lecture_9

August 16, 2019

```
In [46]: ### Define our Connector

         import requests,os,time
         def ratelimit(dt):
             "A function that handles the rate of your calls."
             time.sleep(dt) # sleep one second.

         class Connector():
           def __init__(self,logfile,overwrite_log=False,connector_type='requests',session=Fals
             """This Class implements a method for reliable connection to the internet and mon
             It handles simple errors due to connection problems, and logs a range of informat

             Keyword arguments:
             logfile -- path to the logfile
             overwrite_log -- bool, defining if logfile should be cleared (rarely the case).
             connector_type -- use the 'requests' module or the 'selenium'. Will have differen
             session -- requests.session object. For defining custom headers and proxies.
             path2selenium -- str, sets the path to the geckodriver needed when using selenium
             n_tries -- int, defines the number of retries the *get* method will try to avoid
             timeout -- int, seconds the get request will wait for the server to respond, agai
             """

             ## Initialization function defining parameters.
             self.n_tries = n_tries # For avoiding triviel error e.g. connection errors, this
             self.timeout = timeout # Defining the maximum time to wait for a server to respon
             self.waiting_time = waiting_time # define simple rate_limit parameter.
             ## not implemented here, if you use selenium.
             if connector_type=='selenium':
               assert path2selenium!='', "You need to specify the path to you geckodriver if yo
               from selenium import webdriver
               ## HIN download the latest geckodriver here: https://github.com/mozilla/geckodr

               assert os.path.isfile(path2selenium),'You need to insert a valid path2selenium
               self.browser = webdriver.Firefox(executable_path=path2selenium) # start the bro

             self.connector_type = connector_type # set the connector_type

             if session: # set the custom session
```

```
      self.session = session
    else:
      self.session = requests.session()
    self.logfilename = logfile # set the logfile path
    ## define header for the logfile
    header = ['id','project','connector_type','t', 'delta_t', 'url', 'redirect_url','
    if os.path.isfile(logfile):
      if overwrite_log==True:
        self.log = open(logfile,'w')
        self.log.write(';'.join(header))
      else:
        self.log = open(logfile,'a')
    else:
      self.log = open(logfile,'w')
      self.log.write(';'.join(header))
    ## load log
    with open(logfile,'r') as f: # open file

      l = f.read().split('\n') # read and split file by newlines.
      ## set id
      if len(l)<=1:
        self.id = 0
      else:
        self.id = int(l[-1][0])+1

  def get(self,url,project_name):
    """Method for connector reliably to the internet, with multiple tries and simple 
    Input url and the project name for the log (i.e. is it part of mapping the domain

    Keyword arguments:
    url -- str, url
    project_name -- str, Name used for analyzing the log. Use case could be the 'Mapp
    """

    project_name = project_name.replace(';','-') # make sure the default csv seperato
    if self.connector_type=='requests': # Determine connector method.
      for _ in range(self.n_tries): # for loop defining number of retries with the re
        ratelimit(self.waiting_time)
        t = time.time()
        try: # error handling
          response = self.session.get(url,timeout = self.timeout) # make get call

          err = '' # define python error variable as empty assumming success.
          success = True # define success variable
          redirect_url = response.url # log current url, after potential redirects
          dt = t - time.time() # define delta-time waiting for the server and downloa
          size = len(response.text) # define variable for size of html content of the
          response_code = response.status_code # log status code.
          ## log...
          call_id = self.id # get current unique identifier for the call
          self.id+=1 # increment call id
          #['id','project_name','connector_type','t', 'delta_t', 'url', 'redirect_url
          row = [call_id,project_name,self.connector_type,t,dt,url,redirect_url,size,
          self.log.write('\n'+';'.join(map(str,row))) # write log.
          self.log.flush()
          return response,call_id # return response and unique identifier.

        except Exception as e: # define error condition
          err = str(e) # python error
          response_code = '' # blank response code
          success = False # call success = False
          size = 0 # content is empty.
          redirect_url = '' # redirect url empty
          dt = t - time.time() # define delta t

          ## log...
          call_id = self.id # define unique identifier
          self.id+=1 # increment call_id

          row = [call_id,project_name,self.connector_type,t,dt,url,redirect_url,size,
          self.log.write('\n'+';'.join(map(str,row))) # write row to log.
          self.log.flush()
    else:
      t = time.time()
      ratelimit(self.waiting_time)
      self.browser.get(url) # use selenium get method
      ## log
      call_id = self.id # define unique identifier for the call.
      self.id+=1 # increment the call_id
      err = '' # blank error message
      success = '' # success blank
      redirect_url = self.browser.current_url # redirect url.
      dt = t - time.time() # get time for get method ... NOTE: not necessarily the co
      size = len(self.browser.page_source) # get size of content ... NOTE: not necess
      response_code = '' # empty response code.
      row = [call_id,project_name,self.connector_type,t,dt,url,redirect_url,size,respo
      self.log.write('\n'+';'.join(map(str,row))) # write row to log file.
      self.log.flush()
    # Using selenium it will not return a response object, instead you should call th
    ## connector.browser.page_source will give you the html.
      return None,call_id
```

1 Session 8

1.1 Scraping 2 - Parsing

Snorre Ralund

Yesterday I gave you some powerful tricks. Tricks that will work when the data is already shipped in a neat format. However this is not the rule. Today we shall learn the art of parsing unstructured text and a more principled and advanced method of parsing HTML.

This will help you build ***custom datasets*** within just a few hours or days work, that would have taken ***months*** to curate and clean manually.

1.2 Agenda

1.2.1 Storing

- for **replicability** and **data quality**
- reading and writing of files.

Parsing and cleaning raw data

HTML * Understanding the basics of HTML syntax. * Traversing and Navigating HTML trees using BeautifulSoup. Examples include: * Extracting Text from HTML, * Extracting Tables, * Parsing of "unknown" structures. ______________________ Raw Text * Learning the of Regular Expressions for extracting patterns in strings. * Very valuable when cleaning and validating data, and for information extraction from raw text.

---

2 Sidestep(1): APIS

- Do not look for the backdoor if there is a front.
  - Use the API if provided.
- Create a developer account.
- Learn how to Authenticate.
  - Read the docs.
- Construct your queries

Examples - Twitter, YouTube, Reddit, Facebook, Github, Stackexchange and many more… - Google Translate, Transcribe and ML APIS

3 Sidestep(2): Interactions and Automated Browsing

Sometimes scraping tasks demand interactions (e.g. login, scrolling, clicking), and a no XHR data can be found easily, so you need the browser to execute the scripts before you can get the data.

Here we use the `Selenium` package in combination with the `geckodriver` - download the latest release here. It allows you to animate a browser. I won't go into detail here, but just wanted to mention it.

Installation (and maintainance of compatability) can be a little tough, but instead of trawling through crazy stackoverflow threads about your Issue, my experience tells me that downloaded the latest release, and installing the latest selenium version is always the cure.

```
In [ ]: from selenium import webdriver
        ## download the latest geckodriver here: https://github.com/mozilla/geckodriver/releas
        path2gecko = 'geckodriver' # define path to your geckodriver
        browser = webdriver.Firefox(executable_path=path2gecko) # start the browser with a pat
        browser.get('https://www.google.com') # opens a webpage using the browser objects get
```

3.1 The HTML Tree

HTML has a Tree structure.
Each node in the tree has: - Children, siblings, parents - descendants. - Ids and attributes

3.2 Important syntax and patterns

```
<p>The p tag indicates a paragraph <p/>
```

```
<b>The b tag makes the text bold, giving us a clue to its importance </b>
```

output: **The b tag makes the text bold, giving us a clue to its importance**

```
The <strong>tag</strong> makes the text bold, giving us a clue to its importance
```

output: The **tag** makes the text bold, giving us a clue to its importance

```
<em>The em tag emphasize the text</em>, giving us a clue to its importance
```

output: *The em tag makes emphasize the text*, giving us a clue to its importance

---

```
<h1>h1</h1><h2>h2</h2><h2>h3</h3><b>Headers give similar clues</b>
```

output:
h1
h2
h3
Headers give similar clues ________________

```
<a href="www.google.com">The a tag creates a hyperlink <a/>
```

output: The a tag creates a hyperlink ____________ Finally you have the terrible and confusing iframe:

```
<iframe src="https://www.google.com"></iframe>
```

3.3 How do we find our way around this tree?

- extracting string patterns using .split and regular expresssions as we did yesterday.
- Specifying paths using css-selectors,xpath syntax.
- A more powerful and principled (+readable) way is to use the `python` module `BeautifulSoup` to parse and traverse the tree.

3.3.1 Selectors

Define a unique path to an element in the HTML tree. - quick but has to be hardcoded and also more likely to break.

```
In [4]: # selenium example
        browser.get('https://www.facebook.com')
        # find login button.

        # define username

        # define password

        # click on the submit button
```

3.4 Parsing HTML with BeautifulSoup

BeautifulSoup makes the html tree navigable. It allows you to: * Search for elements by tag name and/or by attribute. * Iterate through them, go up, sideways or down the tree. * Furthermore it helps you with standard tasks such as extracting raw text from html, which would be a very tedious task if you had to hardcode it using `.split` commands and using your own regular expressions will be unstable.

```
In [4]: # scraping newspaper articles example.
        connector = Connector('log_sds_lecture10.csv')
        url = 'https://www.theguardian.com/us-news/2019/aug/14/taco-eating-contest-death-fresno
        response,call_id = connector.get(url,'test_call')
        html = response.text

In [6]: from bs4 import BeautifulSoup
        soup = BeautifulSoup(html,'lxml') # parse the raw html using BeautifoulSoup

In [7]: # extract hyperlinks
        links = soup.find_all('a') # find all a tags -connoting a hyperlink.
        [link['href'] for link in links if link.has_attr('href')][0:5] # unpack the hyperlink

Out[7]: ['#maincontent',
         'https://www.theguardian.com/international',
         'https://support.theguardian.com/contribute?INTCMP=header_support_contribute&acquisit
         'https://support.theguardian.com/subscribe?INTCMP=header_support_subscribe&acquisition
         'https://support.theguardian.com/contribute?INTCMP=header_support_contribute&acquisit
```

```
In [8]: headline = soup.find('h1') # search for the first headline: h1 tag.
        name = headline['class'][0].strip() # use the class attribute name as column name.
        value = headline.text.strip() # extract text using build in method.
        print(name,':',value)

content__headline : Man dies after taco-eating contest in California
```

```
In [9]: article_text = soup.find('div',{'class':'content__article-body from-content-api js-arti
```

Say we are interested in how articles cite sources to back up their story i.e. their hyperlink behaviour within the article, and we want to see if the media has changed their behaviour.

We know how to search for links. But the cool part is that we can search from anywhere in the HTML tree. This means that once we have located the article content node - as above - we can search from there. This results in hyperlinks used within the article text.

```
In [321]: # find the article_content node
          article_content = soup.find('div',{'class':'content__article-body from-content-api js
          # find citations within the article content.
          citations = article_content.find_all('a')
```

```
In [325]: citation_links = [] # define container to the hyperlinks
          for citation in citations: # iterate through each citation node
              if citation.has_attr('data-link-name'): # check if it has the right attribute
                  if citation['data-link-name'] =='in body link': # and if the value of that a
                      #print(citation['href'])
                      citation_links.append(citation['href']) #  add link to the container
```

3.5 Scraping without hardcoding every element to collect.

3.5.1 Hands-on example: Continuing the Cryptomarket scrape.

When scraping a large and diverse website or even crawling many different, it can be useful to design more generic parsing schemes, were you haven't seen all elements you want to keep before hand. In the following example I demonstrate a simple example of this.

Imagine we wanted data on the Cryptomarkets: * Go to the front page of a Cryptomarket page. Looking in the **>Network Monitor<** we find a XHR file (helping their search function) containing links to Cryptocoin. Now we have the link to each page we want to visit.

Visit this example: http://coinmarketcap.com/currencies/ethereum/

Yesterday we saw how to find the XHR file with the underlying data behind the Chart. Now we want all the metadata displayed.

Using the inspector we find that the information we want is located by a node tagged with `ul` - *"unordered list"* tag - with the defining class attribute *'list-unstyled details-panel-item–links'*. And the information is located in the `li` tag *"list item"*.

First we locate this node using the `find` method.

```
In [216]: url = 'https://coinmarketcap.com/currencies/ethereum/' # define the example url
          response = connector.get(url,'crypto_mapping') #
          soup = BeautifulSoup(response.text,'lxml') # parse the HTML
```

```
In [221]: list_node = soup.find('ul',{'class':'list-unstyled details-panel-item--links'}) # se
```

Now starting from this list_node, we can search for each list item - li node, using the find_all method.

```
In [326]: list_items = list_node.find_all('li') # search for all list elements children of the
```

From this we extract the information. Without having to hardcode all extractions we exploit that each html node has a attribute (‘title’). We can therefore just loop through each node, extracting the title attribute and the text. – ***This furthermore allows us to scrape content we did not know was there.***

We use the title as the key in the dictionary. The value we are interested in two things, either the *hyperlink*, or the the *text* on display.

```
In [249]: d = {} # defining our container
          for list_item in list_items:
              key = list_item.span['title'] # attributes of a node can be fetched with dictiona
              if list_item.a!=None: # check if the node has a hyperlink.
                  value = list_item.a['href'] # list_item.a ==list_item.find('a') returns the
              else:
                  value = list_item.text.strip()
              d[key] = value
          d

Out[249]: {'Announcement': 'https://bitcointalk.org/index.php?topic=428589.0',
           'Chat': 'https://gitter.im/orgs/ethereum/rooms',
           'Explorer': 'https://etherchain.org/',
           'Message Board': 'https://forum.ethereum.org/',
           'Rank': 'Rank 2',
           'Source Code': 'https://github.com/ethereum',
           'Tags': 'Coin\nMineable',
           'Website': 'https://www.ethereum.org/'}

In [280]: d['Source Code'],d['Source Code'].split('/')[-1]

Out[280]: 'ethereum'
```

3.6 Storage

** For quality** - interactive process of improvements. - quality assessment based on the log is easier with the raw html responses stored.

For replicability - documentation and transparancy

```
In [53]: # start by defining a Project name
         project = 'erc_funding'
         import os # generel package for interacting with the system
         ### among other things automate folder creation
         import os
         ## create folder
```