python pandas: apply a function with arguments to a series

Ask Question

93

35

I want to apply a function with arguments to a series in python pandas:

```
x = my_series.apply(my_function, more_arguments_1)
y = my_series.apply(my_function, more_arguments_2)
...
```

The documentation describes support for an apply method, but it doesn't accept any arguments. Is there a different method that accepts arguments? Alternatively, am I missing a simple workaround?

Update (October 2017): Note that since this question was originally asked that pandas `apply()` has been updated to handle positional and keyword arguments and the documentation link above now reflects that and shows how to include either type of argument.

python pandas apply

edited Oct 15 '17 at 15:05
JohnE
14.8k 6 35 61

asked Aug 29 '12 at 16:46
Abe
6,799 14 61 97

3 Why not just use `functools.partial` , or `starmap` ? – Joel Cornett Aug 29 '12 at 16:54

1 See `DataFrame.apply` docs and `Series.apply` docs – Martin Thoma Aug 16 '18 at 12:43

4 Answers

116

single parameter. If you want to pass more parameters you should use `functools.partial` as suggested by Joel Cornett in his comment.

An example:

```
>>> import functools
>>> import operator
>>> add_3 = functools.partial(ope
>>> add_3(2)
5
>>> add_3(7)
10
```

You can also pass keyword arguments using `partial`.

Another way would be to create a lambda:

```
my_series.apply((lambda x: your_f
```

But I think using `partial` is better.

Note that newer versions of pandas *do* allow you to pass extra arguments (see the new documentation). So now you can do:

```
my_series.apply(your_function, ar
```

The positional arguments are added *after* the element of the series.

edited Feb 12 '16 at 15:56
user487890

answered Aug 29 '12 at 17:36

Bakuriu
67.1k 12 140 168

9 For a DataFrame apply method accepts `args` argument, which is a tuple holding additional positional arguments or **kwds for named ones. I created an issue to have this also for Series.apply() github.com/pydata/pandas/issues/1829 – Wouter Overmeire Aug 30 '12 at 20:11

21 Feature has been implemented, will be in upcoming pandas release – Wes McKinney Sep 9 '12 at 0:23

1 do we have a pointer now to the aforementioned feature? – watsonic Sep 8 '15 at 21:58

1 This is a nice answer but the first 2/3 of it is really obsolete now. IMO. this

keyword args. Just FWIW and not a criticism of the original answer, just would benefit from an update IMO, especially as it is a frequently read answer. – JohnE Oct 15 '17 at 14:59

@watsonic The documentation has since been updated and clicking on the old links leads to current documentation which now answers the question very well. – JohnE Oct 16 '17 at 16:49

39

Steps:

1. Create a dataframe
2. Create a function
3. Use the named arguments of the function in the apply statement.

Example

```
x=pd.DataFrame([1,2,3,4])

def add(i1, i2):
    return i1+i2

x.apply(add,i2=9)
```

The outcome of this example is that each number in the dataframe will be added to the number 9.

```
    0
0  10
1  11
2  12
3  13
```

Explanation:

The "add" function has two parameters: i1, i2. The first parameter is going to be the value in data frame and the second is whatever we pass to the "apply" function. In this case, we are passing "9" to the apply function using the keyword argument "i2".

answered Apr 17 '17 at 22:15

FistOfFury

3,216 2 32 50

38

```
args : tuple

x = my_series.apply(my_function, a
```

edited May 2 '17 at 14:26

ElBaulP
2,909 3 20 49

answered Nov 13 '14 at 21:12

dani_g
389 3 3

9 Thanks! Can you explain why args = (arg1,) needs a comma after the first argument? – DrMisha May 5 '15 at 18:19

18 @MishaTeplitskiy, you need the comma in order for Python to understand the parentheses' contents to be a tuple of length 1. – prooffreader May 18 '15 at 21:10

1 What about putting in args for the `func` . So if I wish to apply `pd.Series.mean(axis=1)` how do I put in the `axis=1` ? – josh Apr 7 '16 at 10:57

1 As a side note, you can also add a keyword argument without using the <args> parameter (e.g.: x = my_series.apply(my_function, keyword_arg=arg1), where <keyword_arg> is among the input parameters of my_function) – lev Apr 8 '16 at 8:15

1 this response is too short and doesn't explain anything – FistOfFury Apr 17 '17 at 22:08

16

You can pass any number of arguments to the function that `apply` is calling through either unnamed arguments, passed as a tuple to the `args` parameter, or through other keyword arguments internally captured as a dictionary by the `kwds` parameter.

For instance, let's build a function that returns True for values between 3 and 6, and False otherwise.

```
s = pd.Series(np.random.randint(0,
s

0    5
1    3
2    1
3    1
4    6
5    0
6    3
```

```
s.apply(lambda x: x >= 3 and x <=

0     True
1     True
2    False
3    False
4     True
5    False
6     True
7     True
8    False
9     True
dtype: bool
```

This anonymous function isn't very flexible. Let's create a normal function with two arguments to control the min and max values we want in our Series.

```
def between(x, low, high):
    return x >= low and x =< high
```

We can replicate the output of the first function by passing unnamed arguments to `args` :

```
s.apply(between, args=(3,6))
```

Or we can use the named arguments

```
s.apply(between, low=3, high=6)
```

Or even a combination of both

```
s.apply(between, args=(3,), high=6
```

edited Mar 11 at 4:58

NelsonGon

3,674 4 8 34

answered Nov 4 '17 at 2:58

Ted Petrou

24.8k 9 76 69