Module 5: Data Collection & Cleaning Part 2

This module consists of 3 parts.

- **Part 1** - Data Sources
- **Part 2** - Web Scraping
- **Part 3** - Data Preparation

Each part is provided in a separate file. It is recommended that you follow the order of the files.

Table of Contents

Web Scraping

Now that we have an idea of what the data looks like in terms of form, we can begin scraping the data. Here we cover JSON and XML, as these formats cover the majority of web sources. As a special case we also include HTML. Previously, we showed how to download a CSV file from a website and loaded it into a usable `DataFrame` object. In this section we focus on doing the same so that you'll be able to fully utilize web resources as data.

HTML & XML

Python has many libraries for reading and writing data in HTML and XML formats. `lxml` (Behnel, 2018) is a python library that has consistently strong performance in parsing very large files. Lets begin using `lxml` by scraping HTML.

NOTE: In the following example, we can't use `pandas` to scrape. `pd.read_html(trickySite)` will throw an error. This is because the `<table>` element isn't used by the site.

In [7]:
```
# Setup code and importing libraries
from bs4 import BeautifulSoup
import numpy as np
import pandas as pd
import re as re
np.random.seed(12345)
import matplotlib.pyplot as plt
plt.rc('figure', figsize=(10, 6))
np.set_printoptions(precision=4, suppress=True)
pd.options.display.max_rows = 6
```

In [8]:
```
import requests
import warnings
warnings.filterwarnings("ignore")

trickySite = 'https://xmarquez.github.io/democracyData/reference/pacl.html'
# To get started, find the URL you want to extract data from, open it with *request
r = requests.get(trickySite, verify=False, )
r
```

Out[8]:
```
<Response [200]>
```

Now what? We know we have to extract the data, but wasn't this the entire point of using `pandas`? So that we wouldn't have to do this part?

Unfortunately, this part can get very messy, especially for a web-scraping novice. Instead, we will make a simplifying assumption. *__HTML is well formed XML__. *If this is true, then we should be able to manipulate and select items as if they are XML. The easiest way to do this is with XPath** and **XSLT**.

XPath

XML Path Language (XPath) is a query language for selecting nodes from XML or to compute values from XML. XPath is simply a way to select groups of elements, attributes, text, etc., based on the XML tree structure.

For example, the XPath value `/html/body/a` would grab all `a` elements directly in the `body` element in a root `html` element. However, if an `a` element is embedded inside another element in `body`, then it will not be retrieved. XPath can be thought of as a query language using a syntax similar to file directories. File directories can be absolute or relative.

In our example, the path was absolute. If we wanted all `a` elements at any depth inside `body`, then we would use `/html/body//a` as our XPath.

Generally, XPath is used with XML. Since XML defines other formats, XPath is a very useful technology to know when dealing with any markup language. It serves the exact same purpose as Regular Expressions, except for matching tree structures and branches in XML.

XSLT

eXtensible Stylesheet Language Transformations (XSLT) is best described as a way to transform, merge, join, and generally perform operations with XML. The use-case of XSLT is to provide a way to rewrite XML data into other formats. XSLT is a subset of XSL, a stylesheet language. Yet, in order for XSLT to remain general, nothing technology specific can be used. This results in a language specifically designed for expressing transformations concisely for XML.

NOTE: XSLT is itself expressed as an XML data format.

You may notice that XPath is the equivalent to a CSS selector. Both need to find groups of elements and then apply their respective snippets of stylesheet code onto them.

Consider the following benefits of these technologies so far:

- XML data is guaranteed to be cleanly parsed.
- XPath ensures subsets of documents can be retrieved without problems.
- XSLT ensures elements can be rewritten from XML input to any output.

__Thus XML, XPath, and XSLT becomes a powerful generic data manipulation pipeline.__ But more importantly, it allows us to transform the elements we wish to extract into an HTML table, while ignoring all other elements. We are now able to do all of this in very few lines of code \cite{Behnel2018}.

In [9]:
```
import lxml
import lxml.etree
"""
1. find the root
2. match on all `dl` tags and begin producing a `table` of them
3. match on all `dd` and `dt` tags and
  * place them into their own rows (<tr> tags)
    * inside their own cells (<td> tags)
"""
xslt_root = lxml.etree.XML('''\
 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
     <xsl:template match="/">
        <html><body>
             <xsl:for-each select=".//dl">
                 <table>
                     <tr>
                         <xsl:for-each select=".//dd">
```

```
                        <td>
                            <xsl:value-of select="." />
                        </td>
                        </xsl:for-each>
                    </tr>
                    <tr>
                        <xsl:for-each select=".//dt">
                            <td>
                                <xsl:value-of select="." />
                            </td>
                        </xsl:for-each>
                    </tr>
                </table>
            </xsl:for-each>
        </body></html>
    </xsl:template>
 </xsl:stylesheet>''')

# compile it from the XSLT
transform = lxml.etree.XSLT(xslt_root)

# apply the XSLT transformation to an HTML compiled representation of the requested
tree_to_scrape = transform(lxml.etree.HTML(r.content))

## Return string of output. Uncomment to view output
# str(tree_to_scrape)
```

```
None
```

In [10]:

```
"""
1. Convert to string
2. Read with pandas
3. Grab the first/zeroth table
4. Transpose it
5. Switch indexing order for columns (optional)
"""
firstTable = pd.read_html(str(tree_to_scrape))[0].T[[1,0]]

# return pandas `DataFrame`
firstTable
```

Out[10]:

	1	0
0	order	Sequential numbering of rows (1 through 9159)
1	pacl_country	String country identifier.
2	year	Calendar year
...	...	...
68	agedem	Age in years of the current regime as classifi...
69	agereg	Age in years of the current regime as classifi...
70	stra	Sum of past transitions to authoritarianism in...

71 rows × 2 columns

JSON

JavaScript Object Notation (JSON) has become one of the standard formats for sending data by HTTP request between web browsers and other applications. It is a much more flexible data format than a tabular text form like CSV, and much smaller in size than XML files. JSON is very nearly valid Python code.

To convert a JSON string to Python form, use `json.loads`. `json.dumps` on the other hand converts a Python object back to JSON. Because the JSON structure is unpredictable, how you convert a JSON object or list of objects to a DataFrame or some other data structure for analysis will be up to you. Conveniently, we are able to pass a list of JSON objects to the `DataFrame` constructor and select subsets of fields.

In [11]:
```
import requests

# Endpoint for retrieving Yahoo Weather for Toronto, ON.
apiEndpoint =  'http://ws1.metcheck.com/ENGINE/v9_0/json.asp?lat=43.7&lon=-79.4&lid


response_json = pd.read_json(apiEndpoint, lines=False)
response_json
```

Out[11]:

	metcheckData	feedCreation	feedCreator	feedModel	feedModelRun
forecastLocation	{'forecast': [{'temperature': '7', 'dewpoint':...	2024-03-09T01:49:33.00	Metcheck.com	GHX11	21Z

How do we get a pandas `DataFrame` from the results and not the response? `results` actually contains a proper python object. So, it can be manually manipulated or we can

reapply pandas to `results`.

And, this can be repeated. The best way to repeat would be to use the `map()` function and then combine horizontally or vertically depending on the situation. Additionally, it would be preferable to wrap these in a python function to compartmentalize the code used for data manipulation.

In [12]:
```
pd.DataFrame(response_json["metcheckData"]["forecastLocation"])
```

Out[12]:

	forecast	continent	country	location	latitude	longitude	timezone
0	{'temperature': '7', 'dewpoint': '0', 'rain': ...			43.7000/-79.4000	43.7	-79.4	5
1	{'temperature': '6', 'dewpoint': '1', 'rain': ...			43.7000/-79.4000	43.7	-79.4	5
2	{'temperature': '6', 'dewpoint': '2', 'rain': ...			43.7000/-79.4000	43.7	-79.4	5
...	...	...	...	...	...	...	...
150	{'temperature': '0', 'dewpoint': '-11', 'rain'...			43.7000/-79.4000	43.7	-79.4	5
151	{'temperature': '-1', 'dewpoint': '-12', 'rain...			43.7000/-79.4000	43.7	-79.4	5
152	{'temperature': '-4', 'dewpoint': '-11', 'rain...			43.7000/-79.4000	43.7	-79.4	5

153 rows × 7 columns

EXERCISE 1: Scraping data on elected officials

Extract the *Members of Parliament* in the Canadian House of Commons (House of Commons, 2018)

In [15]:
```
## Your Code Here
import io
import requests
import pandas as pd
```

```
r = requests.get('https://www.ourcommons.ca/Parliamentarians/en/members/export?outp
pd.read_csv(filepath_or_buffer=io.StringIO(r.text))
```

Out[15]:

	ï»¿Message	URL
0	The paths for the exports have migrated to a n...	www.ourcommons.ca/members/en
1	Below is a list of links where to find new exp...	NaN
2	Search:	www.ourcommons.ca/members/en/search
...	...	...
8	Party Standings:	www.ourcommons.ca/members/en/party-standings
9	Election Candidates:	www.ourcommons.ca/members/en/election-candidates
10	Chamber Votes:	www.ourcommons.ca/members/en/votes

11 rows × 2 columns

In [16]:

```
### SOLUTION
r = requests.get('https://www.ourcommons.ca/Parliamentarians/en/members/export?outp
pd.read_csv(filepath_or_buffer=io.StringIO(r.text))
```

Out[16]:

	ï»¿Message	URL
0	The paths for the exports have migrated to a n...	www.ourcommons.ca/members/en
1	Below is a list of links where to find new exp...	NaN
2	Search:	www.ourcommons.ca/members/en/search
...	...	...
8	Party Standings:	www.ourcommons.ca/members/en/party-standings
9	Election Candidates:	www.ourcommons.ca/members/en/election-candidates
10	Chamber Votes:	www.ourcommons.ca/members/en/votes

11 rows × 2 columns

Scraping the Easy Way

There is of course nothing wrong with a more interactive or manual approach for webscraping using interactive tools. We only focus on automating the process for repetition.

But it should be noted that websites are not static and will often change. For example, people profiles for Members of Parliament change on a regular basis.

Below are some alternatives for scraping

- Google Chrome Web Scraper extension
- import.io
- Spooky Stuff

Python Libraries for Web Scraping

There are many libraries that can be used for scraping web resources. We chose to use the `requests` library largely due to its ability to fully interact with Web APIs and its ability to reduce most use-cases to one-line of code in comparison to other libraries. Similarly, we make use of `lxml` due to its support for XML, XPath, and XSLT for easier transformation into a format immediately useable by `pandas`. Below are the libraries in Python relevant for the web scraping domain.

- urllib2
- BeautifulSoup
  - **NOTE**: `lxml` is used by the library.
- scrapy
  - **NOTE**: library is for crawling entire websites, and not requesting only one specific Web Resource.
- requests
- RoboBrowser
  - For scraping and filling out forms automatically. Does not render dynamic pages.
- lxml

One topic we don't cover are headless web browsers — which simulate an actual browser — enabling scraping of the *Document Object Model* and scraping of dynamic web resources. The following are libraries and tools specifically for addressing this use case.

- splash
- selenium
- Splinter
- phantompy
  - phantomjs

End of Part 2

This notebook makes up one part of this module. Now that you have completed this part, please proceed to the next notebook in this module. If you have any questions, please reach out to your peers using the discussion boards. If you and your peers are unable to come to a

suitable conclusion, do not hesitate to reach out to your instructor on the designated discussion board.

Links

- http://blog.miguelgrinberg.com/post/easy-web-scraping-with-python
  - About scraping using other python libraries, as well as crawling entire websites.
- http://scrapy.org/
  - About writing scrapers as configeration files via scrapy.
- https://docs.python.org/2/library/urllib2.html
  - Documentation for urlib2 library
- http://docs.python-requests.org/en/latest/
- The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)
  - https://www.joelonsoftware.com/2003/10/08/the-absolute-minimum-every-software-developer-absolutely-positively-must-know-about-unicode-and-character-sets-no-excuses/
- http://import.io
  - A web-based platform for extracting data from websites without writing any code.
- http://www.crummy.com/software/BeautifulSoup/
  - Popular alternative to lxml for web/screen scraping
- http://pbpython.com/web-scraping-mn-budget.html
  - Tutorial using BeautifulSoup with requests library, pandas, numpy and mathplotlib
- Python Regular Expressions Cheat Sheet
  - https://pycon2016.regex.training/cheat-sheet

References

Behnel, S. et al, (2018). Xpath and xslt with lxml online

House of Commons, (2018). Members of Parliament. Retrieved Aug 21, 2018 from http://www.ourcommons.ca/Parliamentarians/en/members

McKinney, W. (2017). Python for data analysis: Data wrangling with Pandas, NumPy, and IPython (2nd Ed.). O'Reilly Media.