Reshaping Pandas DataFrames

Melt, Stack and Pivot functions

Soner Yildirim Follow
Apr 20 · 5 min read ★

Pandas is a very powerful Python data analysis library that expedites the preprocessing steps of your project. The core data structure of Pandas is **DataFrame** which represents data in tabular form with labeled rows and columns. In this post, I will try to explain how to reshape a dataframe by modifying row-column structure.

Photo by Paul Skorupskas on Unsplash

There are multiple ways to reshape a dataframe. We can choose the one that best fits the task at hand. The functions to reshape a dataframe:

- **Melt**
- **Stack and unstack**
- **Pivot**

As always, we start with importing numpy and pandas:

```
import pandas as pd
import numpy as np
```

Melt

Melt is used to convert wide dataframes to narrow ones. What I mean by wide is a dataframe with a high number of columns. Some dataframes are structured in a way that consecutive measurements or variables are represented as columns. In some cases, representing these columns as rows may fit better to our task.

Consider the following dataframe:

```
df1 = pd.DataFrame({'city':['A','B','C'],
                    'day1':[22,25,28],
                    'day2':[10,14,13],
                    'day3':[25,22,26],
                    'day4':[18,15,17],
                    'day5':[12,14,18]})
```

We have three different cities and measurements done on different days. We decide to represent these days as rows in a column. There will also be a column to show the measurements. We can easily accomplish this by using **melt** function:

```
df1.melt(id_vars=['city'])
```

```
df1.melt(id_vars=['city'])
```

	city	variable	value
0	A	day1	22
1	B	day1	25
2	C	day1	28
3	A	day2	10
4	B	day2	14
5	C	day2	13
6	A	day3	25
7	B	day3	22
8	C	day3	26
9	A	day4	18
10	B	day4	15
11	C	day4	17
12	A	day5	12
13	B	day5	14
14	C	day5	18

Variable and value column names are given by default. We can use **var_name** and **value_name** parameters of melt function to assign new column names. It will also look better if we sort the data by city column:

```
df1.melt(id_vars=['city'], var_name = 'date', value_name =
'temperature').sort_values(by='city').reset_index(drop=True)
```

```
df1.melt(id_vars=['city'], var_name = 'date',
         value_name = 'temperature').sort_values(by='city').reset_index(drop=True)
```

	city	date	temperature
0	A	day1	22
1	A	day2	10
2	A	day3	25
3	A	day4	18
4	A	day5	12
5	B	day1	25
6	B	day2	14
7	B	day3	22
8	B	day4	15
9	B	day5	14
10	C	day1	28
11	C	day2	13
12	C	day3	26
13	C	day4	17
14	C	day5	18

```
df1
```

	city	day1	day2	day3	day4	day5
0	A	22	10	25	18	12
1	B	25	14	22	15	14
2	C	28	13	26	17	18

Stack and unstack

Stack function kind of increases the index level of the dataframe. What I mean by increasing the level is:

- If dataframe has a simple column index, stack returns a series whose indices consist of row-column pairs of original dataframe.
- If dataframe has multi-level index, stack increases the index level.

It is better explained with examples. Consider the following dataframe:

```
df1
```

	city	day1	day2	day3	day4	day5
0	A	22	10	25	18	12
1	B	25	14	22	15	14
2	C	28	13	26	17	18

df1 has 3 rows and 6 columns with simple integer column index. If stack function is applied to df1, it will return a series with 3 x 6 = 18 rows. The index of the series will be [(0, ‘city’), (0, ‘day1’), … , (2, ‘day5’)].

Let's also check the shape and index:

```
df1.shape
(3,6)

df1.stack().shape
(18,)

df1.stack().index[0] #multilevel index
(0, 'city')
```

Stack and unstack functions are more commonly used for dataframes with multi-level indices. Let's create a dataframe with multi-level index:

```
tuples = [('A',1),('A',2),('A',3),('B',1),('A',2)]

index = pd.MultiIndex.from_tuples(tuples,
names=['first','second'])

df2 = pd.DataFrame(np.random.randint(10, size=(5,2)),
                   index=index, columns=['column_x',
'column_y'])
```

```
df1.stack()

0  city     A
   day1    22
   day2    10
   day3    25
   day4    18
   day5    12
1  city     B
```

```
   day1    25
   day2    14
   day3    22
   day4    15
   day5    14
2  city     C
   day1    28
   day2    13
   day3    26
   day4    17
   day5    18
dtype: object
```

If we apply stack function of index will be increased:

```
df_stacked = df2.sta
df_stacked
```

Now the names of the colums (column_x and column_y) are part of multi-level index. So the resulting dataframe has one column and a 3-level multi-index.

```
len(df_stacked.index.levels)
3

len(df2.index.levels)
2
```

Unstack is just the opposite of **stack**. If we apply unstack to the stacked dataframe, we will get back the original dataframe:

```
df_stacked.unstack().index
MultiIndex(levels=[['A', 'B'], [1, 2, 3]],
           codes=[[0, 0, 0, 1, 1], [0, 1, 2, 0, 1]],
           names=['first', 'second'])

df2.index
MultiIndex(levels=[['A', 'B'], [1, 2, 3]],
           codes=[[0, 0, 0, 1, 1], [0, 1, 2, 0, 1]],
           names=['first', 'second'])
```

Pivot

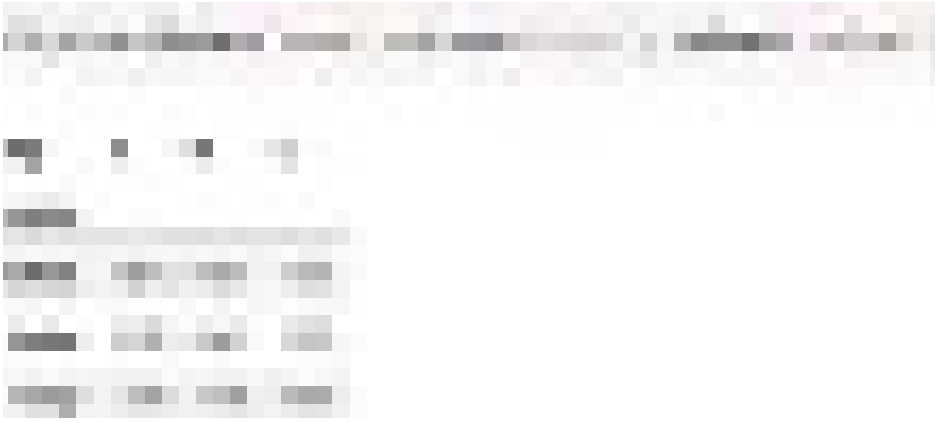
Pivot function can a lataframe from a different perspectiv nong variables by allowing to represe

Consider the following dataframe:

We want to see how values change according to city-name pairs. We can create a new representation of this dataframe with an index of names and columns of cities.

If a city-name pair does not exist, corresponding cell is filled with NaN.

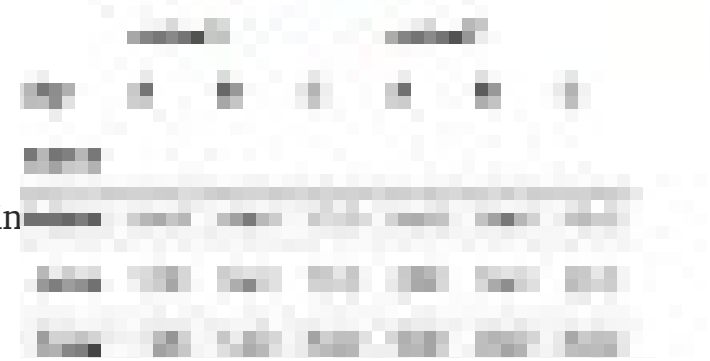
We do not have to see all the values at once. The values to put in the dataframe can be filtered using **values** parameter:



. . .

I think the success and prevalence of Pandas come from the versatile, powerful and easy-to-use functions to manipulate and analyze data. There are almost always multiple ways to do a task with Pandas. Since a big portion of time spent on a data science [illegible]rocessing steps, it is highly encour[illegible]

Thanks for readin[illegible]k.



Data Science Machine Learning Programming Pandas