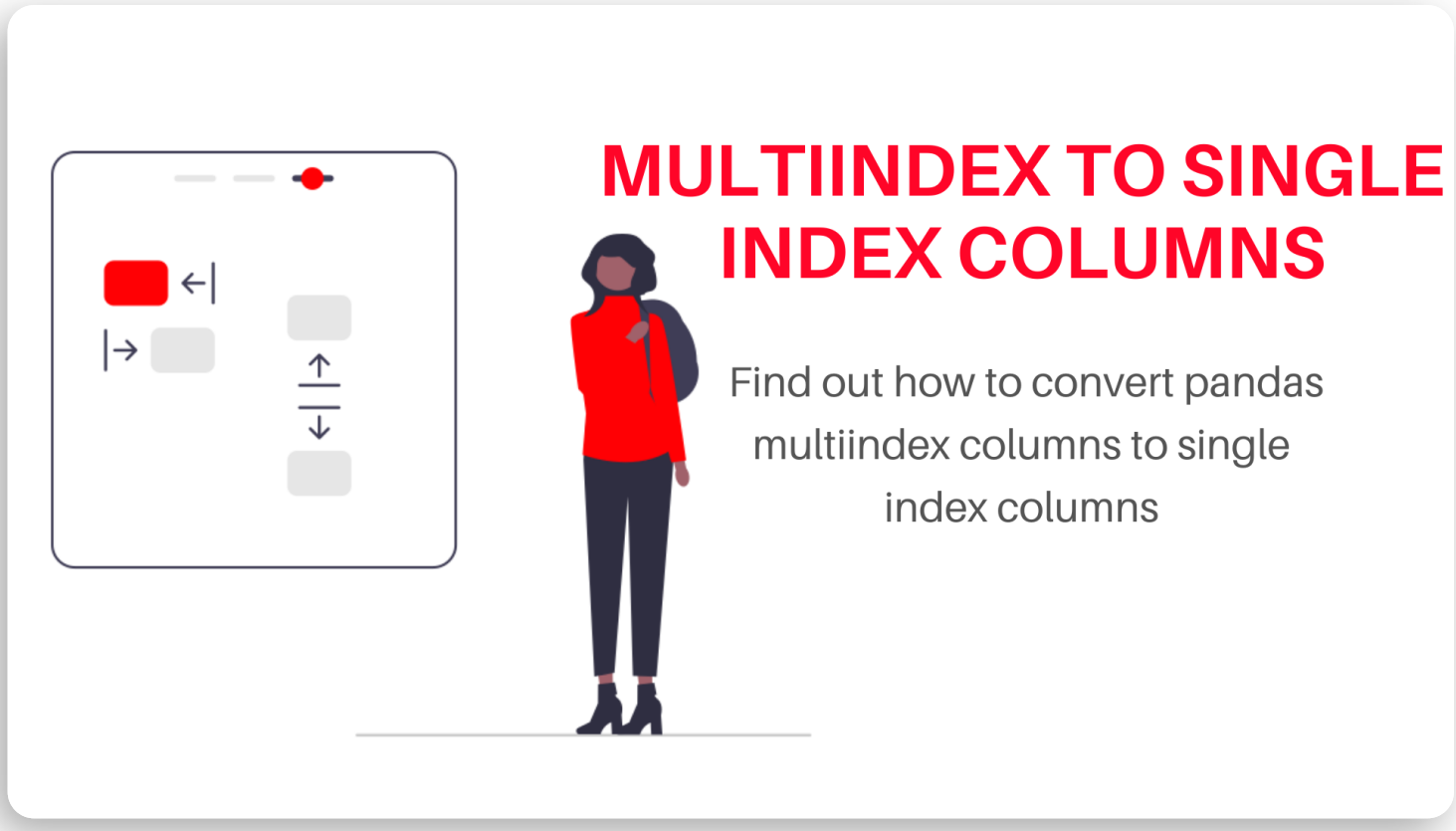
< BUGGY / PROGRAMMER >
Learn with your mistake

Understand How To Convert Pandas Multiindex Columns To Single Index Columns In 2022



July 30, 2022 | Pandas, Pandas Columns, Pandas Multi index columns, pandas Single Index, python pandas, python programming

Hierarchical Indexing or Multi indexing in Pandas Data frames are used when your data is indexed using more than one column. Find out how to convert pandas multiindex columns to single index columns in this article. The columns are indexed using a code from the Pandas Library of Python and this is done in order to derive more information out of raw data in tables.

Note: The tabular form of data is generally indexed using a single column or multiple columns in a table to help with better accessibility of elements in the dataset.

In this article, we'll go over a brief introduction to the Pandas library of Python, what indexing in columns is, how to do indexing in columns and how to make multiple indexes in a data frame, and finally, conversion of a pandas multiindex columns to single index columns.

Also, read -> Tidy Data in Python: Easy ways to Clean your data!

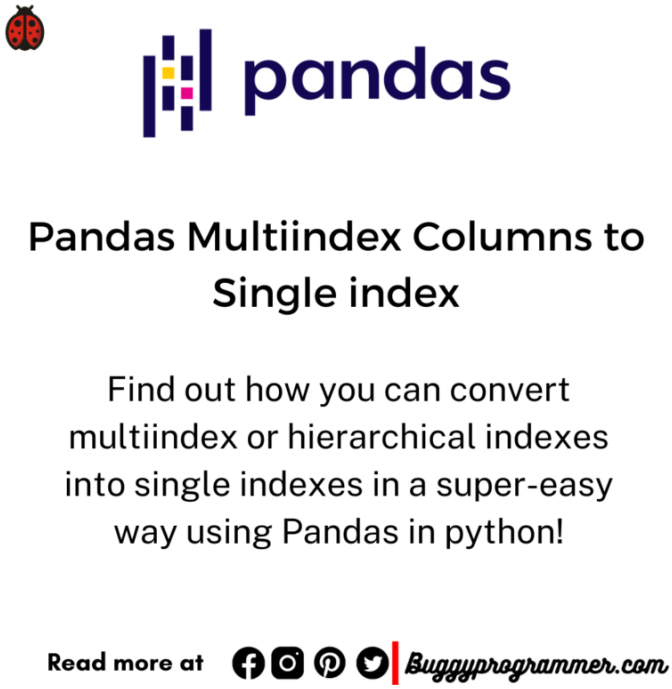
What Is Pandas?

Pandas is — an open-source library developed in 2008, essentially for working with relational or labeled data or both, easily and almost replicating Excel in python. It can work with many data structures and operations for manipulating data in text and/or numbers and time series. Built on top of the NumPy library in python, it is fast and has high performance and usability.

For more about Pandas, check out the official documentation here: Pandas Official website

Check out this free Pandas tutorial if you are new to Pandas: Kaggle Pandas tutorial

What Is Indexing In Columns?



Indexing with the Pandas Library of Python is the selection of particular rows and columns of data from a Pandas DataFrame. It could mean all rows and some columns, some rows & all columns, or some of both rows and columns. If you are aware of how the Set Theory works and how Venn diagrams work, then it is similar to how Subset selection works.

Subsetting or indexing in columns is useful in terms of taking bits and pieces of data to understand more and more from the data. It is therefore essential to know how the conversion of pandas multiindex columns to single index columns works to get better information out of the data.

How To Do Indexing In Columns Using Pandas?

Indexing in Columns with Pandas can happen using any of the following different codes:

- pandas.loc
- pandas.iloc
- pandas.at
- pandas.iat
- pandas.ix

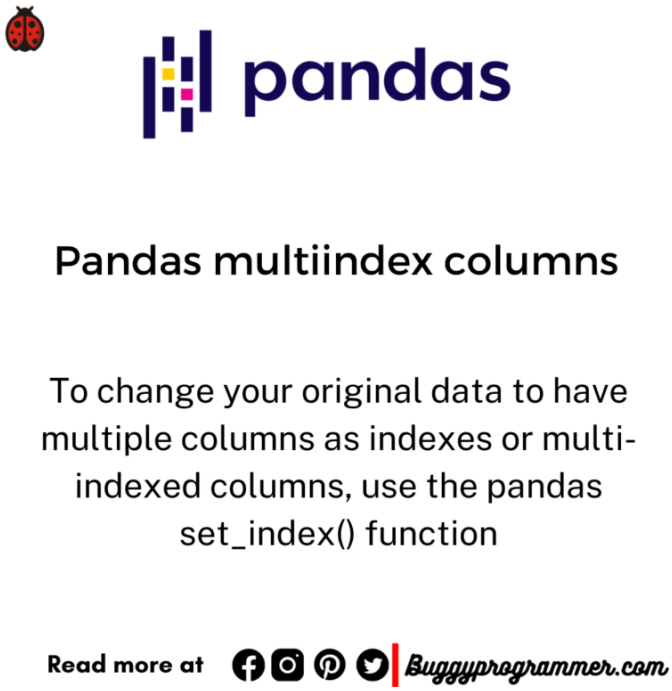
These codes are called indexers and they are used to index data out of tables in data frames. They are multi-axes indexers and can be used to index data out of rows and columns.

To understand more about how they work, check out this youtube video:

How To Make Multiple Indexes In Pandas Dataframes?

To make a multiindex column in a pandas data frame is fairly simple if one understands how to set an index.

Consider your data frame has 5 columns, columns A, B, C, D, and E.



To make columns A, B, and C as the indexes for the table; one can use the following code:

```
import pandas as pd
```

```
df_multi_index = df.set_index(['A','B','C'])
```

As you can observe above, the code presents a fairly easy way to set multiple indexes, using a list as an input in the .set_index() function of Pandas, using which one can set the index easily. To convert pandas multiindex columns to single index columns, we'll use another set of code offered by Pandas.

Check out this example of hierarchical indexing before we head into converting pandas multiindex columns to single index columns.

```
In [11]: df.head()
```

Out[11]:

	location	date	variant	num_sequences	perc_sequences	num_sequences_total
0	Angola	2020-07-06	Alpha	0	0.0	3
1	Angola	2020-07-06	B.1.1.277	0	0.0	3
2	Angola	2020-07-06	B.1.1.302	0	0.0	3
3	Angola	2020-07-06	B.1.1.519	0	0.0	3
4	Angola	2020-07-06	B.1.160	0	0.0	3

```
In [12]: df.set_index(['location','variant'])
```

Out[12]:

		date	num_sequences	perc_sequences	num_sequences_total
location	variant				
Angola	Alpha	2020-07-06	0	0.0	3
	B.1.1.277	2020-07-06	0	0.0	3
	B.1.1.302	2020-07-06	0	0.0	3
	B.1.1.519	2020-07-06	0	0.0	3
	B.1.160	2020-07-06	0	0.0	3
...	...	...	...	...	...
Zimbabwe	Omicron	2021-11-01	0	0.0	6
	S:677H.Robin1	2021-11-01	0	0.0	6
	S:677P.Pelican	2021-11-01	0	0.0	6
	others	2021-11-01	0	0.0	6
	non_who	2021-11-01	0	0.0	6

100416 rows × 4 columns

As you can see, by converting this time series into a multiindex column data frame, the dataset has changed considerably and it is a very straightforward code to change the indexes of the data frame. Let's get to the point in consideration now, i.e., **Conversion of Pandas Multiindex columns to single index columns**.

Find the official documentation of Pandas for setting the index here: pandas.DataFrame.set_index()

Conversion Of Pandas Multiindex Columns To Single Index Columns

Conversion of Pandas multiindex columns to single index columns

To revert back to your original dataset with no changes in the indexing or to convert back from multiindex columns to single index columns, use the reset_index() function with Pandas in Python!

Read more at Buggyprogrammer.com

To convert a pandas data frame with multiindex columns is fairly simple. It is a simple one-liner code or two-liner code that you can use to convert the pandas multiindex columns to single index columns.

Let us look at the code mentioned below and revert or convert back the index in the example given above to arrive at the original dataset as it was.

Note: There are two methods for you to perform this conversion, i.e., using the 'level' method or the column names themselves.

```
import pandas as pd


df.set_index(['A','B','C'])


df.reset_index( level = [0,1] )
###OR
df.reset_index(['A','B','C'])
```

Now from the code above, as you can see that the hierarchical indexes are three columns, in Python, these are levels 0,1 with 0 being the highest in order of hierarchy, followed by 1 being the second level or the next and lowest in order of hierarchy.

Otherwise, one can also reset the index, by using the column names of the columns being used as the hierarchical or multi indexes in the data frame.

The difference between using either of these two methods depends on the number of columns being used as indexes in your dataframe. If your dataframe has over 3 or 4 columns being used as indexes for some reason, then the dataframe indexes can probably be reset using the 'level' method or if your dataframe has columns with complicated names, then it is feasible to use the 'level' method, whereas in a case where there are few columns being used as indexes and can be named easily, the column names can be entered as the input for the reset_index function of Pandas.

Check out how the reset_index function works in both the ways for the aforementioned example:

Method 1

```
In [6]: df1 = df.set_index(['location','variant'])
```

```
In [7]: df1.head()
```

Out[7]:

location	variant	date	num_sequences	perc_sequences	num_sequences_total
Angola	Alpha	2020-07-06	0	0.0	3
	B.1.1.277	2020-07-06	0	0.0	3
	B.1.1.302	2020-07-06	0	0.0	3
	B.1.1.519	2020-07-06	0	0.0	3
	B.1.160	2020-07-06	0	0.0	3

```
In [8]: df1 = df1.reset_index( level = [0])
```

```
In [9]: df1.head()
```

Out[9]:

variant	location	date	num_sequences	perc_sequences	num_sequences_total
Alpha	Angola	2020-07-06	0	0.0	3
B.1.1.277	Angola	2020-07-06	0	0.0	3
B.1.1.302	Angola	2020-07-06	0	0.0	3
B.1.1.519	Angola	2020-07-06	0	0.0	3
B.1.160	Angola	2020-07-06	0	0.0	3

Method 2

```
In [10]: df1 = df.set_index(['location','variant'])
```

```
In [11]: df1.head()
```

Out[11]:

location	variant	date	num_sequences	perc_sequences	num_sequences_total
Angola	Alpha	2020-07-06	0	0.0	3
	B.1.1.277	2020-07-06	0	0.0	3
	B.1.1.302	2020-07-06	0	0.0	3
	B.1.1.519	2020-07-06	0	0.0	3
	B.1.160	2020-07-06	0	0.0	3

```
In [12]: df1 = df1.reset_index(['location','variant'])
```

```
In [13]: df1.head()
```

Out[13]:

	location	variant	date	num_sequences	perc_sequences	num_sequences_total
0	Angola	Alpha	2020-07-06	0	0.0	3
1	Angola	B.1.1.277	2020-07-06	0	0.0	3
2	Angola	B.1.1.302	2020-07-06	0	0.0	3
3	Angola	B.1.1.519	2020-07-06	0	0.0	3
4	Angola	B.1.160	2020-07-06	0	0.0	3

As visible in both the methods, the reset_index function is fairly straightforward and returns the dataset to its original formation once the code is executed. The code can be applied in an easy way and the reset_index function with both its methods is versatile and allows for users to use any of the given methods to allow for easy reversion back to the original data.

If your data has come from a separate analysis elsewhere and has been decided to have multiple indexes, using the reset_index is the one way you can take your data and start around your analysis all over again without any manipulations in the indexes as done by the other researchers.

You can also watch this video tutorial to follow along on how to use the reset_index function in Pandas:

Python Basics Tutorial Pandas DataFrame Reset Index Method

Conclusion

When working with data in tables, it is important to know how to manipulate and change the tables to support the analyses accordingly. Knowing how to change the index does the same thing and hierarchically indexed data is generally found in company financials and in industrial costing methods where every department carries different specifics for costing.

Using Python for such data and converting the pandas multiindex columns to single index columns, ***is essential in order to revert back the data to its original format to allow for other types of analyses to take place.*** If you have your data with multiple indexes to work with, and you need to revert it back to its original form, I hope the aforementioned steps in this article helped you and if you think there is a different method or if you have any remarks about the method for mentioned converting the pandas multiindex columns to single index columns, do drop a comment below!

For more such content, check out our website -> Buggy Programmer