pandas

#178: Clean-up Your Data With Pandas

The data we get is not always in a useful format. With a bit of Pandas magic we can clean-up a lot and automate the process along the way.

The demo data

I was looking for a good dataset for the more interactive plotting tools in Python and found a dataset that is great for a clean-up exercise. Eurostat publishes the **Net greenhouse gas emissions** from the European Environment Agency (EEA). You can play with the data online and download an Excel file with 4 sheets.

For this post we focus on the sheet named "**Sheet 1**", because it offers us these nice challenges:

- There are rows above and below the data that we do not need.
- There are placeholder columns present that we can remove.
- There are formatting issues with the years.
- There are headers we need to modify.
- The axes are inverted what makes plotting an extra challenge.

We solve all those problems in code and start with loading the data into Pandas.

Skip unwanted rows

When we read the Excel file into a Pandas data frame, we can already do a lot to clean-up the dataset. The **read_excel()** method allows us to skip leading rows with **skiprows** parameter and the rows we do not need after our data get ignored with the **skipfooter** parameter:

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

import os
data_file = os.path.abspath(
    os.path.join(os.getcwd(),
    'data',
    'NetGreenhouseGasEmissions_EEA_sdg_13_10.xlsx'))
export_file = os.path.abspath(
    os.path.join(os.getcwd(),
    'data',
    'Greenhouse_cleaned.csv'))

df = pd.read_excel(
    data_file,
    sheet_name='Sheet 1',
    skiprows=8,
    skipfooter=3,
    engine='openpyxl');
```

This gives us a data frame with only the rows we care about:

Drop NaN columns

The **dropna()** method removes missing values (NaN = Not a Number). The parameter **axis=1** means we want to work with the columns and **how='all'** only drops the column if all values are missing:

```
df = df.dropna(axis=1, how='all')
```

Our data frame now has only columns with at least one value:

Fix the headers

The first row would be a good header. But before we can set that header, we need to fix the decimal points for the years and rename the value "Time" to "Country".

The **iloc** property allows us to access the data by position with a 0-based index value. To get the first row, we can use this code:

```
headers = df.iloc[0]
```

This gives us a Pandas Series that we can convert into a string and then do our replacements:

```
headers = headers.astype(str).apply(lambda x: x.replace('.0',''))
headers = headers.apply(lambda x: x.replace('TIME','Country'))
```

We can now use our cleaned header row and set it as the column titles in a new data frame (the data from the second row to the end is now the "data" part):

```
df  = pd.DataFrame(df.values[1:], columns=headers)
```

This gives us a data frame with the correctly formatted header:

Set the Index values

Our first column would be great to name the data in each row. We can use the **set_index()** method to replace the current index (0 – 32) with the values of the column "Country":

```
df = df.set_index('Country')
```

The country names are now the index in our data frame:

Clean up an index value

It is good to know that the country of Germany before 1990 only covered the part of West Germany. But the data starts at 1990, therefore we do not need to have this explanation in our data set. We can replace an index value with the **rename()** method:

```
df = df.rename(index={'Germany (until 1990 former territory of the FRG)':
'Germany'})
```

Instead of "Germany (until 1990 former territory of the FRG)" we now have "Germany" as the name for that row in our data frame:

Drop a row in the middle of your data frame

If we find a row in the middle of our data frame that we no longer need, we can drop() the row by specifying its name:

```
df = df.drop(labels='GEO (Labels)', axis=0)
```

Our data frame no longer has a line with "GEO (Labels)":

Switch axes

If we want to plot how different countries changed their emissions over the years, our data frame is not optimal. It would be much simpler if we could switch the axes and turn the columns into the index and vice versa. The **swapaxes()** method let us do exactly that:

```
df = df.swapaxes(1, 0)
```

We now have a column for each country and the years are the rows:

With this change we can now plot the changes in the greenhouse gas emissions for Switzerland, Norway, and the United Kingdom:

```
df.filter(
    items=['Switzerland', 'United Kingdom', 'Norway']
    ).plot(xlabel='Year', ylabel='Emissions [Indexed]')
```

Save the data frame

As the last step we should export the data frame so that we can reuse the cleaned data set the next time we want to work with it:

```
df.to_csv(export_file, index=True)
```

Next

We turned an interesting data set into a useful data frame. All it took was a few method calls to transform the Excel into a cleaned up CSV file that we can work with. By doing the clean-up in code we automated the process and can rerun the transformations as soon as the EEA releases new data. The more often that happens, the more value is in our process automation.

Next week we do a bit of exploratory data analysis and use Pandas to find our way in a large data frame.