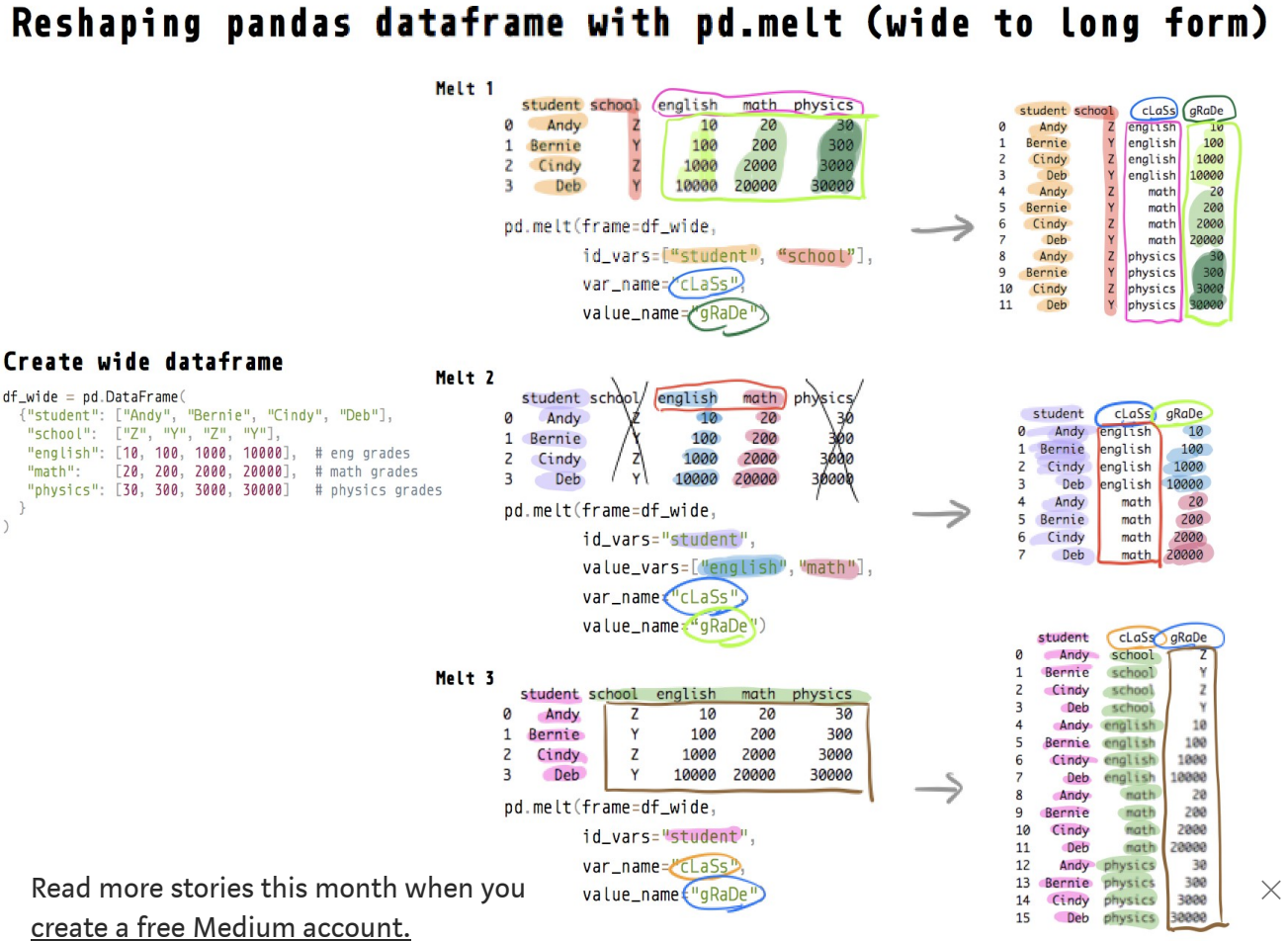
PYTHON PANDAS DATAFRAME RESHAPING

Reshape pandas dataframe with melt in Python — tutorial and visualization

Visualize how pd.melt reshapes pandas dataframes from wide to long form

Hause Follow
May 15 · 4 min read ★



Visual tutorial on how to use pd.melt() to reshape pandas dataframes from wide to long in Python

How does the pandas `melt()` method reshape dataframes? How do you reshape a dataframe from wide to long form? This tutorial will walk you through reshaping dataframes using `pd.melt()` (or the `melt` method associated with pandas dataframes.

I highly recommend you try the code in Python while you read this article. Try running this tutorial on my shared DeepNote notebook (you can only run but not edit this notebook).

Also, you might want to check out my numpy reshape tutorial:

Reshaping numpy arrays in Python — a step-by-step pictorial tutorial

This tutorial and cheatsheet provide visualizations to help you understand how numpy reshapes arrays.

towardsdatascience.com

. . .

Wide versus long dataframe

It's easiest to understand what a **wide** dataframe is or looks like if we look at one and compare it with a long dataframe.

	student	school	english	math	physics
0	Andy	Z	10	20	30
1	Bernie	Y	100	200	300
2	Cindy	Z	1000	2000	3000
3	Deb	Y	10000	20000	30000

Wide pandas dataframe can be melted/stacked using pd.melt()

And below is the corresponding dataframe (with the same information) but in the **long** form:

	student	school	cLaSs	gRaDe
0	Andy	Z	english	10
1	Bernie	Y	english	100
2	Cindy	Z	english	1000
3	Deb	Y	english	10000
4	Andy	Z	math	20
5	Bernie	Y	math	200
6	Cindy	Z	math	2000
7	Deb	Y	math	20000
8	Andy	Z	physics	30
9	Bernie	Y	physics	300
10	Cindy	Z	physics	3000
11	Deb	Y	physics	30000

Long pandas dataframe can be "unmelted" using pd.pivot_table() (not covered in this article)

Before we begin our `pd.melt` tutorial, let's recreate the wide dataframe above in Python with `pd.DataFrame`.

```
import pandas as pd

# create wide dataframe
df_wide = pd.DataFrame(
  {"student": ["Andy", "Bernie", "Cindy", "Deb"],
   "school":  ["Z", "Y", "Z", "Y"],
   "english": [10, 100, 1000, 10000],  # eng grades
   "math":    [20, 200, 2000, 20000],  # math grades
   "physics": [30, 300, 3000, 30000]   # physics grades
  }
)
```

. . .

Melt Example 1

We melt the dataframe by specifying the identifier columns via `id_vars`. The "leftover" non-identifier columns (english, math, physics) will be melted or stacked onto each other into one column.

A new indicator column will be created (contains values english, math, physics) and we can rename this new column (cLaSs) via `var_name`. We can also rename the column in which all the actual grades are contained (gRaDe) via `value_name`.

```
print(df_wide)
> student school  english   math  physics
     Andy      Z       10     20       30
   Bernie      Y      100    200      300
    Cindy      Z     1000   2000     3000
      Deb      Y    10000  20000    30000

df_wide.melt(id_vars=["student", "school"],
             var_name="cLaSs",  # rename
             value_name="gRaDe")  # rename

>  student school    cLaSs  gRaDe
0     Andy      Z  english     10
1   Bernie      Y  english    100
2    Cindy      Z  english   1000
3      Deb      Y  english  10000
4     Andy      Z     math     20
5   Bernie      Y     math    200
6    Cindy      Z     math   2000
7      Deb      Y     math  20000
8     Andy      Z  physics     30
9   Bernie      Y  physics    300
10   Cindy      Z  physics   3000
11     Deb      Y  physics  30000
```

Wide to long: new indicator column ″cLaSs″ + values melted/stacked ″gRaDe″ column

Melt Example 2

You can use `value_vars` to specify which columns you want to melt or stack into column (here, we exclude physics column, so `value_vars=["english", "math"]`). We also drop the school column from `id_vars`.

```
print(df_wide)
> student school  english   math  physics
     Andy      Z       10     20       30
   Bernie      Y      100    200      300
    Cindy      Z     1000   2000     3000
      Deb      Y    10000  20000    30000

df_wide.melt(id_vars="student",
             value_vars=["english", "math"],
             var_name="cLaSs",  # rename
             value_name="gRaDe")  # rename

> student    cLaSs  gRaDe
0    Andy  english     10
1  Bernie  english    100
2   Cindy  english   1000
3     Deb  english  10000
4    Andy     math     20
5  Bernie     math    200
6   Cindy     math   2000
7     Deb     math  2000
```

Melt Example 3

Finally, let's see what happens if we specify only the student column as the identifier column (`id_vars="student"`) but do not specify which columns you want to stack via `value_vars` . As a result, all non-identifier columns (school, english, math, physics) will be stacked into one column.

The resulting long dataframe looks wrong because now the cLaSs and gRaDe columns contain values that shouldn't be there. The point here is to show you how `pd.melt` works.

```
print(df_wide)
> student school  english   math  physics
     Andy      Z       10     20       30
   Bernie      Y      100    200      300
    Cindy      Z     1000   2000     3000
      Deb      Y    10000  20000    30000

df_wide.melt(id_vars="student",
             var_name="cLaSs",  # rename
             value_name="gRaDe")  # rename

>  student    cLaSs  gRaDe
0     Andy   school      Z
1   Bernie   school      Y
2    Cindy   school      Z
3      Deb   school      Y
4     Andy  english     10
5   Bernie  english    100
6    Cindy  english   1000
```

```
7      Deb  english  10000
8     Andy     math     20
9   Bernie     math    200
10   Cindy     math   2000
11     Deb     math  20000
12    Andy  physics     30
13  Bernie  physics    300
14   Cindy  physics   3000
15     Deb  physics  30000
```

Wide to long: school column isn't an identifier column

. . .

Final remarks

I hope now you have a better understanding of how `pd.melt` reshapes dataframes. I look forward to your thoughts and comments.

If you find this post useful, follow me and visit my site for more data science tutorials and also my other articles:

Code and Develop More Productively With Terminal Multiplexer tmux

Simple tmux commands to improve your productivity

medium.com

Data Science Python Pandas Programming Towards Data Science