파이썬 라이브러리를 활용한 데이터 분석

8장 데이터 준비하기: 조인, 병합, 변형

2020.07.02 2h

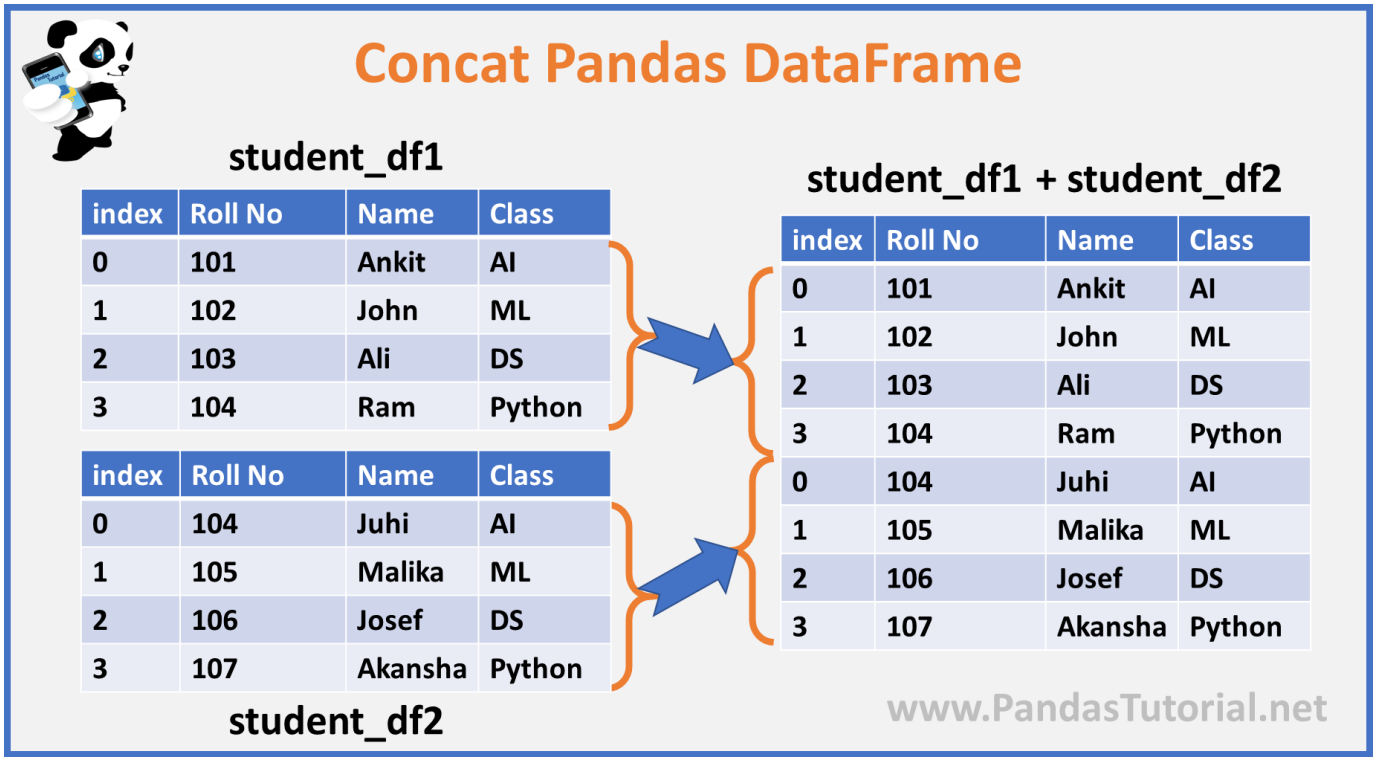
8장 데이터 준비하기: 조인, 병합, 변형

concat

append

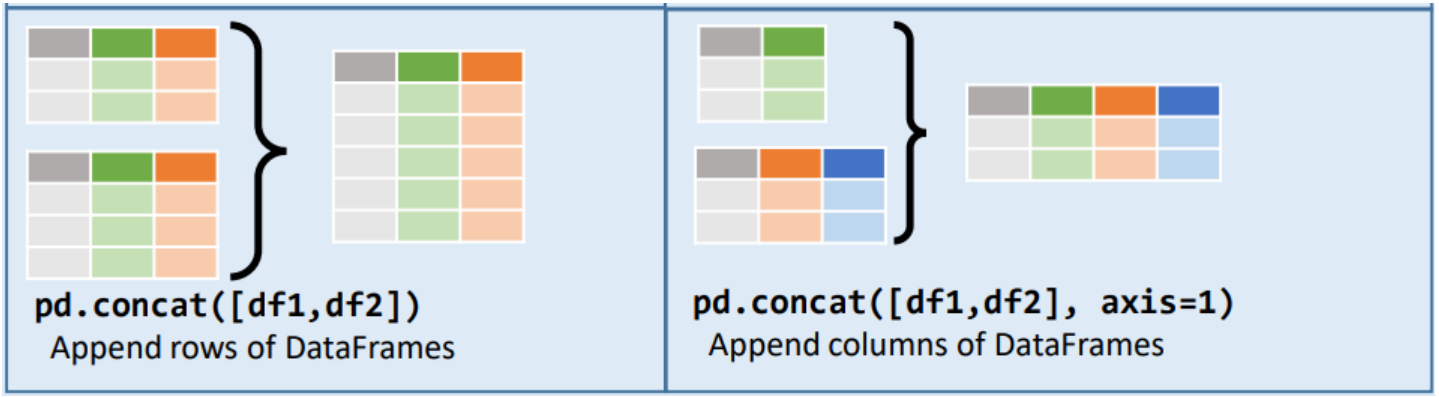
Pandas concat()

- '이어 붙이기'
  - 두 데이터프레임을 무조건 세로로 '이어 붙이기'
    - **구조가 같으면 이해가 매우 쉬움**



8.2 축 따라 이어 붙이기

- **이어 붙이기(concatenate)**
  - Pandas에서는 concate()
  - Numpy의 concatenate() 함수
- **제일 먼저 고민**
  - 기본은 행을 따라 붙이기
  - 어느 축을 따라 붙일 것인가?
    - **행 또는 열**
    - **기본인 인자 axis=0**
      - **세로로 행을 따라 붙이기**
    - **axis=1**
      - **가로로 열을 따라 붙이기**



기본은 축: 행 중심

- 기본은 행(세로)으로 연결

Concatenating Along an Axis

```
In [50]: arr = np.arange(12).reshape((3, 4))
         arr
Out[50]: array([[ 0,  1,  2,  3],
                [ 4,  5,  6,  7],
                [ 8,  9, 10, 11]])

In [51]: np.concatenate([arr, arr], axis=1)
Out[51]: 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]])
```

```
In [53]: s1 = pd.Series([0, 1], index=['a', 'b'])
         s1
Out[53]: a    0
         b    1
         dtype: int64

In [54]: s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
         s2
Out[54]: c    2
         d    3
         e    4
         dtype: int64

In [55]: s3 = pd.Series([5, 6], index=['f', 'g'])
         s3
Out[55]: f    5
         g    6
         dtype: int64

In [56]: pd.concat([s1, s2, s3])
Out[56]: a    0
         b    1
         c    2
         d    3
         e    4
         f    5
         g    6
         dtype: int64
```

인자 axis=1 p327

- **무조건 이어 붙이는 방식**
  - 없는 값은 pd.na

```
In [53]: s1 = pd.Series([0, 1], index=['a', 'b'])
         s1

Out[53]: a    0
         b    1
         dtype: int64
```

```
In [54]: s2 = pd.Series([2, 3, 4], index=['c', 'd', 'e'])
         s2

Out[54]: c    2
         d    3
         e    4
         dtype: int64
```

```
In [55]: s3 = pd.Series([5, 6], index=['f', 'g'])
         s3

Out[55]: f    5
         g    6
         dtype: int64
```

축으로는
무조건 붙이기

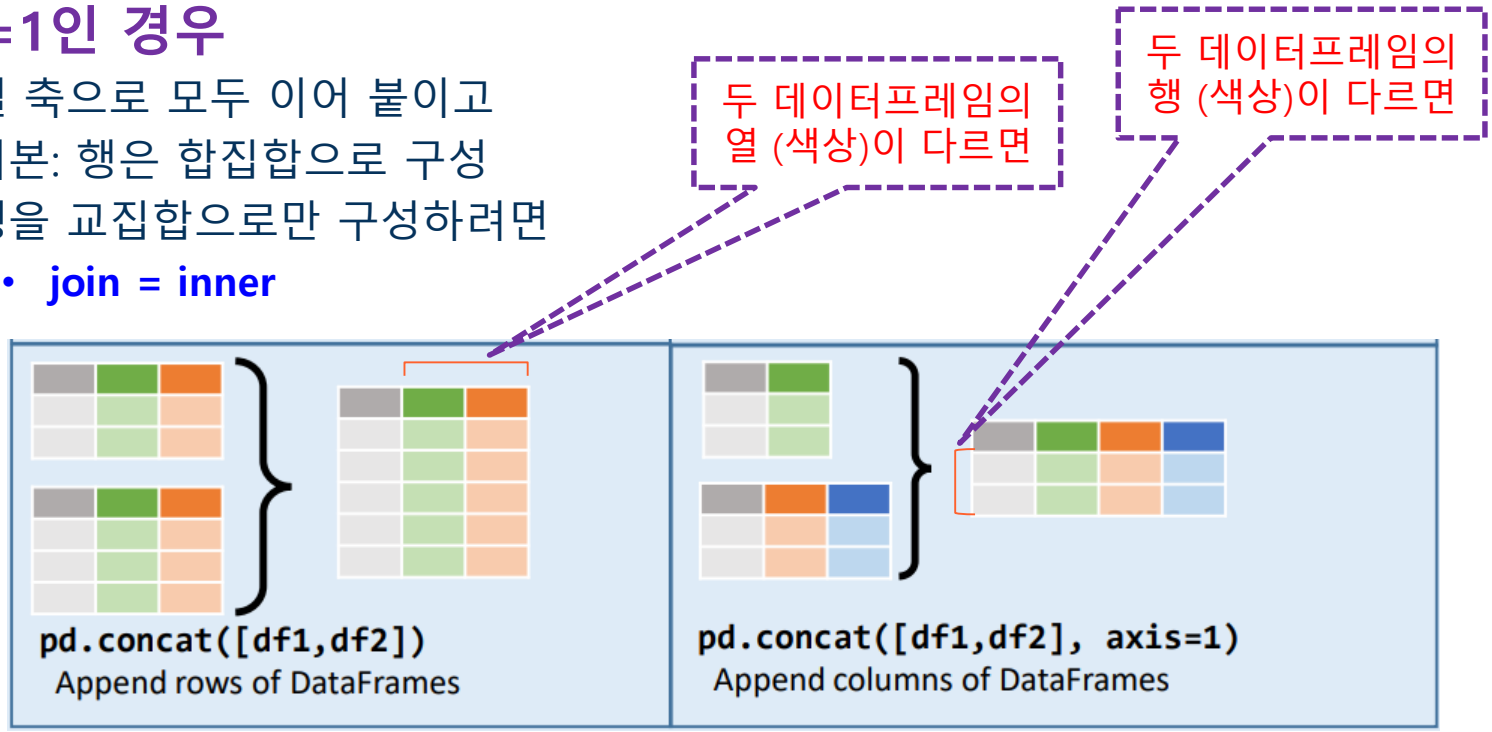
```
In [57]: pd.concat([s1, s2, s3], axis=1)
```

Out[57]:

	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

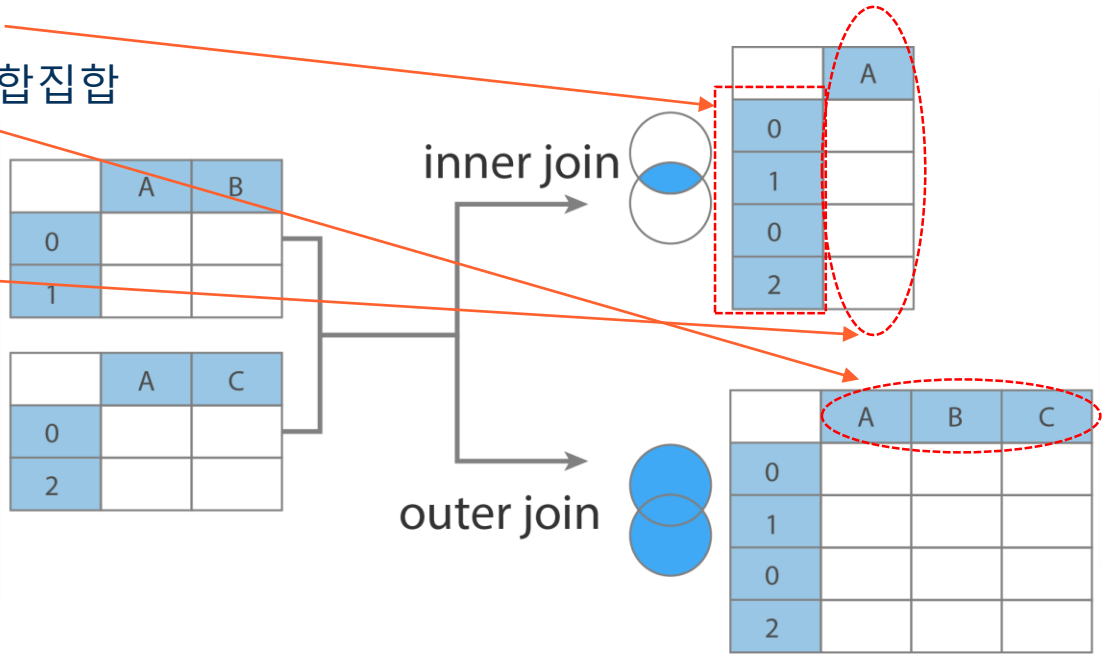
이어 붙인 이후에 겹치는 행과 열을 처리 방법?

- **axis=0인 경우**
  - 행 축으로 모두 이어 붙이고
  - 기본: 칼럼은 합집합으로 구성
  - 칼럼을 교집합으로만 구성하려면
    - **join = inner**
- **axis=1인 경우**
  - 열 축으로 모두 이어 붙이고
  - 기본: 행은 합집합으로 구성
  - 행을 교집합으로만 구성하려면
    - **join = inner**



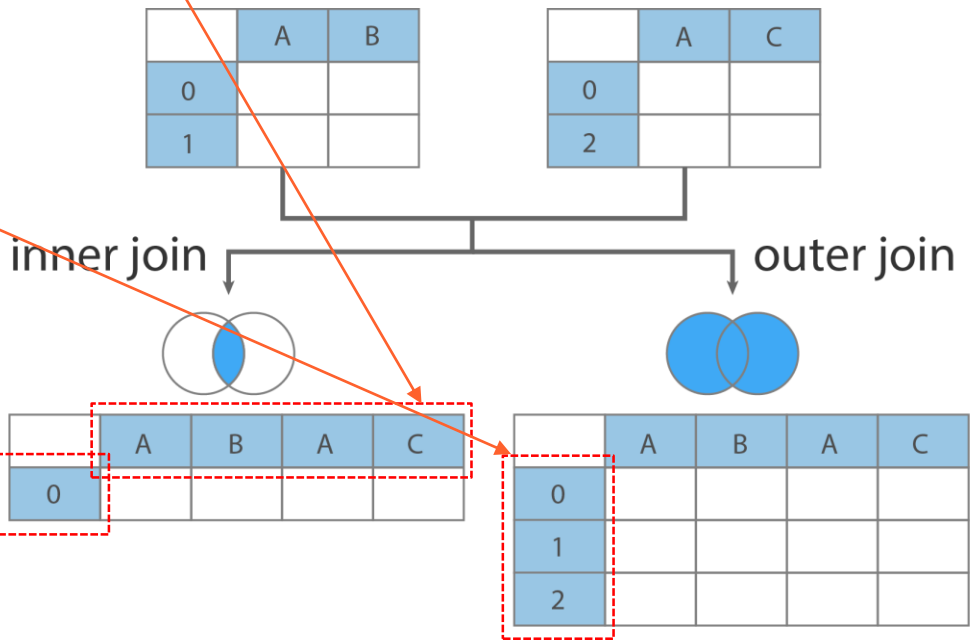
concat axis=0

- **pandas.concat(...)**
  - pandas.concat(objs, axis=0, join='outer', join_axes=None, ignore_index=False, keys=None, levels=None, names=None, verify_integrity=False, sort=None, copy=True)
- **합치는 축. 기본이 세로, 행 축에 따라 무조건 '이어 붙인다'라는 개념**
  - axis=0
    - **인덱스(행)은 모두 추가**
  - 열 기본(join=outer): 열도 합집합
    - **공통 열은 하나로**
    - **다른 열은 모두 추가**
  - 옵션 join=inner
    - **공통인 열만**



concat axis=1

- 합치는 축, 열에 따라, 가로로 무조건 '이어 붙이고'
  - axis=1
  - 열은 모든 열을 추가하고
- 행의 조인 방법 지정
  - join=inner
    - 행은 공통인 것만 선택
  - join=outer, 기본
    - 행은 공통인 것은 하나로
    - 다른 것은 모두 추가



시리즈의 concat

- **기준(key)을 사용하지 않고 단순히 데이터를 연결(concatenate)**
  - 기본적으로는 위/아래로 데이터 행을 연결
    - **단순히 두 시리즈나 데이터프레임을 연결하기 때문에 인덱스 값이 중복 가능**

In [23]:

```
s1 = pd.Series([0, 1], index=['A', 'B'])
s2 = pd.Series([2, 3, 4], index=['A', 'B', 'C'])
```

In [24]:

```
s1
```

```
A    0
B    1
dtype: int64
```

In [25]:

```
s2
```

```
A    2
B    3
C    4
dtype: int64
```

In [26]:

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

```
A    0
B    1
A    2
B    3
C    4
dtype: int64
```

기본은 세로로 붙이기

- 기본, axis=1

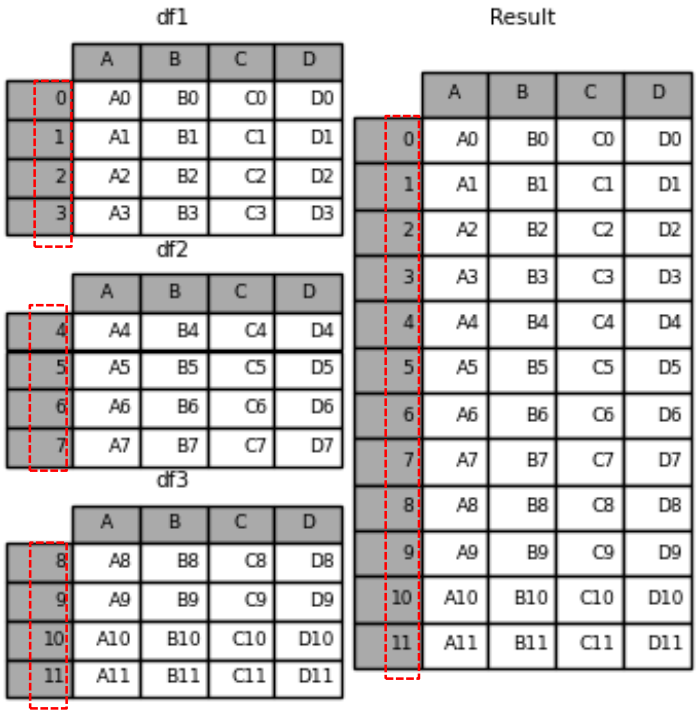
```
In [1]: df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
   ...:                     'B': ['B0', 'B1', 'B2', 'B3'],
   ...:                     'C': ['C0', 'C1', 'C2', 'C3'],
   ...:                     'D': ['D0', 'D1', 'D2', 'D3']},
   ...:                    index=[0, 1, 2, 3])
   ...:

In [2]: df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
   ...:                     'B': ['B4', 'B5', 'B6', 'B7'],
   ...:                     'C': ['C4', 'C5', 'C6', 'C7'],
   ...:                     'D': ['D4', 'D5', 'D6', 'D7']},
   ...:                    index=[4, 5, 6, 7])
   ...:

In [3]: df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
   ...:                     'B': ['B8', 'B9', 'B10', 'B11'],
   ...:                     'C': ['C8', 'C9', 'C10', 'C11'],
   ...:                     'D': ['D8', 'D9', 'D10', 'D11']},
   ...:                    index=[8, 9, 10, 11])
   ...:

In [4]: frames = [df1, df2, df3]

In [5]: result = pd.concat(frames)
```



axis=1

- 만약 옆으로 데이터 열을 연결하고 싶으면
  - axis=1로 인수를 설정
    - 다른 것은 모두 추가
  - 행은 기본이 join='outer'
    - 행은 공통인 것은 하나, 나머지는 모두 추가

In [27]:

```
df1 = pd.DataFrame(
    np.arange(6).reshape(3, 2),
    index=['a', 'b', 'c'],
    columns=['데이터1', '데이터2'])
df1
```

	데이터1	데이터2
a	0	1
b	2	3
c	4	5

In [28]:

```
df2 = pd.DataFrame(
    5 + np.arange(4).reshape(2, 2),
    index=['a', 'c'],
    columns=['데이터3', '데이터4'])
df2
```

	데이터3	데이터4
a	5	6
c	7	8

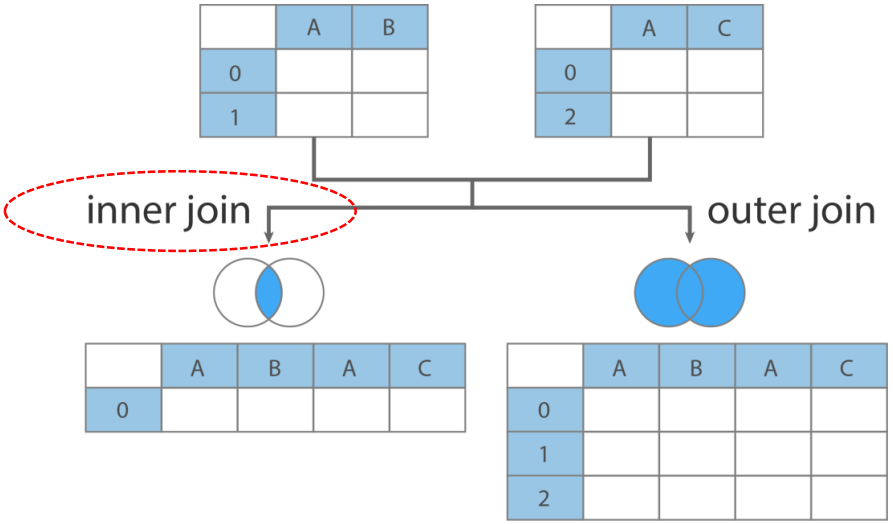
In [29]:

```
pd.concat([df1, df2], axis=1)
```

	데이터1	데이터2	데이터3	데이터4
a	0	1	5.0	6.0
b	2	3	NaN	NaN
c	4	5	7.0	8.0

인자 join

- **axis=1**
  - 가로로 붙이기
  - 기본 outer
  - join=inner
    - **공통인 인덱스만**




```
In [65]: s1
Out[65]: a    0
         b    1
         dtype: int64

In [67]: s4
Out[67]: a    0
         b    1
         f    5
         g    6
         dtype: int64

In [70]: pd.concat([s1, s4], axis=1)
Out[70]:
```

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

```
In [69]: pd.concat([s1, s4], axis=1, join='inner')
Out[69]:
```

	0	1
a	0	0
b	1	1

인자 keys=

- **이어 붙인 축에 대한 색인을 생성**
  - 계층적 색인이 됨

```
In [148]: result = pd.concat([s1, s1, s3], keys=['one', 'two', 'three'])
          result

Out[148]: one    a    0
               b    1
          two  a    0
               b    1
          three f    5
                g    6
          dtype: int64
```

```
In [149]: result.unstack()
```

Out[149]:

	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

인자 ignore_index

- 기본은 False
  - ignore_index=true
    - 이어 붙인 축의 색인을 유지하지않고 새로운 색인을 생성

```
In [80]: df1 = pd.DataFrame(np.random.randn(3, 4), columns=['a', 'b', 'c', 'd'])
         df1
```

Out[80]:

	a	b	c	d
0	-0.204708	0.478943	-0.519439	-0.555730
1	1.965781	1.393406	0.092908	0.281746
2	0.769023	1.246435	1.007189	-1.296221

```
In [81]: df2 = pd.DataFrame(np.random.randn(2, 3), columns=['b', 'd', 'a'])
         df2
```

Out[81]:

	b	d	a
0	0.274992	0.228913	1.352917
1	0.886429	-2.001637	-0.371843

```
In [82]: pd.concat([df1, df2], ignore_index=True)
```

Out[82]:

	a	b	c	d
0	-0.204708	0.478943	-0.519439	-0.555730
1	1.965781	1.393406	0.092908	0.281746
2	0.769023	1.246435	1.007189	-1.296221
3	1.352917	0.274992	NaN	0.228913
4	-0.371843	0.886429	NaN	-2.001637

```
In [83]: pd.concat([df1, df2], ignore_index=False)
```

Out[83]:

	a	b	c	d
0	-0.204708	0.478943	-0.519439	-0.555730
1	1.965781	1.393406	0.092908	0.281746
2	0.769023	1.246435	1.007189	-1.296221
0	1.352917	0.274992	NaN	0.228913
1	-0.371843	0.886429	NaN	-2.001637

```
In [84]: pd.concat([df1, df2])
```

Out[84]:

	a	b	c	d
0	-0.204708	0.478943	-0.519439	-0.555730
1	1.965781	1.393406	0.092908	0.281746
2	0.769023	1.246435	1.007189	-1.296221
0	1.352917	0.274992	NaN	0.228913
1	-0.371843	0.886429	NaN	-2.001637

df1.append(df2)

- **df1과 df2를 행에 따라 붙이기**
  - df1에 없는 열은 추가
  - Pandas concat()와 동일, axis=0

```
In [176]: df = pd.DataFrame([[1, 2], [3, 4]], columns=list('AB'))
          df
```

Out[176]:

	A	B
0	1	2
1	3	4

```
In [180]: df2 = pd.DataFrame([[5, 6], [7, 8]], columns=list('AB'))
          df2
```

Out[180]:

	A	B
0	5	6
1	7	8

```
In [182]: df.append(df2)
```

Out[182]:

	A	B
0	1	2
1	3	4
0	5	6
1	7	8

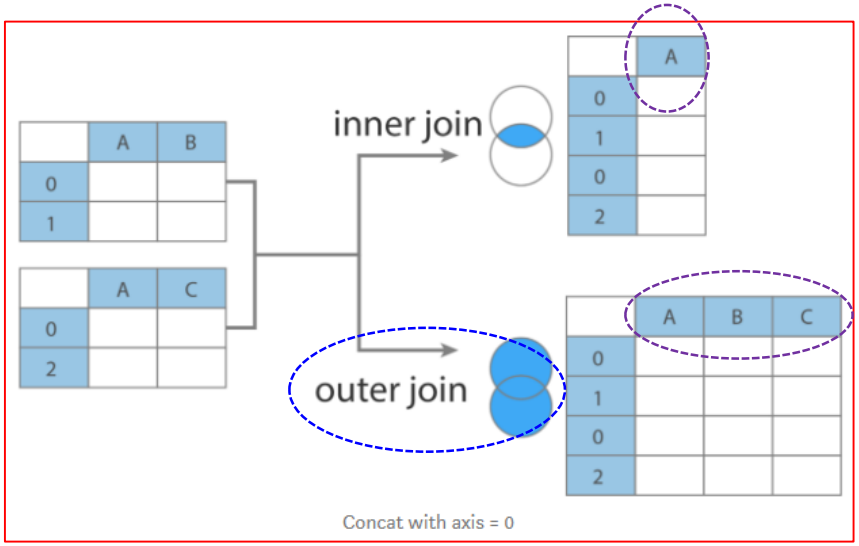
```
In [184]: pd.concat([df, df2])
```

Out[184]:

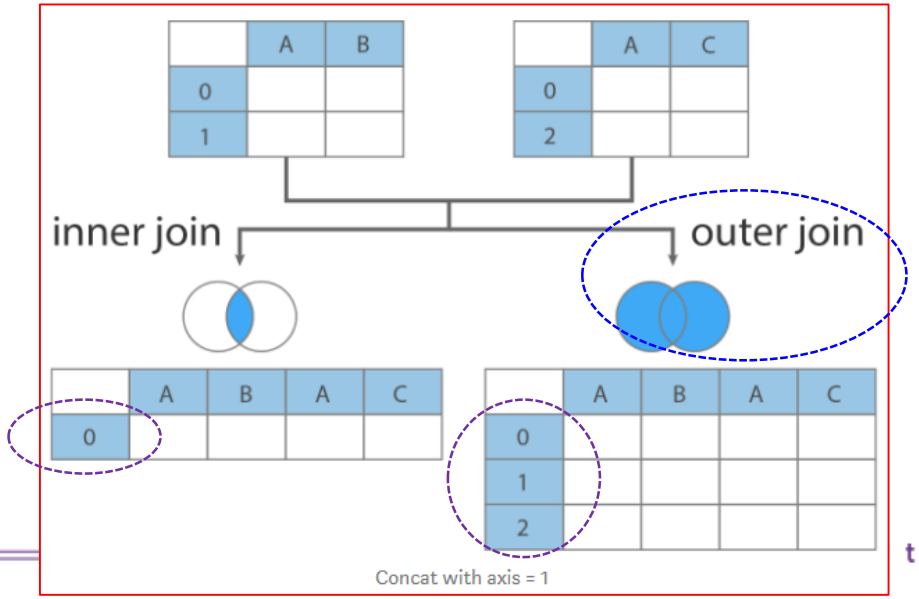
	A	B
0	1	2
1	3	4
0	5	6
1	7	8

이어 붙이기 concat() 요약

- **기본 축 axis=0**
  - 행을 모두 이어 붙이고
  - 열은 기본이 합집합으로
    - **옵션 join='outer'**
  - 공통인 열만
    - **옵션 join='inner'**



- **축 axis=1**
  - 열을 모두 이어 붙이고
  - 행은 기본이 합집합으로
    - **옵션 join='outer'**
  - 공통인 행만
    - **옵션 join='inner'**

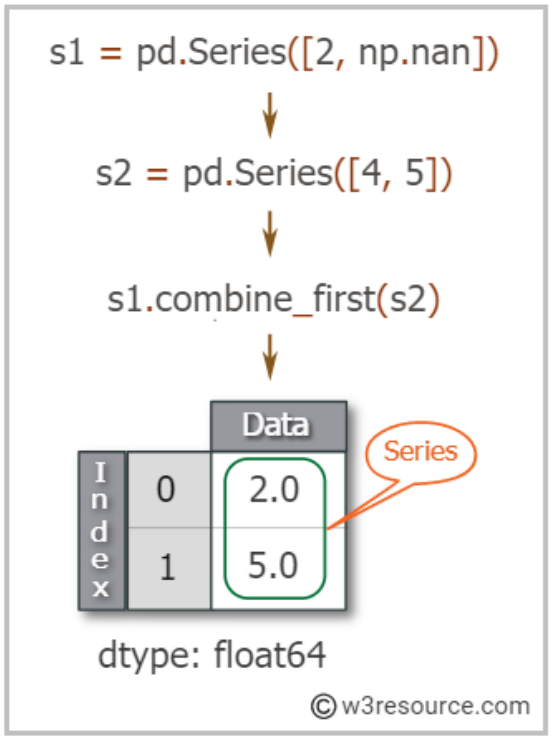


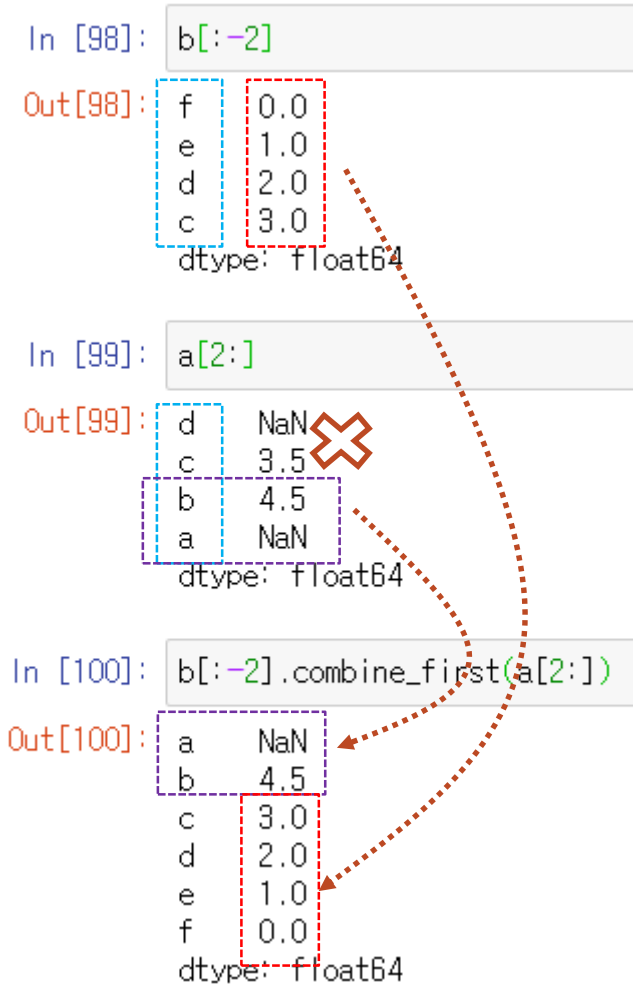
8장 데이터 준비하기: 조인, 병합, 변형

combine_first

겹치는 데이터 합치기

- **a.combine_first(b)**
  - 인덱스에 맞게(행에 대해)
    - **a가 na 아니면 자신, na이면 b,**
    - **남은 b의 것 추가하여, 인덱스로 정렬**





데이터프레임 행과 열에 대해서도 작동

In [101]:
```
df1 = pd.DataFrame({'a': [1., np.nan, 5., np.nan],
                    'b': [np.nan, 2., np.nan, 6.],
                    'c': range(2, 18, 4)})
df1
```

Out[101]:

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

In [102]:
```
df2 = pd.DataFrame({'a': [5., 4., np.nan, 3., 7.],
                    'b': [np.nan, 3., 4., 6., 8.]})
df2
```

Out[102]:

	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

In [103]:
```
df1.combine_first(df2)
```

Out[103]:

	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