Introduction to Python for Stata Users
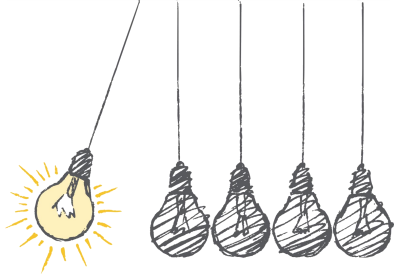
Luis Eduardo San Martin

Development Impact Evaluation (DIME)
The World Bank

WORLD BANK GROUP

i2i
DIME
TRANSFORM DEVELOPMENT

Overview

Web scraping overview

Web scraping - Retrieving raw `html` code

Web scraping - Extracting and arranging information

Web scraping overview

Web scraping overview

- For this second part of the session, we'll do a web scraping exercise
- We'll transform the table shown in this URL: `https://datatables.net/examples/basic_init/zero_configuration.html` into a csv file
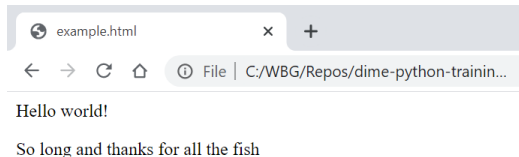
Web scraping overview

Before starting, we'll give a bit more context on web scraping

- You'll probably know that every website you visit has an underlying `html` code
- A simple example:

- Your web browser renders this code and shows you the result

When performing web scraping, we're basically doing two operations

1. We retrieve the underlying raw `html` code of a website

2. From the raw `html` code, we locate the part that is relevant for us to extract and arrange it into an variable we can use

```
</tr><tr role="row" class="odd">
        <td class="sorting_1">Ashton Cox</td>
        <td>Junior Technical Author</td>
        <td>San Francisco</td>
        <td>66</td>
        <td>2009/01/12</td>
        <td>$86,000</td>
</tr><tr role="row" class="even">
        <td class="sorting_1">Bradley Greer</td>
        <td>Software Engineer</td>
        <td>London</td>
        <td>41</td>
        <td>2012/10/13</td>
        <td>$132,000</td>
</tr><tr role="row" class="odd">
        <td class="sorting_1">Brenden Wagner</td>
        <td>Software Engineer</td>
        <td>San Francisco</td>
        <td>28</td>
        <td>2011/06/07</td>
        <td>$206,850</td>
</tr><tr role="row" class="even">
        <td class="sorting_1">Brielle Williamson</td>
        <td>Integration Specialist</td>
        <td>New York</td>
        <td>61</td>
        <td>2012/12/02</td>
        <td>$372,000</td>
</tr><tr role="row" class="odd">
        <td class="sorting_1">Bruno Nash</td>
        <td>Software Engineer</td>
        <td>London</td>
        <td>38</td>
```

	name	position	office	age	start date	salary
0	Tiger Nixon	System Architect	Edinburgh	61	2011/04/25	$320,800
1	Garrett Winters	Accountant	Tokyo	63	2011/07/25	$170,750
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	$86,000
3	Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29	$433,060
4	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700
5	Brielle Williamson	Integration Specialist	New York	61	2012/12/02	$372,000
6	Herrod Chandler	Sales Assistant	San Francisco	59	2012/08/06	$137,500
7	Rhona Davidson	Integration Specialist	Tokyo	55	2010/10/14	$327,900
8	Colleen Hurst	Javascript Developer	San Francisco	39	2009/09/15	$205,500
9	Sonya Frost	Software Engineer	Edinburgh	23	2008/12/13	$103,600
10	Jena Gaines	Office Manager	London	30	2008/12/19	$90,560
11	Quinn Flynn	Support Lead	Edinburgh	22	2013/03/03	$342,000

Web scraping overview

- To execute these operations, we'll need to use three Python libraries.
- Libraries are similar to the user-written Stata commands you download from the ssc repository, except that you have to manually load them in each new Python session you start.
- Each library has custom made data types with operations specific to the topic of the library.
- To install libraries, we would normally have to use Window's or Mac's command lines. However, we'll skip installing for now because Colab has the libraries we need already pre-installed.

Web scraping overview

To load libraries, we use `import`

```
[2]  import requests
     from bs4 import BeautifulSoup
     import pandas as pd
```

- `requests` is an `html` requestor. We'll use it to retrieve the `html` code of our URL
- `BeautifulSoup` is a text parser. We'll use it to extract the information we need from the `html` code
- `pandas` is a dataframe manipulation library. We'll use to structure the information into a dataframe and export the result to a `csv` file

Web scraping - Retrieving raw `html` code

We'll use the command `get()` from `requests` to retrieve the `html` code. We'll start by saving the result of the request in an variable called `response`

```
[9]  url = 'https://datatables.net/examples/basic_init/zero_configuration.html'
     response = requests.get(url)
```

`response` is a data type that contains not only the `html` code we're looking for, but also information about our html request itself. For example, we can use `status_code` to check if our request was successful:

```
[10] print(response.status_code)

    200
```

A value of 200 indicates that the request worked

After we checked that the request was successful, we use an attribute called `.text` on the response data type to extract the html code from the page we requested. The result is a single very long string.

```
[11] response_string = response.text
     type(response_string)

     str
```

`response_string` contains the raw `html` code we need

Web scraping - Extracting and arranging information

Now we need to manually explore the `html` string and locate the information of interest. We can use `print()` for this

Web scraping - Extracting and arranging information

This manual inspection allow us to see that every single observation in our target table is enclosed between `tr` tags, and that every piece of information inside an observation is enclosed in `td` tags

```
<tr>
        <td>Vivian Harrell</td>
        <td>Financial Controller</td>
        <td>San Francisco</td>
        <td>62</td>
        <td>2009/02/14</td>
        <td>$452,500</td>
</tr>
<tr>
        <td>Timothy Mooney</td>
        <td>Office Manager</td>
        <td>London</td>
        <td>37</td>
        <td>2008/12/11</td>
        <td>$136,200</td>
</tr>
<tr>
        <td>Jackson Bradshaw</td>
        <td>Director</td>
        <td>New York</td>
        <td>65</td>
        <td>2008/09/26</td>
        <td>$645,750</td>
</tr>
```

- `tr` - stands for "table row"

- `td` - stands for "table data cell"

Web scraping - Extracting and arranging information

Then, we need to do the following:

1. From the whole string, extract every content enclosed in `tr` tags
2. For each of the contents enclosed in `tr` tags, extract every piece of information enclosed in `td` tags

We'll use `BeautifulSoup` for this - a commonly used library to extract information from an html code

Web scraping - Extracting and arranging information

- BeautifulSoup parses a string by detecting symbols and spacing that creates sections and subsections in plain text
- If used in html code, it knows automatically that tags are used to specify sections

Web scraping - Extracting and arranging information

- To parse an `html` string, we use the following code:

```
[24] soup = BeautifulSoup(response_string)
     type(soup)

     bs4.BeautifulSoup
```

- The result is not a string anymore, but a `BeautifulSoup` data type
- You can think of it as a text with divisions and subdivisions with many operations to work with the content

- The advantage of using this type as opposed to a string is that now we can directly look for any divisions enclosed by the the `tr` tags

- We'll use the `.find_all()` attribute for this and save the result in a new variable named `observations`

- Note that `observations` is a list

```
[58] observations = soup.find_all('tr')
     print(observations)

     [<tr>
     <th>Name</th>
     <th>Position</th>
     <th>Office</th>
     <th>Age</th>
     <th>Start date</th>
     <th>Salary</th>
     </tr>, <tr>
     <td>Tiger Nixon</td>
     <td>System Architect</td>
     <td>Edinburgh</td>
     <td>61</td>
     <td>2011/04/25</td>
     <td>$320,800</td>
     </tr>, <tr>
     <td>Garrett Winters</td>
     <td>Accountant</td>
     <td>Tokyo</td>
```

Web scraping - Extracting and arranging information

- `observations` is a list that contains every piece of text that was enclosed in `tr` tags, including the tags themselves
- Nonetheless, every element of it is not a string, but another `BeautifulSoup` data type
- If we inspect a single element of `observations`, we get this:

```
[59] observations[10]

     <tr>
     <td>Sonya Frost</td>
     <td>Software Engineer</td>
     <td>Edinburgh</td>
     <td>23</td>
     <td>2008/12/13</td>
     <td>$103,600</td>
     </tr>
```

Web scraping - Extracting and arranging information

- You might see by now that we're getting closer to our final result!
- We basically need to iterate through the elements of `observations` and extract everything that is inside the `td` tags
- Since the elements of `observations` are still `Beautiful Soup` data types, we can use once again the `.find_all()` attribute for this

- For a single element, this would be:

```
[26] single_obs = observations[10].find_all('td')
     print(single_obs)

     [<td>Sonya Frost</td>, <td>Software Engineer</td>, <td>Edinburgh</td>, <td>23</td>, <td>2008/12/13</td>, <td>$103,600</td>]
```

- This is *almost* what we want, except that it still includes the `td` tags

To eliminate them, we need to use `.text` with every element of `single_obs`

```
[63] for element in single_obs:
        print(element.text)

    Sonya Frost
    Software Engineer
    Edinburgh
    23
    2008/12/13
    $103,600
```

Now every one of these is a string data type again!

Furthermore, we can modify this code to save the result in a new list

```
[29] result = [] # this creates an empty list
     for element in single_obs:
         print(element.text)
         result.append(element.text) # this adds a new element to "result"
     print(result)

     Sonya Frost
     Software Engineer
     Edinburgh
     23
     2008/12/13
     $103,600
     ['Sonya Frost', 'Software Engineer', 'Edinburgh', '23', '2008/12/13', '$103,600']
```

Now that we finally figured out how to extract the information we want, we just
need to loop over `observations` to generalize our approach and get the entire
information

```
[45] total_results = []

     for observation in observations:

         result = []
         observation_parsed = observation.find_all('td')

         for element in observation_parsed:
             result.append(element.text)
         total_results.append(result)
```

Web scraping - Extracting and arranging information

- You might have noticed that the first and last elements of `total_results` are empty lists

- This is because in our original `html` string, the first and last elements enclosed in `tr` tags didn't contain any elements enclosed in `td` tags. Then, using `observation.find_all("td")` produced an empty lists in both cases

- We can easily eliminate these observations from `total_results` by subsetting the list and excluding the last and first elements

```
[39] total_results = total_results[1:-1]
```

Web scraping - Extracting and arranging information

- total_results contains the result we wanted to have. Great!
- The final step is to export it to a csv file
- Remember that we loaded the pandas library? We'll use it now

Web scraping - Extracting and arranging information

First, we create a pandas dataframe where we insert total_results as data. We call the dataframe df

```
[51] df = pd.DataFrame(data=total_results, columns=['name', 'position', 'city', 'age', 'date', 'salary'])
```

pd.DataFrame() uses two arguments in this case:

1. data: the data we're loading to the dataframe
2. columns: a list of strings with the column names

Pandas dataframes are a data type that is similar to a dataset in Stata

Web scraping - Extracting and arranging information

```
[52] df.head()
```

	name	position	city	age	date	salary
0	Tiger Nixon	System Architect	Edinburgh	61	2011/04/25	$320,800
1	Garrett Winters	Accountant	Tokyo	63	2011/07/25	$170,750
2	Ashton Cox	Junior Technical Author	San Francisco	66	2009/01/12	$86,000
3	Cedric Kelly	Senior Javascript Developer	Edinburgh	22	2012/03/29	$433,060
4	Airi Satou	Accountant	Tokyo	33	2008/11/28	$162,700

Finally, we export `df` into a `csv` file using the attribute `.to_csv()`

```
[57] export_file_name = 'table.csv'
     df.to_csv(export_file_name, index=False)
```

`pandas` by default exports a column with the index numbers. We use `index=False` to omit it

Web scraping - Extracting and arranging information

- Since we were using Colab in this exercise, we just exported `table.csv` to the cloud
- To download the result to your computer, click the folder icon to the left, locate your file, click on the vertical ellipsis next to it and click `Download`