Truth value of a Series is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all()

Ask Question

179

72

Having issue filtering my result dataframe with an `or` condition. I want my result `df` to extract all column `var` values that are above 0.25 and below -0.25.

This logic below gives me an ambiguous truth value however it work when I split this filtering in two separate operations. What is happening here? not sure where to use the suggested `a.empty(), a.bool(), a.item(),a.any() or a.all()` .

```
result = result[(result['var']>0.25) or (result['var']<-0.25)]
```

python pandas dataframe boolean filtering

edited Mar 8 at 21:49

MSeifert
78.9k 19 156 189

asked Apr 28 '16 at 17:46

obabs
1,023 2 5 12

You should add more context. I don't understand what result is and what you are trying to do. – kingledion Apr 28 '16 at 17:49

14 use | instead of `or` – MaxU Apr 28 '16 at 17:54

Here's a workaround: `abs(result['var'])>0.25` – ColinMac Dec 28 '18 at 17:29

Related: Logical operators for boolean indexing in Pandas – cs95 Mar 8 at 22:09

305

The `or` and `and` python statements require `truth` -values. For `pandas` these are considered ambiguous so you should use "bitwise" `|` (or) or `&` (and) operations:

```
result = result[(result['var']>0.
```

These are overloaded for these kind of datastructures to yield the element-wise `or` (or `and`).

Just to add some more explanation to this statement:

The exception is thrown when you want to get the `bool` of a `pandas.Series`:

```
>>> import pandas as pd
>>> x = pd.Series([1])
>>> bool(x)
ValueError: The truth value of a S
a.item(), a.any() or a.all().
```

What you hit was a place where the operator **implicitly** converted the operands to `bool` (you used `or` but it also happens for `and`, `if` and `while`):

```
>>> x or x
ValueError: The truth value of a S
a.item(), a.any() or a.all().
>>> x and x
ValueError: The truth value of a S
a.item(), a.any() or a.all().
>>> if x:
...     print('fun')
ValueError: The truth value of a S
a.item(), a.any() or a.all().
>>> while x:
...     print('fun')
ValueError: The truth value of a S
a.item(), a.any() or a.all().
```

Besides these 4 statements there are several python functions that hide some `bool` calls (like `any`, `all`, `filter`, ...) these are normally not problematic with `pandas.Series` but for completeness I wanted to mention these.

In your case the exception isn't really helpful, because it doesn't mention the **right alternatives**. For `and` and `or` you can use (if you want element-wise comparisons):

- `numpy.logical_or`:

or simply the `|` operator:

```
>>> x | y
```

- `numpy.logical_and`:

```
>>> np.logical_and(x, y)
```

or simply the `&` operator:

```
>>> x & y
```

If you're using the operators then make sure you set your parenthesis correctly because of the operator precedence.

There are several logical numpy functions which *should* work on `pandas.Series`.

The alternatives mentioned in the Exception are more suited if you encountered it when doing `if` or `while`. I'll shortly explain each of these:

- If you want to check if your Series is **empty**:

```
>>> x = pd.Series([])
>>> x.empty
True
>>> x = pd.Series([1])
>>> x.empty
False
```

Python normally interprets the `len`gth of containers (like `list`, `tuple`, ...) as truth-value if it has no explicit boolean interpretation. So if you want the python-like check, you could do: `if x.size` or `if not x.empty` instead of `if x`.

- If your `Series` contains **one and only one** boolean value:

```
>>> x = pd.Series([100])
>>> (x > 50).bool()
True
>>> (x < 50).bool()
False
```

- If you want to check the **first and only item** of your Series (like `.bool()` but works even for not boolean contents):

```
>>> x = pd.Series([100])
>>> x.item()
```

- If you want to check if **all** or **any** item is not-zero, not-empty or not-False:

```
>>> x = pd.Series([0, 1, 2])
>>> x.all()   # because one e
False
>>> x.any()   # because one (
True
```

edited Jan 1 '17 at 18:50

answered Apr 28 '16 at 17:54

MSeifert
78.9k 19 156 189

9 Oh my god! Your comment "If you're using the operators then make sure you set your parenthesis correctly because of the operator precedence" finally solved the problem that's been driving me mad. A very important and, in my case, overlooked point. Thank you! – user4896331 Oct 18 '17 at 21:05

6 one of the most informative answers I have read in a long time – deadcode Jan 24 '18 at 11:54

Why aren't these python operators overloaded to handle pandas series? – Mudit Jain Apr 6 at 14:37

@MuditJain There is no way to directly overload `and`, `or`, and `not` in Python. These operators directly use what `bool` on the operands returns. And in a way Pandas/NumPy overloaded that already to raise the `ValueError` because they consider the truth-value of such a data structure ambiguous. – MSeifert Apr 6 at 15:14

28

For boolean logic, use `&` and `|`.

```
np.random.seed(0)
df = pd.DataFrame(np.random.randn(
```

```
>>> df
          A         B         C
0  1.764052  0.400157  0.978738
1  2.240893  1.867558 -0.977278
2  0.950088 -0.151357 -0.103219
3  0.410599  0.144044  1.454274
4  0.761038  0.121675  0.443863
```

```
1  2.240893  1.867558 -0.977278
3  0.410599  0.144044  1.454274
4  0.761038  0.121675  0.443863
```

To see what is happening, you get a column of booleans for each comparison, e.g.

```
df.C > 0.25
0     True
1    False
2    False
3     True
4     True
Name: C, dtype: bool
```

When you have multiple criteria, you will get multiple columns returned. This is why the the join logic is ambiguous. Using `and` or `or` treats each column separately, so you first need to reduce that column to a single boolean value. For example, to see if any value or all values in each of the columns is True.

```
# Any value in either column is Tr
(df.C > 0.25).any() or (df.C < -0.
True

# All values in either column is T
(df.C > 0.25).all() or (df.C < -0.
False
```

One convoluted way to achieve the same thing is to zip all of these columns together, and perform the appropriate logic.

```
>>> df[[any([a, b]) for a, b in zi
          A         B         C
0  1.764052  0.400157  0.978738
1  2.240893  1.867558 -0.977278
3  0.410599  0.144044  1.454274
4  0.761038  0.121675  0.443863
```

For more details, refer to Boolean Indexing in the docs.

edited Apr 28 '16 at 18:23

answered Apr 28 '16 at 18:15

Alexander
56.5k 14 94 128

8

Or, alternatively, you could use Operator module. More detailed information is here Python docs

```
df = pd.DataFrame(np.random.randn(
df.loc[operator.or_(df.C > 0.25, d
```

```
          A         B         C
0  1.764052  0.400157  0.978738
1  2.240893  1.867558 -0.977278
3  0.410599  0.144044  1.454274
4  0.761038  0.121675  0.4438
```

answered Jan 19 '17 at 7:48

Cảnh Toàn Nguyễn

126 2 7

1

This excellent answer explains very well what is happening and provides a solution. I would like to add another solution that might be suitable in similar cases: using the `query` method:

```
result = result.query("(var > 0.25
```

See also http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-query.

(Some tests with a dataframe I'm currently working with suggest that this method is a bit slower than using the bitwise operators on series of booleans: 2 ms vs. 870 µs)

A piece of warning: At least one situation where this is not straightforward is when column names happen to be python expressions. I had columns named `WT_38hph_IP_2 , WT_38hph_input_2` and `log2(WT_38hph_IP_2/WT_38hph_input_2)` and wanted to perform the following query: `"(log2(WT_38hph_IP_2/WT_38hph_input_2) > 1) and (WT_38hph_IP_2 > 20)"`

I obtained the following exception cascade:

- `KeyError: 'log2'`
- `UndefinedVariableError: name 'log2' is not defined`
- `ValueError: "log2" is not a supported function`

I guess this happened because the query parser was trying to make something from the first two columns instead of identifying the expression

edited Nov 2 '17 at 12:20

answered Nov 2 '17 at 11:13

bli

2,627 2 21 45