Web crawler and web scraping

April 29, 2016

1 Web Crawler and Web Scraping

The first part of this page is almost copied and pasted from this article. The second part is one application.

1.1 Retrieving data from the web

Prepare the environment namespace

```
In [66]: import requests
         from bs4 import BeautifulSoup
         from IPython.core.display import HTML
         import pandas as pd
         from pandas import DataFrame
         import numpy as np
```

requests Use the appropriately named get function to issue a GET request. This is equivalent to typing a URL into your browser and hitting enter.

```
In [21]: # Get the HU Wikipedia page
         req = requests.get("https://en.wikipedia.org/wiki/Harvard_University")
```

The next step is to assign the value of the text property of this Request object to a variable.

```
In [22]: page = req.text
```

Now we have the text of the HU Wikipedia page. But this mess of HTML tags would be a pain to parse manually. Which is why we will use another very cool Python library called BeautifulSoup.

BeautifulSoup One of the problems with HTML is that over the years browsers have evolved to be very forgiving of "malformed" syntax. Your browser is smart enough to detect some common problems, such as open tags, and correct them on the fly.

Unfortunately, we do not have the time or patience to implement all the different corner cases, so we'll let BeautifulSoup do that for us.

BeautifulSoup can deal with HTML or XML data, so the next line parser the contents of the page variable using its HTML parser, and assigns the result of that to the soup variable.

```
In [23]: soup = BeautifulSoup(page, 'html.parser')
```

BeautifulSoup obkects have a cool little method that allows you to see the HTML content in a nice, indented way.

```
In [ ]: soup.prettify()
```

We can now reference elements of the HTML document in different ways. One very convenient way is by using the dot notation, which allows us to access the elements as if they were properties of the object.

```
In [6]: soup.title

Out[6]: <title>Harvard University - Wikipedia, the free encyclopedia</title>
```

But we should make it clear that this is again just syntactic sugar. title is not a property of the soup object and I can prove it:

```
In [7]: "title" in dir(soup)

Out[7]: False
```

This is nice for HTML elements that only appear once per page, such the the title tag. But what about elements that can appear multiple times?

```
In [8]: # Be careful with elements that show up multiple times.
        soup.p

Out[8]: <p><b>Harvard University</b> is a private <a class="mw-redirect" href="/wiki/Research_university

In [9]: len(soup.find_all("p"))

Out[9]: 77
```

If you look at the Wikipedia page on your browser, you'll notice that it has a couple of tables in it. We will be working with the "Demographics" table, but first we need to find it.

One of the HTML attributes that will be very useful to us is the "class" attribute.

Getting the class of a single element is easy.

```
In [11]: soup.table["class"]

Out[11]: ['infobox', 'vcard']
```

List Comprehensions Next we will use a list comprehension to see all the tables that have a "class" attributes. List comprehensions are a very cool Python feature that allows for a loop iteration and a list creation in a single line.

```
In [11]: [t["class"] for t in soup.find_all("table") if t.get("class")]

Out[11]: [[u'infobox', u'vcard'],
          [u'toccolours'],
          [u'metadata', u'plainlinks', u'ambox', u'mbox-small-left', u'ambox-content'],
          [u'infobox', u'vcard'],
          [u'wikitable'],
          [u'metadata', u'plainlinks', u'mbox-small'],
          [u'navbox'],
          [u'nowraplinks', u'collapsible', u'collapsed', u'navbox-inner'],
          [u'nowraplinks', u'navbox-subgroup'],
          [u'nowraplinks', u'navbox-subgroup'],
          [u'navbox'],
          [u'nowraplinks', u'collapsible', u'collapsed', u'navbox-inner'],
          [u'navbox'],
          [u'nowraplinks', u'collapsible', u'autocollapse', u'navbox-inner'],
          [u'navbox'],
          [u'nowraplinks', u'collapsible', u'autocollapse', u'navbox-inner'],
          [u'navbox'],
          [u'nowraplinks', u'collapsible', u'autocollapse', u'navbox-inner'],
          [u'navbox'],
```

```
[u'nowraplinks', u'collapsible', u'autocollapse', u'navbox-inner'],
[u'navbox'],
[u'nowraplinks', u'collapsible', u'autocollapse', u'navbox-inner'],
[u'navbox'],
[u'nowraplinks', u'collapsible', u'autocollapse', u'navbox-inner'],
[u'navbox'],
[u'nowraplinks', u'hlist', u'collapsible', u'autocollapse', u'navbox-inner'],
[u'navbox'],
[u'nowraplinks', u'collapsible', u'autocollapse', u'navbox-inner'],
[u'navbox'],
[u'nowraplinks', u'collapsible', u'autocollapse', u'navbox-inner'],
[u'navbox'],
[u'nowraplinks', u'hlist', u'collapsible', u'autocollapse', u'navbox-inner'],
[u'navbox'],
[u'nowraplinks', u'hlist', u'navbox-inner']]
```

As I mentioned, we will be using the Demographics table for this lab. The next cell contains the HTML elements of said table. We will render it in different parts of the notebook to make it easier to follow along the parsing steps.

```
In [13]: table_html = str(soup.find("table", "wikitable"))
         HTML(table_html)
```

```
Out[13]: <IPython.core.display.HTML object>
```

First we'll use a list comprehension to extract the rows (`tr`) elements.

```
In [14]: rows = [row for row in soup.find("table", "wikitable").find_all("tr")]
         rows
```

```
Out[14]: [<tr>
          <th></th>
          <th>Undergraduate</th>
          <th>Graduate<br/>
          and Professional</th>
          <th>U.S. Census</th>
          </tr>, <tr>
          <th>Asian/Pacific Islander</th>
          <td>17%</td>
          <td>11%</td>
          <td>5%</td>
          </tr>, <tr>
          <th>Black/Non-Hispanic</th>
          <td>6%</td>
          <td>4%</td>
          <td>12%</td>
          </tr>, <tr>
          <th>Hispanics of any race</th>
          <td>9%</td>
          <td>5%</td>
          <td>16%</td>
          </tr>, <tr>
          <th>White/non-Hispanic</th>
          <td>46%</td>
          <td>43%</td>
```

```
<td>64%</td>
</tr>, <tr>
<th>Mixed Race/Other</th>
<td>10%</td>
<td>8%</td>
<td>9%</td>
</tr>, <tr>
<th>International students</th>
<td>11%</td>
<td>27%</td>
<td>N/A</td>
</tr>]
```

We will then use a lambda expression to replace new line characters with spaces.

```
In [15]: # Lambda expressions return the value of the expression inside it.
         # In this case, it will return a string with new line characters replaced by spaces.
         rem_nl = lambda s: s.replace("\n", " ")
```

Splitting the data Next we extract the text value of the columns. If you look at the table above, you'll see that we have three columns and six rows.

Here we're taking the first element (Python indexes start at zero), iterating over the th elements inside it, and taking the text value of those elements. We should end up with a list of column names. But there is one little caveat: the first column of the table is actually an empty string (look at the cell right above the row names). We could add it to our list and then remove it afterwards; but instead we will use the if statement inside the list comprehension to filter that out.

You should be familiar with if statements. They perform a Boolean test and an action if the test was successful. Python considers most values to be equivalent to True. The exceptions are `False, None, 0, ""` (empty string), `[]/{}/(,)...` (empty containers). Here the get_text will return an empty string for the first cell of the table, which means that the test will fail and the value will not be added to the list.

```
In [16]: columns = [rem_nl(col.get_text()) for col in rows[0].find_all("th") if col.get_text()]
         columns

Out[16]: [u'Undergraduate', u'Graduate and Professional', u'U.S. Census']
```

Now let's do the same for the rows. Notice that since we have already parsed the header row, we will continue from the second row. The `[1:]` is a slice notation and in this case it means we want all values starting from the second position.

```
In [17]: indexes = [row.find("th").get_text() for row in rows[1:]]
         indexes

Out[17]: [u'Asian/Pacific Islander',
          u'Black/Non-Hispanic',
          u'Hispanics of any race',
          u'White/non-Hispanic',
          u'Mixed Race/Other',
          u'International students']
```

Here we have another lambda expression that transforms the string on the cells to integers. We start by checking if the last character of the string (Python allows for negative indexes) is a percent sign. If that is true, then we convert the characters before the sign to integers. Lastly, if one of the prior checks fails, we return a value of None.

This is a very common pattern in Python, and it works for two reasons: Python's and and or are "short-circuit" operators. This means that if the first element of an and statement evaluates to False, the second

one is never computed (which in this case would be a problem since we can't convert a non-digit string to an integer). The or statement works the other way: if the first element evaluates to True, the second is never computed. The second reason this works is because these operators will return the value of the last expression that was evaluated, which is this case will be either the integer value or the value None.

One last thing to notice: Python slices are open on the upper bound. So the `[:-1]` construct will return all elements of the string, except for the last.

```
In [18]: to_num = lambda s: s[-1] == "%" and int(s[:-1]) or None
```

Now we use the lambda expression to parse the table values.

Notice that we have two for ... in ... in this list comprehension. That is perfectly valid and somewhat common. Although there is no real limit to how many iterations you can perform at once, having more than two can be visually unpleasant, at which point regular nested loops might be a better solution.

```
In [21]: values = [to_num(value.get_text()) for row in rows[1:] for value in row.find_all("td")]
         values

Out[21]: [17, 11, 5, 6, 4, 12, 9, 5, 16, 46, 43, 64, 10, 8, 9, 11, 27, None]
```

The problem with the list above is that the values lost their grouping.

The zip function is used to combine two sequences element wise. So zip([1,2,3], [4,5,6]) would return [(1, 4), (2, 5), (3, 6)].

This is the first time we see a container bounded by parenthesis. This is a tuple, which you can think of as an immutable list (meaning you can't add, remove, or change elements from it). Otherwise they work just like lists and can be indexed, sliced, etc.

```
In [44]: stacked_values = zip(*[values[i::3] for i in range(len(columns))])
         stacked_values

Out[44]: [(17, 11, 5), (6, 4, 12), (9, 5, 16), (46, 43, 64), (10, 8, 9), (11, 27, None)]
```

1.2 pandas data structures

```
In [46]: df = pd.DataFrame(stacked_values, columns=columns, index=indexes)
         df
```

Out[46]:

	Undergraduate	Graduate and Professional	U.S. Census
Asian/Pacific Islander	17	11	5
Black/Non-Hispanic	6	4	12
Hispanics of any race	9	5	16
White/non-Hispanic	46	43	64
Mixed Race/Other	10	8	9
International students	11	27	NaN

Method 2:

```
In [50]: stacked_by_col = [values[i::3] for i in range(len(columns))]
         df = pd.DataFrame(stacked_by_col).T
         df.columns = columns
         df.index = indexes
```

Method 3:

```
In [51]: data_dicts = [{col: val for col, val in zip(columns, col_values)} for col_values in stacked_val
         df = pd.DataFrame(data_dicts, index=indexes)
```

2 One hand-on application

You are required to find info from web page, extract the information from it, crawl to the profile page of each person (eg. http://www.seas.upenn.edu/directory/profile.php?ID=191), and extract more information. By using chrome, right-click->View Page Source will lead you to the HTML file page. Use online html formatter and you will get a much prettier view of the html file. In case you don't understand HTML syntax, we will explain it in an illustrative way.

```
In [32]: req = requests.get('http://www.seas.upenn.edu/directory/departments.php')
         page = req.text
         soup = BeautifulSoup(page, 'html.parser')

In [71]: # Find all table
         table_html = soup.find_all('table')
```

Now we need to visualize these tables and select informative ones that we want.

```
In [61]: HTML(str(table_html[0]))

Out[61]: <IPython.core.display.HTML object>

In [62]: HTML(str(table_html[1]))

Out[62]: <IPython.core.display.HTML object>

In [63]: HTML(str(table_html[2]))

Out[63]: <IPython.core.display.HTML object>

In [64]: HTML(str(table_html[3]))

Out[64]: <IPython.core.display.HTML object>
```

I just present first four tables. Actually I also check the last three and some random tables in the middle. Then I knew that the first table is empty so I exclude it from analysis. The remaining tables have one regulation. Each parent table has two children tables and the information we want is in the last children. Therefore, we can extract information from these informative tables.

```
In [70]: rem_nl = lambda s: s.replace(": ", "")  #Delete the ': ' in the colums
         rem_nl1 = lambda s: s.replace("\xa0", " ")   #This is to replace "\xa0" with " ".ß
         rem_nl2 = lambda s: s.replace("  ", " ")    #Change double space to one.
         rem_nl3 = lambda s: s.replace("mailto:", "")    #Delete "mailto:" from email address
         columns = [rem_nl(col.get_text()) for col in table_html[3].find_all("strong") if col.get_text()
         columns.append('Email')
         table_out = DataFrame(index=range(int(489/3)), columns = columns)
         for i in range(1, int((len(table_html) - 1)/3)):
             #Get rows.
             rr = tt[3 * i].find_all('tr')
             #Fill in the Name column
             table_out.ix[i - 1, 0] = rem_nl1(rr[0].get_text().lstrip('\xa0Name: ').rstrip('|  View Pro
             #Fill in the Dept column
             rr1 = rr[1].get_text()
             rr1_t = np.array(list(rr1))
             ind_f = np.arange(len(rr1))[rr1_t == '(']
             if len(ind_f) != 0:
                 ind_b = np.arange(len(rr1))[rr1_t == ')']
                 for a in range(len(ind_f)):
```

```
                if a == 0:
                    table_out.ix[i - 1, 1] = rr1[(ind_f[a] + 1):ind_b[a]]
                else:
                    table_out.ix[i - 1, 1] = table_out.ix[i - 1, 1] + ' | ' + rr1[(ind_f[a] + 1):ir
        #Fill in the Research expertise column
        rr2 = rr[2].get_text()
        table_out.ix[i - 1, 2] = rem_nl2(rr2.lstrip('Research Expertise: ').rstrip(' '))
        #Fill in the Email column
        table_out.ix[i - 1, 3] = rem_nl3(rr[3].a["href"])     #Is there an easier way to get email

    #Show top 5 rows in the table
    table_out.head()
```

Out[70]:

	Name	Dept	Research Expertise	Email
0	Firooz Aflatoun	ESE	Integrated Circuits | Silicon Photonics	firooz@seas.upenn.edu
1	Ritesh Agarwa	MSE	lectronic Materials | Nanostructured Materials...	riteshag@seas.upenn.edu
2	Mark G. Allen	ESE | MEAM	MEMS | Nanofabrication	mallen@seas.upenn.edu
3	Rajeev Alu	CIS	mbedded Systems | Algorithms and Complexity | ...	alur@cis.upenn.edu
4	Paulo E. Arratia	MEAM | CBE	Biomechanics | Fluid Mechanics | Mechanics of ...	parratia@seas.upenn.edu