

The results are in! See what nearly 90,000 developers picked as their most loved, dreaded, and desired coding languages and more in the 2019 Developer Survey.

Why isn't my Pandas 'apply' function referencing multiple columns working? [closed]

[Ask Question](#)

▲
207
▼
I have some problems with the Pandas apply function, when using multiple columns with the following dataframe

★
85

```
df = DataFrame({'a' : np.random.randn(6),
                'b' : ['foo', 'bar'] * 3,
                'c' : np.random.randn(6)})
```

and the following function

```
def my_test(a, b):
    return a % b
```

When I try to apply this function with :

```
df['Value'] = df.apply(lambda row: my_test(row[a], row[c]), axis=1)
```

I get the error message:

```
NameError: ("global name 'a' is not defined", u'occurred at index 0')
```

I do not understand this message, I defined the name properly.

I would highly appreciate any help on this issue

Update

Thanks for your help. I made indeed some syntax mistakes with the code, the index should be put ". However I still get the same issue using a more complex function such as:

```
def my_test(a):
    cum_diff = 0
    for ix in df.index():
        cum_diff = cum_diff + (a - df['a'][ix])
    return cum_diff
```

[python](#)[python-2.7](#)[pandas](#)

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

edited Mar 4 at 2:36



smci

15.6k 6 79 109

asked May 3 '13 at 7:25



Andy

2,626 6 26 31

closed as off-topic by [smci](#), [PM 2Ring](#), [Andras Deak](#), [user2357112](#), [C8H10N4O2](#) Mar 8 at 1:39

This question appears to be off-topic. The users who voted to close gave this specific reason:

- "This question was caused by a **problem that can no longer be reproduced** or a **simple typographical error**. While similar questions may be on-topic here, this one was resolved in a manner unlikely to help future readers. This can often be avoided by identifying and closely inspecting [the shortest program necessary to reproduce the problem](#) before posting." – [smci](#), [PM 2Ring](#), [Andras Deak](#), [user2357112](#), [C8H10N4O2](#)

If this question can be reworded to fit the rules in the [help center](#), please [edit the question](#).

- 1 Avoid using `apply` as much as possible. If you're not sure you need to use it, you probably don't. I recommend taking a look at [When should I ever want to use pandas apply\(\) in my code?](#). – [coldspeed](#) Jan 30 at 10:22

This is just about syntax errors referencing a dataframe column, and why do functions need arguments. As to your second question, the function `my_test(a)` doesn't know what `df` is since it wasn't passed in as an argument (unless `df` is supposed to be a global, which would be terrible practice). You need to pass all the values you'll need inside a function as arguments (preferably in order), otherwise how else would the function know where `df` comes from? Also, it's bad practice to program in a namespace littered with global variables, you won't catch errors like this. – [smci](#) Mar 4 at 2:43

6 Answers



```
In [43]: df['Value'] = df.apply(Li
```



```
In [44]: df
```

```
Out[44]:
```

	a	b
0	-1.674308	foo 0.3431
1	-2.163236	bar -2.0461
2	-0.199115	foo -0.4581
3	0.918646	bar -0.0071
4	1.336830	foo 0.5341
5	0.976844	bar -0.7731

BTW, in my opinion, following way is more elegant:

```
In [53]: def my_test2(row):
.....:     return row['a'] % row['b']
.....:
```

```
In [54]: df['Value'] = df.apply(m
```

answered May 3 '13 at 8:40



[waitingkuo](#)

38.6k 17 87 100

Thanks, You are right I forgot the ".
However I have still the same issue with a more complex function. I would highly appreciate your help with that.
Thanks – [Andy](#) May 3 '13 at 8:58

4 [@Andy](#) following [53-54] allow you to apply more complex functions. – [Andy Hayden](#) May 3 '13 at 9:29

[@Andy](#) you can define your complex function like the `In[53]` way. – [waitingkuo](#) May 3 '13 at 9:37

do all apply strategies perform the same? I'm new to pandas and have always found apply slightly enigmatic but your strategy in [53-54] is easy for me to understand (and hopefully remember) ... on a large table is it as quick as the other form of apply presented? – [whytheq](#) Sep 4 '16 at 9:48

Why is it that creating a separate method is considered more elegant - even for tiny methods. I have been doing significant projects in python for 7 years but will likely never be considered a pythonista due to some perspectives including this one. – [javadba](#) Oct 20 '18 at 14:59



If you just want to compute (column

In [7]: df['a'] % df['c']
Out[7]:
 0 -1.132022
 1 -0.939493
 2 0.201931
 3 0.511374
 4 -0.694647
 5 -0.023486
Name: a

answered May 3 '13 at 7:56



herrfz
 3,548 3 20 34

13 I know, it is just an example to show my problem in applying a function to multiple columns – [Andy](#) May 3 '13 at 8:22

Let's say we want to apply a function add5 to columns 'a' and 'b' of DataFrame df

15

def add5(x):
 return x+5

 df[['a', 'b']].apply(add5)

answered Nov 12 '17 at 19:18



Mir_Murtaza
 161 1 4

I am getting following error while trying your code snippet. TypeError: ('must be str, not int', 'occurred at index b') can you please look into that. – [debaonline4u](#) Aug 8 '18 at 5:55

The column b of your dataframe is a string type or object type column, it should be an integer column to be added with a number. – [Mir_Murtaza](#) Aug 8 '18 at 7:59

All of the suggestions above work, but if you want your computations to be more efficient, you should take advantage of numpy vector operations ([as pointed out here](#)).

9

import pandas **as** pd
import numpy **as** np

 df = pd.DataFrame({'a': np.random.
 'b': ['foo', 'bar']
 'c': np.random.randn

Example 1: Inoning with

```
%%timeit
def my_test2(row):
    return row['a'] % row['c']

df['Value'] = df.apply(my_test2, a
```

The slowest run took 7.49 times longer than the fastest. This could mean that an intermediate result is being cached. 1000 loops, best of 3: 481 µs per loop

Example 2: vectorize using `pandas.apply()` :

```
%%timeit
df['a'] % df['c']
```

The slowest run took 458.85 times longer than the fastest. This could mean that an intermediate result is being cached. 10000 loops, best of 3: 70.9 µs per loop

Example 3: vectorize using numpy arrays:

```
%%timeit
df['a'].values % df['c'].values
```

The slowest run took 7.98 times longer than the fastest. This could mean that an intermediate result is being cached. 100000 loops, best of 3: 6.39 µs per loop

So vectorizing using numpy arrays improved the speed by almost two orders of magnitude.

edited Aug 10 '18 at 1:08

answered Apr 27 '18 at 21:14



Blane

194 1 8



1

This is same as the previous solution but I have defined the function in `df.apply` itself:



```
df['Value'] = df.apply(lambda row:
```

answered Sep 30 '18 at 4:47



I have given the comparison of all three discussed above.

0



Using values

```
%timeit df['value'] = df['a'].valu
```

139 μ s \pm 1.91 μ s per loop (mean \pm std. dev. of 7 runs, 10000 loops each)

Without values

```
%timeit df['value'] = df['a']%df['
```

216 μ s \pm 1.86 μ s per loop (mean \pm std. dev. of 7 runs, 1000 loops each)

Apply function

```
%timeit df['Value'] = df.apply(lam
```

474 μ s \pm 5.07 μ s per loop (mean \pm std. dev. of 7 runs, 1000 loops each)

answered Feb 17 at 3:53



Gursewak Singh

56 4