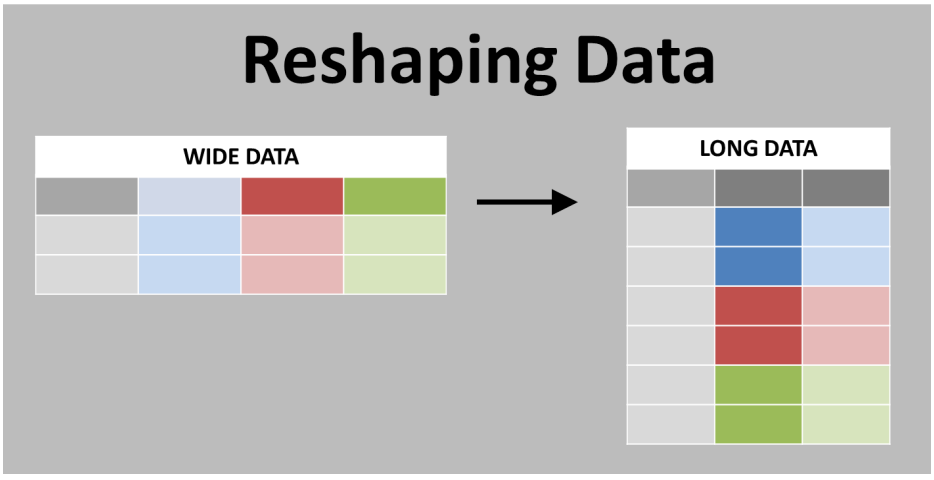
# Seven Clean Steps To Reshape Your Data With Pandas Or How I Use Python Where Excel Fails

Tich Mangono Follow
Nov 16, 2017 · 10 min read

## Concepts: multi-level indexing, pivoting, stacking, apply, lambda, and list-comprehension

## What reshaping data with Pandas LOOKS like...




## What reshaping data with Pandas FEELS like...


A few weeks ago, a colleague sent me a spreadsheet with data on a public health intervention, consisting of many tabs, one tab per organization. The task was to develop a flexible dashboard to explore this data. The problem was that the data was in wide format, but we needed a long format. Before, this would have been just another manual task of copy-and-paste and other dreadfully long and repetitive tasks, but I decided to use Python's Pandas library to automate this task so I set to work developing the script. Within 30 minutes, I had flexible, reusable code which later saved me hours of unnecessary manual labor!

I would like to share with you my process in case it comes up in your own work. I will also show some tricks and explain the approach in detail. Of course, I have sanitized the data and generated dummy numbers for privacy, but the format and concepts to be tackled remain the same. Here's a **road map** of what we will do with Pandas:

1. **Set up the environment and load the data**
2. **Investigate the data**
3. **Parse the different data tabs**
4. **Standardize existing columns and create new ones**
5. **Clean up the data using "apply" and "lambda" functions**

6. **Reshape the data from wide to long by pivoting on multi-level indices and stacking**
7. **Concatenate and save the final results back to Excel**

We will also wrap this up into a neat function that can be reused to automate this task and save time. I have posted the code and data on my github account. Also, check out my blog for more ideas on Machine Learning, Python and Public Health. Let's begin!

## 1. Set up the environment and load the data

As advertised, we only need one Python library to execute this task: Pandas! Our data is an Excel file with several tabs. I like using the ExcelFile object functionality of Pandas as opposed to the read command because it handles multi-tab spreadsheets very well.

```
# Import the relevant libraries...
# By the way, I am using Python 3
import pandas as pd
# Load the raw data using the ExcelFile object
data = pd.ExcelFile('reshaping_data.xlsx')
```

## 2. Investigate the data

We have four tabs in the file, each representing data from a single organization.

```
# First, see the sheetnames available
data.sheet_names


Output: ['ABC_inc', 'HIJ_inc', 'OPQ_inc', 'XYZ_inc']
```

Parsing the first tab for ABC_inc organization, we can see that the format needs a little work before we can use it as a standard data frame. The data contains the targets of a public health intervention. We can see that our column header names start in row 6, and we have information about the location (district, province); entities involved (partner, funding source); and target year (2017 to 2020). Notice also how row 7 has additional information on the target age group for this intervention for each year in the data. The main body of data starts from row 8 down.

```
# Take a peek at the first 10 rows of the first tab
data.parse(sheetname='ABC_inc', skiprows=0).head(10)
```

| | Unnamed: 0 | Unnamed: 1 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 | Unnamed: 5 | Unnamed: 6 | Unnamed: 7 | Unnamed: 8 | Unnamed: 9 | ... | Unnamed: 21 | Unnamed: 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN |
| 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN |
| 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN |
| 3 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN |
| 4 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | Year1 | ... | NaN | NaN |
| 5 | NaN | district | province | partner | funding_source | NaN | NaN | 2017 | 2017 | 2017 | ... | NaN | 2020 |
| 6 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 10-14yrs | 15-29yrs | 30+yrs | ... | NaN | 10-14yrs |
| 7 | NaN | District 1 | Region 1 | partner 1 | Souce 2 | NaN | NaN | 1296 | 383 | 1571 | ... | NaN | 1906 |
| 8 | NaN | District 2 | Region 3 | partner 6 | Souce 5 | NaN | NaN | 722 | 232 | 1848 | ... | NaN | 810 |
| 9 | NaN | District 3 | Region 1 | partner 1 | Souce 2 | NaN | NaN | 545 | 585 | 1736 | ... | NaN | 1890 |

10 rows × 31 columns

# 3. Parse the different data tabs

Make a list of your target tab names. In our case we want all of them. However, if you only wanted say 2 of these for analyses, then you could easily specify a different list.

```
tabnames = data.sheet_names
```

Since the tabs have a similar format, we will use only one them for this demonstration. At the end we will combine all the steps into a single, re-usable function and use iteration to apply the function to all the target tabs. We will then concatenate and save the results. So, parse the tab into a data frame, df, skipping the useless empty rows at the top. I always use “data.head()” to check my result and make sure my code did what I expected.

```
i = 0
df = data.parse(sheetname=tabnames[i], skiprows=7)
df.head(2)
```

| | Unnamed: 0 | district | province | partner | funding_source | Unnamed: 5 | Unnamed: 6 | 2017 | 2017.1 | 2017.2 | ... | Unnamed: 21 | 2020 | 2020.1 | 2020.2 | 2020.3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 10-14yrs | 15-29yrs | 30+yrs | ... | NaN | 10-14yrs | 15-29yrs | 30+yrs | Total |
| 1 | NaN | District 1 | Region 1 | partner 1 | Souce 2 | NaN | NaN | 1296 | 383 | 1571 | ... | NaN | 1906 | 1925 | 931 | 5465 |

## 4. Standardize existing columns and create new ones

Make a list of the default columns. We will throw away some but also strip the rest of them for information to be used for new column names. We will need to retain our information on the particular year, age-group and organization.

```
# make a list of the header row and strip up to the 4th
letter. This is the location and year information
cols1 = list(df.columns)
cols1 = [str(x)[:4] for x in cols1]
```

```
# make another list of the first row,this is the age group
information
# we need to preserve this information in the column name
when we reshape the data
cols2 = list(df.iloc[0,:])
cols2 = [str(x) for x in cols2]
```

```
# now join the two lists to make a combined column name
which preserves our location, year and age-group
information
cols = [x+"_"+y for x,y in zip(cols1,cols2)]
# Assign new column names to the dataframe
df.columns = cols
df.head(1)
```

| | Unna_nan | dist_nan | prov_nan | part_nan | fund_nan | Unna_nan | Unna_nan | 2017_10-14yrs | 2017_15-29yrs | 2017_30+yrs | ... | Unna_nan | 2020_10-14yrs | 2020_15-29yrs |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 10-14yrs | 15-29yrs | 30+yrs | ... | NaN | 10-14yrs | 15-29yrs |

```
# Drop empty columns, Rename the useful columns
# Note when you drop, you should specify axis=1 for columns
and axis=0 for rows
df = df.drop(["Unna_nan"],
axis=1).iloc[1:,:].rename(columns=
{'dist_nan':'district',
'prov_nan': 'province',
'part_nan':'partner',
'fund_nan':'financing_source'})
```

```
df.head(2)
```

| | district | province | partner | financing_source | 2017_10-14yrs | 2017_15-29yrs | 2017_30+yrs | 2017_Total | 2018_10-14yrs | 2018_15-29yrs | ... | 2019_30+yrs | 2019_Total | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | District 1 | Region 1 | partner 1 | Souce 2 | 1296 | 383 | 1571 | 3250 | 189 | 854 | ... | 491 | 2256 | 19 |
| 2 | District 2 | Region 3 | partner 6 | Souce 5 | 722 | 232 | 1848 | 2802 | 972 | 69 | ... | 245 | 2957 | 81 |

```
# Engineer a new column for the organization, grab this
name from the excel tab name
# This should read 'ABC inc' if executed correctly
df['main_organization'] = tabnames[i].split("_")[0] + " "+
tabnames[i].split("_")[1]
df.main_organization.head(2)
```

**Output:**

```
1    ABC inc
2    ABC inc
Name: main_organization, dtype: object
```

Let's pause and look at the structure of our dataframe so far.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 1 to 10
Data columns (total 25 columns):
district             10 non-null object
province             10 non-null object
partner              10 non-null object
financing_source     10 non-null object
2017_10-14yrs        10 non-null object
2017_15-29yrs        10 non-null object
2017_30+yrs          10 non-null object
2017_Total           10 non-null object
2018_10-14yrs        10 non-null object
2018_15-29yrs        10 non-null object
2018_30+yrs          10 non-null object
2018_Total           10 non-null object
2019_10-14yrs        10 non-null object
2019_15-29yrs        10 non-null object
2019_30+yrs          10 non-null object
2019_Total           10 non-null object
2020_10-14yrs        10 non-null object
2020_15-29yrs        10 non-null object
2020_30+yrs          10 non-null object
2020_Total           10 non-null object
2021_10-14yrs        10 non-null object
2021_15-29yrs        10 non-null object
2021_30+yrs          10 non-null object
2021_Total           10 non-null object
main_organization    10 non-null object
dtypes: object(25)
memory usage: 2.0+ KB
```

We see that we have 29 columns in all. However, currently, all of them have the "object" data type and we know some of them should have a numeric data type. We also have redundant "Total" columns. Notice that

the column names still retain multiple levels of information, i.e. the year and the age-group to which the data in that particular column belongs! This is one of the key aspects of this exercise as we will see in the next step. Before that, let's clean up a little bit…

## 5. Clean up the data using "apply" and "lambda" functions

Let's remove more redundant columns and change the data types.

```
# Make lists of the columns which need attention and use
this as reference to execute
# You will notice that I use list comprehension every time
I generate an iterable like a list or dictionary
# This is really amazing python functionality and I never
want to go back to the old looping way of doing this!
to_remove = [c for c in df.columns if "Total" in c] #
redundant
to_change = [c for c in df.columns if "yrs" in c] # numeric
```

```
# drop unwanted columns
# Notice that you need to specify inplace, otherwise pandas
will return the data frame instead of changing it in place
df.drop(to_remove, axis=1, inplace= True)
```

```
# Change the target column data types
for c in to_change:
    df[c] = df[c].apply(lambda x: pd.to_numeric(x))
```

See the changes:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 1 to 10
Data columns (total 20 columns):
district             10 non-null object
province             10 non-null object
partner              10 non-null object
financing_source     10 non-null object
2017_10-14yrs        10 non-null int64
2017_15-29yrs        10 non-null int64
2017_30+yrs          10 non-null int64
2018_10-14yrs        10 non-null int64
2018_15-29yrs        10 non-null int64
2018_30+yrs          10 non-null int64
2019_10-14yrs        10 non-null int64
2019_15-29yrs        10 non-null int64
2019_30+yrs          10 non-null int64
2020_10-14yrs        10 non-null int64
2020_15-29yrs        10 non-null int64
2020_30+yrs          10 non-null int64
2021_10-14yrs        10 non-null int64
2021_15-29yrs        10 non-null int64
2021_30+yrs          10 non-null int64
main_organization    10 non-null object
dtypes: int64(15), object(5)
memory usage: 1.6+ KB
```

Looks like it worked!

## 6. Reshape the data from wide to long by pivoting on multi-level indices and stacking

Now, on to my favorite part—reshaping the data! This functionality gives you so much power when it comes to data.It's one that I use so often for data cleaning and manipulation given the popularity of spreadsheets and survey-type data in my work. To me reshaping data literally feels like the data is clay and I am molding it with my very hands :). So, we will use pivoting on strategic indices and then use stacking to achieve the shape we want. Currently, the data is in a wide format, but we need to change it to long format so that we can easily transfer it to Excel where long formats lend themselves to quick pivot tables and dashboard creation very easily.

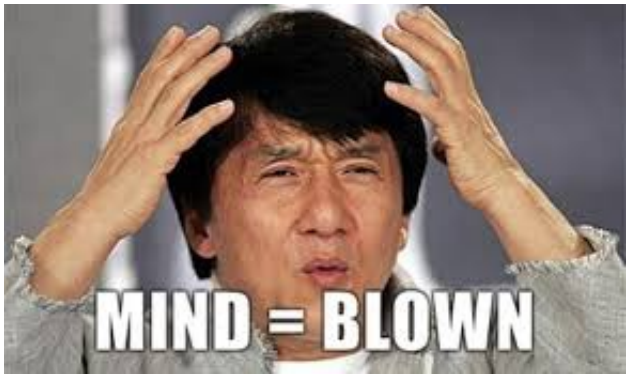
```
# First, select the columns to use for a multi-level index.
This depends on your data
# Generally, you want all the identifier columns to be
included in the multi-index
# For this dataset, this is every non-numeric column
idx =['district','province','partner','financing_source'
,'main_organization']


# Then pivot the dataset based on this multi-level index
multi_indexed_df = df.set_index(idx)
multi_indexed_df.head(2)
```

| | | | | | 2017_10-14yrs | 2017_15-29yrs | 2017_30+yrs | 2018_10-14yrs | 2018_15-29yrs | 2018_30+yrs | 2019_10-14yrs | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| district | province | partner | financing_source | main_organization | | | | | | | | |
| District 1 | Region 1 | partner 1 | Souce 2 | ABC inc | 1296 | 383 | 1571 | 189 | 854 | 339 | 840 | 77 |
| District 2 | Region 3 | partner 6 | Souce 5 | ABC inc | 722 | 232 | 1848 | 972 | 69 | 1205 | 422 | 67 |

After pivoting the data frame on our strategically engineered multi-level index, we will now stack all the numerical columns. This will give us the flexibility to reshape the data back to whatever level we want afterwards, just like an Excel pivot table.

```
# Stack the columns to achieve the baseline long format for
the data
stacked_df = multi_indexed_df.stack(dropna=False)
stacked_df.head(25) # check out the results!
```

```
district    province  partner    financing_source  main_organization
District 1  Region 1  partner 1  Souce 2           ABC inc           2017_10-14yrs    1296
                                                                    2017_15-29yrs     383
                                                                    2017_30+yrs      1571
                                                                    2018_10-14yrs     189
                                                                    2018_15-29yrs     854
                                                                    2018_30+yrs       339
                                                                    2019_10-14yrs     840
                                                                    2019_15-29yrs     773
                                                                    2019_30+yrs       491
                                                                    2020_10-14yrs    1906
                                                                    2020_15-29yrs    1925
                                                                    2020_30+yrs       931
                                                                    2021_10-14yrs      61
                                                                    2021_15-29yrs     353
                                                                    2021_30+yrs      1091
District 2  Region 3  partner 6  Souce 5           ABC inc           2017_10-14yrs     722
                                                                    2017_15-29yrs     232
                                                                    2017_30+yrs      1848
                                                                    2018_10-14yrs     972
                                                                    2018_15-29yrs      69
                                                                    2018_30+yrs      1205
                                                                    2019_10-14yrs     422
                                                                    2019_15-29yrs     676
                                                                    2019_30+yrs       245
                                                                    2020_10-14yrs     810
dtype: int64
```




Mind. Blown! Exactly how I felt when I saw this for the first time too.

```
# Now do a reset to disband the multi-level index, we only
needed it to pivot our data during the reshape
long_df = stacked_df.reset_index()
long_df.head(3)
```

| | district | province | partner | financing_source | main_organization | level_5 | 0 |
|---|---|---|---|---|---|---|---|
| **0** | District 1 | Region 1 | partner 1 | Souce 2 | ABC inc | 2017_10-14yrs | 1296 |
| **1** | District 1 | Region 1 | partner 1 | Souce 2 | ABC inc | 2017_15-29yrs | 383 |
| **2** | District 1 | Region 1 | partner 1 | Souce 2 | ABC inc | 2017_30+yrs | 1571 |

Notice that the “level_5” column contains two pieces of information. This was deliberately done from the beginning so as not to lose any information along the way as we drop some columns and rows. Now, lets use string manipulation to separate these and delete any redundancies after that.

```
# Make series of lists which split year from target age-group
# the .str attribute is how you manipulate the data frame objects and columns with strings in them
col_str = long_df.level_5.str.split("_")
col_str.head(3)
```

```
Output:
0    [2017, 10-14yrs]
1    [2017, 15-29yrs]
2    [2017, 30+yrs]
Name: level_5, dtype: object
```

```
# engineer the columns we want, one columns takes the first item in col_str and another columns takes the second
long_df['target_year'] = [x[0] for x in col_str]
long_df['target_age'] = [x[1] for x in col_str]
long_df['target_quantity'] = long_df[0] # rename this column
long_df.head(2)
```

| | district | province | partner | financing_source | main_organization | level_5 | 0 | target_year | target_age | target_quantity |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | District 1 | Region 1 | partner 1 | Souce 2 | ABC inc | 2017_10-14yrs | 1296 | 2017 | 10-14yrs | 1296 |
| 1 | District 1 | Region 1 | partner 1 | Souce 2 | ABC inc | 2017_15-29yrs | 383 | 2017 | 15-29yrs | 383 |

```
# drop the now redundant columns
df_final = long_df.drop(['level_5', 0], axis=1)
df_final.head(2)
```

| | district | province | partner | financing_source | main_organization | target_year | target_age | target_quantity |
|---|---|---|---|---|---|---|---|---|
| 0 | District 1 | Region 1 | partner 1 | Souce 2 | ABC inc | 2017 | 10-14yrs | 1296 |
| 1 | District 1 | Region 1 | partner 1 | Souce 2 | ABC inc | 2017 | 15-29yrs | 383 |

# 7. Concatenate and save the final results back to Excel

Now we have all the ingredients we need, we can define a function to automate the reshape, use iteration to apply this function to any number of tabs we want and then finally save this to our spreadsheet of choice! Notice that in the function I combine the shapes and add an assertion to check that the inputs are correct, but it mostly the same code.

```
# Now define a function for doing the reshape
def ReshapeFunc(excel_obj, i):
    """ Takes in an excel file object with multiple tabs in
a wide format, and a specified index of the tab to be
parsed and reshaped. Returns a data frame of the specified
tab reshaped to long format"""

    tabnames = data.sheet_names
    assert i < len(tabnames), "Your tab index exceeds the
number of available tabs, try a lower number"

    # parse and clean columns
    df = excel_obj.parse(sheetname=tabnames[i], skiprows=7)
    cols1 = [str(x)[:4] for x in list(df.columns)]
    cols2 = [str(x) for x in list(df.iloc[0,:])]
    cols = [x+"_"+y for x,y in zip(cols1,cols2)]
    df.columns = cols
    df = df.drop(["Unna_nan"],
axis=1).iloc[1:,:].rename(columns={'dist_nan':'district',
'prov_nan': 'province',
'part_nan':'partner',
'fund_nan':'financing_source'})
```

```
    # new columns, drop some and change data type
    df['main_organization'] = tabnames[i].split("_")[0] + "
"+ tabnames[i].split("_")[1]
    df.drop([c for c in df.columns if "Total" in c],
axis=1, inplace= True)

    for c in [c for c in df.columns if "yrs" in c]:
        df[c] = df[c].apply(lambda x: pd.to_numeric(x))
```

```
    # reshape - indexing, pivoting and stacking
    idx = ['district','province',
'partner','financing_source'
,'main_organization']


    multi_indexed_df = df.set_index(idx)
    stacked_df = multi_indexed_df.stack(dropna=False)
```

```
    long_df = stacked_df.reset_index()

    # clean up and finalize
    col_str = long_df.level_5.str.split("_")
    long_df['target_year'] = [x[0] for x in col_str]
    long_df['target_age'] = [x[1] for x in col_str]
    long_df['target_quantity'] = long_df[0] # rename this
column
    df_final = long_df.drop(['level_5', 0], axis=1)


    return df_final
```

Run the function

```
# Check that our function works:
check_df = ReshapeFunc(data, 2)
check_df.head(2)
```

| | district | province | partner | financing_source | main_organization | target_year | target_age | target_quantity |
|---|---|---|---|---|---|---|---|---|
| 0 | District 19 | Region 4 | partner 3 | Souce 4 | OPQ inc | 2017 | 10-14yrs | 1 |
| 1 | District 19 | Region 4 | partner 3 | Souce 4 | OPQ inc | 2017 | 15-29yrs | 974 |

Cool! Now use the function to make a data frame for each tab, put them all in a list, concatenate them into one single data frame and save it back to Excel.

```
dfs_list = [ReshapeFunc(data, i) for i in range(4)]
concat_dfs  = pd.concat(dfs_list)
concat_dfs.to_excel("reshaping_result_long_format.xlsx")
```

This long format can now be easily used for pivot tables and making dashboard charts and analysis by pivoting on any combination of the descriptive, non-numeric columns and aggregating the numeric columns. Mission complete. Happy Coding!