4. Other data reshaping strategies

February 6, 2021

1 4. Other data reshaping strategies

In this notebook, we'll cover: - The `pivot_table()` method - Filling null values - Melting wide data to long data - Building one dataframe out of a directory of files formatted the same way

First, let's import pandas.

[96]:
```
import pandas as pd
```

1.0.1 `pivot_table()`

Sometimes, when you group and aggregate, the `groupby()` method doesn't *quite* give you the view you need of your data, and the `pivot_table()` method can be a more intuitive choice.

For this section, we'll be examining a dataset of foreign eel product imported to the U.S. (source), which you will find in `../data/eels.csv`. Every row is one month's worth of eel product imports to the U.S. from one country.

[97]:
```
df_eels = pd.read_csv('../data/eels.csv')
```

[98]:
```
df_eels.head()
```

[98]:

	year	month	country	product	kilos	dollars
0	2010	1	CHINA	EELS FROZEN	49087	393583
1	2010	1	JAPAN	EELS FRESH	263	7651
2	2010	1	TAIWAN	EELS FROZEN	9979	116359
3	2010	1	VIETNAM	EELS FRESH	1938	10851
4	2010	1	VIETNAM	EELS FROZEN	21851	69955

Let's take a look at import volume by country by year. If we were making this pivot table in Excel, we would drag `country` to Rows, `kilos` to Values and `year` to Columns. But we're gonna do it in pandas! We need to hand the `pivot_table()` method four things: - The data frame you're pivoting (`df_eels`) - The `index` column – what to group your data by (`index='country'`) - The `columns` column – the second grouping factor (`columns='year'`) - The `values` column – what column are we doing math on? (`values='kilos'`) - The `aggfunc` – what function to use to aggregate the data; the default is to use an average, but we'll use Python's built-in `sum` function

[99]:
```
eels_pivoted = pd.pivot_table(df_eels,
                              index='country',
                              columns='year',
                              values='kilos',
                              aggfunc=sum)
```

[100]:
```
eels_pivoted
```

[100]:
```
year                  2010      2011       2012       2013       2014  \
country
BANGLADESH             NaN       NaN       13.0        NaN        NaN
BURMA                  NaN       NaN        NaN        NaN        NaN
CANADA             13552.0   24968.0   110796.0    44455.0    31546.0
CHILE                  NaN       NaN        NaN        NaN     6185.0
CHINA               372397.0  249232.0  1437392.0  1090135.0  1753140.0
CHINA - HONG KONG      NaN       NaN        NaN        NaN        NaN
COSTA RICA             NaN       NaN        NaN        NaN        NaN
INDIA                  NaN       NaN        NaN        NaN        NaN
JAPAN                1326.0    2509.0    32255.0   105758.0    40177.0
MEXICO                 NaN       NaN        NaN     4000.0        NaN
NEW ZEALAND            NaN    2652.0      900.0      270.0        NaN
NORWAY                 NaN       NaN        NaN    17391.0        NaN
PAKISTAN               NaN       NaN        NaN    22453.0        NaN
PANAMA                 NaN       NaN        NaN    11849.0        NaN
PHILIPPINES            NaN       NaN        NaN      610.0        NaN
POLAND                 NaN       NaN     1296.0        NaN      864.0
PORTUGAL            2081.0    3672.0     2579.0     2041.0     7215.0
SENEGAL                NaN    1350.0        NaN        NaN        NaN
SOUTH KOREA         42929.0   41385.0    28146.0    27353.0    37708.0
SPAIN                  NaN       NaN      977.0      275.0     1019.0
TAIWAN              73842.0       NaN    53774.0    39752.0    83478.0
THAILAND             2866.0    5018.0     9488.0     4488.0    15110.0
UKRAINE                NaN       NaN        NaN        NaN        NaN
VIETNAM             63718.0  155488.0   118063.0   100828.0    38112.0

year                  2015       2016       2017
country
BANGLADESH           600.0        NaN        NaN
BURMA                  NaN      699.0        NaN
CANADA             28619.0    68568.0    23571.0
CHILE                  NaN        NaN        NaN
CHINA              4713882.0  4578546.0  1771272.0
CHINA - HONG KONG      NaN      735.0        NaN
COSTA RICA             NaN      563.0        NaN
INDIA                  NaN     2200.0        NaN
JAPAN                69699.0    71748.0    37892.0
MEXICO               16860.0        NaN        NaN
```

```
NEW ZEALAND              NaN        NaN        NaN
NORWAY                   NaN        NaN        NaN
PAKISTAN                 NaN        NaN        NaN
PANAMA                   NaN        NaN      974.0
PHILIPPINES              NaN        NaN        NaN
POLAND                   NaN        NaN        NaN
PORTUGAL              8013.0     9105.0     6747.0
SENEGAL                  NaN        NaN        NaN
SOUTH KOREA           8386.0    14729.0    42904.0
SPAIN                  719.0     1008.0        NaN
TAIWAN               48272.0    99535.0    44087.0
THAILAND             41771.0    26931.0    31884.0
UKRAINE                  NaN    11414.0        NaN
VIETNAM              36859.0    96179.0    28490.0
```

1.0.2 Filling null values

Right now, if a country didn't have any shipments of these eel products in a given year, the values are `NaN`. To fill those values with zeroes instead, which might make more sense in context, you can use the `fillna()` method:

[101]:
```
eels_pivoted_clean = eels_pivoted.fillna(0)
```

[102]:
```
eels_pivoted_clean.head()
```

[102]:
```
year            2010      2011       2012       2013       2014       2015  \
country
BANGLADESH       0.0       0.0       13.0        0.0        0.0      600.0
BURMA            0.0       0.0        0.0        0.0        0.0        0.0
CANADA       13552.0   24968.0   110796.0    44455.0    31546.0    28619.0
CHILE            0.0       0.0        0.0        0.0     6185.0        0.0
CHINA       372397.0  249232.0  1437392.0  1090135.0  1753140.0  4713882.0

year            2016       2017
country
BANGLADESH       0.0        0.0
BURMA          699.0        0.0
CANADA       68568.0    23571.0
CHILE            0.0        0.0
CHINA       4578546.0  1771272.0
```

1.0.3 Your turn

In the cells below: - Sort the `eels_pivoted_clean` dataframe descending on the `2017` column - Show the sum of all eel imports from 2010

[ ]:

[ ]:

1.0.4 Melt wide data to long

Sometimes, you have data that's hard to work with because it's wide instead of long. You can use the pandas `melt()` method to "un-pivot" wide data into tidy data.

For this example, we're going to load in a CSV of African doctor emigration:

[103]:
```
df_doctors = pd.read_csv('../data/africa-physician-emigration.csv')
```

[104]:
```
df_doctors.head()
```

[104]:

	Sending country	UK	USA	France	Canada	Australia	Portugal	Spain	Belgium	So. Africa
0	Algeria	45	50	10,594	10	0	2	60	99	0
1	Angola	16	0	5	25	0	2,006	14	5	31
2	Benin	0	4	206	0	0	0	1	13	0
3	Botswana	28	10	0	0	3	0	0	1	26
4	Burkina Faso	0	0	77	0	0	0	0	1	0

This data would be much easier to work with if every row were the sending country, the receiving country and the number of doctors. According to the documentation, we need to hand this method a couple of keyword arguments: - `id_vars`: The column, or column, that will be the "identifying variable" that won't change – in this case, `Sending country` - `value_vars`: A column name, or a list of column names, with the values we want to melt – in this case, everything but the identifying column (more on this in a minute) - `var_name`: What should we call the resulting column with the variable names in it? In this case, `Receiving country` - `value_name`: What should we call the resulting column with the values in it? In this case, `Number of doctors`

[105]:
```
df_doctors.melt(id_vars='Sending country',
                value_vars=['UK', 'USA', 'France', 'Canada', 'Australia',␣
↪'Portugal', 'Spain', 'Belgium', 'So. Africa'],
                var_name='Receiving country',
                value_name='Number of doctors')
```

[105]:

	Sending country	Receiving country	Number of doctors
0	Algeria	UK	45
1	Angola	UK	16

```
2              Benin                UK                 0
3           Botswana                UK                28
4       Burkina Faso                UK                 0
..               …                  …                 …
472             Togo        So. Africa                 0
473          Tunisia        So. Africa                 0
474           Uganda        So. Africa               179
475           Zambia        So. Africa               203
476         Zimbabwe        So. Africa               643

[477 rows x 3 columns]
```

One thing – having to list out the names of all the columns for the `value_vars` keyword argument was kind of tedious, so here's a more efficient way to do that: by accessing the `columns` attribute of the dataframe and using list indexing to grab just the ones we want:

[106]:
```
df_doctors.columns
```

[106]:
```
Index(['Sending country', 'UK', 'USA', 'France', 'Canada', 'Australia',
       'Portugal', 'Spain', 'Belgium', 'So. Africa'],
      dtype='object')
```

[107]:
```
# what do we get if we leave off the first one?
df_doctors.columns[1:]
```

[107]:
```
Index(['UK', 'USA', 'France', 'Canada', 'Australia', 'Portugal', 'Spain',
       'Belgium', 'So. Africa'],
      dtype='object')
```

Bingo! Let's try that again:

[108]:
```
df_doctors.melt(id_vars='Sending country',
                value_vars=df_doctors.columns[1:],
                var_name='Receiving country',
                value_name='Number of doctors')
```

[108]:
```
    Sending country Receiving country Number of doctors
0           Algeria                UK                45
1            Angola                UK                16
2             Benin                UK                 0
3          Botswana                UK                28
4      Burkina Faso                UK                 0
..               …                  …                 …
472             Togo        So. Africa                 0
473          Tunisia        So. Africa                 0
474           Uganda        So. Africa               179
475           Zambia        So. Africa               203
476         Zimbabwe        So. Africa               643
```

```
[477 rows x 3 columns]
```

1.0.5 Your turn

In the cells below: - Re-run the code from the cell above but assign the new dataframe to a variable - Filter that dataframe to show only records where the receiving country is the USA - Filter that dataframe to show only records where the number of physicians is greater than 100 - Sort the data descending by the number of doctors emigrating – what's the largest outflow?

[ ]:

[ ]:

[ ]:

[ ]:

1.0.6 Building a dataframe from a directory of files

Sometimes, rather than one file you need to load, you have a directory of files with the same format but different data. Let's talk about a strategy for reading them all into a single dataframe – the data for this exercise comes from this wonderful data-driven story from 2019 by C.K. Hickey in *Foreign Policy* on state dinner menus for U.S. presidents (thank you, C.K.!) and can be found in the `../data/state-dinners/` directory.

Our strategy: - Get a list of these files using the `glob` module from the standard library - Use a fun Python data structure called a "list comprehension" in conjunction with the pandas methods `read_csv()` (which we've seen before) and `concat()` (which we have not)

First, we need to import `glob` before we can use it. (FWIW: The customary thing to do is drop all your imports at the top of your script.)

[109]:
```
import glob
```

Get a list of the files using wildcards:

[110]:
```
sd_files = glob.glob('../data/state-dinners/*.csv')
```

[111]:
```
sd_files
```

[111]:
```
['../data/state-dinners/roosevelt.csv',
 '../data/state-dinners/trump.csv',
 '../data/state-dinners/bush43.csv',
 '../data/state-dinners/bush41.csv',
 '../data/state-dinners/eisenhower.csv',
 '../data/state-dinners/obama.csv',
```

```
 '../data/state-dinners/truman.csv',
 '../data/state-dinners/nixon.csv',
 '../data/state-dinners/reagan.csv',
 '../data/state-dinners/clinton.csv',
 '../data/state-dinners/ford.csv',
 '../data/state-dinners/carter.csv',
 '../data/state-dinners/kennedy.csv',
 '../data/state-dinners/johnson.csv']
```

In human language: Go to the `glob` module we just imported and use its `glob` object to get a list of files based on the path and filename wildcards we hand it.

Now let's talk for a sec about list comprehensions. Let's say you had a list of items that you wanted to *do* something to – some math, some filtering, some reading into dataframes. (We're about to do this last one!) One of the main uses for list comprehensions is effeciently "saving" the results of this operation to a new variable.

Here's a simple example – let's say we had the following list of numbers:

[112]:
```
number_list = [1, 2, 3, 4, 5, 6]
```

… and we want to end up with a list of numbers that is each of these numbers multiplied by 10. We could do something like this:

[113]:
```
new_list = []
for x in number_list:
    new_list.append(x*10)
```

[114]:
```
new_list
```

[114]:
```
[10, 20, 30, 40, 50, 60]
```

You could achieve the same thing with a *list comprehension* much quicker and easier:

[115]:
```
new_list_lc = [x*10 for x in number_list]
```

[116]:
```
new_list_lc
```

[116]:
```
[10, 20, 30, 40, 50, 60]
```

Here, `x` is a placeholder for each item in the list, same as the variable defined in the `for` loop.

That's basically what we're going to do here – instead of creating an empty list, looping over each file in the `state_dinners` directory, creating a new dataframe, adding it to the list, then concatenating all those dataframes, we can do it all in one fell swoop:

[117]:
```
df_dinners = pd.concat([pd.read_csv(x) for x in sd_files])
```

Reading this from the inside out as a human sentence: Take each CSV file in the `state_dinners` directory, which we found earlier using the `glob` tool, and read it into a (more or less temporary)

dataframe – then take all of those dataframes and concatenate them together into one dataframe.

[118]:
```
df_dinners.head()
```

[118]:

	president	category	subcategory	item	count
0	Roosevelt	Meat	Seafood	Seafood Cutlets with Mushroom Sauce	1
1	Roosevelt	Meat	Seafood	Planked Shad and Roe - Swiss Dressing	1
2	Roosevelt	Meat	Seafood	Oyster on Half Shell	1
3	Roosevelt	Meat	Seafood	Oyster Cocktail	1
4	Roosevelt	Meat	Seafood	Blue Points	2