Pandas for Panel Data

Thomas J. Sargent and John Stachurski

October 7, 2019

1 Contents

• Overview 2
• Slicing and Reshaping Data 3
• Merging Dataframes and Filling NaNs 4
• Grouping and Summarizing Data 5
• Final Remarks 6
• Exercises 7
• Solutions 8

2 Overview

In an earlier lecture on pandas, we looked at working with simple data sets. Econometricians often need to work with more complex data sets, such as panels. Common tasks include

• Importing data, cleaning it and reshaping it across several axes.
• Selecting a time series or cross-section from a panel.
• Grouping and summarizing data.

pandas (derived from ‘panel’ and ‘data’) contains powerful and easy-to-use tools for solving exactly these kinds of problems.

In what follows, we will use a panel data set of real minimum wages from the OECD to create:

• summary statistics over multiple dimensions of our data
• a time series of the average minimum wage of countries in the dataset
kernel density estimates of wages by continent

We will begin by reading in our long format panel data from a CSV file and reshaping the resulting DataFrame with \texttt{pivot\_table} to build a \texttt{MultiIndex}.

Additional detail will be added to our DataFrame using pandas’ \texttt{merge} function, and data will be summarized with the \texttt{groupby} function.

Most of this lecture was created by Natasha Watkins.

3 Slicing and Reshaping Data

We will read in a dataset from the OECD of real minimum wages in 32 countries and assign it to \texttt{realwage}.

The dataset \texttt{pandas\_panel/realwage.csv} can be downloaded here.

Make sure the file is in your current working directory

\begin{minipage}{\textwidth}
\begin{verbatim}
import pandas as pd
# Display 6 columns for viewing purposes
pd.set_option('display.max_columns', 6)
# Reduce decimal points to 2
pd.options.display.float_format = '{:.2f}'.format
realwage = pd.read_csv('https://github.com/QuantEcon/QuantEcon.lectures.code/raw/master/pandas_panel/realwage.csv')

Let's have a look at what we've got to work with

realwage.head() # Show first 5 rows
\end{verbatim}
\end{minipage}

\begin{verbatim}
      Unnamed: 0   Time       Country Series           Pay period value
0     0       2006-01-01 Ireland In 2015 constant prices at 2015 USD PPPs 17,132.44
1     1       2007-01-01 Ireland In 2015 constant prices at 2015 USD PPPs 18,100.92
2     2       2008-01-01 Ireland In 2015 constant prices at 2015 USD PPPs 17,747.41
3     3       2009-01-01 Ireland In 2015 constant prices at 2015 USD PPPs 18,580.14
4     4       2010-01-01 Ireland In 2015 constant prices at 2015 USD PPPs 18,755.83

The data is currently in long format, which is difficult to analyze when there are several di-

\begin{verbatim}
We will use \texttt{pivot\_table} to create a wide format panel, with a \texttt{MultiIndex} to handle higher dimensional data.

\texttt{pivot\_table} arguments should specify the data (values), the index, and the columns we want in our resulting dataframe.

By passing a list in columns, we can create a \texttt{MultiIndex} in our column axis

realwage = realwage.pivot_table(values='value',
index='Time',
columns=['Country', 'Series', 'Pay period'])
realwage.head()
\end{verbatim}

\end{verbatim}
<table>
<thead>
<tr>
<th>Year</th>
<th>Real Wage (2015 USD PPPs)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006-01-01</td>
<td>20,410.65</td>
<td>10.33</td>
</tr>
<tr>
<td>2007-01-01</td>
<td>21,087.57</td>
<td>10.67</td>
</tr>
<tr>
<td>2008-01-01</td>
<td>20,718.24</td>
<td>10.48</td>
</tr>
<tr>
<td>2009-01-01</td>
<td>20,984.77</td>
<td>10.62</td>
</tr>
<tr>
<td>2010-01-01</td>
<td>20,879.33</td>
<td>10.57</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Real Wage (2015 USD exchange rates)</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006-01-01</td>
<td>23,826.64</td>
<td>12.59</td>
</tr>
<tr>
<td>2007-01-01</td>
<td>24,616.84</td>
<td>13.84</td>
</tr>
<tr>
<td>2008-01-01</td>
<td>24,185.70</td>
<td>13.78</td>
</tr>
<tr>
<td>2009-01-01</td>
<td>24,496.84</td>
<td>14.58</td>
</tr>
<tr>
<td>2010-01-01</td>
<td>24,373.76</td>
<td>14.38</td>
</tr>
</tbody>
</table>

To more easily filter our time series data, later on, we will convert the index into a `DateTimeIndex`.

```python
realwage.index = pd.to_datetime(realwage.index)
type(realwage.index)
```

The columns contain multiple levels of indexing, known as a `MultiIndex`, with levels being ordered hierarchically (Country > Series > Pay period).

A `MultiIndex` is the simplest and most flexible way to manage panel data in pandas.

```python
type(realwage.columns)
pandas.core.indexes.multi.MultiIndex
realwage.columns.names
FrozenList(['Country', 'Series', 'Pay period'])
```

Like before, we can select the country (the top level of our `MultiIndex`).

```python
realwage['United States'].head()
```
### Series In 2015 constant prices at 2015 USD PPPs

<table>
<thead>
<tr>
<th>Time</th>
<th>Pay period</th>
<th>Real wage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006-01-01</td>
<td>Annual</td>
<td>12,594.40</td>
</tr>
<tr>
<td>2007-01-01</td>
<td>Annual</td>
<td>12,974.40</td>
</tr>
<tr>
<td>2008-01-01</td>
<td>Annual</td>
<td>14,097.56</td>
</tr>
<tr>
<td>2009-01-01</td>
<td>Annual</td>
<td>15,756.42</td>
</tr>
<tr>
<td>2010-01-01</td>
<td>Annual</td>
<td>16,391.31</td>
</tr>
</tbody>
</table>

### Series In 2015 constant prices at 2015 USD exchange rates

<table>
<thead>
<tr>
<th>Time</th>
<th>Pay period</th>
<th>Real wage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006-01-01</td>
<td>Annual</td>
<td>12,594.40</td>
</tr>
<tr>
<td>2007-01-01</td>
<td>Annual</td>
<td>12,974.40</td>
</tr>
<tr>
<td>2008-01-01</td>
<td>Annual</td>
<td>14,097.56</td>
</tr>
<tr>
<td>2009-01-01</td>
<td>Annual</td>
<td>15,756.42</td>
</tr>
<tr>
<td>2010-01-01</td>
<td>Annual</td>
<td>16,391.31</td>
</tr>
</tbody>
</table>

Stacking and unstacking levels of the `MultiIndex` will be used throughout this lecture to reshape our dataframe into a format we need.

`.stack()` rotates the lowest level of the column `MultiIndex` to the row index (.unstack() works in the opposite direction - try it out)
<table>
<thead>
<tr>
<th>Country</th>
<th>Series</th>
<th>Time</th>
<th>Pay period</th>
<th>Hourly</th>
<th>Annual</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2006-01-01</td>
<td>Annual</td>
<td>12,594.40</td>
<td>6.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2007-01-01</td>
<td>Annual</td>
<td>12,828.00</td>
<td>6.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2008-01-01</td>
<td>Annual</td>
<td>14,097.56</td>
<td>6.25</td>
</tr>
<tr>
<td>Australia</td>
<td></td>
<td>2006-01-01</td>
<td>Annual</td>
<td>20,410.65</td>
<td>10.33</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Belgium</td>
<td>Annual</td>
<td>21,042.28</td>
<td>10.09</td>
</tr>
<tr>
<td>Brazil</td>
<td></td>
<td>2006-01-01</td>
<td>Annual</td>
<td>3,310.51</td>
<td>1.41</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Canada</td>
<td>Annual</td>
<td>13,649.69</td>
<td>6.56</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Chile</td>
<td>Annual</td>
<td>5,281.65</td>
<td>2.22</td>
</tr>
</tbody>
</table>

We can also pass in an argument to select the level we would like to stack.

```python
realwage.stack(level='Country').head()
```

Using a `DatetimeIndex` makes it easy to select a particular time period.

Selecting one year and stacking the two lower levels of the MultiIndex creates a cross-section of our panel data.

```python
realwage['2015'].stack(level=(1, 2)).transpose().head()
```

For the rest of lecture, we will work with a dataframe of the hourly real minimum wages across countries and time, measured in 2015 US dollars.

To create our filtered dataframe (realwage_f), we can use the `xs` method to select values at lower levels in the multiindex, while keeping the higher levels (countries in this case).

```python
realwage_f = realwage.xs({'Hourly', 'In 2015 constant prices at 2015 USD exchange rates'}, level=('Pay period', 'Series'), axis=1)
realwage_f.head()
```
4 Merging Dataframes and Filling NaNs

Similar to relational databases like SQL, pandas has built in methods to merge datasets together.

Using country information from WorldData.info, we’ll add the continent of each country to realwage_f with the merge function.

The CSV file can be found in pandas_panel/countries.csv and can be downloaded here.

```python
worlddata = pd.read_csv('https://github.com/QuantEcon/QuantEcon.lectures.code/raw/master/pandas_panel/countries.csv', sep=';')
worlddata.head()
```

First, we’ll select just the country and continent variables from worlddata and rename the column to ‘Country’

```python
worlddata = worlddata[['Country (en)', 'Continent']]
worlddata = worlddata.rename(columns={'Country (en)': 'Country'})
worlddata.head()
```

We want to merge our new dataframe, worlddata, with realwage_f.
The pandas `merge` function allows dataframes to be joined together by rows.

Our dataframes will be merged using country names, requiring us to use the transpose of `realwage_f` so that rows correspond to country names in both dataframes.

```
realwage_f.transpose().head()
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Country</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Australia</td>
<td>12.06</td>
<td>12.46</td>
<td>12.24</td>
<td></td>
<td>12.67</td>
<td>12.83</td>
</tr>
<tr>
<td>Belgium</td>
<td>9.70</td>
<td>9.82</td>
<td>9.87</td>
<td></td>
<td>10.01</td>
<td>9.95</td>
</tr>
<tr>
<td>Brazil</td>
<td>0.87</td>
<td>0.92</td>
<td>0.96</td>
<td></td>
<td>1.21</td>
<td>1.21</td>
</tr>
<tr>
<td>Canada</td>
<td>6.89</td>
<td>6.96</td>
<td>7.24</td>
<td></td>
<td>8.22</td>
<td>8.35</td>
</tr>
<tr>
<td>Chile</td>
<td>1.42</td>
<td>1.45</td>
<td>1.44</td>
<td></td>
<td>1.76</td>
<td>1.81</td>
</tr>
</tbody>
</table>
```

We can use either left, right, inner, or outer join to merge our datasets:

- left join includes only countries from the left dataset
- right join includes only countries from the right dataset
- outer join includes countries that are in either the left and right datasets
- inner join includes only countries common to both the left and right datasets

By default, `merge` will use an inner join.

Here we will pass `how='left'` to keep all countries in `realwage_f`, but discard countries in `worlddata` that do not have a corresponding data entry `realwage_f`.

This is illustrated by the red shading in the following diagram:

```
```

We will also need to specify where the country name is located in each dataframe, which will be the **key** that is used to merge the dataframes ‘on’.
Our ‘left’ dataframe (`realwage_f.transpose()`) contains countries in the index, so we set `left_index=True`.

Our ‘right’ dataframe (`worlddata`) contains countries in the ‘Country’ column, so we set `right_on='Country'

```python
merged = pd.merge(realwage_f.transpose(), worlddata, how='left', left_index=True, right_on='Country')
merged.head()
```

```
2006-01-01 00:00:00 2007-01-01 00:00:00 2008-01-01 00:00:00 ... 
17 12.06 12.46 12.24 ... 
23 9.70 9.82 9.87 ... 
32 6.89 6.96 7.24 ... 
38 1.42 1.45 1.44 ... 

2016-01-01 00:00:00 Country Continent
17 12.08 Australia Australia
23 8.76 Belgium Europe
32 1.24 Brazil South America
100 8.48 Canada North America
38 1.91 Chile South America
```

[5 rows x 13 columns]

Countries that appeared in `realwage_f` but not in `worlddata` will have `NaN` in the Continent column.

To check whether this has occurred, we can use `.isnull()` on the continent column and filter the merged dataframe

```python
merged[merged['Continent'].isnull()]
```

```
2006-01-01 00:00:00 2007-01-01 00:00:00 2008-01-01 00:00:00 ... 
247 3.42 3.74 3.87 ... 
247 0.23 0.45 0.39 ... 
247 1.50 1.64 1.71 ... 

2016-01-01 00:00:00 Country Continent
247 5.28 Korea NaN
247 0.55 Russian Federation NaN
247 2.08 Slovak Republic NaN
```

[3 rows x 13 columns]

We have three missing values!

One option to deal with `NaN` values is to create a dictionary containing these countries and their respective continents.

`.map()` will match countries in `merged['Country']` with their continent from the dictionary.

Notice how countries not in our dictionary are mapped with `NaN`

```python
missing_continents = {'Korea': 'Asia',
                      'Russian Federation': 'Europe',
                      'Slovak Republic': 'Europe'}
merged['Country'].map(missing_continents)
```

```
17  NaN
23  NaN
32  NaN
100 NaN
38  NaN
```
We don’t want to overwrite the entire series with this mapping. `.fillna()` only fills in `NaN` values in `merged['Continent']` with the mapping, while leaving other values in the column unchanged.

```
merged['Continent'] = merged['Continent'].fillna(merged['Country'].map(missing_continents))
```

# Check for whether continents were correctly mapped
```python
merged[merged['Country'] == 'Korea']
```

We will also combine the Americas into a single continent - this will make our visualization nicer later on.

To do this, we will use `.replace()` and loop through a list of the continent values we want to replace.

```
replace = ['Central America', 'North America', 'South America']
for country in replace:
    merged['Continent'].replace(to_replace=country, value='America', inplace=True)
```

Now that we have all the data we want in a single `DataFrame`, we will reshape it back into panel form with a `MultiIndex`.

We should also ensure to sort the index using `.sort_index()` so that we can efficiently filter our dataframe later on.

By default, levels will be sorted top-down
While merging, we lost our `DatetimeIndex`, as we merged columns that were not in datetime format

```python
merged.columns
```

```
Index([‘2006-01-01 00:00:00’, ‘2007-01-01 00:00:00’, ‘2008-01-01 00:00:00’, ‘2009-01-01 00:00:00’, ‘2010-01-01 00:00:00’, ‘2011-01-01 00:00:00’, ‘2012-01-01 00:00:00’, ‘2013-01-01 00:00:00’, ‘2014-01-01 00:00:00’, ‘2015-01-01 00:00:00’, ‘2016-01-01 00:00:00’], dtype=’datetime64[ns]’, name=’Time’, freq=None)
```

Now that we have set the merged columns as the index, we can recreate a `DatetimeIndex` using `.to_datetime()`

```python
merged.columns = pd.to_datetime(merged.columns)
merged.columns = merged.columns.rename(‘Time’)
```

The `DatetimeIndex` tends to work more smoothly in the row axis, so we will go ahead and transpose `merged`

```python
merged = merged.transpose()
merged.head()
```

```
Continent America Brazil Canada Chile Colombia Costa Rica
2006-01-01 0.87 6.89 1.42 1.01 nan nan
2007-01-01 0.92 6.96 1.45 1.02 nan nan
2008-01-01 0.96 7.24 1.44 1.01 nan nan
2009-01-01 1.03 7.67 1.52 1.03 nan nan
2010-01-01 1.08 7.94 1.56 1.08 nan nan
```

[5 rows x 32 columns]

5 Grouping and Summarizing Data

Grouping and summarizing data can be particularly useful for understanding large panel datasets.
A simple way to summarize data is to call an aggregation method on the dataframe, such as .\texttt{mean}() or .\texttt{max}().

For example, we can calculate the average real minimum wage for each country over the period 2006 to 2016 (the default is to aggregate over rows)

```python
merged.mean().head(10)
```

<table>
<thead>
<tr>
<th>Continent</th>
<th>Country</th>
<th>Real Minimum Wage</th>
</tr>
</thead>
<tbody>
<tr>
<td>America</td>
<td>Brazil</td>
<td>1.09</td>
</tr>
<tr>
<td></td>
<td>Canada</td>
<td>7.82</td>
</tr>
<tr>
<td></td>
<td>Chile</td>
<td>1.82</td>
</tr>
<tr>
<td></td>
<td>Colombia</td>
<td>1.87</td>
</tr>
<tr>
<td></td>
<td>Costa Rica</td>
<td>2.53</td>
</tr>
<tr>
<td></td>
<td>Mexico</td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>United States</td>
<td>7.15</td>
</tr>
<tr>
<td>Asia</td>
<td>Israel</td>
<td>5.95</td>
</tr>
<tr>
<td></td>
<td>Japan</td>
<td>6.18</td>
</tr>
<tr>
<td></td>
<td>Korea</td>
<td>4.22</td>
</tr>
</tbody>
</table>

dtype: float64

Using this series, we can plot the average real minimum wage over the past decade for each country in our data set

```python
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.style.use('seaborn')
merged.mean().sort_values(ascending=False).plot(kind='bar', title="Average real minimum\-wage 2006 - 2016")

#Set country labels
country_labels = merged.mean().sort_values(ascending=False).index.
get_level_values('Country').tolist()
plt.xticks(range(0, len(country_labels)), country_labels)
plt.xlabel('Country')
plt.show()
```
Passing in `axis=1` to `.mean()` will aggregate over columns (giving the average minimum wage for all countries over time)

```python
[26]: merged.mean(axis=1).head()
```

<table>
<thead>
<tr>
<th>Time</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2006-01-01</td>
<td>4.69</td>
</tr>
<tr>
<td>2007-01-01</td>
<td>4.84</td>
</tr>
<tr>
<td>2008-01-01</td>
<td>4.90</td>
</tr>
<tr>
<td>2009-01-01</td>
<td>5.08</td>
</tr>
<tr>
<td>2010-01-01</td>
<td>5.11</td>
</tr>
<tr>
<td>dtype:</td>
<td>float64</td>
</tr>
</tbody>
</table>

We can plot this time series as a line graph

```python
[27]: merged.mean(axis=1).plot()
plt.title('Average real minimum wage 2006 - 2016')
plt.ylabel('2015 USD')
plt.xlabel('Year')
plt.show()
```
We can also specify a level of the MultiIndex (in the column axis) to aggregate over

```python
merged.mean(level='Continent', axis=1).head()
```

<table>
<thead>
<tr>
<th>Continent</th>
<th>America</th>
<th>Asia</th>
<th>Australia</th>
<th>Europe</th>
</tr>
</thead>
<tbody>
<tr>
<td>2006-01-01</td>
<td>2.80</td>
<td>4.29</td>
<td>10.25</td>
<td>4.80</td>
</tr>
<tr>
<td>2007-01-01</td>
<td>2.85</td>
<td>4.44</td>
<td>10.73</td>
<td>4.94</td>
</tr>
<tr>
<td>2008-01-01</td>
<td>2.99</td>
<td>4.45</td>
<td>10.76</td>
<td>4.99</td>
</tr>
<tr>
<td>2009-01-01</td>
<td>3.23</td>
<td>4.53</td>
<td>10.97</td>
<td>5.16</td>
</tr>
<tr>
<td>2010-01-01</td>
<td>3.34</td>
<td>4.53</td>
<td>10.95</td>
<td>5.17</td>
</tr>
</tbody>
</table>

We can plot the average minimum wages in each continent as a time series

```python
merged.mean(level='Continent', axis=1).plot()
plt.title('Average real minimum wage')
plt.ylabel('2015 USD')
plt.xlabel('Year')
plt.show()
```
We will drop Australia as a continent for plotting purposes.

```python
merged = merged.drop('Australia', level='Continent', axis=1)
merged.mean(level='Continent', axis=1).plot()
plt.title('Average real minimum wage')
plt.ylabel('2015 USD')
plt.xlabel('Year')
plt.show()
```
`.describe()` is useful for quickly retrieving a number of common summary statistics

<table>
<thead>
<tr>
<th>Continent</th>
<th>America</th>
<th>Asia</th>
<th>Europe</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>69.00</td>
<td>44.00</td>
<td>200.00</td>
</tr>
<tr>
<td>mean</td>
<td>3.19</td>
<td>4.70</td>
<td>5.15</td>
</tr>
<tr>
<td>std</td>
<td>3.02</td>
<td>1.56</td>
<td>3.82</td>
</tr>
<tr>
<td>min</td>
<td>0.52</td>
<td>2.22</td>
<td>0.23</td>
</tr>
<tr>
<td>25%</td>
<td>1.03</td>
<td>3.37</td>
<td>2.02</td>
</tr>
<tr>
<td>50%</td>
<td>1.44</td>
<td>5.48</td>
<td>3.54</td>
</tr>
<tr>
<td>75%</td>
<td>6.96</td>
<td>5.95</td>
<td>9.70</td>
</tr>
<tr>
<td>max</td>
<td>8.48</td>
<td>6.65</td>
<td>12.39</td>
</tr>
</tbody>
</table>

This is a simplified way to use `groupby`.

Using `groupby` generally follows a ‘split-apply-combine’ process:

- split: data is grouped based on one or more keys
- apply: a function is called on each group independently
- combine: the results of the function calls are combined into a new data structure

The `groupby` method achieves the first step of this process, creating a new `DataFrameGroupBy` object with data split into groups.

Let’s split `merged` by continent again, this time using the `groupby` function, and name the resulting object `grouped`

```python
grouped = merged.groupby(level='Continent', axis=1)
```

```python
grouped = <pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f1a911a9e48>
```

Calling an aggregation method on the object applies the function to each group, the results of which are combined in a new data structure.

For example, we can return the number of countries in our dataset for each continent using `.size()`.

In this case, our new data structure is a `Series`

```python
grouped.size()
```

```
Continent    America 7  Asia 4  Europe 19
dtype: int64
```

Calling `.get_group()` to return just the countries in a single group, we can create a kernel density estimate of the distribution of real minimum wages in 2016 for each continent.

`grouped.groups.keys()` will return the keys from the `groupby` object

```python
import seaborn as sns
continents = grouped.groups.keys()
for continent in continents:
sns.kdeplot(grouped.get_group(continent)['2015'].unstack(), label=continent, shade=True)
plt.title('Real minimum wages in 2015')
plt.xlabel('US dollars')
```
6 Final Remarks

This lecture has provided an introduction to some of pandas’ more advanced features, including multiindices, merging, grouping and plotting.

Other tools that may be useful in panel data analysis include xarray, a python package that extends pandas to N-dimensional data structures.

7 Exercises

7.1 Exercise 1

In these exercises, you’ll work with a dataset of employment rates in Europe by age and sex from Eurostat.

The dataset pandas_panel/employ.csv can be downloaded here.

Reading in the CSV file returns a panel dataset in long format. Use .pivot_table() to construct a wide format dataframe with a MultiIndex in the columns.

Start off by exploring the dataframe and the variables available in the MultiIndex levels.

Write a program that quickly returns all values in the MultiIndex.
7.2 Exercise 2

Filter the above dataframe to only include employment as a percentage of ‘active population’. Create a grouped boxplot using seaborn of employment rates in 2015 by age group and sex. **Hint: GEO includes both areas and countries.**

8 Solutions

8.1 Exercise 1

```python
[35]:
employ = pd.read_csv('https://github.com/QuantEcon/QuantEcon.lectures.code/raw/master/~pandas_panel/employ.csv')
employ = employ.pivot_table(values='Value',
                           index=['DATE'],
                           columns=['UNIT', 'AGE', 'SEX', 'INDIC_EM', 'GEO'])
employ.index = pd.to_datetime(employ.index) # ensure that dates are datetime format
employ.head()

[35]:
UNIT  Percentage of total population ...
AGE  From 15 to 24 years ...
SEX  Females ...
INDIC_EM  Active population ...
GEO  Austria Belgium Bulgaria ...
DATE ...
2007-01-01  56.00  31.60  26.00 ...
2008-01-01  56.20  30.80  26.10 ...
2009-01-01  56.20  29.90  24.80 ...
2010-01-01  54.00  29.80  26.60 ...
2011-01-01  54.80  29.80  24.80 ...

UNIT  Thousand persons ...
AGE  From 55 to 64 years ...
SEX  Total ...
INDIC_EM  Total employment (resident population concept - LFS) ...
GEO  Switzerland Turkey ...
DATE ...
2007-01-01  nan  1,282.00 ...
2008-01-01  nan  1,354.00 ...
2009-01-01  nan  1,449.00 ...
2010-01-01  640.00  1,583.00 ...
2011-01-01  661.00  1,760.00 ...

UNIT ...
AGE ...
SEX ...
INDIC_EM ...
GEO United Kingdom ...
DATE ...
2007-01-01  4,131.00 ...
2008-01-01  4,204.00 ...
2009-01-01  4,193.00 ...
2010-01-01  4,186.00 ...
2011-01-01  4,184.00 ...

[5 rows x 1440 columns]

This is a large dataset so it is useful to explore the levels and variables available

[36]:
employ.columns.names

[36]:
FrozenList(['UNIT', 'AGE', 'SEX', 'INDIC_EM', 'GEO'])

Variables within levels can be quickly retrieved with a loop

17
8.2 Exercise 2

To easily filter by country, swap GEO to the top level and sort the MultiIndex

```python
employ.columns = employ.columns.swaplevel(0, -1)
employ = employ.sort_index(axis=1)
```

We need to get rid of a few items in GEO which are not countries.

A fast way to get rid of the EU areas is to use a list comprehension to find the level values in GEO that begin with ‘Euro’

```python
geo_list = employ.columns.get_level_values('GEO').unique().tolist()
countries = [x for x in geo_list if not x.startswith('Euro')]
employ = employ[countries]
employ.columns.get_level_values('GEO').unique()
```

Select only percentage employed in the active population from the dataframe

```python
employ_f = employ.xs(('Percentage of total population', 'Active population'), level=('UNIT', 'INDIC_EM'), axis=1)
employ_f.head()
```
2010-01-01  54.00  62.60  58.30  ...  51.10  69.20
2011-01-01  54.80  63.60  59.20  ...  51.30  68.40

GEO
AGE
SEX  Total
DATE
2007-01-01 59.30
2008-01-01 59.80
2009-01-01 60.30
2010-01-01 60.00
2011-01-01 59.70

[5 rows x 306 columns]

Drop the ‘Total’ value before creating the grouped boxplot

```python
employ_f = employ_f.drop('Total', level='SEX', axis=1)

box = employ_f['2015'].unstack().reset_index()
sns.boxplot(x='AGE', y=0, hue='SEX', data=box, palette=['husl'], showfliers=False)
plt.xlabel('')
plt.xticks(rotation=35)
plt.ylabel('Percentage of population (%)')
plt.title('Employment in Europe (2015)')
plt.legend(bbox_to_anchor=(1.0, 0.5))
plt.show()
```

References