

Notice that indices are aligned correctly irrespective of their order in the two objects, and indices in the result are sorted. As was the case with `Series`, we can use the associated object's arithmetic method and pass any desired `fill_value` to be used in place of missing entries. Here we'll fill with the mean of all values in `A` (which we compute by first stacking the rows of `A`):

```
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

Table 3-1 lists Python operators and their equivalent Pandas object methods.

Table 3-1. Mapping between Python operators and Pandas methods

Python operator	Pandas method(s)
+	<code>add()</code>
-	<code>sub()</code> , <code>subtract()</code>
*	<code>mul()</code> , <code>multiply()</code>
/	<code>truediv()</code> , <code>div()</code> , <code>divide()</code>
//	<code>floordiv()</code>
%	<code>mod()</code>
**	<code>pow()</code>

## Ufuncs: Operations Between DataFrame and Series

When you are performing operations between a `DataFrame` and a `Series`, the index and column alignment is similarly maintained. Operations between a `DataFrame` and a `Series` are similar to operations between a two-dimensional and one-dimensional NumPy array. Consider one common operation, where we find the difference of a two-dimensional array and one of its rows:

```
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]])
```

According to NumPy's broadcasting rules (see “[Computation on Arrays: Broadcasting](#)” on page 63), subtraction between a two-dimensional array and one of its rows is applied row-wise.

In Pandas, the convention similarly operates row-wise by default:

```
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
2	3	-7	1	4

If you would instead like to operate column-wise, you can use the object methods mentioned earlier, while specifying the `axis` keyword:

```
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

Note that these `DataFrame`/`Series` operations, like the operations discussed before, will automatically align indices between the two elements:

```
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

This preservation and alignment of indices and columns means that operations on data in Pandas will always maintain the data context, which prevents the types of silly errors that might come up when you are working with heterogeneous and/or mis-aligned data in raw NumPy arrays.

## Handling Missing Data

The difference between data found in many tutorials and data in the real world is that real-world data is rarely clean and homogeneous. In particular, many interesting datasets will have some amount of data missing. To make matters even more complicated, different data sources may indicate missing data in different ways.