Getting Started with pandas

Part 5

Essential Functionality

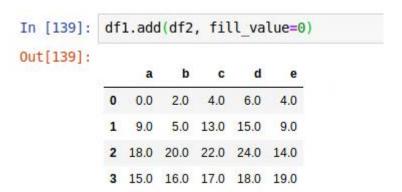
Part 3

Arithmetic methods with fill values

• In arithmetic operations between differently indexed objects, you might want to fill with a special value, like 0, when an axis label is found in one object but not the other.

```
In [134]: df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)),
                              columns=list('abcd'))
          df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)),
                              columns=list('abcde'))
In [135]: df2.loc[1, 'b'] = np.nan
In [136]: df1
Out[136]:
                           d
                       C
           0 0.0 1.0 2.0
                          3.0
           1 4.0 5.0 6.0
                          7.0
           2 8.0 9.0 10.0 11.0
In [137]: df2
Out[137]:
                         C
                             d
              0.0
                   1.0 2.0
                            3.0 4.0
           1 5.0 NaN
                       7.0
                            8.0
                                9.0
           2 10.0 11.0 12.0 13.0 14.0
           3 15.0 16.0 17.0 18.0 19.0
In [138]: df1 + df2
Out[138]:
              0.0
                   2.0
                       4.0
                            6.0 NaN
           1 9.0 NaN 13.0 15.0 NaN
           2 18.0 20.0 22.0 24.0 NaN
           3 NaN NaN NaN NaN NaN
```

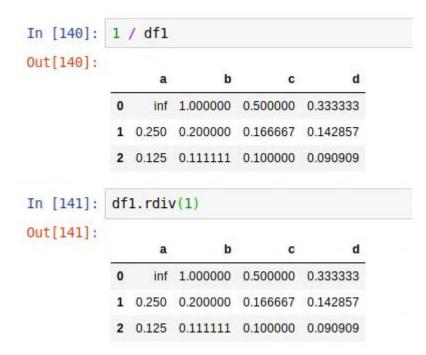
• Using the add method on df1, I pass df2 and an argument to fill value:



• Flexible arithmetic methods

Method	Description
add, radd	Methods for addition (+)
sub, rsub	Methods for subtraction (-)
div, rdiv	Methods for division (/)
floordiv, rfloordiv	Methods for floor division (//)
mul, rmul	Methods for multiplication (*)
pow, rpow	Methods for exponentiation (**)

- Each of them has a counterpart, starting with the letter \mathbf{r} , that has arguments flipped.
- So these two statements are equivalent:



 Relatedly, when reindexing a Series or DataFrame, you can also specify a different fill value:

Operations between DataFrame and Series

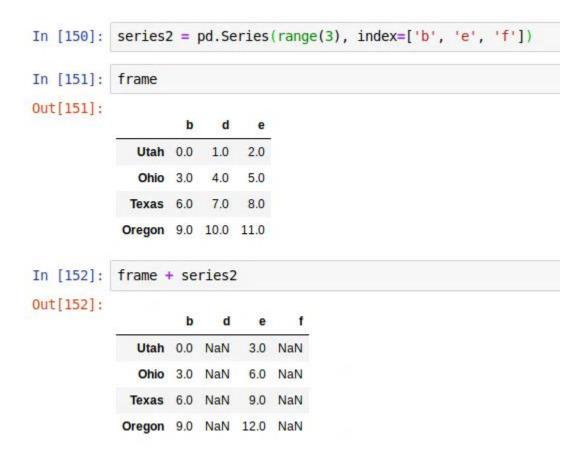
- As with NumPy arrays of different dimensions, arithmetic between DataFrame and Series is also defined.
- First, as a motivating example, consider the difference between a twodimensional array and one of its rows:

- When we subtract arr[0] from arr, the subtraction is performed once for each row.
- This is referred to as broadcasting.

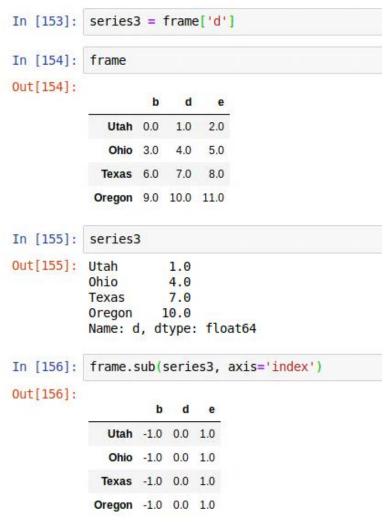
• Operations between a DataFrame and a Series are similar:

```
In [146]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                                columns=list('bde'),
                                index=['Utah', 'Ohio', 'Texas', 'Oregon'])
           series = frame.iloc[0]
In [147]: frame
Out[147]:
                       d e
                   b
             Utah 0.0 1.0 2.0
             Ohio 3.0
                      4.0 5.0
            Texas 6.0 7.0 8.0
           Oregon 9.0 10.0 11.0
In [148]: series
Out[148]: b
               0.0
               1.0
               2.0
          Name: Utah, dtype: float64
In [149]:
          frame - series
Out[149]:
                   b
                       d e
             Utah 0.0 0.0 0.0
             Ohio 3.0 3.0 3.0
            Texas 6.0 6.0 6.0
           Oregon 9.0 9.0 9.0
```

• If an index value is not found in either the DataFrame's columns or the Series's index, the objects will be reindexed to form the union:



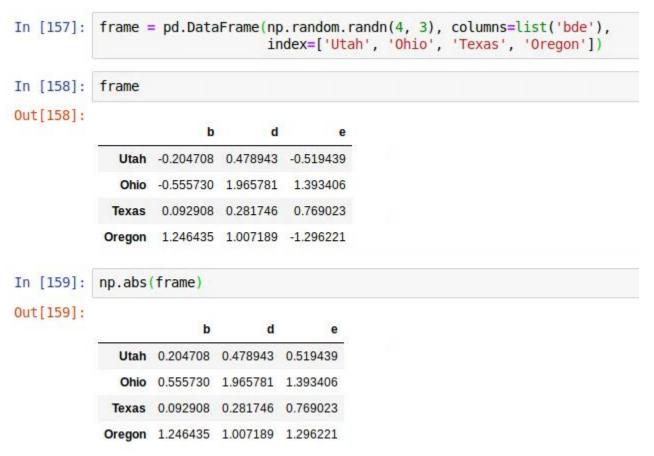
• If you want to instead broadcast over the columns, matching on the rows, you have to use one of the arithmetic methods.



- The axis number that you pass is the axis to match on.
- In this case we mean to match on the DataFrame's row index (axis='index' or axis=0) and broadcast across.

Function Application and Mapping

 NumPy ufuncs (element-wise array methods) also work with pandas objects:



- Another frequent operation is applying a function on one-dimensional arrays to each column or row.
- DataFrame's apply method does exactly this:

- Here the function £, which computes the difference between the maximum and minimum of a Series, is invoked once on each column in frame.
- The result is a Series having the columns of frame as its index.

• If you pass axis='columns' to apply, the function will be invoked once per row instead:

• Many of the most common array statistics (like sum and mean) are DataFrame methods, so using apply is not necessary.

• The function passed to apply need not return a scalar value; it can also return a Series with multiple values:

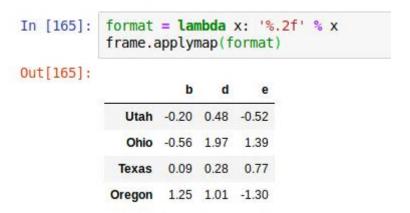
```
In [163]: def f(x):
    return pd.Series([x.min(), x.max()], index=['min', 'max'])

In [164]: frame.apply(f)

Out[164]:
    b    d    e

    min -0.555730  0.281746 -1.296221
    max  1.246435  1.965781  1.393406
```

- Element-wise Python functions can be used, too.
- Suppose you wanted to compute a formatted string from each floatingpoint value in frame.
- You can do this with applymap:



 The reason for the name applymap is that Series has a map method for applying an element-wise function:

```
In [166]: frame['e'].map(format)

Out[166]: Utah -0.52
Ohio 1.39
Texas 0.77
Oregon -1.30
Name: e, dtype: object
```