데이터세트 결합:

데이터의 가장 흥미로운 연구 중 일부는 서로 다른 데이터 소스를 결합하는 데서 나온다. 이 연산들은 두 개의 다른 데이터를 매우 간단하게 연결하는 것부터 데이터 간 겹치는 부분을 제대로 처리하는 복잡한 데이터베이스 스타일을 조인하고 병합하는 것까지 다양하게 사용될 수 있다. Series 와 DataFrame 은 이 유형의 연산을 염두에 두고 만들어진 것으로, Pandas 는 이러한 유형의 데이터 랭글링(data wrangling)을 빠르고 간단하게 할 수 있 는 함수와 메서드를 제공한다.

여기서는 pd.concat 함수를 이용한 Series 와 DataFrame 의 간단한 연결에 대해 알아볼 것이며, 이어서 Pandas 에 구현된 더 복잡한 인메모리 병합과 조인에 대해 자세히 살펴보겠다.

```
In [1]:import pandas as pd
       import numpy as np
```

편의상 앞으로 사용할 특정 형태의 `DataFrame` 을 생성하는 함수를 다음과 같이 정의하겠다.

```
In [2]:def make_df(cols, ind):
           """Quickly make a DataFrame"""
           data = {c: [str(c) + str(i) for i in ind]
                   for c in cols}
           return pd.DataFrame(data, ind)
# DataFrame 예제
       make_df('ABC', range(3))
```

Out[2]:

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2

복습: NumPy 배열 연결

Series 와 DataFrame 객체의 연결은 NumPy 배열의 기초에서 살펴본 np.concatenate 함수를 이용하면 두 개이상의 배열의 콘텐츠를 하나의 배열로 결합할 수 있다는 점을 기억하자.

```
In [4]:x = [1, 2, 3]
       y = [4, 5, 6]
       z = [7, 8, 9]
       np.concatenate([x, y, z])
Out[4]:array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

첫번째 인수는 연결할 배열의 리스트나 튜플이다. 게다가 axis 키워드를 사용해 결과를 어느 축에 따라 연결할 것인지 지정할 수 있다.

```
In [5]:x = [[1, 2],
            [3, 4]]
        np.concatenate([x, x], axis=1)
Out[5]:array([[1, 2, 1, 2],
              [3, 4, 3, 4]])
```

`pd.concat` 을 이용한 간단한 연결

Pandas 에는 np.concatenate 와 구분이 매우 비슷하지만 다양한 옵션을 가진 pd.concat()함수가 있다. 옵션에 대해서는 잠시 후 알아보겠다.

```
# Signature in Pandas v0.18

pd.concat(objs, axis=0, join='outer', join_axes=None, ignore_index=False,

          keys=None, levels=None, names=None, verify_integrity=False,

          copy=True)
```

np.concatenate()를 배열을 간단하게 연결하는 데 사용할 수 있는 것처럼 pd.concat()은 Series 나 DataFrame 객체를 간단하게 연결할 때 사용할 수 있다.

```
In [6]:ser1 = pd.Series(['A', 'B', 'C'], index=[1, 2, 3])
       ser2 = pd.Series(['D', 'E', 'F'], index=[4, 5, 6])
       pd.concat([ser1, ser2])
Out[6]:1    A
       2    B
       3    C
       4    D
       5    E
       6    F
       dtype: object
```

pd.concat()을 이용하면 `DataFrames` 같은 고차원 객체를 연결할 수도 있다.

```
In [7]:df1 = make_df('AB', [1, 2])
       df2 = make_df('AB', [3, 4])
       print(df1);print( df2); print(pd.concat([df1, df2]))
Out[7]:df1
```

	A	B
1	A1	B1

	A	B
2	A2	B2

df2

	A	B
3	A3	B3
4	A4	B4

```
pd.concat([df1, df2])
```

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4

기본적으로 연결은 DataFrame 내에서 행 단위(즉, axis = 0)로 일어난다. np.concatenate 처럼 pd.concat 도 어느 축을 따라 연결할 것인지 지정할 수 있다. 아래 예제를 생각해보자

```
In [8]:df3 = make_df('AB', [0, 1])
       df4 = make_df('CD', [0, 1])
       print(df3); print(df4); print(pd.concat([df3, df4], axis=1))
Out[8]:df3
```

	A	B
0	A0	B0
1	A1	B1

df4

	C	D
0	C0	D0
1	C1	D1

```
pd.concat([df3, df4], axis=1)
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1

인덱스 복제

np.concaenate 와 pd.concat 의 중요한 차이는 Pandas 에서의 연결은 그 결과가 복제된 인덱스를 가지더라도 인덱스를 유지한다는데 있다.:

```
In [9]:x = make_df('AB', [0, 1])
        y = make_df('AB', [2, 3])
        y.index = x.index  # 복제 인덱스 생성
        print(x);print(y); print(pd.concat([x, y]))
Out[9]:
x
    A   B
0   A0  B0
1   A1  B1
y
    A   B
0   A2  B2
1   A3  B3
pd.concat([x, y])
    A   B
0   A0  B0
1   A1  B1
0   A2  B2
1   A3  B3
```

결과에서 인덱스가 반복되는 것을 주목하자. 이것은 DataFrame 내에서는 유효하지만 결과가 바람직하지 않은 경우가 종종 있다. pd.concat()은 이 문제를 처리 하는 몇 가지 방법을 제공한다.

반복을 에러로 잡아낸다.

pd.concat()의 결과에서 인덱스가 겹치지 않는지 간단히 검증하고 싶으면 verify_integrity 프래그를 지정하면 된다. 이 플래그를 True 로 설정하면 연결 작업에서 중복 인덱스가 있을 때 예외가 발생한다. 확인을 위해 오류를 잡아내고 메세지를 출력하는 다음 예제를 보자.

```
In [10]:try:
            pd.concat([x, y], verify_integrity=True)
```

```
        except ValueError as e:
            print("ValueError:", e)
ValueError: Indexes have overlapping values: [0, 1]
```

인덱스를 무시한다.

인덱스 자체가 중요하지 않은 경우에는 그냥 인덱스를 무시하고 싶을 것이다. Ignore_index 플래그를 사용해 이 옵션을 지정하면 된다. 이 플래그를 True 로 설정하면 연결 작업에서 중복 인덱스가 있을 때 예의가 발생한다. 확인을 위해 오류를 잡아내고 메시지를 출력하는 다음 예제를 보자.

```
In [11]: print(x); print(y); print(pd.concat([x, y], ignore_index=True))
Out[11]:x
```

	A	B
0	A0	B0
1	A1	B1

```
y
```

	A	B
0	A2	B2
1	A3	B3

```
pd.concat([x, y], ignore_index=True)
```

	A	B
0	A0	B0
1	A1	B1
2	A2	B2
3	A3	B3

다중 인덱스 키를 추가한다.

또 다른 방법은 데이터 소스에 대한 레이블 지정하는데 keys 옵션을 사용하는 것이다. 결과는 그 데이터를 포함하는 계층적 인덱스를 자신 시리즈가 될 것이다.

```
In [12]: print(x); print(y); print(pd.concat([x, y], keys=['x', 'y']))
Out[12]:x
```

	A	B
0	A0	B0
1	A1	B1

```
y
```

	A	B
0	A2	B2
1	A3	B3

```
pd.concat([x, y], keys=['x', 'y'])
```

		A	B
x	0	A0	B0
	1	A1	B1
y	0	A2	B2
	1	A3	B3

결과는 다중 인덱스를 가지는 DataFrame 이며 '계층적 인덱싱'에서 살펴본 도구를 사용해 이 데이터를 관심 있는 표현 방식으로 전환할 수 있다.

조인을 이용한 연결

방금 살펴본 간단한 예제에서는 주로 공유된 열이름으로 DataFrame 을 연결했다. 실무에서는 다른 소스에서 가져온 데이터는 다른 열 이름 집합을 가질 수도 있는데, pd.concat 이 이 경우를 위한 몇 가지 옵션을 제공한다. 다음과 같이 공통적인 열 몇개(전부 아님)를 가지고 있는 두개의 DataFrame 을 연결하는 것을 생각해 보자.

```
In [13]: df5 = make_df('ABC', [1, 2])
         df6 = make_df('BCD', [3, 4])
         print(df5); print(df6); print(pd.concat([df5, df6]))
Out[13]: df5
```

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

```
df6
```

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

```
pd.concat([df5, df6])
```

	A	B	C	D
1	A1	B1	C1	NaN

	A	B	C	D
2	A2	B2	C2	NaN
3	NaN	B3	C3	D3
4	NaN	B4	C4	D4

채울 값이 없는 항목은 기본적으로 NA 값으로 채워진다. 이 값을 바꾸려면 연결 함수의 join 과 join_axes 매개변수에 대한 여러 옵션 중 하나를 지정하면 된다. 기본적으로 조인은 입력 열의 합집합(join = 'outer')이지만, join = 'inner'를 사용해 이를 열의 교집합으로 변경할 수 있다.

```
In [14]: print(df5); print(df6); print(pd.concat([df5, df6], join='inner'))
Out[14]:df5
```

	A	B	C
1	A1	B1	C1
2	A2	B2	C2

```
   df6
```

	B	C	D
3	B3	C3	D3
4	B4	C4	D4

```
   pd.concat([df5, df6], join='inner')
```

	B	C
1	B1	C1
2	B2	C2
3	B3	C3
4	B4	C4

`append()` 메서드

배열을 직접 연결하는 것이 매우 일반적이라서 Series 와 DataFrame 객체는 더 작은 키 입력으로 똑같은 작업을 수행할 수 있는 append 메서드를 가지고 있다. 예를 들어, pd.concat([df1, df2])를 호출하지 않고 간단하게 df1.append(df2) 를 호출한다.

```
In [16]: print(df1); print(df2); print(df1.append(df2))
Out[16]:df1
```

	A	B
1	A1	B1
2	A2	B2

```
df2
```

	A	B
3	A3	B3
4	A4	B4

```
df1.append(df2)
```

	A	B
1	A1	B1
2	A2	B2
3	A3	B3
4	A4	B4

파이썬 리스트의 append(), extend()메서드와 달리 Pandas 의 append()메서드는 원래의 객체를 변경하지 않는 대신 결합된 데이터를 가지는 새로운 객체를 만든다는 사실을 유념하자. 이 방법 역시 새 인덱스와 데이터 버퍼를 생성하기 때문에 매우 효율적인 방식이라고 보기 어렵다. 따라서 append 연산을 여러번 수행할 계획이라면 일반적으로 DataFrame 의 목록을 만들고 그것들을 concat()함수에 한 번에 전달하는 것이 바람직하다.