Speeding up the Housing Search in San Francisco with Data Science

[image]

One of the most helpful pictures I had in finding housing

I was asked a while ago how I was able to find housing in San Francisco. It was one year ago before I found my current sofa-in-a-converted-living-room that I started my search for housing, and it quickly became an overwhelming experience trying to find any kind of rent for less than an arm and a leg. I devoted more and more time to online searching and expanded my search from the Financial District to several regions of San Francisco including the notorious Tenderloins. The search slowly consumed me to the point where I would wake up and spend at least an hour everyday clicking on Craigslist seeing familiar places with a worn out face.

I thought to myself, “there must have been some way to expedite the housing search!” As an aspiring data scientist, my solution was to create a web scraper that could collect the information of several regions of San Francisco and present them in a concise way without all the fluff. While it was no guarantee to find a house, at least I could reduce my 1+ hour searches to 10 minutes.

TLDR: I made a script to scrape Craigslist with locations and pricing as parameters. The code is located at the bottom, but otherwise continue reading to see the entire web-scraping process!

[image 1]

One of the most helpful pictures I had in finding housing

**Step 0: Know your enemy**

[image 2]

What the Craigslist interface looks like

[image 3]

What I wanted the results to look like

Often times, it’s easy to get ahead of oneself and dive right into the problem. One thing that significantly helped my coding skills was to start off with smaller goals and building it up into a big, grand goal. My big goal is to get the headline of each post as well as it’s link to the individual page. I then want to move from one region of San Francisco (say Glen Park) to another one (say Balboa Park). Looking at the Craigslist interface, it appears as though each post has a title and a blue hyperlink that would be useful to show up beside the description.

So breaking down the big goal, my plan of attack is to:

1. Open up a remote controlled page using Selenium
2. Scrape a single page of Craigslist housing to get the titles
3. Scrape it further to get the links to seller’s posts
4. Move to another location without rewriting anything
5. Repeat this process until all locations are scraped
6. Bring it all together to a nice function

**Step 1: Bring in the webscraping tools**

CODE BLOCK:

from selenium import webdriver  
from bs4 import BeautifulSoup  
import urllib2  
import re  
import pandas as pd  
import numpy as np  
pd.set\_option('max\_colwidth', 5000)  
pd.options.display.max\_rows = 999

**Step 2: Open up a webpage with Selenium**

“[Selenium](http://www.seleniumhq.org/" \t "_blank) is a tool that lets us automate actions within a browser as if we were clicking or typing as a human. This lets us login and do anything else a human can do when using the browser. Perhaps most importantly, it allows us to scrape data from webpages that generate HTML using JavaScript executing within the browse” — Professor Terence Parr at the University of San Francisco

To initialize the Selenium tools, we need to call it from the path from which the Chromedriver (or geckodriver) is located. [(Link to download Selenium)](http://www.seleniumhq.org/download/" \t "_blank)

driver = webdriver.Chrome(executable\_path="/Users/shsu/Downloads/chromedriver")

From there, we can use the driver to open up certain webpages, for example:

driver.get('http://google.com')

This should open up a new browser on your Chrome which can then be automated with code. Try it out with any website of your choice!

**Step 3: Scrape post headlines from a page**

[image]

A general query for my housing limitations

As mentioned earlier, the first goal in our grand scheme of things is to scrape a single page of Craigslist. Let’s open up a single page, physically enter in some filters (parameters), and look at the content (to the image on the left). For my budget, I was looking for a place less than $900 and wanted the most recent posts to show up first. When I look at the hyperlink created by this query, I get this:

“[https://sfbay.craigslist.org/search/roo?query=](https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max_price=1000&availabilityMode=0" \t "_blank)**[glen+park](https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max_price=1000&availabilityMode=0" \t "_blank)**[&sort=](https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max_price=1000&availabilityMode=0" \t "_blank)**[date](https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max_price=1000&availabilityMode=0" \t "_blank)**[&max\_price=](https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max_price=1000&availabilityMode=0" \t "_blank)**[900](https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max_price=1000&availabilityMode=0" \t "_blank)**[&availabilityMode=0](https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max_price=1000&availabilityMode=0" \t "_blank)”

What do ya know? It looks like the headers in the url include exactly the parameters I entered, also known as a [query string](https://en.wikipedia.org/wiki/Query_string" \t "_blank). Let’s start with that for our code.

url = [https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max\_price=1000&availabilityMode=0](https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max_price=1000&availabilityMode=0" \t "_blank)driver.get(url)

At this point, you might be wondering why I chose Chromedriver instead of other web drivers. The main reason is because of the developer tools which can be opened with these three keys:

**Option + Command + U**

By clicking those three keys simultaneously, you can view the HTML code behind any webpage.

[image]

Inside the developer source with the tag information we need

By searching for a single title post, we can then extract the tags and headers of that post. In this case, notice the “p” tag with class “result-info” at the top of the image above. Within this tag includes the key text (specifically the “Room for Rent in Lafayette”, “$1100”, and “(lafayetter / orinda / moraga)” that we can scrape with a .text.

craig\_title = []  
all\_posts = driver.find\_elements\_by\_class\_name("result-row")  
for post in all\_posts:  
 craig\_title.append(post.text)

Now we’ve got all the important information from the headlines of posts! The next thing we’ll want to do to scrape this single page is to get not just the title information, but also the webpage that the title leads to.

**Step 4: Scrape the link to the individual post**

Here comes the tricky part: how can I get the link to the individual seller’s page if I can’t click on the link and get it’s hyperlink without redirecting the Selenium browser?

Solution: use our web-scraping tools to look for a title and then generalize it to all posts on a page.

[image]

What we see on the website

[image]

What the code on the website looks like

By searching for the original title post, we can see the “a” tag with class “href” equal to post link that we’re searching for. We’ll first open the url with urllib2 (because we don’t want to remotely open another page with selenium), and then follow a similar process as Step 2 to get the post hyperlink.

link\_list = []  
html\_page = urllib2.urlopen(url)  
soup = BeautifulSoup(html\_page, "lxml")  
for element in soup.findAll("a", {"class": "result-title hdrlnk"}):  
print element['href']

We have now scraped the post titles and their respective links.

**Step 5 and 6: Iterating through the process and generalizing**

We can replace all parameters with variables to generalize our scraping.

For example, turn this:

craig\_title = []  
url = **"https://sfbay.craigslist.org/search/roo?query=glen+park&sort=date&max\_price=1000&availabilityMode=0"**  
driver.get(url)  
allthing = driver.find\_elements\_by\_class\_name("result-row")  
for post in allthing:  
 craig\_title.append(post.text)

Into this:

**place = ['glen+park']  
price = '900'**craig\_title = []  
url = **"[https://sfbay.craigslist.org/search/roo?query=](https://sfbay.craigslist.org/search/roo?query=" \t "_blank)"+ \  
 str(place)+   
 "&sort=date&max\_price="+  
 str(price)+  
 "&availabilityMode=0"**  
driver.get(url)  
all\_posts = driver.find\_elements\_by\_class\_name("result-row")  
for post in all\_posts:  
 craig\_title.append(post.text)

We can repeat this for all the other variables such as while we loop through the pages and append to the lists.

**Step 7: Bring it all together**

We have the pieces to (1) scrape a page’s titles and links, (2) move to another page and repeat step 1, and (3) append information all to a list. So now, all we have to do is create a nice little function that allows us to include a number of arguments and return a data frame. Let’s call it: **craig\_list()**.

from selenium import webdriver  
from bs4 import BeautifulSoup  
import urllib2  
import re  
import pandas as pd  
import numpy as np  
pd.set\_option('max\_colwidth', 5000)  
pd.options.display.max\_rows = 999  
from IPython.display import display, HTMLdriver = webdriver.Chrome(executable\_path="/Users/shsu/Downloads/chromedriver")def craig\_list(price, location\_list=list):  
 """  
 craig\_list is a function that creates the condensed Craigslist post information  
 Inputs:  
 price\_max: integer of the maximum price of houses one wants  
 places: list of strings where the string represents the region of choice   
 Output:  
 a dataframe with the description and the link of the post   
 """  
 craig\_title = []  
 link\_list = []  
 for place in location\_list:  
 print("--------- MOVING TO PLACE: " + str(place) + " -----")  
 link\_list.append(" ")  
 craig\_title.append(str(place).upper())  
   
 url = "[https://sfbay.craigslist.org/search/roo?query=](https://sfbay.craigslist.org/search/roo?query=" \t "_blank)"+ \  
 str(place)+ "&sort=date&max\_price="+ str(price)+"&availabilityMode=0"  
 driver.get(url)  
 all\_posts = driver.find\_elements\_by\_class\_name("result-row")  
 for post in all\_posts:  
 craig\_title.append(post.text)  
   
 html\_page = urllib2.urlopen(url)  
 soup = BeautifulSoup(html\_page, "lxml")  
 for pid in soup.findAll("a", {"class": "result-title hdrlnk"}):  
 link\_list.append(pid['href'])  
 craig\_df = pd.DataFrame(np.column\_stack([craig\_title,link\_list]),   
columns = ["Info", "Link"])  
 return craig\_dfplaces = ["glen+park", "balboa+park"]  
price\_max = 900  
craig\_df = craig\_list(price\_max, places)  
display(craig\_df)

Just like that, you too can now save 40 minutes of your day searching Craigslist for housing to finally rent a sofa in a converted living room!

[image]

Even San Francisco is not safe from satire

So in conclusion, we learned that while scraping appears difficult and time consuming, it can be broken down into easier parts, visualized between the actual user interface and the HTML source behind it, and finally generalized to mimic the entire human behavior. Thanks for reading and good luck on your housing search!

P.S. A lot of posts are spammers that copy real posts and give a generic message to “Please send me your contact number and I will communicate with you as soon as possible.”

References:

1. [Selenium Download Page](http://www.seleniumhq.org/download/" \t "_blank)
2. [Selenium tutorial from Professor Terence Parr](https://github.com/parrt/msan692/blob/master/notes/selenium.md" \t "_blank)
3. [Beautiful Soup Documentation](https://www.crummy.com/software/BeautifulSoup/bs4/doc/" \t "_blank)
4. [Query strings](https://en.wikipedia.org/wiki/Query_string" \t "_blank)
5. [Web scraping with Beautifulsoup tutorial](http://web.stanford.edu/~zlotnick/TextAsData/Web_Scraping_with_Beautiful_Soup.html" \t "_blank)
6. [San Francisco Craigslist](https://sfbay.craigslist.org/" \t "_blank)