

파이썬 라이브러리를 활용한 데이터 분석

8장 데이터 준비하기:
조인, 병합, 변형

2020.07.02 3h

다중 색인, 조인, 병합, 변형

- 데이터틀 합치고 재배열 필요
 - 원천 데이터는 분석하기 어려운 형태로 기록되어 제공
- 주요 내용
 - 계층 색인(다중 색인)
 - Multi-index
 - 데이터 합치기
 - Merge
 - Join
 - Concat
 - Combine_first
 - 재형성과 피벗
 - Stack
 - Unstack
 - Pivot
 - Melt

참고 사이트

- 국내

- https://freelife1191.github.io/dev/2018/05/07/dev-data_analysis-22.python_data_analysis/
- <https://rfriend.tistory.com/276>

- 국외

- https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#advanced-hierarchical
- https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html
- <https://towardsdatascience.com/python-pandas-dataframe-join-merge-and-concatenate-84985c29ef78>
- <http://talimi.se/p/pandas/>

파일 ch08-study.ipynb

8장 데이터 준비하기: 조인, 병합, 변형

Multi-index

계층 색인(다중 인덱스)

• 다중 인덱스(multi-index)를 설정: MultiIndex class

- 행이나 열에 여러 계층을 가지는 인덱스
 - 칼럼
 - 생성 시, `columns` 인수에 리스트 항목으로 리스트(행렬) 형태로 인덱스를 지정
- 열 인덱스들의 이름 지정
 - `columns` 객체의 `names` 속성에 리스트를 넣어서 지정

In [6]:

```
np.random.seed(0)
df3 = pd.DataFrame(np.round(np.random.randn(5, 4), 2),
                    columns=[["A", "A", "B", "B"],
                             ["C1", "C2", "C1", "C2"]])
df3
```

	A		B	
	C1	C2	C1	C2
0	1.76	0.40	0.98	2.24
1	1.87	-0.98	0.95	-0.15
2	-0.10	0.41	0.14	1.45
3	0.76	0.12	0.44	0.33
4	1.49	-0.21	0.31	-0.85

In [7]:

```
df3.columns.names = ["Cidx1", "Cidx2"]
df3
```

Cidx1	A		B	
	C1	C2	C1	C2
0	1.76	0.40	0.98	2.24
1	1.87	-0.98	0.95	-0.15
2	-0.10	0.41	0.14	1.45
3	0.76	0.12	0.44	0.33
4	1.49	-0.21	0.31	-0.85

다중 (행) 인덱스

• index 인수

- 리스트의 리스트(행렬) 형태로 인덱스를 넣으면
- 행 인덱스들의 이름 지정
 - index 객체의 names 속성에 리스트를 넣어서 지정

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

교재 8.1 계층적 색인

p307

- 계층적 색인

- 높은 차원의 테이블을 낮은 차원의 형식으로 다룰 수 있게 해주는 기능

Hierarchical Indexing

```
In [3]: 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[3]: a 1 -0.204708
        2  0.478943
        3 -0.519439
       b 1 -0.555730
        3  1.965781
       c 1  1.393406
        2  0.092908
       d 2  0.281746
        3  0.769023
dtype: float64
```

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

```
Out[4]: MultiIndex([(a, 1),
                    (a, 2),
                    (a, 3),
                    (b, 1),
                    (b, 3),
                    (c, 1),
                    (c, 2),
                    (d, 2),
                    (d, 3)],
                    )
```

```
In [6]: idx = data.index
        type(idx)
```

```
Out[6]: pandas.core.indexes.multi.MultiIndex
```


색인 중 내부 색인을 칼럼으로 변환

- Unstack
- Stack
 - 칼럼을 색인으로 변환

In [13]: data

Out[13]:

a	1	-0.204708
	2	0.478943
	3	-0.519439
b	1	-0.555730
	3	1.965781
c	1	1.393406
	2	0.092908
d	2	0.281746
	3	0.769023

dtype: float64

In [15]: data.unstack().stack()

Out[15]:

a	1	-0.204708
	2	0.478943
	3	-0.519439
b	1	-0.555730
	3	1.965781
c	1	1.393406
	2	0.092908
d	2	0.281746
	3	0.769023

dtype: float64

In [14]: data.unstack()

Out[14]:

	1	2	3
a	-0.204708	0.478943	-0.519439
b	-0.555730	NaN	1.965781
c	1.393406	0.092908	NaN
d	NaN	0.281746	0.769023

다중 색인의 생성 p311

- 미리 생성해 재사용 가능

```
In [24]: midx = pd.MultiIndex.from_arrays([[ 'Ohio', 'Ohio', 'Colorado'], [ 'Green', 'Red', ' '],
                                         names=[ 'state', 'color']])
```

```
In [25]: fm = pd.DataFrame(np.arange(12).reshape((4, 3)),
                           index=[ 'a', 'a', 'b', 'b'], [1, 2, 1, 2]),
                           columns=midx)
fm
```

Out[25]:

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

색인 계층의 순서 바꾸기

- **swaplevel()**
 - 데이터는 무관

In [26]: frame

Out[26]:

		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.swaplevel('key1', 'key2')

Out[28]:

		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	

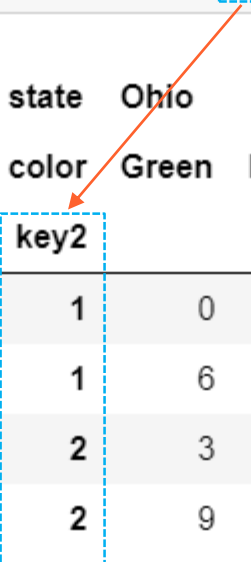
색인 정렬에 따른 자료 정렬

• `sort_index()`

- level: 정수나 이름 문자열 지정 가능
- axis: 기본 0이고 (행) 색인이며, 1이면 칼럼

```
In [36]: frame.sort_index(level='key2')
```

```
Out[36]:
```



		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

8장 데이터 준비하기: 조인, 병합, 변형

set_index()
reset_index()

(행) 인덱스 지정

- 반드시 열만 지정
 - 기존의 열은 기본적으로 제거
- 제거하지 않으려면
 - 인자 drop=False

Indexing with a DataFrame's columns

```
In [40]: 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[40]:
```

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

```
Out[47]:
```

	a	b	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 [41]: frame2 = frame.set_index(['c', 'd'])
frame2
```

```
Out[41]:
```

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

=

reset_index()

- 기존 인덱스를 열로 이동
 - 행 인덱스가 모두 칼럼으로 이동
 - 결과의 인덱스는 정수 인덱스

In [49]: frame2

Out[49]:

		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 [48]: frame2.reset_index()

Out[48]:

	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

8장 데이터 준비하기: 조인, 병합, 변형

8.2 데이터 합치기

8.2 데이터 합치기 p315

- 데이터프레임 합치기

- merge
 - 하나 이상의 키로 데이터프레임의 로우를 합치기
- Join
 - 인덱스로 데이터프레임의 로우를 합치기
- concat
 - 하나의 축에 따라 '이어 붙이기'
- combine_first
 - 두 객체를 포개서 한 객체에서 누락된 데이터를 다른 객체에 있는 값으로 채우기

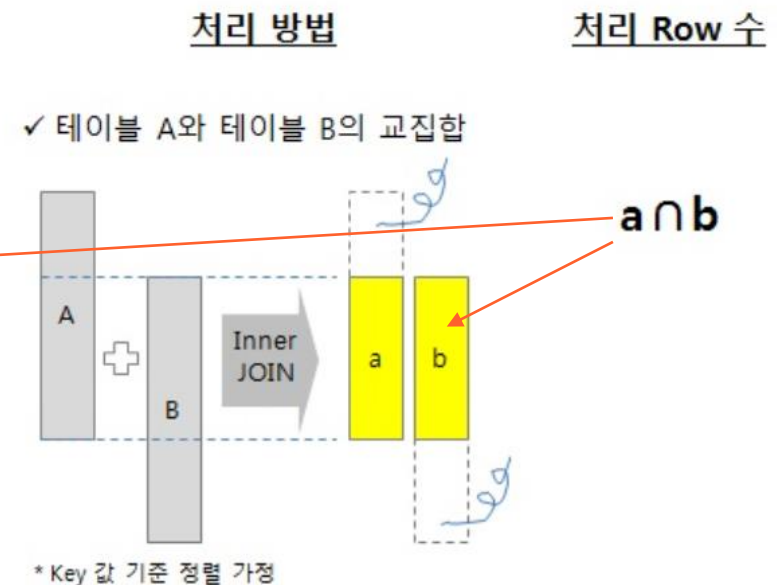
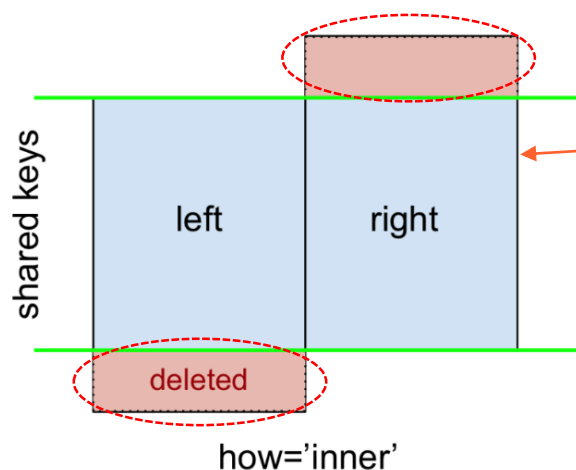
8장 데이터 준비하기: 조인, 병합, 변형

Merge

병합 개요: 기본 연산, 내부 병합

• 합치다 merge

- 두 데이터 프레임의 공통 열(혹은 인덱스)를 기준으로 두 개의 테이블을 합침
- 공통 키에 대한 합침
 - 기본은 내부 병합
 - 교집합 공통 키인 모든 행 구성
 - 공통 키가 없는 행은 제외
- 결과 열은 항상 두 데이터 프레임 모든 열의 합
 - 왼쪽만 있는 열, 오른쪽만 있는 열 포함
 - 공통 열은 하나만 포함



병합 개요: 외부 병합

• 합치다 merge

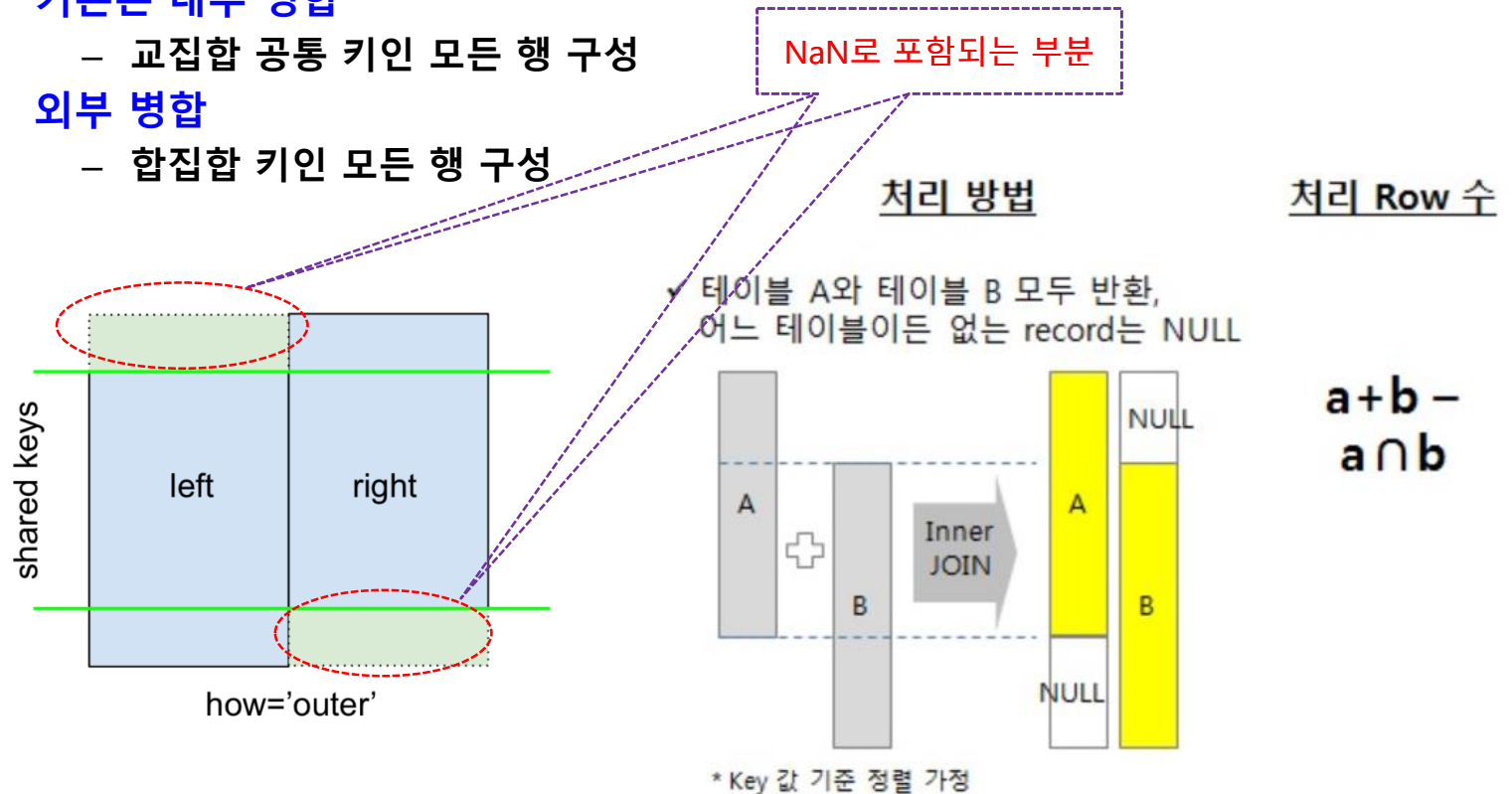
- 두 데이터 프레임의 공통 열 혹은 인덱스를 기준으로 두 개의 테이블을 합침
- 공통 키에 대한 합침

• 기본은 내부 병합

- 교집합 공통 키인 모든 행 구성

• 외부 병합

- 합집합 키인 모든 행 구성



merge

• merge 기본

- 두 데이터 프레임의 공통 열 혹은 인덱스를 기준으로 두 개의 테이블을 합침
 - 이 때 기준이 되는 열, 행의 데이터를 키 (key)
- 공통 열인 고객번호 열을 기준으로 데이터를 찾아서 합침
 - 기본적으로 양쪽 데이터프레임에 모두 키가 존재하는 데이터만 보여주는 inner join 방식을 사용

In [1]:

```
df1 = pd.DataFrame({
    '고객번호': [1001, 1002, 1003, 1004, 1005, 1006, 1007],
    '이름': ['둘리', '도우너', '또치', '길동', '희동', '마이콜', '영희']
}, columns=['고객번호', '이름'])
df1
```

	고객번호	이름
0	1001	둘리
1	1002	도우너
2	1003	또치
3	1004	길동
4	1005	희동
5	1006	마이콜
6	1007	영희

In [2]:

```
df2 = pd.DataFrame({
    '고객번호': [1001, 1001, 1005, 1006, 1008, 1001],
    '금액': [10000, 20000, 15000, 5000, 100000, 30000]
}, columns=['고객번호', '금액'])
df2
```

	고객번호	금액
0	1001	10000
1	1001	20000
2	1005	15000
3	1006	5000
4	1008	100000
5	1001	30000

how=

- **inner**
 - 키
 - 공통 열인 고객 번호 열
 - 기준으로 데이터를 찾아서 합침
- **outer**
 - 키 값이 한쪽에만 있어도 데이터를 보여줌

	고객번호	이름
0	1001	둘리
1	1002	도우너
2	1003	또치
3	1004	길동
4	1005	희동
5	1006	마이클
6	1007	영희

	고객번호	금액
0	1001	10000
1	1001	20000
2	1005	15000
3	1006	5000
4	1008	100000
5	1001	30000

In [3]:

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

	고객번호	이름	금액
0	1001	둘리	10000
1	1001	둘리	20000
2	1001	둘리	30000
3	1005	희동	15000
4	1006	마이클	5000

outer join 방식은 키 값이 한쪽에만 있어도 데이터를 보여준다.

In [4]:

```
pd.merge(df1, df2, how='outer')
```

	고객번호	이름	금액
0	1001	둘리	10000.0
1	1001	둘리	20000.0
2	1001	둘리	30000.0
3	1002	도우너	NaN
4	1003	또치	NaN
5	1004	길동	NaN
6	1005	희동	15000.0
7	1006	마이클	5000.0
8	1007	영희	NaN
9	1008	NaN	100000.0

중복 열 지정 on=

p316

- 공통인 열, key에서 교집합인 행만 표시

```
In [104]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                             'data1': range(7)})
df1
```

Out[104]:

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

```
In [105]: df2 = pd.DataFrame({'key': ['a', 'b', 'd'],
                             'data2': range(3)})
df2
```

Out[105]:

	key	data2
0	a	0
1	b	1
2	d	2

```
In [106]: pd.merge(df1, df2)
```

Out[106]:

	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

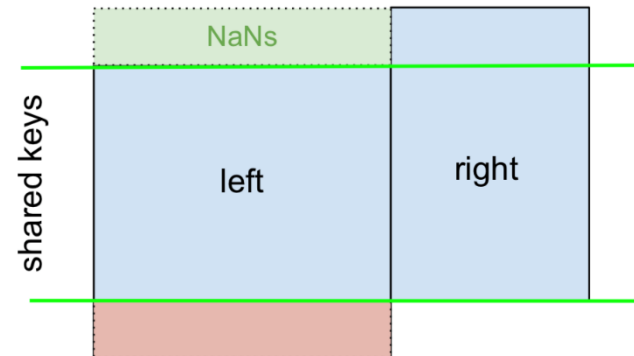
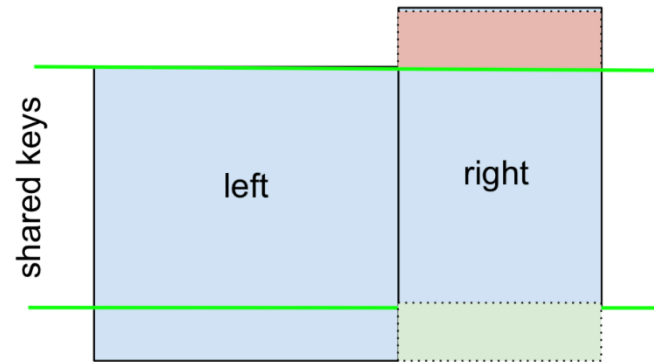
```
In [107]: pd.merge(df1, df2, on='key')
```

Out[107]:

	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

병합 방법 4가지

- **inner, outer**
 - 기본은 내부 병합
- **left**
 - 왼쪽 키로만 구성
- **right**
 - 오른쪽 키로만 구성



how=

- **left**
 - 키
 - 공통 열인 고객번호 열
 - 왼쪽의 키로 구성된 모든 행 선택
- **right**
 - 오른쪽의 키로 구성된 모든 행 선택

	고객번호	이름
0	1001	둘리
1	1002	도우너
2	1003	또치
3	1004	길동
4	1005	희동
5	1006	마이콜
6	1007	영희

	고객번호	금액
0	1001	10000
1	1001	20000
2	1005	15000
3	1006	5000
4	1008	100000
5	1001	30000

In [5]:

```
pd.merge(df1, df2, how='left')
```

	고객번호	이름	금액
0	1001	둘리	10000.0
1	1001	둘리	20000.0
2	1001	둘리	30000.0
3	1002	도우너	NaN
4	1003	또치	NaN
5	1004	길동	NaN
6	1005	희동	15000.0
7	1006	마이콜	5000.0
8	1007	영희	NaN

In [6]:

```
pd.merge(df1, df2, how='right')
```

	고객번호	이름	금액
0	1001	둘리	10000
1	1001	둘리	20000
2	1001	둘리	30000
3	1005	희동	15000
4	1006	마이콜	5000
5	1008	NaN	100000

가능한 조합 구성

- 키 값이 같은 데이터가 여러 개 있는 경우
 - 있을 수 있는 모든 경우의 수를 따져서 조합을 구성
 - 키 값 setosa
 - 대해 왼쪽 데이터프레임은 1.4와 1.3라는 2개의 데이터
 - 오른쪽 데이터프레임에 0.4라는 1개의 데이터
 - 병합된 데이터에는 setosa가 (1.4, 0.4), (1.3, 0.4) 두 개의 데이터가 생김
 - 키 값 virginica
 - 왼쪽 데이터프레임에 1.5와 1.3라는 2개의 데이터
 - 오른쪽 데이터프레임에 0.3와 0.5라는 2개의 데이터
 - 2개와 2개의 조합에 의해 (2*2)4가지 값

In [7]:

```
df1 = pd.DataFrame({
    '품종': ['setosa', 'setosa', 'virginica', 'virginica'],
    '꽃잎길이': [1.4, 1.3, 1.5, 1.3]},
    columns=['품종', '꽃잎길이'])
df1
```

	품종	꽃잎길이
0	setosa	1.4
1	setosa	1.3
2	virginica	1.5
3	virginica	1.3

In [8]:

```
df2 = pd.DataFrame({
    '품종': ['setosa', 'virginica', 'virginica', 'versicolor'],
    '꽃잎너비': [0.4, 0.3, 0.5, 0.3]},
    columns=['품종', '꽃잎너비'])
df2
```

	품종	꽃잎너비
0	setosa	0.4
1	virginica	0.3
2	virginica	0.5
3	versicolor	0.3

In [9]:

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

	품종	꽃잎길이	꽃잎너비
0	setosa	1.4	0.4
1	setosa	1.3	0.4
2	virginica	1.5	0.3
3	virginica	1.5	0.5
4	virginica	1.3	0.3
5	virginica	1.3	0.5

인자 how=outer

- **left**
 - 왼쪽의 모든 행을 포함
- **right**
 - 오른쪽의 모든 행을 포함
- **outer**
 - 모든 합집합

In [111]: df1

Out[111]:

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

In [112]: df2

Out[112]:

	key	data2
0	a	0
1	b	1
2	d	2

In [113]: pd.merge(df1, df2, how='outer')

Out[113]:

	key	data1	data2
0	b	0.0	1.0
1	b	1.0	1.0
2	b	6.0	1.0
3	a	2.0	0.0
4	a	4.0	0.0
5	a	5.0	0.0
6	c	3.0	NaN
7	d	NaN	2.0

인자 how=left

```
In [114]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                             'data1': range(6)})
df1
```

Out[114]:

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

```
In [115]: df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                             'data2': range(5)})
df2
```

Out[115]:

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

```
In [116]: pd.merge(df1, df2, on='key', how='left')
```

Out[116]:

	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

인자 how=inner

```
In [114]: df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                             'data1': range(6)})
df1
```

Out[114]:

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

```
In [115]: df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                             'data2': range(5)})
df2
```

Out[115]:

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

```
In [117]: pd.merge(df1, df2, how='inner')
```

Out[117]:

	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

공통 기준열 명시 on=

• on 인수로 기준 열을 명시

- 두 데이터프레임에서 이름이 같은 열은 모두 키
- 만약 이름이 같아도 키가 되면 안되는 열이 있다면 on으로 지정

• 사례

- 첫번째 데이터프레임의 "데이터"
 - 실제로는 금액을 나타내는 데이터
- 두번째 데이터프레임의 "데이터"
 - 실제로는 성별을 나타내는 데이터
- 이름이 같아도 다른 데이터
 - 따라서 이 열은 기준 열이 되면 안됨
- 기준(키) 열이 아니면서 이름이 같은 열에는 _x 또는 _y 와 같은 접미사가 붙음

In [10]:

```
df1 = pd.DataFrame({
    '고객명': ['춘향', '춘향', '몽룡'],
    '날짜': ['2018-01-01', '2018-01-02', '2018-01-01'],
    '데이터': ['20000', '30000', '100000']})
df1
```

	고객명	날짜	데이터
0	춘향	2018-01-01	20000
1	춘향	2018-01-02	30000
2	몽룡	2018-01-01	100000

In [11]:

```
df2 = pd.DataFrame({
    '고객명': ['춘향', '몽룡'],
    '데이터': ['여자', '남자']})
df2
```

In [12]:

```
pd.merge(df1, df2, on='고객명')
```

	고객명	날짜	데이터_x	데이터_y
0	춘향	2018-01-01	20000	여자
1	춘향	2018-01-02	30000	여자
2	몽룡	2018-01-01	100000	남자

인자 left_on= right_on=

- left_on, right_on 인수를 사용하여 기준 열을 명시
 - 만일 키가 되는 기준열의 이름이 두 데이터프레임에서 다르다면
 - 다음처럼 열 이름이 다르면 모두 구성

In [13]:

```
df1 = pd.DataFrame({
    '이름': ['영희', '철수', '철수'],
    '성적': [1, 2, 3]})
df1
```

	성적	이름
0	1	영희
1	2	철수
2	3	철수

In [14]:

```
df2 = pd.DataFrame({
    '성명': ['영희', '영희', '철수'],
    '성적2': [4, 5, 6]})
df2
```

	성명	성적2
0	영희	4
1	영희	5
2	철수	6

In [15]:

```
pd.merge(df1, df2, left_on='이름', right_on="성명")
```

	성적	이름	성명	성적2
0	1	영희	영희	4
1	1	영희	영희	5
2	2	철수	철수	6
3	3	철수	철수	6

좌우의 키를 지정, left_on= right_on p317

```
In [108]: df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                             'data1': range(7)})
df3
```

Out[108]:

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

```
In [110]: pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

Out[110]:

	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

```
In [109]: df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
                             'data2': range(3)})
df4
```

Out[109]:

	rkey	data2
0	a	0
1	b	1
2	d	2

내부 조인을 수행해 교집합인 결과를 반환

인자 left_index= right_index=

- 일반 데이터 열이 아닌 인덱스를 기준 열로 사용하려면
 - left_index 또는 right_index 인수를 True 로 설정

In [16]:

```
df1 = pd.DataFrame({
    '도시': ['서울', '서울', '서울', '부산', '부산'],
    '연도': [2000, 2005, 2010, 2000, 2005],
    '인구': [9853972, 9762546, 9631482, 3655437, 3512547]})
df1
```

	도시	연도	인구
0	서울	2000	9853972
1	서울	2005	9762546
2	서울	2010	9631482
3	부산	2000	3655437
4	부산	2005	3512547

In [17]:

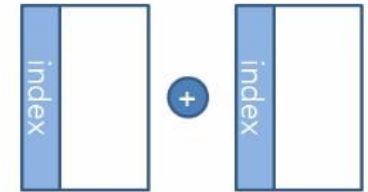
```
df2 = pd.DataFrame(
    np.arange(12).reshape((6, 2)),
    index=[['부산', '부산', '서울', '서울', '서울', '서울'],
           [2000, 2005, 2000, 2005, 2010, 2015]],
    columns=['데이터1', '데이터2'])
df2
```

		데이터1	데이터2
부산	2000	0	1
	2005	2	3
서울	2000	4	5
	2005	6	7
	2010	8	9
	2015	10	11

DataFrame을 index 기준으로 합치기 (merge, join on index)



```
pd.merge(df1, df2,
         left_index=True,
         right_index=True,
         how='left')
```



In [18]:

```
pd.merge(df1, df2, left_on=['도시', '연도'], right_index=True)
```

	도시	연도	인구	데이터1	데이터2
0	서울	2000	9853972	4	5
1	서울	2005	9762546	6	7
2	서울	2010	9631482	8	9
3	부산	2000	3655437	0	1
4	부산	2005	3512547	2	3

교재 예제: p321

```
In [123]: left1 = pd.DataFrame({'key': ['a', 'b', 'a', 'a', 'b', 'c'],
                                'value': range(6)})
left1
```

Out[123]:

	key	value
0	a	0
1	b	1
2	a	2
3	a	3
4	b	4
5	c	5

```
In [124]: right1 = pd.DataFrame({'group_val': [3.5, 7]}, index=['a', 'b'])
right1
```

Out[124]:

	group_val
a	3.5
b	7.0

```
In [125]: pd.merge(left1, right1, left_on='key', right_index=True)
```

Out[125]:

	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

여러 키의 공통으로 병합

p319

```
In [76]: left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                             'key2': ['one', 'two', 'one'],
                             'lval': [1, 2, 3]})
left
```

Out[76]:

	key1	key2	lval
0	foo	one	1
1	foo	two	2
2	bar	one	3

```
In [77]: right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                              'key2': ['one', 'one', 'one', 'two'],
                              'rval': [4, 5, 6, 7]})
right
```

Out[77]:

	key1	key2	rval
0	foo	one	4
1	foo	one	5
2	bar	one	6
3	bar	two	7

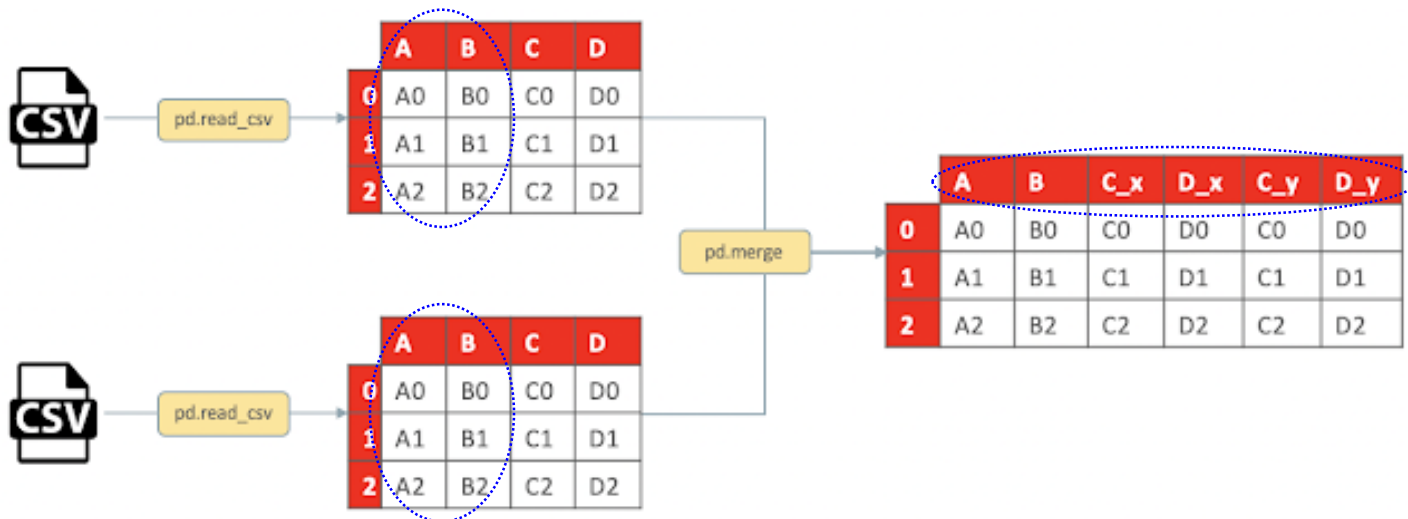
```
In [79]: pd.merge(left, right, on=['key1', 'key2'], how='outer')
```

Out[79]:

	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
4	bar	two	NaN	7.0

열 이름이 같지만 키에서 제외

- 각각의 열이 결과 데이터프레임에 추가되려면
 - 열 이름 수정이 필요
 - 자동으로 `_x`, `_y`가 왼쪽 오른쪽 이름 뒤에 붙음
- `pd.merge(left, right, on = ['A', 'B'])`



병합된 동일 칼럼 이름 재설정 : 옵션 **suffixes**

• 기본

- 이름_x, 이름_y
- suffixes로 지정 가능

```
In [76]: left = pd.DataFrame({'key1': ['foo', 'foo', 'bar'],
                             'key2': ['one', 'two', 'one'],
                             'lval': [1, 2, 3]})
left
```

```
Out[76]:
```

	key1	key2	lval
0	foo	one	1
1	foo	two	2
2	bar	one	3

```
In [77]: right = pd.DataFrame({'key1': ['foo', 'foo', 'bar', 'bar'],
                              'key2': ['one', 'one', 'one', 'two'],
                              'rval': [4, 5, 6, 7]})
right
```

```
Out[77]:
```

	key1	key2	rval
0	foo	one	4
1	foo	one	5
2	bar	one	6
3	bar	two	7

```
In [83]: pd.merge(left, right, on='key1')
```

```
Out[83]:
```

	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

```
In [84]: pd.merge(left, right, on='key1', suffixes=('_left', '_right'))
```

```
Out[84]:
```

	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

merge()의 인자 목록

p320

- `pd.merge(...)`
 - `pd.merge(left, right, # merge할 DataFrame 객체 이름`
 - `how='inner', # left, right, inner (default), outer`
 - `on=None, # merge의 기준이 되는 Key 변수`
 - `left_on=None, # 왼쪽 DataFrame의 변수를 Key로 사용`
 - `right_on=None, # 오른쪽 DataFrame의 변수를 Key로 사용`
 - `left_index=False, # 만약 True 라면, 왼쪽 DataFrame의 index를 merge Key로 사용`
 - `right_index=False, # 만약 True 라면, 오른쪽 DataFrame의 index를 merge Key로 사용`
 - `sort=True, # merge 된 후의 DataFrame을 join Key 기준으로 정렬`
 - `suffixes=('_x', '_y'), # 중복되는 변수 이름에 대해 접두사 부여 (defaults to '_x', '_y')`
 - `copy=True, # merge할 DataFrame을 복사`
 - `indicator=False) # 병합된 이후의 DataFrame에 left_only, right_only, both 등의 출처를 알 수 있는 부가 정보 변수 추가`

Merge 요약

```
In [29]: lt = pd.DataFrame([[1, 2], [10, 20]],
                          index = list('ab'),
                          columns = list('AB'))
```

lt

Out[29]:

	A	B
a	1	2
b	10	20

```
In [30]: rt = pd.DataFrame([[1, 2], [100, 200]],
                          index = list('ac'),
                          columns = list('AC'))
```

rt

Out[30]:

	A	C
a	1	2
c	100	200

```
In [31]: lt.merge(rt)
```

Out[31]:

	A	B	C
0	1	2	2

```
In [32]: lt.merge(rt, how='outer')
```

Out[32]:

	A	B	C
0	1	2.0	2.0
1	10	20.0	NaN
2	100	NaN	200.0

```
In [33]: lt.merge(rt, how='left')
```

Out[33]:

	A	B	C
0	1	2	2.0
1	10	20	NaN

```
In [34]: lt.merge(rt, how='right')
```

Out[34]:

	A	B	C
0	1	2.0	2
1	100	NaN	200

공통

모두합

왼쪽

오른쪽