Pandas 에서 데이터 연산하기

NumPy 의 기본 중 하나는 기본 산술 연산(덧셈, 뺄셈, 곱셈)과 복잡한 연산(삼각함수, 지수와 로그 함수 등) 모두에서 요소 단위의 연산을 빠르게 수행할 수 있다는 점이다. Pandas 는 NumPy 로 부터 이 기능의 대부분을 상속받았다.

Pandas 는 몇 가지 유용한 특수 기능을 포함하고 있다. 부정 함수와 삼각함수 같은 단항 연산의 경우에는 이 유니버설 함수가 결과물에 인덱스와 열 레이블을 보존하고, 덧셈과 곱셈 같은 이항 연산의 경우에는 Pandas 가 유니버설 함수에 객체를 전달할 때 자동으로 인덱스를 정렬한다. 다시 말해 Pandas 를 이용하면 데이터의 맥락을 유지하고 다른 소스에서 가져온 데이터를 결합하는 작업(둘 다 원시 NumPy 배열로는 오류 가 발생하기 쉬운 작업)을 근본적으로 실패할 일이 없다는 뜻이다. 이 밖에도 1 차원 Series 규조체와 2 차원 DataFrame 구조체 사이에 잘 정의된 연산에 대해 알아보자.

Ufuncs: Index 보존

Pandas 는 NumPy 와 함께 작업하도록 설계됐기 때문에 NumPy 의 유니버설 함수가 Pandas Series 와 DataFramer 객체 동작한다. 먼저 이를 보여줄 간단한 Series 와 DataFrame 을 정의하자.

```
In [1]:import pandas as pd
       import numpy as np
In [2]:rng = np.random.RandomState(42)
       ser = pd.Series(rng.randint(0, 10, 4))
       ser
Out[2]:0    6
       1    3
       2    7
       3    4
       dtype: int64
In [3]:df = pd.DataFrame(rng.randint(0, 10, (3, 4)),
                         columns=['A', 'B', 'C', 'D'])
       df
```

Out[3]:

	A	B	C	D
0	6	9	2	6
1	7	4	3	7
2	7	2	5	4

NumPy 유니버설 함수를 이 객체 중 하나에 적용하면 그 결과는 인덱스가 그대로 보존된 다른 Pandas 객체가 될 것이다.

```
In [4]:np.exp(ser)
Out[4]:0     403.428793
       1      20.085537
       2    1096.633158
       3      54.598150
       dtype: float64
```

다음은 약간 더 복잡한 계산이다.

```
In [5]:np.sin(df * np.pi / 4)
```

Out[5]:	A	B	C	D
0	-1.000000	7.071068e-01	1.000000	-1.000000e+00
1	-0.707107	1.224647e-16	0.707107	-7.071068e-01
2	-0.707107	1.000000e+00	-0.707107	1.224647e-16

NumPy 배열 연산 : 유니버설 함수에서 논의한 유니버설 함수는 모두 비슷한 방식으로 사용할 수 있다.

UFuncs: Index 정렬

두 개의 Series 또는 DataFrame 객체에 이항 연산을 적용하는 겨우, Pandas 는 연산을 수행하는 과정에서 인덱스를 정렬한다. 이는 다음에 나올 몇가지 예제에서 보는 바와 같이 불완전한 데이터로 작업할 때 매우 편리하다.

Series 에서 인덱스 정렬

두개의 다른 데이터 소스를 결합해 미국 주에서 면적 기준 상위 세개의 주와 인구 기준상위 세개의 주를 찾는다고 가정하자

```
In [6]:area = pd.Series({'Alaska': 1723337, 'Texas': 695662,
                         'California': 423967}, name='area')
        population = pd.Series({'California': 38332521, 'Texas': 26448193,
                                'New York': 19651127}, name='population')
```

둘을 나누어 인구 밀도를 계산하면 어떤 일이 일어나는지 보자:

```
In [7]:population / area
Out[7]:Alaska            NaN
       California    90.413926
       New York           NaN
```

```
Texas         38.018740
dtype: float64
```

결과 배열은 두 입력 배열의 인덱스의 합집합을 담고 있으며, 그 합집합은 이 인덱스에 표준 파이썬 집합 연산을 사용해 결정된다.

```
In [8]:area.index | population.index
Out[8]:Index(['Alaska', 'California', 'New York', 'Texas'], dtype='object')
```

둘 중 하나라도 같이 없는 항목은 Pandas 가 누락된 데이터를 표시하는 방식에 따라 NaN, 즉, '숫자가 아님(Not a Number)'으로 표시된다. 이 인덱스 매칭은 파이썬에 내장된 산술표현식에 대해서도 같은 방식으로 구현돼 있다. 누락된 같은 기본으로 NaN 으로 채운다.

```
In [9]:A = pd.Series([2, 4, 6], index=[0, 1, 2])
       B = pd.Series([1, 3, 5], index=[1, 2, 3])
       A + B
Out[9]:0    NaN
       1    5.0
       2    9.0
       3    NaN
       dtype: float64
```

NaN 값 사용을 원치 않을 경우, 연산자 대신에 적절한 객체 메서드를 사용해 채우기 값을 수정할 수 있다. 예를 들면, A.add(B)를 호출하면 A + B 를 호출하는 것과 같지만, A 나 B 에서 누락된 요소의 채우기 값을 선택해 명시적으로 지정할 수 있다.

```
In [10]:A.add(B, fill_value=0)
Out[10]:0    2.0
        1    5.0
        2    9.0
        3    5.0
        dtype: float64
```

DataFrame 에서 인덱스 정렬

`DataFrame` 에서 연산을 수행할 때 열과 인덱스 모두에서 비슷한 유형의 정렬이 발생한다.

```
In [11]:A = pd.DataFrame(rng.randint(0, 20, (2, 2)),
                         columns=list('AB'))
        A
```

Out[11]:	A	B
0	1	11

```
Out[11]:   A  B
1          5  1
```

```
In [12]: B = pd.DataFrame(rng.randint(0, 10, (3, 3)),
                          columns=list('BAC'))
         B
```

```
Out[12]:   B  A  C
0          4  0  9
1          5  8  0
2          9  2  6
```

```
In [13]: A + B
```

```
Out[13]:   A     B     C
0          1.0   15.0  NaN
1          13.0  6.0   NaN
2          NaN   NaN   NaN
```

두 객체의 순서와 상관없이 인덱스가 올바르게 정렬되고 결과 인덱스가 정렬된다. Series 와 마찬가지로 관련 객체의 산술 연산 메서드를 사용해 누락된 값 대신 원하는 fill_value 를 전달할 수 있다. 여기서는 A 에 있는 모든 값(먼저 A 의 행을 쌓아서 계산한)의 평균값으로 채운다.

```
In [14]: fill = A.stack().mean()
         A.add(B, fill_value=fill)
```

```
Out[14]:   A     B     C
0          1.0   15.0  13.5
1          13.0  6.0   4.5
2          6.5   13.5  10.5
```

아래 표에 파이썬 연산자와 그에 상응하는 Pandas 객체 메서드를 정리했다.

Python Operator	Pandas Method(s)
+	add()
-	sub(), subtract()

Python Operator	Pandas Method(s)
*	mul(), multiply()
/	truediv(), div(), divide()
//	floordiv()
%	mod()
**	pow()

Ufuncs: DataFrame 과 Series 간의 연산

DataFrame 과 Series 사이에서 연산할 때 인덱스와 열의 순서는 비슷하게 유지된다. DataFrame 과 Series 사이의 연산은 2 차원 NumPy 배열과 1 차원 NumPy 배열 사이의 연산과 비슷하다. 2 차원 배열과 그 배열의 행 하나와의 차이를 알아내는 일반적인 연산을 생각해 보자.

```
In [15]: A = rng.randint(10, size=(3, 4))
         A
Out[15]: array([[3, 8, 2, 4],
                [2, 6, 4, 8],
                [6, 1, 3, 8]])
In [16]: A - A[0]
Out[16]: array([[ 0,  0,  0,  0],
                [-1, -2,  2,  4],
                [ 3, -7,  1,  4]])
```

NumPy 브로드캐스팅 규칙에 따르면 2 차원 배열에서 그 배열의 행 하나를 빼는 것은 행 방향으로 적용된다.

Pandas 에서도 연산 규칙이 기본적으로 행 방향으로 적용된다.

```
In [17]: df = pd.DataFrame(A, columns=list('QRST'))
         df - df.iloc[0]
```

Out[17]:

	Q	R	S	T
0	0	0	0	0
1	-1	-2	2	4

Out[17]:

	Q	R	S	T
2	3	-7	1	4

열 방향으로 연산하고자 한다면 앞에서 언급한 객체 메서드를 사용하면서 axis 키워드를 지정하면 된다.

```
In [18]: df.subtract(df['R'], axis=0)
```

Out[18]:

	Q	R	S	T
0	-5	0	-6	-4
1	-4	0	-2	2
2	5	0	2	7

`DataFrame`/`Series` 연산은 앞에서 언급했던 연산과 마찬가지로 두 요소 간의 인덱스를 자동으로 맞춘다.

```
In [19]: halfrow = df.iloc[0, ::2]
         halfrow
Out[19]: Q    3
         S    2
         Name: 0, dtype: int64
In [20]: df - halfrow
```

Out[20]:

	Q	R	S	T
0	0.0	NaN	0.0	NaN
1	-1.0	NaN	2.0	NaN
2	3.0	NaN	1.0	NaN

이렇게 인덱스와 열을 맞추고 보존한다는 것은 Pandas 에서의 데이터 연산이 항상 데이터 맥락을 유지하기 때문에 원시 NumPy 배열에서 이종의 정렬되지 않은 데이터로 작업할 때 발생할 수 있는 멍청한 오류를 방지할 수 있다는 뜻이다.