How to change the order of DataFrame columns?

Asked 10 years, 1 month ago Modified 9 months ago Viewed 2.0m times

1462

I have the following `DataFrame` (`df`):

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.rand(10, 5))
```

I add more column(s) by assignment:

```
df['mean'] = df.mean(1)
```

How can I move the column `mean` to the front, i.e. set it as first column leaving the order of the other columns untouched?

python pandas dataframe

Share Edit Follow

edited Jan 20, 2019 at 13:47
nick
1,032 1 11 24

asked Oct 30, 2012 at 22:22
Timmie
14.8k 3 14 7

3 possible duplicate of Python Pandas - Re-ordering columns in a dataframe based on column name – Laurence Jan 4, 2013 at 11:31

2 For a generalized NumPy-based solution see How to move a column in a pandas dataframe, assumes one column level only, i.e. no `MultiIndex`. – jpp Oct 3, 2018 at 8:31

1 After searching enough, I got this best link for columns re-arranging multiple logics in pretty simple terms [columns re-arrange logic for pandas] [datasciencemadesimple.com/… – ravibeli Jun 6, 2020 at 13:49

1 In the end, the point is: `df = df[ list with newly arranged column names ]` ;D – starriet Apr 12 at 0:40

41 Answers

Sorted by: Highest score (default)

1 2 Next

1283

One easy way would be to reassign the dataframe with a list of the columns, rearranged as needed.

This is what you have now:

```
In [6]: df
Out[6]:
          0         1         2         3         4      mean
0  0.445598  0.173835  0.343415  0.682252  0.582616  0.445543
1  0.881592  0.696942  0.702232  0.696724  0.373551  0.670208
2  0.662527  0.955193  0.131016  0.609548  0.804694  0.632596
3  0.260919  0.783467  0.593433  0.033426  0.512019  0.436653
4  0.131842  0.799367  0.182828  0.683330  0.019485  0.363371
5  0.498784  0.873495  0.383811  0.699289  0.480447  0.587165
6  0.388771  0.395757  0.745237  0.628406  0.784473  0.588529
7  0.147986  0.459451  0.310961  0.706435  0.100914  0.345149
```

```
8  0.394947  0.863494  0.585030  0.565944  0.356561  0.553195
9  0.689260  0.865243  0.136481  0.386582  0.730399  0.561593

In [7]: cols = df.columns.tolist()

In [8]: cols
Out[8]: [0L, 1L, 2L, 3L, 4L, 'mean']
```

Rearrange `cols` in any way you want. This is how I moved the last element to the first position:

```
In [12]: cols = cols[-1:] + cols[:-1]

In [13]: cols
Out[13]: ['mean', 0L, 1L, 2L, 3L, 4L]
```

Then reorder the dataframe like this:

```
In [16]: df = df[cols]  #    OR    df = df.ix[:, cols]

In [17]: df
Out[17]:
       mean         0         1         2         3         4
0  0.445543  0.445598  0.173835  0.343415  0.682252  0.582616
1  0.670208  0.881592  0.696942  0.702232  0.696724  0.373551
2  0.632596  0.662527  0.955193  0.131016  0.609548  0.804694
3  0.436653  0.260919  0.783467  0.593433  0.033426  0.512019
4  0.363371  0.131842  0.799367  0.182828  0.683330  0.019485
5  0.587165  0.498784  0.873495  0.383811  0.699289  0.480447
6  0.588529  0.388771  0.395757  0.745237  0.628406  0.784473
7  0.345149  0.147986  0.459451  0.310961  0.706435  0.100914
8  0.553195  0.394947  0.863494  0.585030  0.565944  0.356561
9  0.561593  0.689260  0.865243  0.136481  0.386582  0.730399
```

Share Edit Follow

edited Oct 31, 2012 at 15:51

answered Oct 30, 2012 at 22:38

Aman
44.3k 7 35 37

37 incase you get "cannot concatenate 'str' and 'list' objects" make sure you [] the str value in cols: cols = [cols[7]] + cols[:7] + cols[8:] – moeabdol Jan 9, 2015 at 16:59

4 @FooBar That's not a set union it's a concatenation of two ordered lists. – Aman Oct 6, 2016 at 22:08

3 @Aman I'm just pointing out that your code is deprecated. Your handling of your post is at your discretion. – FooBar Oct 7, 2016 at 6:13

2 @FooBar, the type of `cols` is `list`; it even allows duplicates (which will be discarded when used on the dataframe). You are thinking of `Index` objects. – alexis Feb 28, 2017 at 15:19

17 This implies copying ALL the data, which is highly inefficient. I wished pandas had a way to do that without creating a copy. – Konstantin Nov 27, 2017 at 8:48

779

You could also do something like this:

```
df = df[['mean', '0', '1', '2', '3']]
```

You can get the list of columns with:

```
cols = list(df.columns.values)
```

The output will produce:

```
['0', '1', '2', '3', 'mean']
```

...which is then easy to rearrange manually before dropping it into the first function

Share Edit Follow

answered May 19, 2014 at 15:20
freddygv
8,538 1 14 9

10 You could also get the list of columns with list(df.columns) – Jim Oct 9, 2015 at 22:14

28 or `df.columns.tolist()` – Jim Oct 9, 2015 at 22:22

7 I don't think this is a good answer as it does not provide code how to change column order of any dataframe. Say i import a csv file as pandas pd as `pd.read_csv()` . How can your answer be used to change the column order? – Robvh Jul 25, 2019 at 8:22

6 @Robvh, the second line of code explains how to get the existing column names. From there, you can copy the output into the first line of code, and re-arrange as desired. The only other piece of information to know is that without a header, the default column names are integers, not strings. – daniel brandstetter Sep 9, 2019 at 23:27

395

Just assign the column names in the order you want them:

```
In [39]: df
Out[39]: 
          0         1         2         3         4  mean
0  0.172742  0.915661  0.043387  0.712833  0.190717     1
1  0.128186  0.424771  0.590779  0.771080  0.617472     1
2  0.125709  0.085894  0.989798  0.829491  0.155563     1
3  0.742578  0.104061  0.299708  0.616751  0.951802     1
4  0.721118  0.528156  0.421360  0.105886  0.322311     1
5  0.900878  0.082047  0.224656  0.195162  0.736652     1
6  0.897832  0.558108  0.318016  0.586563  0.507564     1
7  0.027178  0.375183  0.930248  0.921786  0.337060     1
8  0.763028  0.182905  0.931756  0.110675  0.423398     1
9  0.848996  0.310562  0.140873  0.304561  0.417808     1

In [40]: df = df[['mean', 4,3,2,1]]
```

Now, 'mean' column comes out in the front:

```
In [41]: df
Out[41]: 
   mean         4         3         2         1
0     1  0.190717  0.712833  0.043387  0.915661
1     1  0.617472  0.771080  0.590779  0.424771
2     1  0.155563  0.829491  0.989798  0.085894
3     1  0.951802  0.616751  0.299708  0.104061
4     1  0.322311  0.105886  0.421360  0.528156
5     1  0.736652  0.195162  0.224656  0.082047
6     1  0.507564  0.586563  0.318016  0.558108
7     1  0.337060  0.921786  0.930248  0.375183
8     1  0.423398  0.110675  0.931756  0.182905
9     1  0.417808  0.304561  0.140873  0.310562
```

Share Edit Follow

edited Mar 28, 2018 at 8:13

answered Apr 28, 2015 at 14:19
fixxxer
15.2k 15 58 76

13 Does it make a copy? – user3226167 Jun 2, 2017 at 2:02

44 @NicholasMorley - This isn't the best answer if you have, say, 1000 columns in your df. – AGS Jul 21, 2017 at 20:19

2 it doesn't seem like you're assigning to `<df>.columns` like you claim initially – 3pitt Feb 19, 2018 at 19:09

15 This is the best answer for a small number of columns. – RyanC Apr 19, 2018 at 23:12

7 This is just a copy of @freddygv 's earlier answer. That one should be the accepted answer, not this. – James Hirschorn May 4, 2018 at 21:40

300

For pandas >= 1.3 (Edited in 2022):

```
df.insert(0, 'mean', df.pop('mean'))
```

How about (for Pandas < 1.3, the original answer)

```
df.insert(0, 'mean', df['mean'])
```

https://pandas.pydata.org/pandas-docs/stable/user_guide/dsintro.html#column-selection-addition-deletion

Share Edit Follow

edited Feb 7 at 23:25
Jongwook Choi
7,396 3 22 18

answered Nov 9, 2012 at 21:04
Wes McKinney
97.1k 30 140 108

73 Could this be a future feature add to `pandas` ? something like `df.move(0,df.mean)` ? – jason May 27, 2014 at 1:23

8 Beautiful. And it happens in place, too. – cucu8 Aug 2, 2018 at 10:29

7 This is a scalable solution since other solutions are manually typing column names. – CKM Aug 19, 2019 at 17:00

15 This works for the OP's question, when creating a new column, but it doesn't for moving a column; attempt to move results in `*** ValueError: cannot insert mean, already exists` – SpinUp __ A Davis Feb 24, 2020 at 21:07

9 This is a clean solution. The modern API method is: `df.insert(0, 'mean', df['mean'])` – Sumax Jul 29, 2020 at 5:22

188

In your case,

```
df = df.reindex(columns=['mean',0,1,2,3,4])
```

will do exactly what you want.

In my case (general form):

```
df = df.reindex(columns=sorted(df.columns))
df = df.reindex(columns=(['opened'] + list([a for a in df.columns if a !=
'opened']) ))
```

Share Edit Follow

edited Jul 8, 2019 at 23:01
Mr_and_Mrs_D
30.7k 37 174 353

answered Aug 30, 2016 at 21:57
Alvaro Silvino
9,131 11 50 80

2 I tried to set `copy=False` but it looks like `reindex_axis` still creates a copy. – Konstantin Nov 27, 2017 at 9:10

1 @Konstantin can you create another question about this issue? It would be better to have more context – Alvaro Silvino Nov 27, 2017 at 14:11

111

```
import numpy as np
import pandas as pd
df = pd.DataFrame()
column_names = ['x','y','z','mean']
for col in column_names:
    df[col] = np.random.randint(0,100, size=10000)
```

You can try out the following solutions :

Solution 1:

```
df = df[ ['mean'] + [ col for col in df.columns if col != 'mean' ] ]
```

Solution 2:

```
df = df[['mean', 'x', 'y', 'z']]
```

Solution 3:

```
col = df.pop("mean")
df = df.insert(0, col.name, col)
```

Solution 4:

```
df.set_index(df.columns[-1], inplace=True)
df.reset_index(inplace=True)
```

Solution 5:

```
cols = list(df)
cols = [cols[-1]] + cols[:-1]
df = df[cols]
```

solution 6:

```
order = [1,2,3,0] # setting column's order
df = df[[df.columns[i] for i in order]]
```

Time Comparison:

Solution 1:

> CPU times: user 1.05 ms, sys: 35 µs, total: 1.08 ms Wall time: 995 µs

Solution 2:

> CPU times: user 933 µs, sys: 0 ns, total: 933 µs Wall time: 800 µs

Solution 3:

> CPU times: user 0 ns, sys: 1.35 ms, total: 1.35 ms Wall time: 1.08 ms

Solution 4:

> CPU times: user 1.23 ms, sys: 45 µs, total: 1.27 ms Wall time: 986 µs

Solution 5:

> CPU times: user 1.09 ms, sys: 19 µs, total: 1.11 ms Wall time: 949 µs

Solution 6:

> CPU times: user 955 µs, sys: 34 µs, total: 989 µs Wall time: 859 µs

Share Edit Follow

edited Nov 9, 2019 at 6:57

answered Nov 9, 2019 at 6:24
Pygirl
12.6k 4 28 41

2 solution 1 is what I needed as I have too many columns(53), thanks – ratnesh Apr 1, 2020 at 13:15

2 @Pygirl wich value shows real comsumed time? (user, sys, total or wall time) – sergzemsk Apr 10, 2020 at 21:46

2 This is for me the best answer for the problem. So many solutions(including one that I needed) and simple approach. Thanks! – Gustavo Rottgering May 15, 2020 at 0:36

2 **Solution 6** (no list comprehension): `df = df.iloc[:, [1, 2, 3, 0]]` – Dmitriy Work May 20, 2020 at 16:55

2 @sergzemsk: stackoverflow.com/a/55702033/6660373. I compare by wall time. – Pygirl Sep 29, 2020 at 4:58

81

You need to create a new list of your columns in the desired order, then use `df = df[cols]` to rearrange the columns in this new order.

```
cols = ['mean']  + [col for col in df if col != 'mean']
df = df[cols]
```

You can also use a more general approach. In this example, the last column (indicated by -1) is inserted as the first column.

```
cols = [df.columns[-1]] + [col for col in df if col != df.columns[-1]]
df = df[cols]
```

You can also use this approach for reordering columns in a desired order if they are present in the DataFrame.

```
inserted_cols = ['a', 'b', 'c']
cols = ([col for col in inserted_cols if col in df]
          + [col for col in df if col not in inserted_cols])
df = df[cols]
```

Share Edit Follow

edited Nov 19, 2019 at 13:16
Kevin Markham
5,448 1 25 36

answered Aug 21, 2015 at 2:18
Alexander
101k 28 192 186

66

Suppose you have `df` with columns `A` `B` `C`.

The most simple way is:

```
df = df.reindex(['B','C','A'], axis=1)
```

Share Edit Follow

edited Jan 15, 2021 at 15:35
Asclepius
53.3k 16 155 138

answered May 30, 2020 at 5:12
liangli
1,074 8 11

1 Note that this will only return a reindexed data frame - not change the `df` instance which is being used. If you want to use the reindexed df, simply use the returned value: `df2 = df.reindex(['B', 'C', 'A'], axis=1)`. Thanks for this answer! – Andreas Forslöw Dec 8, 2020 at 9:13

64

If your column names are too-long-to-type then you could specify the new order through a list of integers with the positions:

Data:

```
          0         1         2         3         4      mean
0  0.397312  0.361846  0.719802  0.575223  0.449205  0.500678
1  0.287256  0.522337  0.992154  0.584221  0.042739  0.485741
2  0.884812  0.464172  0.149296  0.167698  0.793634  0.491923
3  0.656891  0.500179  0.046006  0.862769  0.651065  0.543382
4  0.673702  0.223489  0.438760  0.468954  0.308509  0.422683
5  0.764020  0.093050  0.100932  0.572475  0.416471  0.389390
6  0.259181  0.248186  0.626101  0.556980  0.559413  0.449972
7  0.400591  0.075461  0.096072  0.308755  0.157078  0.207592
8  0.639745  0.368987  0.340573  0.997547  0.011892  0.471749
9  0.050582  0.714160  0.168839  0.899230  0.359690  0.438500
```

Generic example:

```
new_order = [3,2,1,4,5,0]
print(df[df.columns[new_order]])

          3         2         1         4      mean         0
0  0.575223  0.719802  0.361846  0.449205  0.500678  0.397312
1  0.584221  0.992154  0.522337  0.042739  0.485741  0.287256
2  0.167698  0.149296  0.464172  0.793634  0.491923  0.884812
3  0.862769  0.046006  0.500179  0.651065  0.543382  0.656891
4  0.468954  0.438760  0.223489  0.308509  0.422683  0.673702
5  0.572475  0.100932  0.093050  0.416471  0.389390  0.764020
6  0.556980  0.626101  0.248186  0.559413  0.449972  0.259181
7  0.308755  0.096072  0.075461  0.157078  0.207592  0.400591
8  0.997547  0.340573  0.368987  0.011892  0.471749  0.639745
9  0.899230  0.168839  0.714160  0.359690  0.438500  0.050582
```

Although it might seem like I'm just explicitly typing the column names in a different order, the fact that there's a column 'mean' should make it clear that `new_order` relates to actual positions and not column names.

For the specific case of OP's question:

```
new_order = [-1,0,1,2,3,4]
df = df[df.columns[new_order]]
print(df)

       mean         0         1         2         3         4
0  0.500678  0.397312  0.361846  0.719802  0.575223  0.449205
1  0.485741  0.287256  0.522337  0.992154  0.584221  0.042739
2  0.491923  0.884812  0.464172  0.149296  0.167698  0.793634
3  0.543382  0.656891  0.500179  0.046006  0.862769  0.651065
4  0.422683  0.673702  0.223489  0.438760  0.468954  0.308509
5  0.389390  0.764020  0.093050  0.100932  0.572475  0.416471
6  0.449972  0.259181  0.248186  0.626101  0.556980  0.559413
7  0.207592  0.400591  0.075461  0.096072  0.308755  0.157078
8  0.471749  0.639745  0.368987  0.340573  0.997547  0.011892
9  0.438500  0.050582  0.714160  0.168839  0.899230  0.359690
```

The main problem with this approach is that calling the same code multiple times will create different results each time, so one needs to be careful :)

Share Edit Follow

edited Feb 9, 2021 at 14:44

answered Aug 20, 2018 at 17:35
Yuca
5,890 3 23 40

58

This question has been answered before but `reindex_axis` is deprecated now so I would suggest to use:

```
df = df.reindex(sorted(df.columns), axis=1)
```

For those who want to specify the order they want instead of just sorting them, here's the solution spelled out:

```
df = df.reindex(['the','order','you','want'], axis=1)
```

Now, how you want to sort the list of column names is really not a `pandas` question, that's a Python list manipulation question. There are many ways of doing that, and I think this answer has a very neat way of doing it.

Share Edit Follow

edited Jan 15, 2021 at 15:40
Asclepius
53.3k 16 155 138

answered Jan 4, 2013 at 6:04
dmvianna
14.4k 18 78 105

21 No, that's different. There the user wants to sort all columns by name. Here they want to move one column to the first column while leaving the order of the other columns untouched. – smci Apr 17, 2013 at 13:06

2 What if you don't want them sorted? – Chankey Pathak Jun 8, 2017 at 10:16

1 @mins I hope the edit above is clear enough. :) – dmvianna Dec 16, 2020 at 0:14

27

I think this is a slightly neater solution:

```
df.insert(0, 'mean', df.pop("mean"))
```

This solution is somewhat similar to @JoeHeffer 's solution but this is one liner.

Here we remove the column `"mean"` from the dataframe and attach it to index `0` with the same column name.

Share Edit Follow

edited Jan 15, 2021 at 15:38
Asclepius
53.3k 16 155 138

answered Nov 5, 2019 at 16:33
erncyp
1,479 20 22

1 Any new column you create is added to the end, so I guess it would be `df["mean"] = df.pop("mean")` – erncyp Jun 12, 2020 at 15:55

27

You can reorder the dataframe columns using a list of names with:

```
df = df.filter(list_of_col_names)
```

Share Edit Follow

answered Apr 13, 2021 at 13:36
Sam Murphy
745 9 22

21

I ran into a similar question myself, and just wanted to add what I settled on. I liked the `reindex_axis() method` for changing column order. This worked:

```
df = df.reindex_axis(['mean'] + list(df.columns[:-1]), axis=1)
```

An alternate method based on the comment from @Jorge:

```
df = df.reindex(columns=['mean'] + list(df.columns[:-1]))
```

Although `reindex_axis` seems to be slightly faster in micro benchmarks than `reindex`, I think I prefer the latter for its directness.

Share Edit Follow

edited Aug 10, 2018 at 7:11

answered Aug 27, 2014 at 19:49
clocker
1,346 9 16

7 This was a nice solution, but reindex_axis will be deprecated. I used reindex, and it worked just fine. – Jorge Aug 8, 2018 at 21:32

20

This function avoids you having to list out every variable in your dataset just to order a few of them.

```
def order(frame,var):
    if type(var) is str:
        var = [var] #let the command take a string or list
    varlist =[w for w in frame.columns if w not in var]
    frame = frame[var+varlist]
    return frame
```

It takes two arguments, the first is the dataset, the second are the columns in the data set that you want to bring to the front.

So in my case I have a data set called Frame with variables A1, A2, B1, B2, Total and Date. If I want to bring Total to the front then all I have to do is:

```
frame = order(frame,['Total'])
```

If I want to bring Total and Date to the front then I do:

```
frame = order(frame,['Total','Date'])
```

EDIT:

Another useful way to use this is, if you have an unfamiliar table and you're looking with variables with a particular term in them, like VAR1, VAR2,... you may execute something like:

```
frame = order(frame,[v for v in frame.columns if "VAR" in v])
```

Share Edit Follow

edited Jul 23, 2018 at 23:10

answered Jul 29, 2014 at 19:30
seeiespi
3,408 2 32 36

19

Simply do,

```
df = df[['mean'] + df.columns[:-1].tolist()]
```

Share Edit Follow

edited Jul 12, 2017 at 8:23

answered Apr 28, 2015 at 9:50
Napitupulu Jon
7,513 2 21 23

1 A variation of this worked well for me. With an existing list, `headers` , that was used to create a dict that was then used to create the DataFrame, I called `df.reindex(columns=headers)` . The only problem I ran into was I had already called `df.set_index('some header name', inplace=True)` , so when the reindex was done, it added another column named `some header name` since the original column was now the index. As for the syntax specified above, `['mean'] + df.columns` in the python interpreter gives me `Index(u'meanAddress', u'meanCity', u'meanFirst Name'...` – hlongmore Jun 20, 2017 at 19:41

1 @hlongmore: I don't know your prior code is, but the edit should work (using 0.19.2) – Napitupulu Jon Jun 21, 2017 at 0:56

18

Here's a way to move one existing column that will modify the existing dataframe in place.

```
my_column = df.pop('column name')
df.insert(3, my_column.name, my_column)  # Is in-place
```

Share Edit Follow

edited Aug 24, 2021 at 22:45
Asclepius
53.3k 16 155 138

answered Jan 4, 2018 at 13:25
Joe Heffer
605 7 9

12

You could do the following (borrowing parts from Aman's answer):

```
cols = df.columns.tolist()
cols.insert(0, cols.pop(-1))

cols
>>>['mean', 0L, 1L, 2L, 3L, 4L]

df = df[cols]
```

Share Edit Follow

answered Dec 8, 2016 at 15:22
otteheng
594 1 8 26

11

Just type the column name you want to change, and set the index for the new location.

```
def change_column_order(df, col_name, index):
    cols = df.columns.tolist()
    cols.remove(col_name)
    cols.insert(index, col_name)
    return df[cols]
```

For your case, this would be like:

```
df = change_column_order(df, 'mean', 0)
```

Share Edit Follow

answered May 6, 2016 at 11:39
ccerhan
612 6 7

11

Moving any column to any position:

```
import pandas as pd
df = pd.DataFrame({"A": [1,2,3],
                   "B": [2,4,8],
                   "C": [5,5,5]})

cols = df.columns.tolist()
column_to_move = "C"
new_position = 1

cols.insert(new_position, cols.pop(cols.index(column_to_move)))
df = df[cols]
```

Share Edit Follow

answered Feb 27, 2018 at 14:05
pomber
22.1k 10 77 91

7

I wanted to bring two columns in front from a dataframe where I do not know exactly the names of all columns, because they are generated from a pivot statement before. So, if you are in the same situation: To bring columns in front that you know the name of and then let them follow by "all the other columns", I came up with the following general solution:

```
df = df.reindex_axis(['Col1','Col2'] + list(df.columns.drop(['Col1','Col2'])),
axis=1)
```

Share Edit Follow

edited Jan 15, 2021 at 15:44
Asclepius
53.3k 16 155 138

answered Jul 27, 2017 at 9:21
matthhias
201 3 6

6

Here is a very simple answer to this(only one line).

You can do that after you added the 'n' column into your df as follows.

```
import numpy as np
import pandas as pd

df = pd.DataFrame(np.random.rand(10, 5))
df['mean'] = df.mean(1)
df
           0           1           2           3           4        mean
0   0.929616    0.316376    0.183919    0.204560    0.567725    0.440439
1   0.595545    0.964515    0.653177    0.748907    0.653570    0.723143
2   0.747715    0.961307    0.008388    0.106444    0.298704    0.424512
3   0.656411    0.809813    0.872176    0.964648    0.723685    0.805347
4   0.642475    0.717454    0.467599    0.325585    0.439645    0.518551
5   0.729689    0.994015    0.676874    0.790823    0.170914    0.672463
6   0.026849    0.800370    0.903723    0.024676    0.491747    0.449473
7   0.526255    0.596366    0.051958    0.895090    0.728266    0.559587
8   0.818350    0.500223    0.810189    0.095969    0.218950    0.488736
9   0.258719    0.468106    0.459373    0.709510    0.178053    0.414752


### here you can add below line and it should work
# Don't forget the two (()) 'brackets' around columns names.Otherwise, it'll
give you an error.

df = df[list(('mean',0, 1, 2,3,4))]
df

        mean           0           1           2           3           4
0   0.440439    0.929616    0.316376    0.183919    0.204560    0.567725
1   0.723143    0.595545    0.964515    0.653177    0.748907    0.653570
2   0.424512    0.747715    0.961307    0.008388    0.106444    0.298704
3   0.805347    0.656411    0.809813    0.872176    0.964648    0.723685
4   0.518551    0.642475    0.717454    0.467599    0.325585    0.439645
5   0.672463    0.729689    0.994015    0.676874    0.790823    0.170914
6   0.449473    0.026849    0.800370    0.903723    0.024676    0.491747
7   0.559587    0.526255    0.596366    0.051958    0.895090    0.728266
8   0.488736    0.818350    0.500223    0.810189    0.095969    0.218950
9   0.414752    0.258719    0.468106    0.459373    0.709510    0.178053
```

Share Edit Follow

answered Jun 18, 2020 at 19:30

rra
745 5 16

6

You can use a set which is an *unordered collection of unique elements* to do keep the "order of the other columns untouched":

```
other_columns = list(set(df.columns).difference(["mean"])) #[0, 1, 2, 3, 4]
```

Then, you can use a lambda to move a specific column to the front by:

```
In [1]: import numpy as np

In [2]: import pandas as pd

In [3]: df = pd.DataFrame(np.random.rand(10, 5))

In [4]: df["mean"] = df.mean(1)

In [5]: move_col_to_front = lambda df, col:
```

```
df[[col]+list(set(df.columns).difference([col]))]

In [6]: move_col_to_front(df, "mean")
Out[6]:
       mean         0         1         2         3         4
0  0.697253  0.600377  0.464852  0.938360  0.945293  0.537384
1  0.609213  0.703387  0.096176  0.971407  0.955666  0.319429
2  0.561261  0.791842  0.302573  0.662365  0.728368  0.321158
3  0.518720  0.710443  0.504060  0.663423  0.208756  0.506916
4  0.616316  0.665932  0.794385  0.163000  0.664265  0.793995
5  0.519757  0.585462  0.653995  0.338893  0.714782  0.305654
6  0.532584  0.434472  0.283501  0.633156  0.317520  0.994271
7  0.640571  0.732680  0.187151  0.937983  0.921097  0.423945
8  0.562447  0.790987  0.200080  0.317812  0.641340  0.862018
9  0.563092  0.811533  0.662709  0.396048  0.596528  0.348642

In [7]: move_col_to_front(df, 2)
Out[7]:
          2         0         1         3         4      mean
0  0.938360  0.600377  0.464852  0.945293  0.537384  0.697253
1  0.971407  0.703387  0.096176  0.955666  0.319429  0.609213
2  0.662365  0.791842  0.302573  0.728368  0.321158  0.561261
3  0.663423  0.710443  0.504060  0.208756  0.506916  0.518720
4  0.163000  0.665932  0.794385  0.664265  0.793995  0.616316
5  0.338893  0.585462  0.653995  0.714782  0.305654  0.519757
6  0.633156  0.434472  0.283501  0.317520  0.994271  0.532584
7  0.937983  0.732680  0.187151  0.921097  0.423945  0.640571
8  0.317812  0.790987  0.200080  0.641340  0.862018  0.562447
9  0.396048  0.811533  0.662709  0.596528  0.348642  0.563092
```

Share Edit Follow

edited Jul 4, 2020 at 10:34

answered Jul 4, 2020 at 10:19

Mathia Haure-Touzé
301 2 6

5

Just flipping helps often.

```
df[df.columns[::-1]]
```

Or just shuffle for a look.

```
import random
cols = list(df.columns)
random.shuffle(cols)
df[cols]
```

Share Edit Follow

edited Apr 10, 2020 at 11:45

answered Apr 10, 2020 at 11:39

plhn
4,787 4 43 47

4

You can use `reindex` which can be used for both axis:

```
df
#           0         1         2         3         4      mean
# 0  0.943825  0.202490  0.071908  0.452985  0.678397  0.469921
# 1  0.745569  0.103029  0.268984  0.663710  0.037813  0.363821
# 2  0.693016  0.621525  0.031589  0.956703  0.118434  0.484254
# 3  0.284922  0.527293  0.791596  0.243768  0.629102  0.495336
# 4  0.354870  0.113014  0.326395  0.656415  0.172445  0.324628
# 5  0.815584  0.532382  0.195437  0.829670  0.019001  0.478415
# 6  0.944587  0.068690  0.811771  0.006846  0.698785  0.506136
# 7  0.595077  0.437571  0.023520  0.772187  0.862554  0.538182
# 8  0.700771  0.413958  0.097996  0.355228  0.656919  0.444974
# 9  0.263138  0.906283  0.121386  0.624336  0.859904  0.555009

df.reindex(['mean', *range(5)], axis=1)

#        mean         0         1         2         3         4
# 0  0.469921  0.943825  0.202490  0.071908  0.452985  0.678397
# 1  0.363821  0.745569  0.103029  0.268984  0.663710  0.037813
# 2  0.484254  0.693016  0.621525  0.031589  0.956703  0.118434
# 3  0.495336  0.284922  0.527293  0.791596  0.243768  0.629102
# 4  0.324628  0.354870  0.113014  0.326395  0.656415  0.172445
# 5  0.478415  0.815584  0.532382  0.195437  0.829670  0.019001
# 6  0.506136  0.944587  0.068690  0.811771  0.006846  0.698785
# 7  0.538182  0.595077  0.437571  0.023520  0.772187  0.862554
# 8  0.444974  0.700771  0.413958  0.097996  0.355228  0.656919
# 9  0.555009  0.263138  0.906283  0.121386  0.624336  0.859904
```

Share Edit Follow

answered Dec 18, 2017 at 15:24
silgon
6,762 6 42 66

4

Hackiest method in the book

```
df.insert(0, "test", df["mean"])
df = df.drop(columns=["mean"]).rename(columns={"test": "mean"})
```

Share Edit Follow

edited Jan 14, 2021 at 22:19
Asclepius
53.3k 16 155 138

answered Apr 11, 2019 at 17:58
Kaustubh J
672 8 9

4

A pretty straightforward solution that worked for me is to use `.reindex` on `df.columns`:

```
df = df[df.columns.reindex(['mean', 0, 1, 2, 3, 4])[0]]
```

Share Edit Follow

edited Jan 15, 2021 at 15:43
Asclepius
53.3k 16 155 138

answered May 8, 2020 at 15:42
CSQL
106 1 3

3

Here is a function to do this for any number of columns.

```
def mean_first(df):
    ncols = df.shape[1]        # Get the number of columns
    index = list(range(ncols)) # Create an index to reorder the columns
    index.insert(0,ncols)      # This puts the last column at the front
    return(df.assign(mean=df.mean(1)).iloc[:,index]) # new df with last column
(mean) first
```

Share Edit Follow

edited Feb 28, 2018 at 11:49

answered Jan 29, 2018 at 18:57
freeB
91 3

3

A simple approach is using `set()` , in particular when you have a long list of columns and do not want to handle them manually:

```
cols = list(set(df.columns.tolist()) - set(['mean']))
cols.insert(0, 'mean')
df = df[cols]
```

Share Edit Follow

edited Jan 14, 2021 at 22:18
Asclepius
53.3k 16 155 138

answered Sep 12, 2017 at 2:06
Shoresh
2,493 1 16 9

2 One caution: the order of columns goes away if you put it into set – pnv Mar 6, 2018 at 5:31

3

How about using `T` ?

```
df = df.T.reindex(['mean', 0, 1, 2, 3, 4]).T
```

Share Edit Follow

edited Jan 15, 2021 at 15:43
Asclepius
53.3k 16 155 138

answered Jun 26, 2016 at 23:46
ZEE
188 1 11

2

I believe @Aman's answer is the best if you know the location of the other column.

If you don't know the location of `mean` , but only have its name, you cannot resort directly to `cols = cols[-1:] + cols[:-1]` . Following is the next-best thing I could come up with:

```
meanDf = pd.DataFrame(df.pop('mean'))
# now df doesn't contain "mean" anymore. Order of join will move it to left or
right:
meanDf.join(df) # has mean as first column
df.join(meanDf) # has mean as last column
```

Share Edit Follow

edited May 23, 2017 at 12:10
Community Bot
1 1

answered Mar 22, 2015 at 14:43
FooBar
15.2k 18 76 163