How to add multiple columns to pandas dataframe in one assignment?

Asked 3 years, 11 months ago Active 3 months ago Viewed 163k times

120

41

I'm new to pandas and trying to figure out how to add multiple columns to pandas simultaneously. Any help here is appreciated. Ideally I would like to do this in one step rather than multiple repeated steps...

```
import pandas as pd

df = {'col_1': [0, 1, 2, 3],
        'col_2': [4, 5, 6, 7]}
df = pd.DataFrame(df)

df[[ 'column_new_1', 'column_new_2','column_new_3']] = [np.nan, 'dogs',3]  #thought this
would work here...
```

python pandas dataframe

edited Apr 19 at 7:44 — smci — 22.6k 12 92 131

asked Aug 20 '16 at 4:40 — runningbirds — 3,581 8 33 68

You need to state what error you got. When I try this on pandas 1.0 I get `KeyError: "None of [Index(['column_new_1', 'column_new_2', 'column_new_3'], dtype='object')] are in the [columns]"` – smci Apr 19 at 7:24

8 Answers

Active | Oldest | Votes

183

I would have expected your syntax to work too. The problem arises because when you create new columns with the column-list syntax (`df[[new1, new2]] = ...`), pandas requires that the right hand side be a DataFrame (note that it doesn't actually matter if the columns of the DataFrame have the same names as the columns you are creating).

Your syntax works fine for assigning scalar values to *existing* columns, and pandas is also happy to assign scalar values to a new column using the single-column syntax (`df[new1] = ...`). So the solution is either to convert this into several single-column assignments, or create a suitable DataFrame for the right-hand side.

Here are several approaches that *will* work:

```
import pandas as pd
import numpy as np

df = pd.DataFrame({
    'col_1': [0, 1, 2, 3]
```

Then one of the following:

1) Three assignments in one, using list unpacking:

```
df['column_new_1'], df['column_new_2'], df['column_new_3'] = [np.nan, 'dogs', 3]
```

2) `DataFrame` conveniently expands a single row to match the index, so you can do this:

```
df[['column_new_1', 'column_new_2', 'column_new_3']] = pd.DataFrame([[np.nan, 'dogs',
3]], index=df.index)
```

3) Make a temporary data frame with new columns, then combine with the original data frame later:

```
df = pd.concat(
    [
        df,
        pd.DataFrame(
            [[np.nan, 'dogs', 3]],
            index=df.index,
            columns=['column_new_1', 'column_new_2', 'column_new_3']
        )
    ], axis=1
)
```

4) Similar to the previous, but using `join` instead of `concat` (may be less efficient):

```
df = df.join(pd.DataFrame(
    [[np.nan, 'dogs', 3]],
    index=df.index,
    columns=['column_new_1', 'column_new_2', 'column_new_3']
))
```

5) Using a dict is a more "natural" way to create the new data frame than the previous two, but the new columns will be sorted alphabetically (at least before Python 3.6 or 3.7):

```
df = df.join(pd.DataFrame(
    {
        'column_new_1': np.nan,
        'column_new_2': 'dogs',
        'column_new_3': 3
    }, index=df.index
))
```

I like this variant on @zero's answer a lot, but like the previous one, the new columns will always be sorted alphabetically, at least with early versions of Python:

```
df = df.assign(column_new_1=np.nan, column_new_2='dogs', column_new_3=3)
```

7) This is interesting (based on https://stackoverflow.com/a/44951376/3830997), but I don't know when it would be worth the trouble:

```
new_cols = ['column_new_1', 'column_new_2', 'column_new_3']
new_vals = [np.nan, 'dogs', 3]
df = df.reindex(columns=df.columns.tolist() + new_cols)   # add empty cols
df[new_cols] = new_vals  # multi-column assignment works for existing cols
```

8) In the end it's hard to beat three separate assignments:

```
df['column_new_1'] = np.nan
df['column_new_2'] = 'dogs'
df['column_new_3'] = 3
```

Note: many of these options have already been covered in other answers: Add multiple columns to DataFrame and set them equal to an existing column, Is it possible to add several columns at once to a pandas DataFrame?, Add multiple empty columns to pandas DataFrame

edited Apr 19 at 7:32
smci
22.6k 12 92 131

answered Aug 20 '16 at 5:49
Matthias Fripp
11.4k 3 22 33

Wouldn't approach #7 (`.reindex`) alter the dataframe's index? Why would someone want to needlessly alter the index when adding columns unless it's an explicit goal... – Acumenus Feb 18 at 20:12

1 `.reindex()` is used with the `columns` argument, so it only changes the column "index" (names). It doesn't alter the row index. – Matthias Fripp Feb 18 at 23:19

for some of the approaches, you can use `OrderedDict` : for instance, `df.join(pd.DataFrame( OrderedDict([('column_new_2', 'dogs'),('column_new_1', np.nan),('column_new_3', 3)]), index=df.index ))` – hashmuke Mar 11 at 12:12

@hashmuke That makes sense for early versions of Python. It may appeal especially to people using dictionaries for multiple things in Pandas, e.g., `df = pd.DataFrame({'before': [1, 2, 3], 'after': [4, 5, 6]})` vs. `df = pd.DataFrame(OrderedDict([('before', [1, 2, 3]), ('after', [4, 5, 6])])` – Matthias Fripp Mar 13 at 18:32

2 In case you are using the option with `join` , make sure that you don't have duplicates in your index (or use a `reset_index` first). Might save you a few hours debugging. – Guido Apr 30 at 13:53

You could use `assign` with a dict of column names and values.

```
0      0      4       dogs          3        NaN
1      1      5       dogs          3        NaN
2      2      6       dogs          3        NaN
3      3      7       dogs          3        NaN
```

answered Oct 4 '17 at 20:02

Zero
52.3k 10 109 125

Is there a way of doing the same that maintains a specific ordering of the columns? – user48956 May 31 '18 at 0:58

1 You can maintain a specific ordering with earlier versions of Python by calling assign multiple times: `df.assign(**{'col_new_1': np.nan}).assign(**{'col2_new_2': 'dogs'}).assign(**{'col3_new_3': 3})` – skasch Apr 2 at 18:35

9

With the use of concat:

```
In [128]: df
Out[128]:
   col_1  col_2
0      0      4
1      1      5
2      2      6
3      3      7

In [129]: pd.concat([df, pd.DataFrame(columns = [ 'column_new_1',
'column_new_2','column_new_3'])])
Out[129]:
   col_1  col_2 column_new_1 column_new_2 column_new_3
0    0.0    4.0          NaN          NaN          NaN
1    1.0    5.0          NaN          NaN          NaN
2    2.0    6.0          NaN          NaN          NaN
3    3.0    7.0          NaN          NaN          NaN
```

Not very sure of what you wanted to do with `[np.nan, 'dogs',3]` . Maybe now set them as default values?

```
In [142]: df1 = pd.concat([df, pd.DataFrame(columns = [ 'column_new_1',
'column_new_2','column_new_3'])])
In [143]: df1[[ 'column_new_1', 'column_new_2','column_new_3']] = [np.nan, 'dogs', 3]

In [144]: df1
Out[144]:
   col_1  col_2  column_new_1 column_new_2  column_new_3
0    0.0    4.0           NaN         dogs             3
1    1.0    5.0           NaN         dogs             3
2    2.0    6.0           NaN         dogs             3
3    3.0    7.0           NaN         dogs             3
```

answered Aug 20 '16 at 5:00

Nehal J Wani

if there was a way to do your 2nd part in one step - yes constant values in the columns as an example. – runningbirds Aug 20 '16 at 5:23

3

use of list comprehension, `pd.DataFrame` and `pd.concat`

```
pd.concat(
    [
        df,
        pd.DataFrame(
            [[np.nan, 'dogs', 3] for _ in range(df.shape[0])],
            df.index, ['column_new_1', 'column_new_2','column_new_3']
        )
    ], axis=1)
```

	col_1	col_2	column_new_1	column_new_2	column_new_3
0	0	4	NaN	dogs	3
1	1	5	NaN	dogs	3
2	2	6	NaN	dogs	3
3	3	7	NaN	dogs	3

edited Aug 20 '16 at 15:09

answered Aug 20 '16 at 6:49

piRSquared
216k 32 282 451

3

if adding a lot of missing columns (a, b, c ,....) with the same value, here 0, i did this:

```
new_cols = ["a", "b", "c" ]
df[new_cols] = pd.DataFrame([[0] * len(new_cols)], index=df.index)
```

It's based on the second variant of the accepted answer.

answered May 2 '19 at 14:15

A. Rabus
352 3 9

0

Just want to point out that option2 in @Matthias Fripp's answer

(2) I wouldn't necessarily expect DataFrame to work this way, but it does

df[['column_new_1', 'column_new_2', 'column_new_3']] = pd.DataFrame([[np.nan, 'dogs', 3]], index=df.index)

You can pass a list of columns to [] to select columns in that order. If a column is not contained in the DataFrame, an exception will be raised. **Multiple columns can also be set in this manner.** You may find this useful for applying a transform (**in-place**) to a subset of the columns.

edited Jun 20 at 9:12
Community ♦
1 1

answered Sep 15 '17 at 13:55
halfmoonhalf
77 5

I think this is pretty standard for multi-column assignment. The part that surprised me was that `pd.DataFrame([[np.nan, 'dogs', 3]], index=df.index)` replicates the one row it is given to create a whole dataframe the same length as the index. – Matthias Fripp Nov 10 '17 at 6:57

0

If you just want to add empty new columns, reindex will do the job

```
df
   col_1  col_2
0      0      4
1      1      5
2      2      6
3      3      7

df.reindex(list(df)+['column_new_1', 'column_new_2','column_new_3'], axis=1)
   col_1  col_2  column_new_1  column_new_2  column_new_3
0      0      4           NaN           NaN           NaN
1      1      5           NaN           NaN           NaN
2      2      6           NaN           NaN           NaN
3      3      7           NaN           NaN           NaN
```

full code example

```
import numpy as np
import pandas as pd

df = {'col_1': [0, 1, 2, 3],
        'col_2': [4, 5, 6, 7]}
df = pd.DataFrame(df)
print('df',df, sep='\n')
print()
df=df.reindex(list(df)+['column_new_1', 'column_new_2','column_new_3'], axis=1)
print('''df.reindex(list(df)+['column_new_1', 'column_new_2','column_new_3'],
axis=1)''',df, sep='\n')
```

otherwise go for zeros answer with assign

answered Jul 23 '19 at 11:23
Markus Dutschke
3,257 2 24 32

0

```
df.columns
Index(['A123', 'B123'], dtype='object')

df=pd.concat([df,pd.DataFrame(columns=list('CDE'))])

df.rename(columns={
    'C':'C123',
    'D':'D123',
    'E':'E123'
},inplace=True)


df.columns
Index(['A123', 'B123', 'C123', 'D123', 'E123'], dtype='object')
```

edited May 12 at 12:25

Nensi Kasundra

906 1 6 22

answered May 12 at 9:57

Alex

1 1