Select rows from a DataFrame based on values in a column in pandas

Ask Question

1272

773

How to select rows from a DataFrame based on values in some column in pandas?
In SQL I would use:

```
select * from table where colume_name = some_value.
```

I tried to look at pandas documentation but did not immediately find the answer.

python pandas dataframe

edited Apr 17 '16 at 21:49
Thanos
1,475 1 7 25

asked Jun 12 '13 at 17:42
szli
7,527 6 23 33

Check here: github.com/debaonline4u/Python_Programming/tree/master/… – debaonline4u Jul 10 '18 at 19:49

`df.query` and `pd.eval` seem like good fits for this use case. For information on the `pd.eval()` family of functions, their features and use cases, please visit Dynamic Expression Evaluation in pandas using pd.eval(). – coldspeed Dec 16 '18 at 4:54

This is a Comparison with SQL: pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html where you can run pandas as SQL. – i_th Jan 5 at 8:07

14 Answers

2480

To select rows whose column value equals a scalar, `some_value`, use `==`:

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

To select rows whose column value is in an iterable, `some_values`, use `isin`:

```
df.loc[df['column_name'].isin(some
```

Combine multiple conditions with `&`:

```
df.loc[(df['column_name'] >= A) &
```

Note the parentheses. Due to Python's [operator precedence rules](), `&` binds more tightly than `<=` and `>=`. Thus, the parentheses in the last example are necessary. Without the parentheses

```
df['column_name'] >= A & df['colum
```

is parsed as

```
df['column_name'] >= (A & df['colu
```

which results in a [Truth value of a Series is ambiguous error]().

To select rows whose column value *does not equal* `some_value`, use `!=`:

```
df.loc[df['column_name'] != some_v
```

`isin` returns a boolean Series, so to select rows whose value is *not* in `some_values`, negate the boolean Series using `~`:

```
df.loc[~df['column_name'].isin(som
```

For example,

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'A': 'foo bar f
                   'B': 'one one t
                   'C': np.arange(
print(df)
#      A      B  C   D
# 0  foo    one  0   0
# 1  bar    one  1   2
# 2  foo    two  2   4
# 3  bar  three  3   6
# 4  foo    two  4   8
# 5  bar    two  5  10
# 6  foo    one  6  12
# 7  foo  three  7  14

print(df.loc[df['A'] == 'foo'])
```

yields

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

If you have multiple values you want to include, put them in a list (or more generally, any iterable) and use `isin` :

```
print(df.loc[df['B'].isin(['one','
```

yields

```
     A      B  C   D
0  foo    one  0   0
1  bar    one  1   2
3  bar  three  3   6
6  foo    one  6  12
7  foo  three  7  14
```

Note, however, that if you wish to do this many times, it is more efficient to make an index first, and then use `df.loc` :

```
df = df.set_index(['B'])
print(df.loc['one'])
```

yields

```
       A  C   D
B
one  foo  0   0
one  bar  1   2
one  foo  6  12
```

or, to include multiple values from the index use `df.index.isin` :

```
df.loc[df.index.isin(['one','two']
```

yields

```
       A  C   D
B
one  foo  0   0
one  bar  1   2
two  foo  2   4
two  foo  4   8
two  bar  5  10
one  foo  6  12
```

edited Jan 18 at 2:47

answered Jun 12 '13 at 17:44

unutbu
562k 105 1211

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

df[df['colume_name']==some_value] also works. But my first attempt, df.where(df['colume_name']==some_value) does not work... not sure why... – szli Jun 12 '13 at 18:12

7 When you use `df.where(condition)` , the condition has to have the same shape as `df` . – unutbu Jun 12 '13 at 18:19

6 FYI: If you want to select a row based upon two (or more) labels (either requiring both or either), see stackoverflow.com/questions/31756340/… – Shane Aug 1 '15 at 0:18

5 What about the negative "isnotin" does that exist? – BlackHat Mar 24 '16 at 6:13

7 @BlackHat: `isin` returns a boolean mask. To find rows not in `some_iterable` , negate the boolean mask using `~` (a tilde). That is, `df.loc[~df['column_name'].isin(some_values)]` – unutbu Mar 24 '16 at 10:27

212

tl;dr

The pandas equivalent to

```
select * from table where column_na
```

is

```
table[table.column_name == some_va
```

Multiple conditions:

```
table[(table.column_name == some_va
```

or

```
table.query('column_name == some_va
```

Code example

```
import pandas as pd

# Create data set
d = {'foo':[100, 111, 222],
     'bar':[333, 444, 555]}
df = pd.DataFrame(d)
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
# 0  333   100
# 1  444   111
# 2  555   222

# Output only the row(s) in df whe
df[df.foo == 222]

# Shows:
#    bar  foo
# 2  555  222
```

In the above code it is the line `df[df.foo == 222]` that gives the rows based on the column value, `222` in this case.

Multiple conditions are also possible:

```
df[(df.foo == 222) | (df.bar == 44
#    bar  foo
# 1  444  111
# 2  555  222
```

But at that point I would recommend using the query function, since it's less verbose and yields the same result:

```
df.query('foo == 222 | bar == 444'
```

edited Jun 28 '18 at 15:30

answered Jul 8 '15 at 15:17

imolit
4,092 3 17 24

3 I really like the approach here. Thanks for having added it. It seems a bit more elegant than the accepted answer - which is still ok but this is great thanks. – kiltannen Apr 22 '18 at 5:21

1 `query` is the only answer here that is compatible with method chaining. It seems like it's the pandas analog to `filter` in dplyr. – Berk U. Apr 23 '18 at 17:26

2 Hi, in your third example (multiple columns) I think you need square brackets `[` not round brackets `(` on the outside. – user2739472 Jun 28 '18 at 12:40

1 at first I thought that `|` was for AND, but of course it is OR-operator... – O95 Nov 7 '18 at 9:32

I like `query` a lot as it is very readable. It is worth noting that it also works for multi-index dataframes where one can also query on different index levels (see the answer here). – Cleb Nov 25 '18 at 15:09

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

151

+500

1. Boolean indexing
2. Positional indexing
3. Label indexing
4. API

For each base type, we can keep things simple by restricting ourselves to the pandas API or we can venture outside the API, usually into `numpy` , and speed things up.

I'll show you examples of each and guide you as to when to use certain techniques.

Setup
The first thing we'll need is to identify a condition that will act as our criterion for selecting rows. The OP offers up `column_name == some_value` . We'll start there and include some other common use cases.

Borrowing from @unutbu:

```
import pandas as pd, numpy as np

df = pd.DataFrame({'A': 'foo bar fo
                   'B': 'one one tw
                   'C': np.arange(8
```

Assume our criterion is column `'A'` = `'foo'`

1.
Boolean indexing requires finding the true value of each row's `'A'` column being equal to `'foo'` , then using those truth values to identify which rows to keep. Typically, we'd name this series, an array of truth values, `mask` . We'll do so here as well.

```
mask = df['A'] == 'foo'
```

We can then use this mask to slice or index the data frame

```
df[mask]

     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

This is one of the simplest ways to accomplish this task and if

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

consider an alternative way of creating the `mask`.

2.
Positional indexing has its use cases, but this isn't one of them. In order to identify where to slice, we first need to perform the same boolean analysis we did above. This leaves us performing one extra step to accomplish the same task.

```
mask = df['A'] == 'foo'
pos = np.flatnonzero(mask)
df.iloc[pos]

     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

3.
Label indexing can be very handy, but in this case, we are again doing more work for no benefit

```
df.set_index('A', append=True, dro

     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

4.
`pd.DataFrame.query` is a very elegant/intuitive way to perform this task. But is often slower. **However**, if you pay attention to the timings below, for large data, the query is very efficient. More so than the standard approach and of similar magnitude as my best suggestion.

```
df.query('A == "foo"')

     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

My preference is to use the `Boolean mask`

Actual improvements can be made by modifying how we create our `Boolean mask`.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

forgo the overhead of creating another `pd.Series`

```
mask = df['A'].values == 'foo'
```

I'll show more complete time tests at the end, but just take a look at the performance gains we get using the sample data frame. First, we look at the difference in creating the `mask`

```
%timeit mask = df['A'].values == '
%timeit mask = df['A'] == 'foo'

5.84 µs ± 195 ns per loop (mean ±
166 µs ± 4.45 µs per loop (mean ±
```

Evaluating the `mask` with the `numpy` array is ~ 30 times faster. This is partly due to `numpy` evaluation often being faster. It is also partly due to the lack of overhead necessary to build an index and a corresponding `pd.Series` object.

Next, we'll look at the timing for slicing with one `mask` versus the other.

```
mask = df['A'].values == 'foo'
%timeit df[mask]
mask = df['A'] == 'foo'
%timeit df[mask]

219 µs ± 12.3 µs per loop (mean ±
239 µs ± 7.03 µs per loop (mean ±
```

The performance gains aren't as pronounced. We'll see if this holds up over more robust testing.

`mask` alternative 2

We could have reconstructed the data frame as well. There is a big caveat when reconstructing a dataframe—you must take care of the `dtypes` when doing so!

Instead of `df[mask]` we will do this

```
pd.DataFrame(df.values[mask], df.i
```

If the data frame is of mixed type, which our example is, then when we get `df.values` the resulting array is of `dtype` `object` and consequently, all columns of the new data frame will be of `dtype` `object` . Thus requiring the `astype(df.dtypes)` and killing any potential performance gains.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
216 µs ± 10.4 µs per loop (mean ± s
1.43 ms ± 39.6 µs per loop (mean ±
```

However, if the data frame is not of mixed type, this is a very useful way to do it.

Given

```
np.random.seed([3,1415])
d1 = pd.DataFrame(np.random.randint

d1

   A  B  C  D  E
0  0  2  7  3  8
1  7  0  6  8  6
2  0  2  0  4  9
3  7  3  2  4  3
4  3  6  7  7  4
5  5  3  7  5  9
6  8  7  6  4  7
7  6  2  6  6  5
8  2  8  7  5  8
9  4  7  6  1  5
```

```
%%timeit
mask = d1['A'].values == 7
d1[mask]

179 µs ± 8.73 µs per loop (mean ± s
```

Versus

```
%%timeit
mask = d1['A'].values == 7
pd.DataFrame(d1.values[mask], d1.in

87 µs ± 5.12 µs per loop (mean ± st
```

We cut the time in half.

`mask` alternative 3

@unutbu also shows us how to use `pd.Series.isin` to account for each element of `df['A']` being in a set of values. This evaluates to the same thing if our set of values is a set of one value, namely `'foo'`. But it also generalizes to include larger sets of values if needed. Turns out, this is still pretty fast even though it is a more general solution. The only real loss is in intuitiveness for those not familiar with the concept.

```
mask = df['A'].isin(['foo'])
df[mask]

     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

However, as before, we can utilize `numpy` to improve performance while sacrificing virtually nothing. We'll use `np.in1d`

```
mask = np.in1d(df['A'].values, ['fo
df[mask]

     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

Timing
I'll include other concepts mentioned in other posts as well for reference.
Code Below

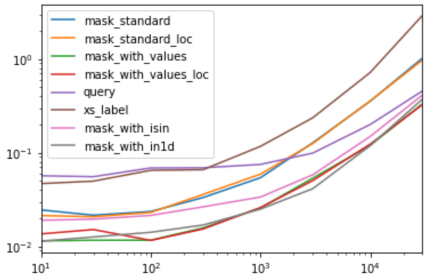
Each Column in this table represents a different length data frame over which we test each function. Each column shows relative time taken, with the fastest function given a base index of `1.0`.

```
res.div(res.min())

                            10
10000     30000
mask_standard         2.156872  1.8
2.981326  3.131151
mask_standard_loc     1.879035  1.1
2.998112  2.990103
mask_with_values      1.010166  1.0
1.007824  1.016919
mask_with_values_loc  1.196843  1.1
1.037020  1.000000
query                 4.997304  4.1
1.680447  1.398190
xs_label              4.124597  4.2
6.032809  8.950255
mask_with_isin        1.674055  1.6
1.253554  1.264760
mask_with_in1d        1.000000  1.0
1.000000  1.144175
```

You'll notice that fastest times seem to be shared between `mask_with_values` and `mask_with_in1d`

```
res.T.plot(loglog=True)
```



By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
def mask_standard(df):
    mask = df['A'] == 'foo'
    return df[mask]

def mask_standard_loc(df):
    mask = df['A'] == 'foo'
    return df.loc[mask]

def mask_with_values(df):
    mask = df['A'].values == 'foo'
    return df[mask]

def mask_with_values_loc(df):
    mask = df['A'].values == 'foo'
    return df.loc[mask]

def query(df):
    return df.query('A == "foo"')

def xs_label(df):
    return df.set_index('A', appen

def mask_with_isin(df):
    mask = df['A'].isin(['foo'])
    return df[mask]

def mask_with_in1d(df):
    mask = np.in1d(df['A'].values,
    return df[mask]
```

Testing

```
res = pd.DataFrame(
    index=[
        'mask_standard', 'mask_sta
'mask_with_values_loc',
        'query', 'xs_label', 'mask_
    ],
    columns=[10, 30, 100, 300, 100
    dtype=float
)

for j in res.columns:
    d = pd.concat([df] * j, ignore_
    for i in res.index:a
        stmt = '{}(d)'.format(i)
        setp = 'from __main__ impo
        res.at[i, j] = timeit(stmt
```

Special Timing

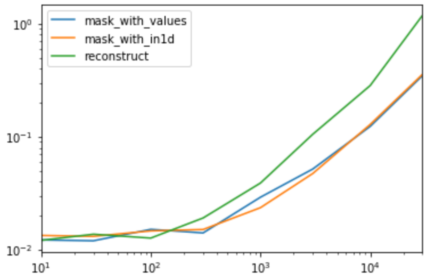
Looking at the special case when we have a single non-object `dtype` for the entire data frame. *Code Below*

```
spec.div(spec.min())

                     10        30
10000     30000
mask_with_values  1.009030  1.0000
1.000000  1.000000
mask_with_in1d    1.104638  1.0945
1.040043  1.027100
reconstruct       1.000000  1.1428
2.294913  3.406735
```

Turns out, reconstruction isn't worth it past a few hundred rows.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.



Functions

```
np.random.seed([3,1415])
d1 = pd.DataFrame(np.random.randin

def mask_with_values(df):
    mask = df['A'].values == 'foo'
    return df[mask]

def mask_with_in1d(df):
    mask = np.in1d(df['A'].values,
    return df[mask]

def reconstruct(df):
    v = df.values
    mask = np.in1d(df['A'].values,
    return pd.DataFrame(v[mask], d

spec = pd.DataFrame(
    index=['mask_with_values', 'mas
    columns=[10, 30, 100, 300, 100
    dtype=float
)
```

Testing

```
for j in spec.columns:
    d = pd.concat([df] * j, ignore_
    for i in spec.index:
        stmt = '{}(d)'.format(i)
        setp = 'from __main__ impo
        spec.at[i, j] = timeit(stm
```

edited Dec 18 '18 at 15:08

Prakash Pazhanisamy
876 1 9 23

answered Sep 11 '17 at 22:14

piRSquared
160k 24 159 303

3 Fantastic answer! 2 questions though, i) how would `.iloc(numpy.where(..))` compare in this scheme? ii) would you expect the rankings to be the same when using multiple conditions? – posdef Mar 6 '18 at 13:49

For performance of `pd.Series.isin`, note it *does* use `np.in1d` under the hood in a specific scenario, uses khash in others, and implicitly applies a trade-off between cost of hashing versus performance in specific situations. This answer has more detail. – jpp Jun 17 '18 at 19:08

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

50

I find the syntax of the previous answers to be redundant and difficult to remember. Pandas introduced the `query()` method in v0.13 and I much prefer it. For your question, you could do `df.query('col == val')`

Reproduced from http://pandas.pydata.org/pandas-docs/version/0.17.0/indexing.html#indexing-query

```
In [167]: n = 10

In [168]: df = pd.DataFrame(np.ranc

In [169]: df
Out[169]:
          a         b         c
0  0.687704  0.582314  0.281645
1  0.250846  0.610021  0.420121
2  0.624328  0.401816  0.932146
3  0.011763  0.022921  0.244186
4  0.590198  0.325680  0.890392
5  0.598892  0.296424  0.007312
6  0.634625  0.803069  0.123872
7  0.924168  0.325076  0.303746
8  0.116822  0.364564  0.454607
9  0.986142  0.751953  0.561512

# pure python
In [170]: df[(df.a < df.b) & (df.b
Out[170]:
          a         b         c
3  0.011763  0.022921  0.244186
8  0.116822  0.364564  0.454607

# query
In [171]: df.query('(a < b) & (b <
Out[171]:
          a         b         c
3  0.011763  0.022921  0.244186
8  0.116822  0.364564  0.454607
```

You can also access variables in the environment by prepending an `@`.

```
exclude = ('red', 'orange')
df.query('color not in @exclude')
```

answered Feb 9 '16 at 1:36
fredcallaway
826 9 5

1 You only need package `numexpr` installed. – MERose Mar 13 '16 at 9:16

3 In my case I needed quotation because val is a string. df.query('col == "val"') – smerlung Aug 10 '17 at 18:34

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
In [76]: df.iloc[np.where(df.A.valu
Out[76]:
     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

Timing comparisons:

```
In [68]: %timeit df.iloc[np.where(
1000 loops, best of 3: 380 µs per

In [69]: %timeit df.loc[df['A'] ==
1000 loops, best of 3: 745 µs per

In [71]: %timeit df.loc[df['A'].is
1000 loops, best of 3: 562 µs per

In [72]: %timeit df[df.A=='foo']
1000 loops, best of 3: 796 µs per

In [74]: %timeit df.query('(A=="fo
1000 loops, best of 3: 1.71 ms per
```

edited Oct 3 '17 at 16:17
Brian Burns
7,082 5 46 45

answered Jul 5 '17 at 16:34
shivsn
3,997 11 24

15

Here is a simple example

```
from pandas import DataFrame

# Create data set
d = {'Revenue':[100,111,222],
     'Cost':[333,444,555]}
df = DataFrame(d)


# mask = Return True when the valu
mask = df['Revenue'] == 111

print mask

# Result:
# 0    False
# 1     True
# 2    False
# Name: Revenue, dtype: bool


# Select * FROM df WHERE Revenue =
df[mask]

# Result:
#    Cost    Revenue
# 1  444     111
```

answered Jun 13 '13 at 11:49
DataByDavid
599 2 7 19

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

11

I just tried editing this, but I wasn't logged in, so I'm not sure where my edit went. I was trying to incorporate multiple selection. So I think a better answer is:

For a single value, the most straightforward (human readable) is probably:

```
df.loc[df['column_name'] == some_va
```

For lists of values you can also use:

```
df.loc[df['column_name'].isin(some_
```

For example,

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'A': 'foo bar fo
               'B': 'one one two th
               'C': np.arange(8),
print(df)
#      A      B  C   D
# 0  foo    one  0   0
# 1  bar    one  1   2
# 2  foo    two  2   4
# 3  bar  three  3   6
# 4  foo    two  4   8
# 5  bar    two  5  10
# 6  foo    one  6  12
# 7  foo  three  7  14

print(df.loc[df['A'] == 'foo'])
```

yields

```
     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

If you have multiple criteria you want to select against, you can put them in a list and use 'isin':

```
print(df.loc[df['B'].isin(['one','t
```

yields

```
      A      B  C   D
0  foo    one  0   0
1  bar    one  1   2
3  bar  three  3   6
6  foo    one  6  12
7  foo  three  7  14
```

Note, however, that if you wish to do this many times, it is more efficient to make A the index first, and then use df.loc:

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

yields

```
  A      B  C   D
foo    one  0   0
foo    two  2   4
foo    two  4   8
foo    one  6  12
foo  three  7  14
```

answered Jan 25 '15 at 23:27

Jeff Ellen
514 3 9

7

If you finding rows based on some integer in a column, then

```
df.loc[df['column_name'] == 2017]
```

If you are finding value based on string

```
df.loc[df['column_name'] == 'string
```

If based on both

```
df.loc[(df['column_name'] == 'stri
```

answered Nov 16 '18 at 7:26

prateek singh
91 1 6

6

```
df = pd.DataFrame({'A': 'foo bar fo
                   'B': 'one one tw
                   'C': np.arange(8
df[df['A']=='foo']

OUTPUT:
   A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

answered Mar 6 '16 at 6:02

user15051990
710 7 17

5 How is this any different from imolit's answer? – MERose Mar 13 '16 at 9:15

To append to this famous question (though a bit too late): You can also

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

specified column having a particular value. E.g.

```
import pandas as pd
df = pd.DataFrame({'A': 'foo bar fo
                   'B': 'one one tw
print("Original dataframe:")
print(df)

b_is_two_dataframe =
pd.DataFrame(df.groupby('B').get_gr
= 1)
#NOTE: the final drop is to remove
object
print('Sub dataframe where B is two
print(b_is_two_dataframe)
```

Run this gives:

```
Original dataframe:
     A      B
0  foo    one
1  bar    one
2  foo    two
3  bar  three
4  foo    two
5  bar    two
6  foo    one
7  foo  three
Sub dataframe where B is two:
     A    B
0  foo  two
1  foo  two
2  bar  two
```

answered Nov 18 '16 at 12:10

TuanDT
1,190 6 22

6

For selecting only specific columns out of multiple columns for a given value in pandas:

```
select col_name1, col_name2 from ta
```

Options:

```
df.loc[df['column_name'] == some_va
```

or

```
df.query['column_name' == 'some_va
```

edited Jun 22 '18 at 7:44

firelynx
15.3k 3 65 81

answered Dec 7 '17 at 10:39

SP001
69 1 3

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

4

those whose column's value is NOT any of a list of values, here's how to flip around unutbu's answer for a list of values above:

```
df.loc[~df['column_name'].isin(some
```

(To not include a single value, of course, you just use the regular not equals operator, `!=`.)

Example:

```
import pandas as pd
df = pd.DataFrame({'A': 'foo bar fo
                   'B': 'one one tw
print(df)
```

gives us

```
     A      B
0  foo    one
1  bar    one
2  foo    two
3  bar  three
4  foo    two
5  bar    two
6  foo    one
7  foo  three
```

To subset to just those rows that AREN'T `one` or `three` in column `B`:

```
df.loc[~df['B'].isin(['one', 'three
```

yields

```
     A    B
2  foo  two
4  foo  two
5  bar  two
```

answered Nov 12 '15 at 20:03

Bonnie
461 4 6

1

You can also use .apply:

```
df.apply(lambda row: row[df['B'].is
```

It actually works row-wise (i.e., applies the function to each row).

The output is

```
   A      B  C   D
0  foo    one  0   0
1  bar    one  1   2
3  bar  three  3   6
6  foo    one  6  12
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

The results is the same as using as mentioned by @unutbu

```
df[[df['B'].isin(['one','three'])]]
```

answered Dec 7 '18 at 17:38

Vahidn
52 1 10

1

```
df.loc[df['column_name'] == some_va
```

answered Feb 10 at 19:36

John Nero
1

---

protected by jezrael Feb 24 '18 at 18:33

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.