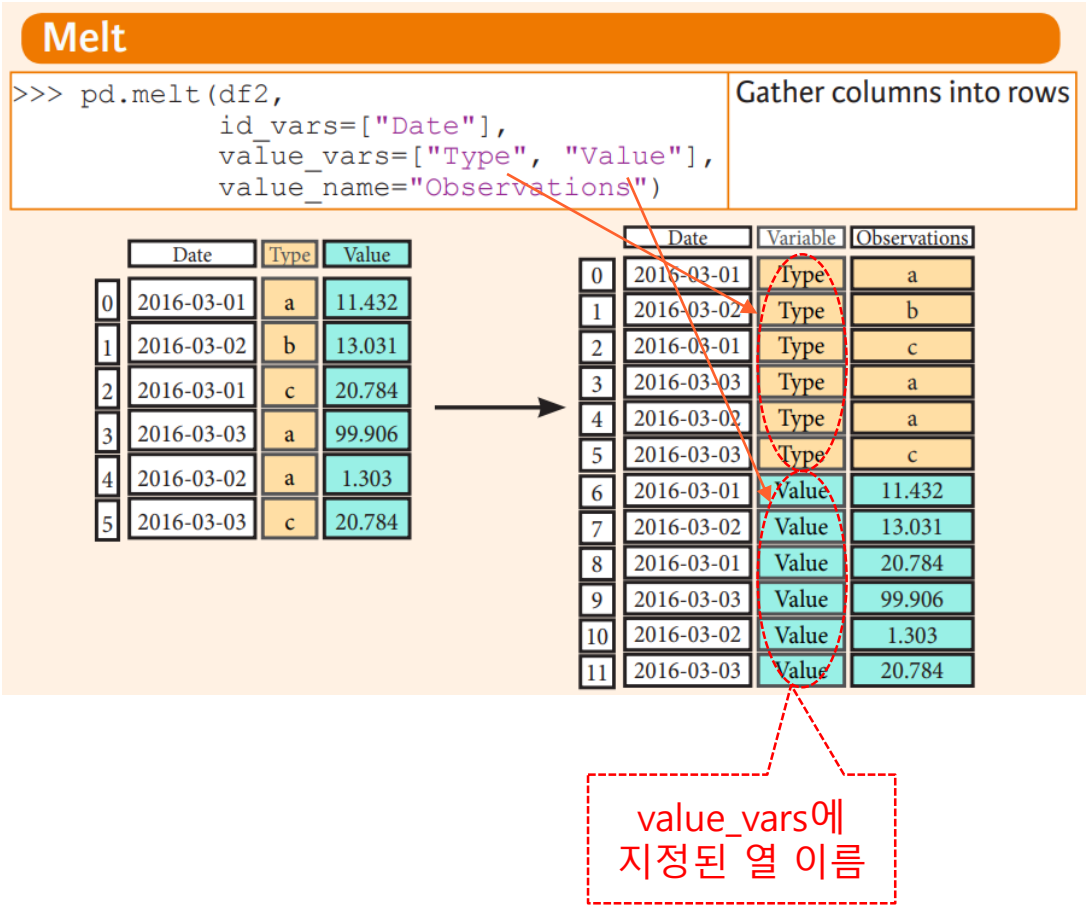
Out[9]:

	이름	과목	점수
0	홍길동	영어	99
1	임걱정	영어	79
2	아무개	영어	55

열 variable, value 의 이름을 각각 var_name, value_name으로 수정

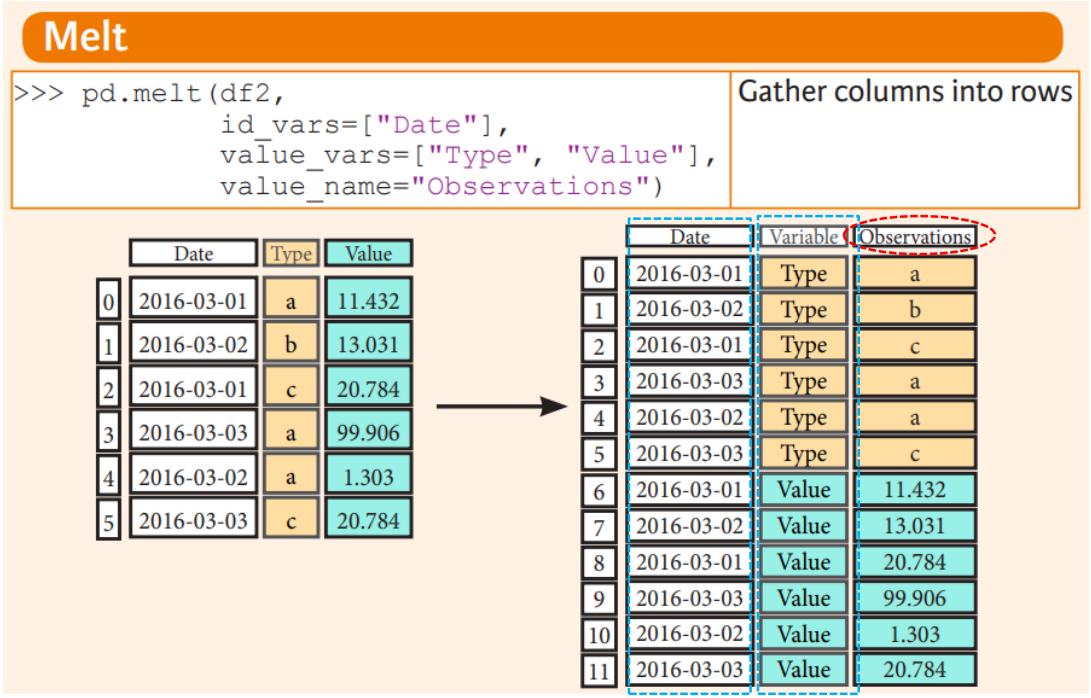
함수 melt 인자 활용

- 넓은 형식에서 긴 형식으로
  - id_vars:
    - 하나 이상의 열을 식별자 집합으로 지정
  - value_vars:
    - 열 variable, value에 사용할 열 들을 지정
      - 없으면 id_vars 외의 다른 열은 모두 variable과 value로 지정
- 결과적으로 간단히 재형성
  - id_vars와 'variable', 'value' 열만 남음
  - 'variable'
    - 기존의 열 이름이 저장
    - variable 열 이름
      - var_name으로 수정 가능
  - 'value'
    - 실제 값이 저장
    - value 열 이름
      - value_name으로 수정 가능



칼럼 이름 수정 인자

- **함수**
  - pd.melt(
  - frame: pandas.core.frame.DataFrame,
  - id_vars=None,
  - value_vars=None,
  - var_name=None,
  - value_name='value',
  - col_level=None,
  -)



- **variable열은 var_name으로 수정 가능**
- **value열은 value_name으로 수정 가능**

구분자 인자: id_vars

- **id_vars = ['key']**
  - id_vars를 식별자 집합으로 사용
  - 다른 2개의 열을 열 variable과 value에 저장
    - **열 이름과 값으로 지정해 사용**

```
In [157]: df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
   .....:                    'A': [1, 2, 3],
   .....:                    'B': [4, 5, 6],
   .....:                    'C': [7, 8, 9]})

In [158]: df
Out[158]:
   A  B  C  key
0  1  4  7  foo
1  2  5  8  bar
2  3  6  9  baz
```

The 'key' column may be a group indicator, and the other columns are data values. When using `pandas.melt`, we must indicate which columns (if any) are group indicators. Let's use 'key' as the only group indicator here:

```
In [159]: melted = pd.melt(df, ['key'])

In [160]: melted
Out[160]:
   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
6  foo        C      7
7  bar        C      8
8  baz        C      9
```

인자 없이 첫번째 위치한 것이 id_vars

인자 value_vars

- **value_vars: tuple, list, or ndarray, optional**
  - 데이터 값으로 사용할 칼럼을 지정
    - **열 variable에 값으로 삽입될 column 이름 또는 목록**
      - **If not specified, uses all columns that are not set as id_vars**

```
>>> df = pd.DataFrame({'A': {0: 'a', 1: 'b', 2: 'c'},
...                    'B': {0: 1, 1: 3, 2: 5},
...                    'C': {0: 2, 1: 4, 2: 6}})
>>> df
   A  B  C
0  a  1  2
1  b  3  4
2  c  5  6
```

```
>>> pd.melt(df, id_vars=['A'], value_vars=['B'])
   A variable  value
0  a        B      1
1  b        B      3
2  c        B      5
```

```
>>> pd.melt(df, id_vars=['A'], value_vars=['B', 'C'])
   A variable  value
0  a        B      1
1  b        B      3
2  c        B      5
3  a        C      2
4  b        C      4
5  c        C      6
```

```
In [102]: df
```

Out[102]:

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

```
In [101]: pd.melt(df, id_vars=['key'], value_vars=['A', 'B'])
```

Out[101]:

	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

식별자 변수 없이도 가능

- **인자 id_vars 없이**
  - 열 variable, value만 보임

```
In [283]: df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
                             'A': [1, 2, 3],
                             'B': [4, 5, 6],
                             'C': [7, 8, 9]})
          df
```

Out[283]:

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

```
In [106]: pd.melt(df, value_vars=['A', 'B', 'C'])
```

Out[106]:

	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

```
In [107]: pd.melt(df, value_vars=['key', 'A', 'B'])
```

Out[107]:

	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

pivot()으로 원래의 형식으로 복원

- 원 df를 녹인 후
  - 다시 피봇하여
    - reset_index()
      - 원 df

In [292]:
```
df = pd.DataFrame({'key': ['foo', 'bar', 'baz'],
                   'A': [1, 2, 3],
                   'B': [4, 5, 6],
                   'C': [7, 8, 9]})
df
```

Out[292]:

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

In [285]:
```
pd.melt?
```

In [293]:
```
melted = pd.melt(df, ['key'])
melted
```

Out[293]:

	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
6	foo	C	7
7	bar	C	8
8	baz	C	9

index columns value

In [294]:
```
reshaped = melted.pivot('key', 'variable', 'value')
reshaped
```

Out[294]:

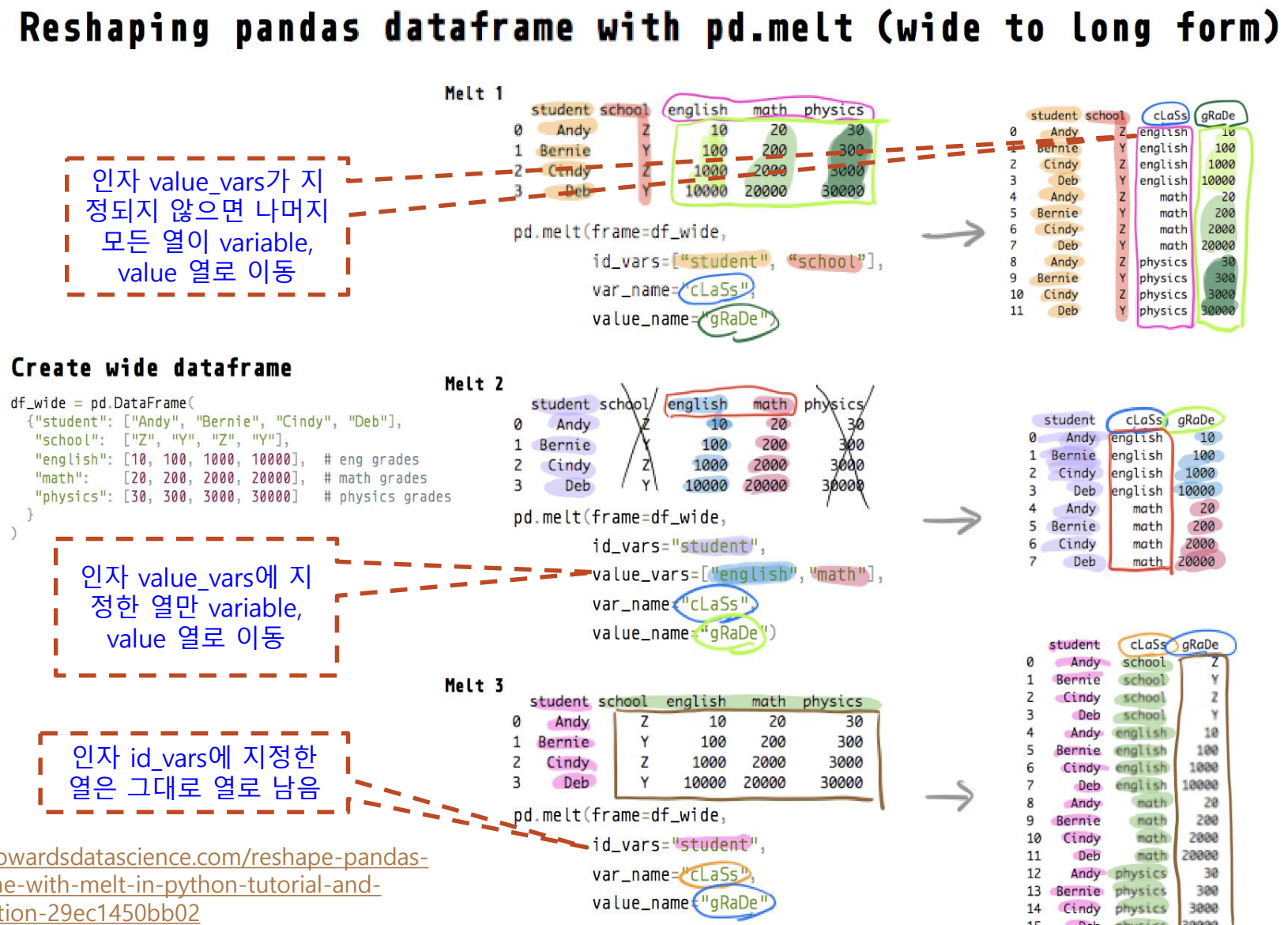
variable	A	B	C
key			
bar	2	5	8
baz	3	6	9
foo	1	4	7

In [295]:
```
reshaped.reset_index()
```

Out[295]:

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

변형 메소드 melt() 요약



https://towardsdatascience.com/reshape-pandas-dataframe-with-melt-in-python-tutorial-and-visualization-29ec1450bb02

Reshaping by melt

- The top-level melt() function and the corresponding DataFrame.melt() are useful to massage a DataFrame into a format where one or more columns are identifier variables, while all other columns, considered measured variables, are "unpivoted" to the row axis, leaving just two non-identifier columns, "variable" and "value".
  - **The names of those columns can be customized by supplying the var_name and value_name parameters.**

https://pandas.pydata.org/docs/user_guide/reshaping.html

df3

	first	last	height	weight
0	John	Doe	5.5	130
1	Mary	Bo	6.0	150

df3.melt(id_vars=['first', 'last'])

	first	last	variable	value
0	John	Doe	height	5.5
1	Mary	Bo	height	6.0
2	John	Doe	weight	130
3	Mary	Bo	weight	150