데이터세트 결합하기: 병합과 조인

Pandas 가 제공하는 기본 기능의 하나는 고성능 인메모리 조인과 병합 연산이다. 데이터베이스로 작업해 본 경험이 있다면 이러한 유형의 데이터 작업에 익숙할 것이다. 이를 위한 주요 인터페이스는 pd.merge 함수이며, 뒤에서 예제 몇개를 통해 이 함수가 실제로 어떻게 동작하는지 살펴본다.

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

관계 대수

pd.merge()에는 관계 데이터(relational data)를 조작하는 규칙의 정형집합이자 대부분의 데이터베이스에서 사용할 수 있는 연산의 개념적 기반을 현성하는 관계 대수(relational algebra)의 하위 집합에 해당하는 행위가 구현돼 있다. 관계 대수 방식의 강점은 그것이 데이터베이스나 다른 프로그램에서 효율적으로 구현된 기초 연산의 어휘를 사용하면 매우 복잡한 작업을 다양하게 수행할 수 있다.

Pandas 에는 Series 와 DataFrame 의 pd.merge()함수의 이와 관련된 join()메서드의 기본 구성요소가 몇가지 구현돼 있다. 이것들을 이용하면 서로 다른 소스에서 나온 데이터를 효율적으로 연결할 수 있다.

조인 작업의 분류

pd.merge() 함수는 일대일, 다대일, 다대다 조인 같은 여러 가지 조인 유형을 구현한다. 이 세가지 유형의 조인은 모두 pd.merge() 인터페이스에서 동일한 호출을 통해 사용할 수 있다. 수행하는 조인의 유형은 입력 데이터의 형태에 따라 다르다. 여기서는 세가지 유형의 병합에 간단한 예제를 보여주고 이어서 자세한 옵션을 알아보겠다.

일 대 일 조인

아마 가장 간단한 유형의 병합 표현식은 '데이터 세트 결합: concat 과 Append 에서 본 열 단위의 연결과 여러면에서 매우 유사한 일 대 일 조인일 것이다. 구체적인 예로, 회사의 직원 몇 명에 대한 정보를 포함하는 다음 두개의 DataFrame 을 생각해보자.

```
In [2]:df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                           'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
        df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                            'hire_date': [2004, 2008, 2012, 2014]})
        print(df1); print(df2)
Out[2]:df1
```

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

```
df2
```

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

이 정보를 하나의 DataFrame 으로 결합하려면 pd.merge()함수를 사용하면된다.

```
In [3]:df3 = pd.merge(df1, df2)
        df3
Out[3]:
```

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

pd.merge()함수는 각 DataFrame 이 'employee'열을 가지고 있다 는 것을 알고 자동으로 이 열을 키로 사용해 조인한다. 병합 결과로 두 입력값으로부터 얻은 정보를 결합한 새로운 DataFrame 을 얻게 된다. 각 열의 항목 순서가 반드시 유지되는 것은 아니다. 이 경우, df1 과 df2 에서 'employee'열의 순서가 다른데, pd.merge()함수가 정확하게 이를 맞추어 연산한다. 아울러 병합은 인덱스 별로 병합하는 특별한 경우를 제외하고는 일반적으로 인덱스를 버린다는 점을 명심하자.

다대일(Many-to-one) 조인

다대일 조인은 두 개의 키 열 중 하나가 중복된 항목을 포함하는 경우의 조인을 의미한다. 다대일 조인의 경우 결과 DataFrame 은 이 중복 항목을 타당한 것으로 보존한다.:

```
In [4]:df4 = pd.DataFrame({'group': ['Accounting', 'Engineering', 'HR'],
                           'supervisor': ['Carly', 'Guido', 'Steve']})
```

```
print(df3); print(df4); print(pd.merge(df3,df4));
```

```
Out[4]:df3
```

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

```
df4
```

	Group	supervisor
0	Accounting	Carly
1	Engineering	Guido
2	HR	Steve

```
pd.merge(df3, df4)
```

	employee	group	hire_date	supervisor
0	Bob	Accounting	2008	Carly
1	Jake	Engineering	2012	Guido
2	Lisa	Engineering	2004	Guido
3	Sue	HR	2014	Steve

결과 DataFrame 에는 'supervisor'정보를 담고 있는 추가 열이 있는데, 그 정보는 입력값에 따라 하나 이상의 위치에 반복해서 등장한다.

다대다(Many-to-many) 조인

다대다 조인은 개념적으로 조금 혼란을 주지만 그래도 잘 정의돼 있다. 왼쪽과 오른쪽 배열의 키 열에 모두 중복 항목이 존재하면 결과는 다대다 병합이 된다. 구체적인 예제를 보면 잘 이해할 수 있을 것이다.

특정 그룹과 연결된 하나 이상의 기술을 보여주는 다음 DataFrame 을 생각해보자.

```
In [5]:df5 = pd.DataFrame({'group': ['Accounting', 'Accounting',
                                     'Engineering', 'Engineering', 'HR', 'HR'],
                           'skills': ['math', 'spreadsheets', 'coding', 'linux',
                                      'spreadsheets', 'organization']})
print(df1); print(df5); print(pd.merge(df1,df5))
```

```
Out[5]:df1
```

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

```
df5
```

	group	skills
0	Accounting	math
1	Accounting	spreadsheets
2	Engineering	coding
3	Engineering	linux
4	HR	spreadsheets
5	HR	organization

```
pd.merge(df1, df5)
```

	employee	group	skills
0	Bob	Accounting	math
1	Bob	Accounting	spreadsheets
2	Jake	Engineering	coding
3	Jake	Engineering	linux
4	Lisa	Engineering	coding
5	Lisa	Engineering	linux
6	Sue	HR	spreadsheets
7	Sue	HR	organization

이 세가지 유형의 조인은 다른 Pandas 도구와 함께 사용해 다양한 기능을 구현할 수 있다. 하지만 실제 데이터세트가 여기서 다룬 것만큼 깨끗한 경우는 드물다.

병합키 지정

앞에서 pd.merge()의 기본 동작 방식을 알아봤다. pd.merge()는 두개의 입력 값 사이에 일치하는 하나 이상의 열 이름을 찾아 그것을 키로 사용한다. 그러나 열 이름이 그렇게 잘 일치하는 경우는 흔하지 않으며, pd.merge()가 이 문제를 처리하기 위한 다양한 옵션을 제공한다.

on 키워드

가장 간단한 방법은 열 이름이나 열 이름의 이스트를 취하는 on 키워드를 사용해 키 열의 이름을 명시적으로 지정하는 것이다.

```
In [6]: print(df1); print(df2); print(pd.merge(df1, df2, on='employee'))
Out[6]:df1
```

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

```
df2
```

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

```
pd.merge(df1, df2, on='employee')
```

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

이 옵션은 왼쪽과 오른쪽 DataFrame 이 모두 지정된 열 이름을 가진 경우에만 동작한다.

Left_on 과 right_on 키워드

때로는 다른 열 이름을 가진 두 테이터세트를 병합하고 싶을 수도 있다. 예를 들면, 직원 이름 레이블이 'employee'가 아니라 'name'인 데이터세트를 가지고 있는 경우가 그렇다. 이 경우, left_on 과 right_on 키워드를 사용해 두열 이름을 지정할 수 있다.:

```
In [7]:df3 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                           'salary': [70000, 80000, 120000, 90000]})
        print(df1);      print(df3);      print(pd.merge(df1,      df3,      left_on="employee",
right_on="name"))
Out[7]:df1
```

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

```
df3
```

	name	salary
0	Bob	70000
1	Jake	80000
2	Lisa	120000
3	Sue	90000

```
pd.merge(df1, df3, left_on="employee", right_on="name")
```

	employee	group	name	salary
0	Bob	Accounting	Bob	70000
1	Jake	Engineering	Jake	80000
2	Lisa	Engineering	Lisa	120000
3	Sue	HR	Sue	90000

그 결과는 불필요하게 중복된 열을 갖게 되며, 원하는 경우 DataFrame 의 drop()메서드를 사용해 삭제할 수 있다.

```
In [8]:pd.merge(df1, df3, left_on="employee", right_on="name").drop('name', axis=1)
Out[8]:
```

	employee	group	salary
0	Bob	Accounting	70000
1	Jake	Engineering	80000
2	Lisa	Engineering	120000
3	Sue	HR	90000

`left_index` 와 `right_index` 키워드

때로는 열을 병합하는 대신 인덱스로 병합해야 하는 경우도 있다. 예를들어, 다음과 같은 데이터가 있다고 하자.:

```
In [9]:df1a = df1.set_index('employee')
        df2a = df2.set_index('employee')
               print(df1a); print(df2a)
```

```
Out[9]:df1a
```

employee	group
Bob	Accounting
Jake	Engineering
Lisa	Engineering
Sue	HR

```
      df2a
```

employee	hire_date
Lisa	2004
Bob	2008
Jake	2012
Sue	2014

여기서 `pd.merge()`의 left_iondex 나 right_index 를 지정해 병합 키로 인덱스를 사용할 수 있다.

```
In [10]: print(df1a); print(df2a)
```

```
print(pd.merge(df1a, df2a, left_index=True, right_index=True))
```

```
Out[10]:df1a
```

Employee	group
Bob	Accounting
Jake	Engineering
Lisa	Engineering
Sue	HR

```
df2a
```

Employee	hire_date
Lisa	2004
Bob	2008
Jake	2012
Sue	2014

```
pd.merge(df1a, df2a, left_index=True, right_index=True)
```

employee	group	hire_date
Lisa	Engineering	2004
Bob	Accounting	2008
Jake	Engineering	2012
Sue	HR	2014

편의를 위해 DataFrame 은 기본적으로 인덱스 기반으로 조인하는 병합을 수행하는 join()메서드를 구현하다.

```
In [11]:print(df1a);print(df2a); print(df1a.join(df2a))
Out[11]:df1a
```

Employee	group
Bob	Accounting
Jake	Engineering

	group
Employee	
Lisa	Engineering
Sue	HR

df2a

	hire_date
Employee	
Lisa	2004
Bob	2008
Jake	2012
Sue	2014

df1a.join(df2a)

	group	hire_date
employee		
Bob	Accounting	2008
Jake	Engineering	2012
Lisa	Engineering	2004
Sue	HR	2014

인덱스와 열을 섞고자 한다면 `left_index`를 `right_on`과 결합하거나 `left_on`을 `right_index`와 결합해 원하는 결과를 얻을 수 있다.

```
In [12]:print(df1a); print(df3);
        print(pd.merge(df1a, df3, left_index=True, right_on='name'))
```

Out[12]:df1a

	group
Employee	
Bob	Accounting
Jake	Engineering
Lisa	Engineering
Sue	HR

df3

	Name	salary
0	Bob	70000
1	Jake	80000
2	Lisa	120000
3	Sue	90000

```
pd.merge(df1a, df3, left_index=True, right_on='name')
```

	group	name	salary
0	Accounting	Bob	70000
1	Engineering	Jake	80000
2	Engineering	Lisa	120000
3	HR	Sue	90000

이 옵션 모두 다중 인덱스나 다중 열에서도 동작한다. 이 행위에 대한 인터페이스는 매우 직관적인다.

조인을 위한 집합 연산 지정하기

조인을 수행하는 데 있어한 가지 중요한 고려사항인 조인에 사용되는 집합연산 유형에 대해 넘어갔다. 이것은 어떤키 열이 등장하는 값이 다른키 열에서 는 등장하지 않는 경우의 문제가 된다.

```
In [13]:df6 = pd.DataFrame({'name': ['Peter', 'Paul', 'Mary'],
                            'food': ['fish', 'beans', 'bread']},
                                    columns=['name', 'food'])
        df7 = pd.DataFrame({'name': ['Mary', 'Joseph'],
                            'drink': ['wine', 'beer']},
                                    columns=['name', 'drink'])
        print(df6); print(df7); print(pd.merge(df6, df7))
Out[13]:df6
```

	name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

```
df7
```

	name	drink
0	Mary	wine

	name	drink
1	Joseph	beer

```
pd.merge(df6, df7)
```

	name	food	drink
0	Mary	bread	wine

여기서는 'name'에서 공통 항목으로 유일하게 Mary 를 가지고 있는 두 테이터세트를 병합했다. 기본적으로 결과에는 입력값의 두 집합에 대한 교집합이 들어가는데, 이것이 바로 내부 조인(inner join)이다. 이것은 기본적으로 'inner'로 설정된 how 키워드를 사용해 명시적으로 저장할 수 있다.

```
In [14]:pd.merge(df6, df7, how='inner')
Out[14]:
```

	name	food	drink
0	Mary	bread	wine

How 키워드의 다른 옵션으로 'outer', 'left','right'가 있다. 외부 조인(outer join)은 입력 데이터 열의 합집합으로 조인한 결과를 반환하고 누락된 값은 NA 로 채운다.

```
In [15]:print(df6); print(df7); print(pd.merge(df6, df7, how='outer'))
Out[15]:df6
```

	Name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

```
df7
```

	Name	drink
0	Mary	wine
1	Joseph	beer

```
pd.merge(df6, df7, how='outer')
```

	Name	food	drink
0	Peter	fish	NaN
1	Paul	beans	NaN
2	Mary	bread	wine

	Name	food	drink
3	Joseph	NaN	beer

왼쪽 조인 (left join)과 오른쪽 조인(right join)은 각각 왼쪽 항목과 오른쪽 항목을 기준으로 조인한다.

```
In [16]:print(df6); print(df7); print(pd.merge(df6, df7, how='left'))
Out[16]:df6
```

	Name	food
0	Peter	fish
1	Paul	beans
2	Mary	bread

```
df7
```

	Name	drink
0	Mary	wine
1	Joseph	beer

```
pd.merge(df6, df7, how='left')
```

	name	food	drink
0	Peter	fish	NaN
1	Paul	beans	NaN
2	Mary	bread	wine

이번에는 결과값의 행이 입력값의 항목에 대응한다. How = 'right'를 사용해도 비슷한 방식으로 동작한다.

이 옵션들 모두 이전 조인 유형에 간단하게 적용할 수 있다.

열 이름이 겹치는 경우 : suffixes 키워드

마지막으로 두 개의 입력 DataFrame 이 충돌하는 열 이름을 가진 경우를 살펴보자. :

```
In [17]:df8 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                            'rank': [1, 2, 3, 4]})
         df9 = pd.DataFrame({'name': ['Bob', 'Jake', 'Lisa', 'Sue'],
                            'rank': [3, 1, 4, 2]})
         print(pd.merge(df8, df9, on="name"))
Out[17]:df8
```

	name	rank
0	Bob	1
1	Jake	2
2	Lisa	3
3	Sue	4

```
df9
```

	Name	rank
0	Bob	3
1	Jake	1
2	Lisa	4
3	Sue	2

```
pd.merge(df8, df9, on="name")
```

	name	rank_x	rank_y
0	Bob	1	3
1	Jake	2	1
2	Lisa	3	4
3	Sue	4	2

결과값에 두 개의 충돌하는 열 이름이 있기 때문에 병합 함수가 결과 열을 고유하게 만들려고 자동으로 접미사_x 나 _y 를 덧 붙인다. 이 기본값이 적절하지 않다면 suffixes 키워드를 사용해 접미사를 별도로 지정할 수 있다:

```
In [18]:print(df8); print(df9);
        print(pd.merge(df8, df9, on="name", suffixes=["_L", "_R"]))
Out[18]:df8
```

	Name	rank
0	Bob	1
1	Jake	2
2	Lisa	3
3	Sue	4

```
df9
```

	Name	rank
0	Bob	3

	Name	rank
1	Jake	1
2	Lisa	4
3	Sue	2

```
pd.merge(df8, df9, on="name", suffixes=["_L", "_R"])
```

	name	rank_L	rank_R
0	Bob	1	3
1	Jake	2	1
2	Lisa	3	4
3	Sue	4	2

이 접미사는 모든 조인 패턴에서 사용할 수 있으며 중첩된 열이 여러개 있어도 동작한다.

예제: 미국 주 데이터

병합과 조인 연산은 서로 다른 소스에서 나온 데이터를 연결할 때 가장 자주 사용한다. 여기서는 미국 주와 주 별 인구데이터를 이용한 예제를 생각해 볼 것이다.

Pandas read_csv()함수를 사용해 세개의 데이터 세트를 살펴보자.

```
In [20]: pop = pd.read_csv('data/state-population.csv')
         areas = pd.read_csv('data/state-areas.csv')
         abbrevs = pd.read_csv('data/state-abbrevs.csv')

                  print(pop.head()); print(areas.head()); print(abbrevs.head())
Out[20]: pop.head()
```

	state/region	ages	year	population
0	AL	under18	2012	1117489.0
1	AL	total	2012	4817528.0
2	AL	under18	2010	1130966.0
3	AL	total	2010	4785570.0
4	AL	under18	2011	1125763.0

```
areas.head()
```

	State	area (sq. mi)
0	Alabama	52423

	State	area (sq. mi)
1	Alaska	656425
2	Arizona	114006
3	Arkansas	53182
4	California	163707

```
abbrevs.head()
```

	State	abbreviation
0	Alabama	AL
1	Alaska	AK
2	Arizona	AZ
3	Arkansas	AR
4	California	CA

이 정보가 주어진 상태에서 비교적 간단하게 2010 인구 밀도 기준으로 미국 주와 지역 순위를 계산하고 싶다고 하자. 이 결과를 얻을 수 있는 데이터는 확실히 있지만, 그 데이터 세트를 결합해야 결과를 얻을 수 있다.

먼저 인구(population)DataFrame 에 전체 주의 이름을 제공하는 다대일 병합으로 시작하자. 여기서는 pop 의 state/region 열과 abbrevs 의 sbbreviation 열을 기준으로 병합하려고 한다. 레이블이 일치하지 않는다는 이유로 데이터가 제거되지 않도록 how='outer'를 사용할 것이다.

```
In [21]: merged = pd.merge(pop, abbrevs, how='outer',
                           left_on='state/region', right_on='abbreviation')
         merged = merged.drop('abbreviation', 1) # drop duplicate info
         merged.head()
```

Out[21]:

	state/region	ages	year	population	state
0	AL	under18	2012	1117489.0	Alabama
1	AL	total	2012	4817528.0	Alabama
2	AL	under18	2010	1130966.0	Alabama
3	AL	total	2010	4785570.0	Alabama
4	AL	under18	2011	1125763.0	Alabama

이 코드에 불일치하는 항목이 있는지 다시 한번 확인해 보자., 널값을 가진 행을 찾으면 된다.:

```
In [22]: merged.isnull().any()
Out[22]:
state/region    False
ages            False
year            False
population       True
state            True
dtype: bool
```

일부 population 정보가 널 값이다. 어떤 항목이 널인지 확인해 보자.

```
In [23]: merged[merged['population'].isnull()].head()
Out[23]:
```

	state/region	ages	year	population	State
2448	PR	under18	1990	NaN	NaN
2449	PR	total	1990	NaN	NaN
2450	PR	total	1991	NaN	NaN
2451	PR	under18	1991	NaN	NaN
2452	PR	total	1993	NaN	NaN

인구 데이터에서 널 값은 모두 2000 년 이전의 푸에르토리코에서 비롯됐음을 알 수 있다. 이는 원본 소스에 사용할 수 있는 데이터가 없기 때문이다.

```
In [24]: merged.loc[merged['state'].isnull(), 'state/region'].unique()
Out[24]: array(['PR', 'USA'], dtype=object)
```

이 이슈는 신속하게 추론할 수 있다. 예제에 사용한 인구 데이터가 푸에르토리코(PR)와 전체 미국(USA)에 대한 항목을 포함하고 있지만, 이 항목들이 주 이름의 약어 키(abbreviation key)에는 등장하지 않는다. 이 문제는 적절한 항목을 채워 넣음으로써 쉽게 해결할 수 있다.

```
In [25]: merged.loc[merged['state/region'] == 'PR', 'state'] = 'Puerto Rico'
         merged.loc[merged['state/region'] == 'USA', 'state'] = 'United States'
         merged.isnull().any()
Out[25]: state/region    False
         ages            False
         year            False
         population       True
         state           False
```

```
dtype: bool
```

state 열에 더 이상 널 값이 없으니 연산을 수행할 준비가 끝났다.!

이제 비슷한 절차를 이용해 면적 데이터가 포함된 결과를 병합할 수 있다. 결과값을 검사해 보면 양쪽의 state 열을 기준으로 조인하고 싶을 것이다.

```
In [26]: final = pd.merge(merged, areas, on='state', how='left')
         final.head()
Out[26]:
```

	state/region	ages	year	population	state	area (sq. mi)
0	AL	under18	2012	1117489.0	Alabama	52423.0
1	AL	total	2012	4817528.0	Alabama	52423.0
2	AL	under18	2010	1130966.0	Alabama	52423.0
3	AL	total	2010	4785570.0	Alabama	52423.0
4	AL	under18	2011	1125763.0	Alabama	52423.0

다시 말해, 일치하지 않는 항목이 있는지 보기 위해 널 값 여부를 확인하자.

```
In [27]: final.isnull().any()
Out[27]:
state/region     False
ages             False
year             False
population        True
state            False
area (sq. mi)     True
dtype: bool
```

area 열에 널 값이 있다. 다음 코드로 어느 지역이 누락됐는지 찾을 수 있다.

```
In [28]: final['state'][final['area (sq. mi)'].isnull()].unique()
Out[28]: array(['United States'], dtype=object)
```

Areas DataFrame 이 전체 미국 면적을 담고 있지 않음을 알 수 있다. 적절한 값(예를 들면, 모든 주 면적의 합계 사용)을 삽입할 수 있지만, 이 경우에는 미국 전체의 인구 밀도가 현재 논의하는 내용과 관련이 없기 때문에 그냥 널 값을 삭제할 것이다.

```
In [29]: final.dropna(inplace=True)
         final.head()
Out[29]:
```

	state/region	ages	year	population	state	area (sq. mi)
0	AL	under18	2012	1117489.0	Alabama	52423.0
1	AL	total	2012	4817528.0	Alabama	52423.0
2	AL	under18	2010	1130966.0	Alabama	52423.0
3	AL	total	2010	4785570.0	Alabama	52423.0
4	AL	under18	2011	1125763.0	Alabama	52423.0

이제 필요한 모든 데이터를 갖췄다. 궁금한 질문에 답변하기 위해 먼저 2010 년과 전체 인구(total population)에 해당하는 데이터 부분을 선택하자. 이 작업을 신속히 처리하기 위해 여기서는 query()함수를 사용하겠다.(이 함수를 사용하려면 numexpr 패키지가 설치돼 있어야 한다.)

```
In [30]: data2010 = final.query("year == 2010 & ages == 'total'")
         data2010.head()
Out[30]:
```

	state/region	ages	year	population	state	area (sq. mi)
3	AL	total	2010	4785570.0	Alabama	52423.0
91	AK	total	2010	713868.0	Alaska	656425.0
101	AZ	total	2010	6408790.0	Arizona	114006.0
189	AR	total	2010	2922280.0	Arkansas	53182.0
197	CA	total	2010	37333601.0	California	163707.0

이제 인구 밀도를 계산하고 그것을 순서대로 표시해 보자.우선 주 기준으로 데이터 인덱스를 재배열하고 나서 결과를 계산할 것이다.

```
In [31]: data2010.set_index('state', inplace=True)
         density = data2010['population'] / data2010['area (sq. mi)']
In [32]: density.sort_values(ascending=False, inplace=True)
         density.head()
Out[32]:
state
District of Columbia    8898.897059
Puerto Rico             1058.665149
New Jersey              1009.253268
Rhode Island             681.339159
Connecticut              645.600649
dtype: float64
```

결과는 미국 주와 워싱턴 DC, 푸에르토리코를 1 제곱마일당 주민 수로 계산한 2010 년 인구 밀도 기준으로 순위를 매겼다. 지금까지 이 데이터세트에서 가장 밀도가 높은 지역은 워싱턴 DC 이며, 가장 밀도가 높은 주는 뉴저지주임을 알 수 있다.

```
In [33]: density.tail()
Out[33]:
state
South Dakota    10.583512
North Dakota     9.537565
Montana          6.736171
Wyoming          5.768079
Alaska           1.087509
dtype: float64
```

가장 밀도가 적은 주는 알래스카로 1 제곱마일당 평균 주민 수가 1 명이 조금이 넘는다.

이러한 유형의 지저분한 데이터 병합이 현실 세계의 데이터 소스를 사용해 질문에 대한 답을 얻고자 하는 경우 일반적으로 수행하는 작업이다.