Difference between map, applymap and apply methods in Pandas

Can you tell me when to use these vectorization methods with basic examples? I see that `map` is a `Series` method whereas the rest are `DataFrame` methods. I got confused about `apply` and `applymap` methods though. Why do we have two methods for applying a function to a DataFrame? Again, simple examples which illustrate the usage would be great!

Thanks!

python numpy pandas vectorization

edited Jul 16 '14 at 18:18

asked Nov 5 '13 at 20:20
marillion
1,276 6 19 30

5 Answers

Straight from Wes McKinney's Python for Data Analysis book, pg. 132 (I highly recommended this book):

> Another frequent operation is applying a function on 1D arrays to each column or row. DataFrame's apply method does exactly this:

```
In [116]: frame = DataFrame(np.random.randn(4, 3), columns=list('bde'), index=['Utah',
'Ohio', 'Texas', 'Oregon'])

In [117]: frame
Out[117]:
               b         d         e
Utah   -0.029638  1.081563  1.280300
Ohio    0.647747  0.831136 -1.549481
Texas   0.513416 -0.884417  0.195343
Oregon -0.485454 -0.477388 -0.309548

In [118]: f = lambda x: x.max() - x.min()

In [119]: frame.apply(f)
Out[119]:
b    1.133201
d    1.965980
e    2.829781
dtype: float64
```

> Many of the most common array statistics (like sum and mean) are DataFrame methods, so using apply is not necessary.
>
> Element-wise Python functions can be used, too. Suppose you wanted to compute a formatted string from each floating point value in frame. You can do this with applymap:

```
In [120]: format = lambda x: '%.2f' % x

In [121]: frame.applymap(format)
Out[121]:
            b      d      e
Utah    -0.03   1.08   1.28
Ohio     0.65   0.83  -1.55
Texas    0.51  -0.88   0.20
Oregon  -0.49  -0.48  -0.31
```

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

```
In [122]: frame['e'].map(format)
Out[122]:
Utah       1.28
Ohio      -1.55
Texas      0.20
Oregon    -0.31
Name: e, dtype: object
```

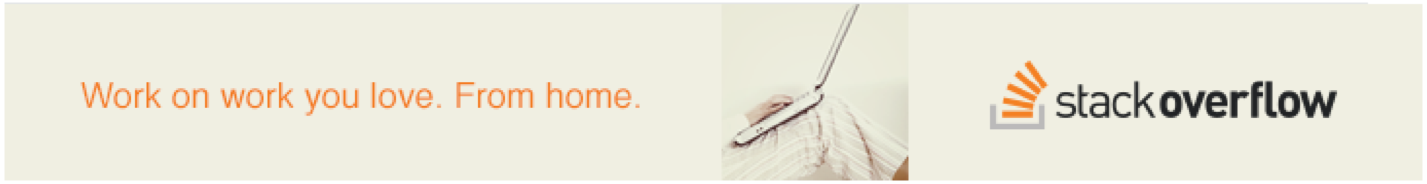
Summing up, `apply` works on a row / column basis of a DataFrame, `applymap` works element-wise on a DataFrame, and `map` works element-wise on a Series.

answered Nov 5 '13 at 20:40
jeremiahbuddha
2,124 1 13 22

5 strictly speaking, applymap internally is implemented via apply with a little wrap-up over passed function parameter (rougly speaking replacing `func` to `lambda x: [func(y) for y in x]` , and applying column-wise) – alko Nov 5 '13 at 20:53

1 Thanks for the explanation. Since `map` and `applymap` both work element-wise, I would expect a single method (either `map` or `applymap`) which would work both for a Series and a DataFrame. Probably there are other design considerations, and Wes McKinney decided to come up with two different methods. – marillion Nov 5 '13 at 21:58

It's on page 129 in my copy for some reason. There's no label for second edition or anything. – Jody Jan 26 at 21:34



@jeremiahbuddha mentioned that apply works on row/columns, while applymap works element-wise. But it seems you can still use apply for element-wise computation....

```
frame.apply(np.sqrt)
Out[102]:
               b         d         e
Utah         NaN  1.435159       NaN
Ohio    1.098164  0.510594  0.729748
```

```
Texas         NaN  0.456436  0.697337
Oregon  0.359079       NaN       NaN

frame.applymap(np.sqrt)
Out[103]:
               b         d         e
Utah         NaN  1.435159       NaN
Ohio    1.098164  0.510594  0.729748
Texas        NaN  0.456436  0.697337
Oregon  0.359079       NaN       NaN
```

answered Dec 19 '13 at 17:21

user2921752
166 6

7 Good catch with this. The reason this works in your example is because np.sqrt is a ufunc, i.e. if you give it an array, it will broadcast the sqrt function onto each element of the array. So when apply pushes np.sqrt on each columns, np.sqrt works itself on each of the elements of the columns, so you are essentially getting the same result as applymap. – jeremiahbuddha Jan 16 '14 at 0:22

Adding to the other answers, in a `Series` there are also map and apply.

Apply can make a DataFrame out of a series; however, map will just put a series in every cell of another series, which is probably not what you want.

```
In [41]: p=pd.Series([1,2,3])

In [42]: p.apply(lambda x: pd.Series([x, x]))
Out[42]:
   0  1
0  1  1
1  2  2
2  3  3

In [43]: p.map(lambda x: pd.Series([x, x]))
Out[43]:
0    0    1
1    1
dtype: int64
1    0    2
1    2
dtype: int64
```

```
2    0    3
1    3
dtype: int64
dtype: object
```

Also if I had a function with side effects, such as "connect to a web server", I'd probably use `apply` just for the sake of clarity.

```
series.apply(download_file_for_every_element)
```

`Map` can use not only a function, but also a dictionary or another series. Let's say you want to manipulate permutations.

Take

```
1 2 3 4 5
2 1 4 5 3
```

The square of this permutation is

```
1 2 3 4 5
1 2 5 3 4
```

You can compute it using `map` . Not sure if self-application is documented, but it works in `0.15.1` .

```
In [39]: p=pd.Series([1,0,3,4,2])

In [40]: p.map(p)
Out[40]:
0    0
1    1
2    4
3    2
4    3
dtype: int64
```

answered Dec 8 '14 at 23:30
osa
2,677 19 28

Just wanted to point out, as I struggled with this for a bit

```
def f(x):
    if x < 0:
        x = 0
    elif x > 100000:
        x = 100000
    return x

df.applymap(f)
df.describe()
```

this does not modify the dataframe itself, has to be reassigned

```
df = df.applymap(f)
df.describe()
```

answered Sep 26 '15 at 1:30

muon
168 10

I sometimes have trouble in figuring out whether you have to reassign or not after doing something with the df. It's mostly trial and error for me, but I bet there is a logic to how it works (that I am missing out). – marillion Apr 13 at 16:19

in general, a pandas dataframe is only modified by either reassigning `df = modified_df` or if you set `inplace=True` flag. Also dataframe will change if you pass a dataframe to a function by reference and the function modifies the dataframe – muon Apr 13 at 16:30

This is not entirely true, think of `.ix` or `.where` etc. Not sure what the full explanation is for when you need to re-assign and when not. – Thanos Apr 14 at 20:45

Probably simplest explanation the difference between apply and applymap:

apply takes the whole column as a parameter and then assign the result to this column

applymap takes the separate cell value as a parameter and assign the result back to this cell.

NB If apply returns the single value you will have this value instead of the column after assigning and eventually will have just a row instead of matrix.

answered May 20 at 2:10

Kath
1,379 7 13