Web Scraping and RESTful APIs

Jamie Saxon

Introduction to Programming for Public Policy

November 2, 2016

A Spectrum

- (No data or senseless tabulations.)
- Unformatted data and locked down websites...
- Tables on nice, clean webpages.
- Slightly-hidden APIs, needing a bit of scraping.
- Documented APIs, ready for consumption.

Examples of Different Resources

Qualities and techniques best demonstrated by example:

1. Google has many straightforward APIs for **mapping**.
2. Twitter famously provides one of the most-comfortable, featureful, compulsively perfect **APIs** ever (**example**).
3. Census provides both a 'consumer-level' **site** and **an API** for retrieving well-formatted data at any level.
4. Bureau of Labor Statistics provides a **respectable API** for time series data whose **coding** is quite abstruse (**example**).
5. The **Virginia Department of Elections** provides a clean (but hidden) interface for retrieving data, but no clean way to *query it*.
6. Lots of Wikipedia **articles** have "pretty clean" tables.
7. Illinois State Board of Education **Report Card** provides a lot of data, but back-breaking methods for accessing it.
8. Some websites go to great lengths to keep you out: **India Water** or **google trends** [**ex**]. (It is *their data*, after all...)

Five Tools

1. **requests** library: retrieve web resources in python.
    - Provides methods for authentication, `POST`ing data, etc.
    - Basically, `curl` for python.
2. **selenium** has similar functionality, but completes javascript loads.
3. **pandas.read_html**: for well-formatted tables, pandas does great.
4. **beautifulsoup** library: provides mechanisms for 'quickly' accessing and extracting elements of web pages in python.
5. curl – our old friend!

Scraping

What is Scraping?

- Look at the actual html and the individual requests, using the developpers tools in your browser.
- **requests**: Identify patterns in HTML and URLs that allow you to download the appropriate resources.
- **BeautifulSoup**: If necessary, extract data from those resources.
- Necessarily a one-off for each site, and often for each part of a site.

Downloading Data: Requests

- Download data directly – for example, our **chickens** page.

```
#!/usr/bin/env python

import requests

addr = "https://harris-ippp.github.io/lectures/"

resp = requests.get(addr) # this is it!!
# pt = requests.put('addr', data = {'key':'value'})
# ... and options, delete, etc.

s = resp.status_code
t = resp.text
# j = resp.json()
print(s, t)
```

Extracting Data: Beautiful Soup 4

Beauiful Soup: full documentation **here**.

- If you can identify the objects, `find()` and `find_all()` are usually the fastest accessors.
  - These yield the first, and all instances, respectively.
- Consider this example, from our website:

```
from bs4 import BeautifulSoup as bs

addr = "https://harris-ippp.github.io/lectures/"
resp = requests.get(addr)
html = resp.content
soup = bs(html, "html.parser")
```

- Find all the rows in the table:

```
soup.find_all("tr")
```

BS4: Parts of Elements

Accessing parts of elements get/dictionary:

- `soup.find("img").get("src")`
- `soup.find("ul").get("class")`
- `soup.find("em").contents`
- `soup.find("ul")["class"]`
- `soup.find("a")["href"]`

BS4: Children and Contents

- Select all of the ‘true’ conditions of chicken health benefits.
- The problem is that the truth is in a different element from the item.
- We need to to look at rows in their entirety, printing the first column if the second is true.
- `children()` provides an iterable, and `.contents` provides a list.

Solution

For “Trivial” Tables: Pandas

Pandas will extract tables from a page, as a list of DataFrames.

```python
import pandas as pd

pd.read_html("http://harris-ippp.github.io/lectures")

wiki =  "https://en.wikipedia.org/wiki/"
wiki += "List_of_colleges_and_universities_"
wiki += "in_the_United_States_by_endowment"
pd.read_html(wiki)
```

Beautiful Soup

- For more-difficult sites, there is an ‘art’ to scanning the raw html/resources and finding the tag you want.
- Take a **school** from the **Illinois State Report Card** (which we used last week): 560 schools in the district, not doing it by hand.
  - Check the html to see if you can find the address? No!?
  - Chrome: View → Developper → Developper Tools → Elements.
  - Firefox: Tools → Developper → Toggle Tools → Inspector.
  - Find the element of interest... find its source.
- Most complex webpages load content from many different sources; they may not all be rendered as part of the base URL.
  - If watching for multiple components, watch “Network,” also in Developper Tools.
- Best case: find it in the webpage or wait for it to load.

Beautiful Soup: “Real” Example

- Please grab the url from this **link**.
- Go to View/Tools → Source, and look for the address.

```python
import requests
from bs4 import BeautifulSoup

sch_ad = "http://iircapps.niu.edu/Apps_3_0/"
sch_ad += "/en/School/150162990250849/Profile"

resp = requests.get(sch_ad)
soup = BeautifulSoup(resp.content, "html.parser")

street = soup.find("span", "street").contents[0]
```

Scraping with a WebDriver: Selenium

- On **many** web pages, you must wait for elements of a page to load from other sources. These won't be in the html.
- Browsers aren't scalable, so use a python a web driver!

```
conda install -c conda-forge selenium
conda install phantomjs
```

```
from selenium import webdriver

addr = "http://illinoisreportcard.com/District.aspx?"
addr += "source=schoolsindistrict&Districtid=150162990

driver = webdriver.PhantomJS()
driver.get(addr)
driver.page_source # like requests.get(...).text
```

Reducing the Pain:

RESTful APIs

Representational State Transfer: Doing it the ‘Right’ Way.

REST in Practice

- Application Programming Interfaces (APIs) are exposed resources with documented parameters for returning formatted data.
- Usually: `GET` requests that return json, csv, or xml data.
  - Variables follow a ?, are specified by =, and separated by &.

APIs: Fancy-Coded URLs

- You can then access these resources with `requests`, `curl`, etc.

RESTful Principles*

Roy Fielding standardized good principles for HTTP 1.1 and RESTful services. REST is a consistent style of organizing resources. The philosophy is that:

- Client and server are stateless and separated (server doesn't 'remember' anything about a session).
- Service is scalable and cachable; this allows for expansion.
- RESTful web services (usually) use HTTP methods as meaningful verbs, and web addresses as functions.
  - `GET` is a pure retrieval and `POST` corresponds to a send.
  - `DELETE`, `PUT`, etc. may also be defined.

*The most-readable resource on this I have found is **here**.

Census API

- Typical, excellent, APIs from the **Census**.
- For example, five-year **ACS estimates** as of 2014 (**variables**).

http://api.census.gov/data/2014/acs5/profile?get=DP02_0037PE,NAME&for=state:*

- Returning to last week:

```
import requests, pandas as pd

addr =  "http://api.census.gov/data/2014/acs5/"
addr += "profile?get=DP02_0037PE,NAME&for=state:*"
j = requests.get(addr).json()

df = pd.DataFrame(j[1:], columns = j[0])
print(df)
```

Google Maps API

Well-documented APIs for **geocoding**, **directions**, **distances**, etc.

- Returns a beautiful json response.
- For more than a few calls, requires a (free) API key.

```
api = "https://maps.googleapis.com/maps/api/"

geoc = "geocode/json?address=Harris School"
dire = "directions/json?origin=Philadelphia&" + \
       "destination=Chicago"
dist = "distancematrix/json?origins=Chicago,IL|" + \
       "Tucson,AZ&destinations=Philly"

key  = "&key=YOUR_KEY"

j = requests.get(api + geoc).json()
```

Unexposed APIs

Some websites establish pretty good RESTful interfaces, and make no effort to restrict their use – but don't publicize them either.

- Let's try **healthdata.org**.
- Watch the Network as we navigate the visualization, filter for gbd-compare/api.
- Just need the metadata, which shows up at the **beginning**.

Some websites make you work a little bit harder:

- Google **trends** requires a token, but it's fairly easy to find it. [**ex**]

Some sites make it fairly hard to retrieve their data (changing tokens/authentication).

Scraping Take-Aways

- Enormous variability in ease of scraping and tools required.
- Hope for good APIs: immediate retrieval with requests.
- Well-formatted tables also fine: pandas saves the day.
- “Deep” scraping takes BeautifulSoup, practice, and patience.

Notes and Rants

Twitter API

- Fantastic, clear API: **https://api.twitter.com/1.1/**
- **Many methods**, e.g., show users or tweets by user:

```
/users/show.json?screen_name=iamjohnoliver
/statuses/user_timeline.json?screen_name=iamjohnoliver
```

- Basically: carefully follow each model. Query starts by ? and terms are separated by & (except in `search`).
- Requires access keys, readily generated with a Twitter account.[†]

[†] Follow **these instructions** and checkout **this script** if you want to try this.

Two Notes on APIs

1. A lot of APIs end up with 3rd party python libaries.
   - For two examples, these are tweepy and sunlightlabs/census.
   - Both of them are good! But they obfuscate the original intention, and often are not as well documented as the original site.
   - I usually find it easier just to understand the API.

2. Many cities/states use Socrata or CKAN (open source/data.gov). Socrata comes with the SODA API, for many (all?) datasets (e.g., **Chicago Divvy**).

A Soapbox on Standards

- Tremendous range in how hard you have to work to get data.
- Lots of jurisdictions and agencies are touting their open data efforts. But very often, the released data sets are awful.
- Even when they're very good (e.g., city crime, education) they may not be standardized across jurisdictions.
- Need standards (schemas) to minimize overlapping efforts.