CMU MSP 36602: Web Scraping in Python

H. Seltman, Mon. Jan. 28, 2019

1) Basic scraping in R in Python

a) Example 1: a **text file** (or html as text), e.g., http://www.stat.cmu.edu/~hseltman/scrape1.txt

i) Python 3 (scrape1.py):

```
>>> from urllib import request
>>> site = 'http://www.stat.cmu.edu/~hseltman/'
>>> page1 = 'scrape1.txt'
>>> response = request.urlopen(site + page1)
>>> response.geturl()
'http://www.stat.cmu.edu/~hseltman/scrape1.txt'
>>> response.getcode()
200
>>> txt = response.read()
>> txt
b'Congratulations!\nYou have scraped a text file from the web.\n'
>>> [x.decode("utf-8") for x in txt.splitlines()]
['Congratulations!', 'You have scraped a text file from the web.']
>>> response.closed
False
>>> response.close()
>>> response.closed
True
```

b) Example 2: a **csv file**, e.g., http://www.stat.cmu.edu/~hseltman/scrape2.csv

i) Python 3 (scrape2.py):

```
>>> from urllib import request
>>> import numpy as np
>>> import pandas as pd
>>> site = 'http://www.stat.cmu.edu/~hseltman/'
>>> page2 = 'scrape2.csv'
>>> csv = request.urlopen(site + page2).read()
>>> len(csv)
106
>>> csv
b'id, message, punctuation\n1, Congratulations, exclamation
point\n2, You have scraped, NA\n3, the web, period\n'
>>> lst = [[y.strip() for y in x.split(sep=",")]
...         for x in csv.decode('utf8').splitlines()]
>>> dtf = pd.DataFrame(lst[1:], columns=lst[0])
>>> dtf['id'] = dtf['id'].astype('int64')
>>> dtf.replace('NA', np.nan, inplace=True)
>>> dtf
   id           message        punctuation
0   1   Congratulations  exclamation point
1   2  You have scraped               NaN
2   3           the web            period
```

c) Example 3: a **json file** (just JavaScript Object Notation: a format, not a web protocol)

i) Python 3 (stations.py):

```
>>> import requests
>>> url = "https://feeds.citibikenyc.com/stations/stations.json"
>>> bikes = requests.get(url).json()
>>> type(bikes)
<class 'dict'>
>>> bikes.keys()
dict_keys(['executionTime', 'stationBeanList'])
>>> type(bikes['executionTime'])
<class 'str'>
>>> len(bikes['executionTime'])
22
>>> bikes['executionTime']
'2019-01-25 02:16:32 PM'

>>> type(bikes['stationBeanList'])
<class 'list'>
>>> len(bikes['stationBeanList'])
811
>>> bikes['stationBeanList'][0]
{'id': 304, 'stationName': 'Broadway & Battery Pl',
'availableDocks': 14, 'totalDocks': 33, 'latitude': 40.70463334,
'longitude': -74.01361706, 'statusValue': 'In Service', 'statusKey':
1, 'availableBikes': 14, 'stAddress1': 'Broadway & Battery Pl',
'stAddress2': '', 'city': '', 'postalCode': '', 'location': '',
'altitude': '', 'testStation': False,
'lastCommunicationTime': '2019-01-25 02:13:29 PM', 'landMark': ''}
>>> bikes['stationBeanList'][1]
{'id': 359, 'stationName': 'E 47 St & Park Ave', 'availableDocks':
32, 'totalDocks': 64, ...
>>> bikes = pd.DataFrame(bikes['stationBeanList'])
>>> bikes.shape
(811, 18)
>>> bikes.describe()
       availableBikes  availableDocks           id    latitude   longitude
count      811.000000      811.000000   811.000000  811.000000  811.000000
mean        12.295931       17.641184  2373.699137   40.733150  -73.976025
std         10.381547       12.065489  1418.079900    0.039101    0.030579
min          0.000000        0.000000    72.000000   40.655400  -74.083639
25%          4.000000        8.000000   468.500000   40.702073  -73.993716
50%         10.000000       17.000000  3193.000000   40.730473  -73.976323
75%         18.000000       25.000000  3461.500000   40.763257  -73.953684
max         61.000000       70.000000  3726.000000   40.814394  -73.907744

            statusKey  totalDocks
count      811.000000  811.000000
mean         1.014797   30.992602
std          0.171495   10.725875
min          1.000000    0.000000
25%          1.000000   23.000000
50%          1.000000   29.000000
75%          1.000000   38.000000
max          3.000000   79.000000
```

2) Intermediate scraping in Python: Using Beautiful Soup

a) soup.py:

```
>>> import numpy as np
>>> import pandas as pd

>>> import urllib.request
>>> import bs4  # Beautiful Soup
>>> import requests  # for get() which has .json() method
#                      and for post()

>>> page3 = 'scrape3.html'
>>> with urllib.request.urlopen(site + page3) as req:
...     soup = bs4.BeautifulSoup(req, "lxml")

>>> print(soup.prettify())
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<!-- Anything offset like this is a comment -->
<!-- The top line is not html; it tells the browser what version -->
<!-- of html you are using. -->
<!-- The real web page starts here -->
<html>
 <head>
...
</body>
</html>

>>> ols = soup.find_all("ol")
>>> type(ols)
<class 'bs4.element.ResultSet'>
>>> type(soup.find("ol"))
<class 'bs4.element.Tag'>
>>> len(ols)
2
>>> ols
[<ol>
<li> milk
      </li><li> eggs
      </li><li> bread
    </li></ol>, <ol>
<li> niúnǎi
      </li><li> jīdàn
      </li><li> miànbāo
    </li></ol>]
>>> type(ols[0])
<class 'bs4.element.Tag'>
>>> ols[0]
<ol>
<li> milk
      </li><li> eggs
      </li><li> bread
    </li></ol>
>>> [type(e) for e in ols[0]]
[<class 'bs4.element.NavigableString'>, <class 'bs4.element.Tag'>,
<class 'bs4.element.Tag'>, <class 'bs4.element.Tag'>]
```

```
>>> lis = ols[0].find_all("li")
>>> len(lis)
3
>>> lis
[<li> milk
      </li>, <li> eggs
      </li>, <li> bread
    </li>]
```

Explore the page

```
>>> def taste(Tag):
...     """ Examine the contents of a BeautifulSoup bs4.element.Tag """
...     if Tag is None:
...         raise(TypeError("'Tag' is None"))
...     if not isinstance(Tag, bs4.element.Tag):
...         raise(TypeError("'Tag' is a '" + str(type(Tag)) +
...                         "rather than a 'bs4.element.Tag'"))
...     def show(e):
...         if isinstance(e, bs4.element.Tag):
...             return e.name + " Tag"
...         elif isinstance(e, bs4.element.NavigableString):
...             return e.string
...         else:
...             return str(type(e)) + " type"
...     return [show(e) for e in Tag]
...
>>> taste(soup)
['HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"', ' Anything
offset like this is a comment ', ' The top line is not html; it tells
the browser what version ', ' of html
you are using. ', ' The real web page starts here ', 'html Tag', '\n']
>>> taste(soup.head)
['\n', ' The next line tells the browser what encoding to use ', '\n',
'meta Tag', '\n', 'title Tag', '\n']
>>> taste(soup.body)
['\n', 'h2 Tag', '\n', 'p Tag', '\n', 'p Tag', '\n', 'hr Tag', '\n', 'a
Tag', '\n', 'p Tag', '\n', 'table Tag', '\n', 'hr Tag', '\n\n    Text
outside of tags is a bad idea, but works.\n\n    ', 'p Tag', '\n', 'ol
Tag', '\n', 'p Tag', '\n', '
http://pinyin.info/unicode/unicode_test.html#lcnumbered ', '\n', 'ol
Tag', '\n']
>>> taste(soup.body.ol)  # first "ol" in the body
['\n', 'li Tag', 'li Tag', 'li Tag']
>>> lis = soup.body.ol.find_all("li")
>>> len(lis)
3
>>> taste(lis[0])
[' milk\n      ']
>>> taste(lis[1])
[' eggs\n      ']
```

Pull off the English shopping list

```
>>> shop = [x.text for x in lis]
>>> shop = [x.strip().replace("\n", "") for x in shop]
>>> shop
['milk', 'eggs', 'bread']
```

Pull of the table

```
>>> table = soup.body.table
>>> taste(table)
['\n', 'tr Tag', '\n', 'tr Tag', '\n']
>>> table = table.find_all("tr")
>>> table = table.find_all("tr")
>>> taste(table[0])
[' ', 'td Tag', '\n', 'td Tag', '\n', 'td Tag', '\n']
>>> taste(table[1])
[' ', 'td Tag', '\n', 'td Tag', '\n', 'td Tag', '\n']
>>> table = [r.find_all("td") for r in table]
>>> table
[[<td> R1C1 </td>, <td> R1C2 </td>, <td> R1C3 </td>], [<td> R2C1 </td>,
<td> R2C2 </td>, <td> R2C3 </td>]]table = [[v.text for v in row] for
row in table]
>>> table = [[v.text for v in row] for row in table]
>>> table
[[' R1C1 ', ' R1C2 ', ' R1C3 '], [' R2C1 ', ' R2C2 ', ' R2C3 ']]
>>> dtf = pd.DataFrame(table)
>>> dtf
        0       1       2
0   R1C1    R1C2    R1C3
1   R2C1    R2C2    R2C3
```

Simulate a form "get"

```
>>> url = "http://www.statsci.org/cgibin/searchwg.pl"
>>> Names = ["Terms"]
>>> Values = ["logistic"]
>>> pairs = [n + "=" + v for (n, v) in zip(Names, Values)]
>>> pairs = "&".join(pairs)

>>> with urllib.request.urlopen(url + "?" + pairs) as html:
...     soup = bs4.BeautifulSoup(html, "lxml")

>>> taste(soup.body)
['\n', 'p Tag', '\n', 'h1 Tag', '\n', 'hr Tag', '\n', 'p Tag', 'ol
Tag', '\n', 'p Tag', '\n', 'p Tag', '\n', 'center Tag', '\n']
>>> taste(soup.body.ol)
['\n', 'p Tag', 'li Tag', '\n', 'p Tag', 'li Tag', '\n', 'p Tag', 'li
Tag', '\n', 'p Tag', 'li Tag', '\n', 'p Tag', 'li Tag', '\n', 'p Tag',
'li Tag', '\n', 'p Tag', 'li Tag',
'\n', 'p Tag', 'li Tag', '\n', 'p Tag', 'li Tag', '\n', 'p Tag', 'li
Tag', '\n', 'p Tag', 'li Tag', '\n', 'p Tag', 'li Tag', '\n', 'p Tag',
'li Tag', '\n', 'p Tag', 'li Tag', '\n', 'p Tag', 'li Tag', '\n', 'p
Tag', 'li Tag', '\n', 'p Tag', 'li Tag', '\n', 'p Tag', 'li Tag', '\n',
'p Tag', 'li Tag', '\n', 'p Tag', 'li Tag', '\n']
>>> lis = soup.body.ol.find_all("li")
>>> taste(lis[0])
['a Tag', '. Documentation and code for an S-Plus function to compute
predicted values and confidence  intervals for logistic regression. ',
'small Tag']
>>> lis[0]
```

```
<li><a href="/s/predlogi.html"><strong>Predicted values for logistic
regression</strong></a>. Documentation and code for an S-Plus function
to compute predicted values and confidence  intervals for logistic
regression. <small><i>5 K - 31 Dec 2015</i>.</small></li>
>>> links = [x.string for ls in lis for x in ls
...           if isinstance(x, str)]
>>> links = [x[1:].strip() for x in links]
>>> links
['Documentation and code for an S-Plus function to compute predicted
values and confidence  intervals for logistic regression.', 'A Matlab
function for ordinal logistic regression.', 'A Matlab function for
logistic regression.', 'A introduction to the statistical reseearch
area of generalized linear models.', 'List of statistical tools for
Excel.', ...]
```

POST example from http://www.avcodes.co.uk/ (Airport Codes)

```
>>> values = {'iataapt': 'ABZ', 'B1': 'Submit'}
>>> r = requests.post("http://www.avcodes.co.uk/aptcoderes.asp",
...     values)
>>> asoup = bs4.BeautifulSoup(r.text, "lxml")
>>> myT = asoup.html.body.find_all("table")[3]
>>> taste(myT)
['\n', 'tr Tag', '\n', 'tr Tag', '\n', 'tr Tag', '\n', 'tr Tag', '\n',
'tr Tag', '\n', 'tr Tag', '\n']
>>> rows = [e.find_all("td") for e in myT
        if isinstance(e, bs4.element.Tag)]
>>> dat = [[e.text for e in r] for r in rows]
>>> print(dat)
[['\xa0Aberdeen / \xa0Dyce'], ['IATA Code:ABZ', 'ICAO Code:EGPD', 'FAA
Code:\xa0'], ['State / Province / Time Zone:\xa0', 'Country:United
Kingdom'], ['Website URLwww.baa.co.uk/main/airports/aberdeen/', ''],
['Longitude:002 12 01 W', 'Latitude:057 12 15 N', '\xa0Google Airport
Map'], ['Remarks: ']]
```

b) Bigger examples and/or homework: AirportCodes.py and WeatherScrape.py and HTMLTableParser.py

3) Advanced scraping

i) Some pages use hidden data to limit usage and/or keep "session" information. If this shows up as hidden form fields, you can include those fields in your POST, and things will work normally. Remember that these will usually change from session to session, so you need to scrape the hidden form text each time.

ii) If you get a "Forbidden" message, the web site may be avoiding scraping by block requests that appear to come from Python (etc.) via examination of the "User Agent". If you choose so, you can change the default User Agent to the one that would be seen if you accessed the page directly from your web browser (https://www.whoishostingthis.com/tools/user-agent/).

iii) Scrapers may also use proxy sites to get around limitations on the number of requests per user.

iv) If you are looking at the data you want on a web page, but cannot see the data in a Page View, then the web page is using advanced techniques (some form of Javascript), and some more complex scraping is needed.