Sezgin Isguzar

Sezgin Isguzar

Data Scientist
61 Followers
Follow

Aug 28, 2021
·
6 min read
·
Listen

Scraping Google Reviews with Selenium(Python)

Disclaimer
While web scraping is a powerful tool, it can also lead you into ethical and legal gray areas. To start, it is possible to make hundreds of requests a second to a website. Browsing at superhuman speeds such as this is apt to get noticed. Large volumes of requests such as this are apt to bog down a website's servers and in extreme cases could be considered a denial of service attack. Similarly, any website requiring login may contain information that is thereby not considered public and

scraping said websites could leave you in legal jeopardy.
The purpose of this article is solely educational.

There are different scenarios when a developer may need to work with Google(Map) Reviews. Anyone familiar with Google My Business API, would know that it's necessary to have an *accountId* for each location(business) to obtain reviews through the API. In some cases where a developer needs to work with the reviews of multiple sites (or not having access to a google business account), scraping reviews can be a big help.

Here I'll share an example code to gather google reviews via Selenium and BeautifulSoup.

What is Web Scraping

Web scraping is the process of extracting data from websites. There are several libraries to help you with it, such as:

Beautiful Soup: An excellent tool to parse the DOM, it is simply pulling data out of HTML and XML files

Scrapy: An open-source package for scraping larger datasets at scale

Selenium: Browser automation to help with interacting javascript for complex scraping projects (clicks, scrolls, filling in forms, drag and drop, moving between windows and frames, etc.)

We will use Selenium to navigate the page, upload more content, and then use Beautiful Soup to parse the HTML file.

Selenium

You can install Selenium via pip or conda (package management systems):

```
#Installing with pip
pip install selenium#Installing with conda
conda install -c conda-forge selenium
```

Selenium requires a driver to interface with the chosen browser. I'll be using chrome driver as a personal preference. Here are the links to some of the more popular browser drivers:

Chrome: https://sites.google.com/a/chromium.org/chromedriver/downloads

Firefox: https://github.com/mozilla/geckodriver/releases

Safari: https://webkit.org/blog/6900/webdriver-support-in-safari-10/

BeautifulSoup

We will use BeautifulSoup to parse the HTML page and extract the information we are looking for.(review text, reviewer, date, etc. in our example)

To install BeautifulSoup

```
#Installing with pip
pip install beautifulsoup4#Installing with conda
conda install -c anaconda beautifulsoup4
```

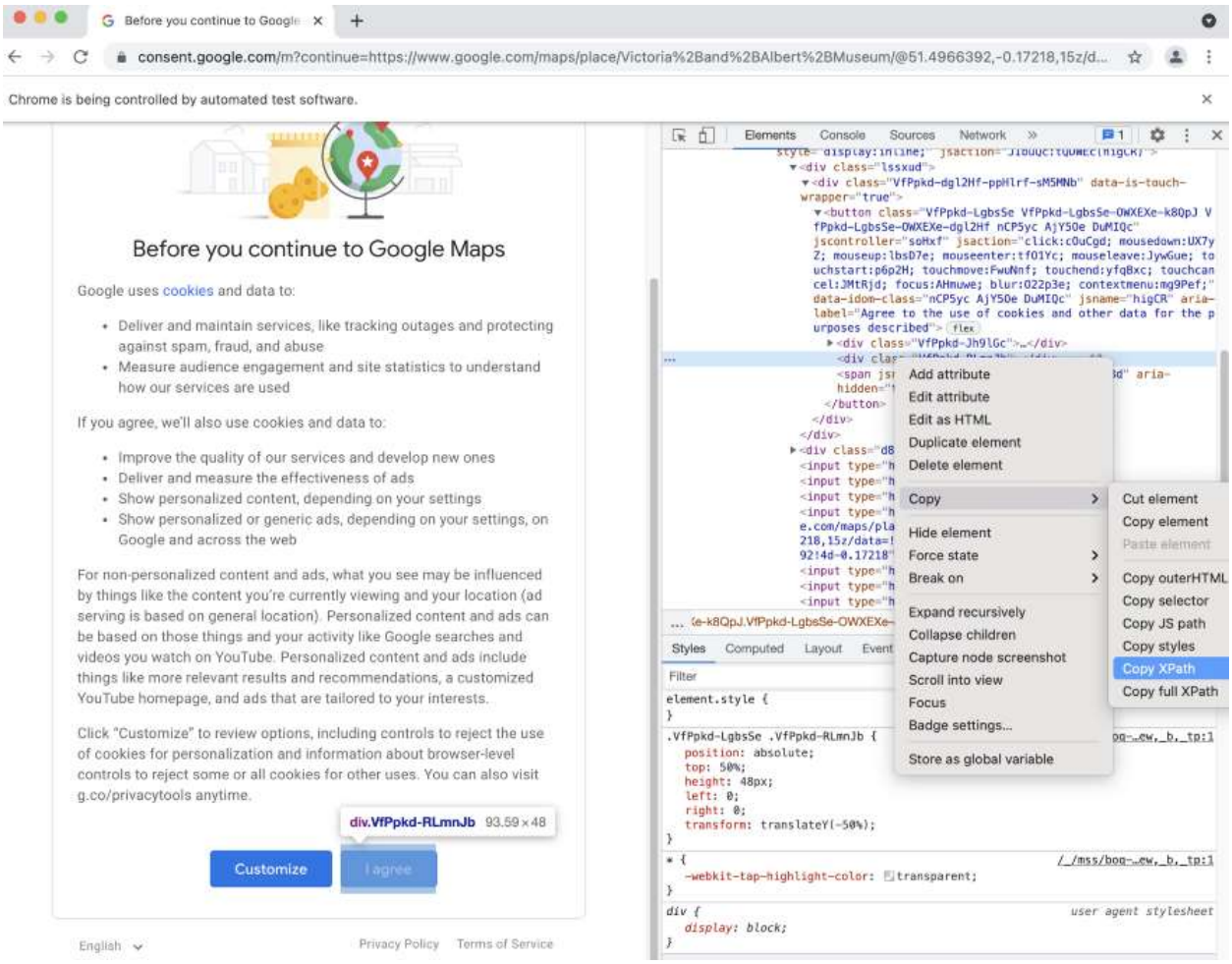
Web Driver Initialisation & URL Input

Initialising the web driver is simple, import and initialize it. Then with the get method, we will pass the Google Maps URL of the location that we want to get the reviews of:

```
from selenium import webdriver
driver = webdriver.Chrome()#London Victoria & Albert Museum URL
url =
'https://www.google.com/maps/place/Victoria+and+Albert+Museum/@51.4
966392,-
0.17218,15z/data=!4m5!3m4!1s0x0:0x9eb7094dfdcd651f!8m2!3d51.4966392
!4d-0.17218'
driver.get(url)
```

Navigating

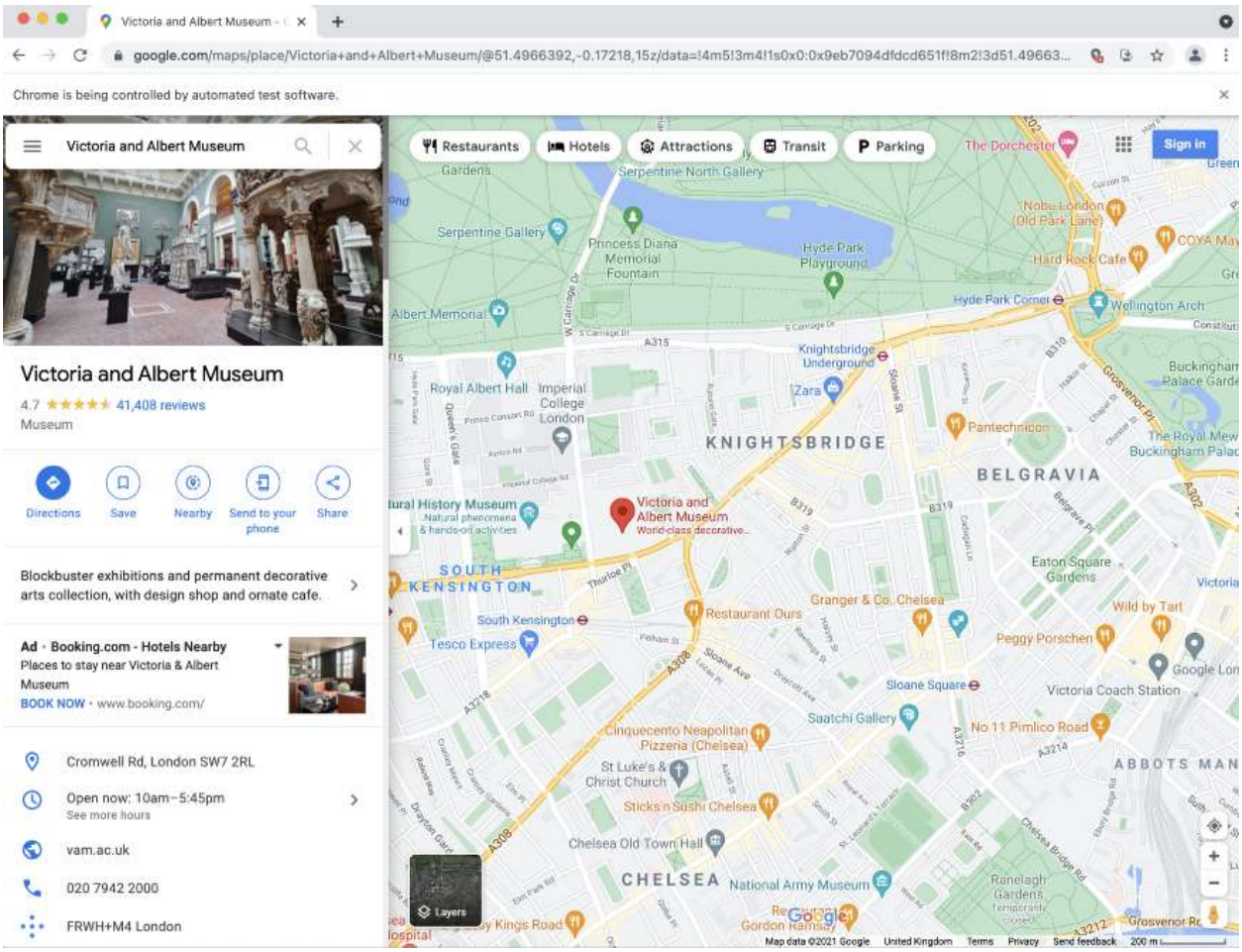
Highly likely web driver will bump into the consent google page to agree with cookies before heading to the actual web address we gave via URL variable. If that's the case, we can click the "I agree" button and continue.



To copy the XPath, click inspect anywhere on the page, and once the source code frame comes on the right side, click inspect again on the "I agree" button. Right-click (or double) on the code and choose Copy XPath from the copy options.

Selenium provides several methods to navigate the page; here I used *find_element_by_xpath():*

```
driver.find_element_by_xpath('//*[@id="yDmH0d"]/c-wiz/div/div/div/div[2]/div[1]/div[4]/form/div[1]/div/button').click()#to make sure content is fully loaded we can use time.sleep() after navigating to each page
import time
time.sleep(3)
```

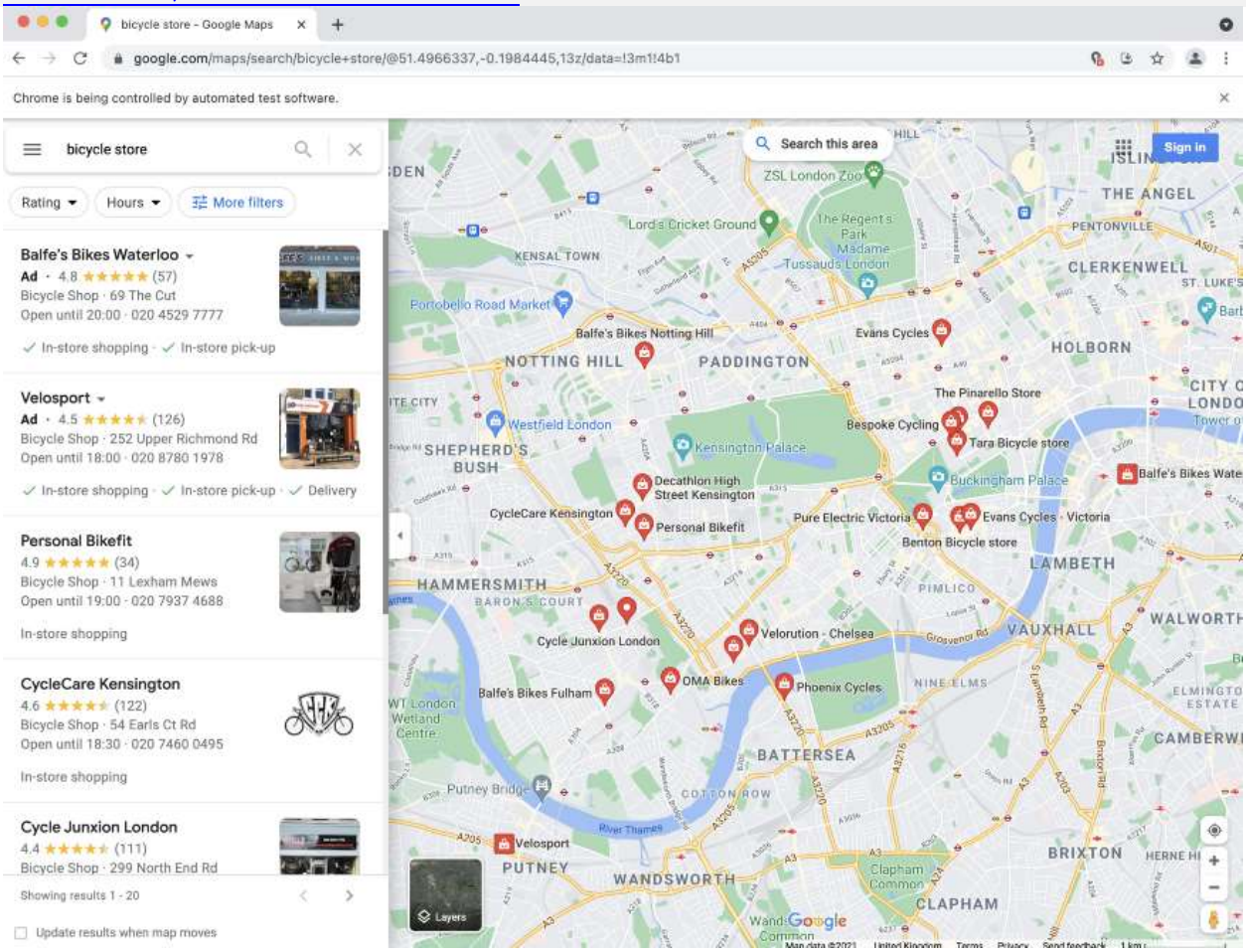


Here we are. Victoria & Albert Museum is one of the best museums in London with more than 40k reviews. It would be a hard job to go through one by one.

Here is a little tricky. In our example, the URL brought us to the exact address we wanted to display to have found and clicked on the 41,408 reviews button to load reviews.

However, there are a few different types of profile pages on Google maps, and on many occasions, the location would be displayed with a bunch of other places listed on the left side, and highly likely there will be some ads displayed on the top of the list. For example following url would take us to this page:

```
url =
'https://www.google.com/maps/search/bicycle+store/@51.5026862,-
0.1430242,13z/data=!3m1!4b1'
```



To not stuck on these different types of layouts or load the wrong page, we will write an exception handler code and navigate to load reviews.

```
try:
    driver.find_element(By.CLASS_NAME, "widget-pane-link").click()
except Exception:
    response = BeautifulSoup(driver.page_source, 'html.parser')
    # Check if there are any paid ads and avoid them
    if response.find_all('span', {'class': 'ARktye-badge'}):
        ad_count = len(response.find_all('span', {'class': 'ARktye-
badge'}))
        li = driver.find_elements(By.CLASS_NAME, "a4gq8e-aVTXAb-
haAclf-jRmmHf-hSRGPd")
        li[ad_count].click()
    else:
        driver.find_element(By.CLASS_NAME, "a4gq8e-aVTXAb-haAclf-
```

```
jRmmHf-hSRGPd").click()
        time.sleep(5)
    driver.find_element(By.CLASS_NAME, "widget-pane-link").click()
```

The code above is following these steps:

1-)try to find the reviews button by class name (driver landed on a page as in the first scenario) and click it
2-)if not parse the HTML with BeautifulSoup check he contents of the page
3-)if response(so the page) includes paid ads, count the ads, skip and navigate to google map profile page for the target location
else navigate to google map profile page for the target location
4-)open reviews (click reviews button — class_name=widget-pane-link)

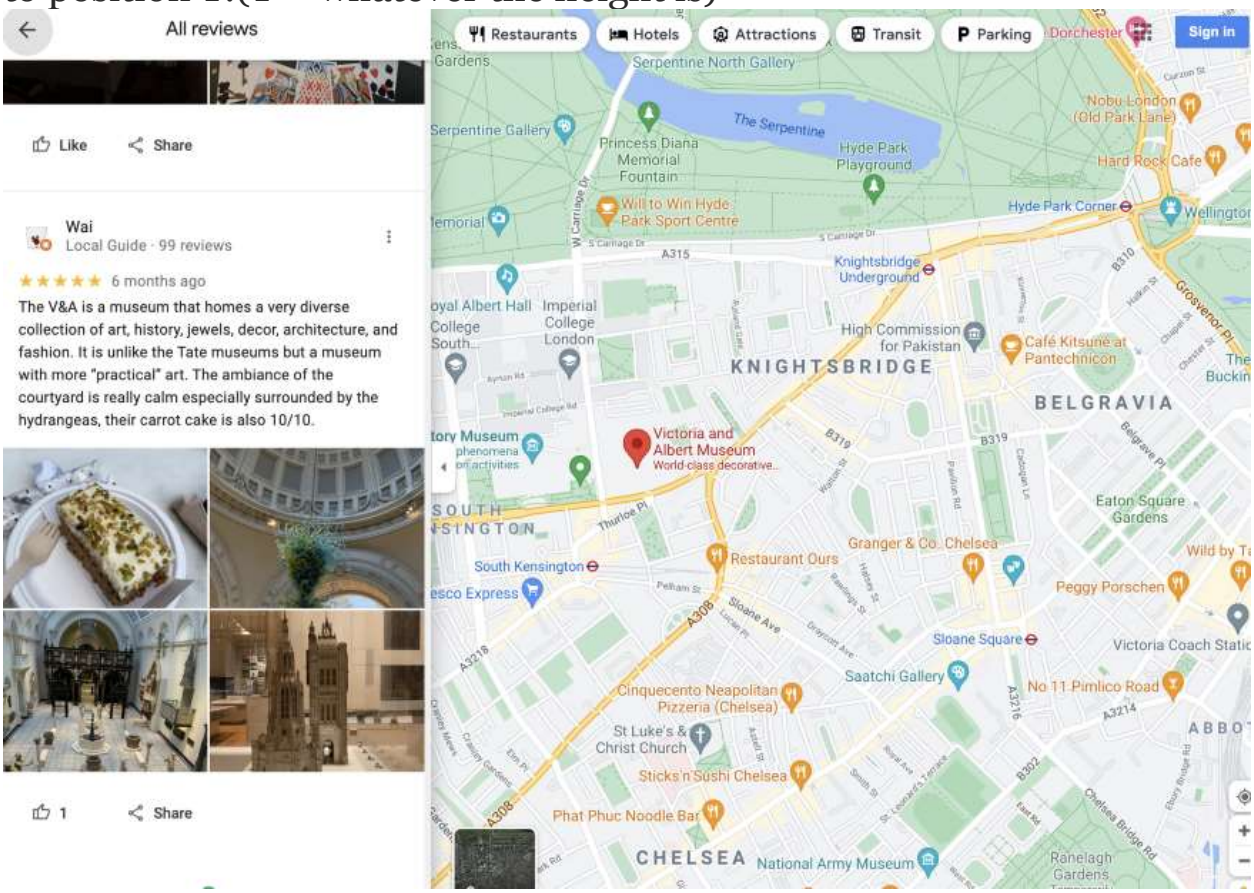
Scrolling and Loading More Reviews

This should take us to the page where the reviews are displayed. Although, the first load would bring only ten reviews and another ten with each scroll. To get all the reviews made for the location, we will calculate how many times we need to scroll and use the *execute_script()* method of the chrome driver.

```
#Find the total number of reviews
total_number_of_reviews =
driver.find_element_by_xpath('//*[@id="pane"]/div/div[1]/div/div/di
v[2]/div[2]/div/div[2]/div[2]').text.split(" ")[0]
total_number_of_reviews =
int(total_number_of_reviews.replace(',','')) if ',' in
total_number_of_reviews else int(total_number_of_reviews)#Find
scroll layout
scrollable_div =
driver.find_element_by_xpath('//*[@id="pane"]/div/div[1]/div/div/di
```

```
v[2]')#Scroll as many times as necessary to load all reviews
for i in range(0,(round(total_number_of_reviews/10 - 1))):
        driver.execute_script('arguments[0].scrollTop =
arguments[0].scrollHeight',
                              scrollable_div)
        time.sleep(1)
```

Inside the for loop above, firstly, we find the scrollable element. Initially, the location of the scroll bar is at the top, which means the vertical location is 0. Through passing a simple JavaScript snippet to the chrome driver(*driver.execute_script*) we set the scroll element's(*scrollable_div*) vertical position(*.scrollTop*) to it's height (*.scrollHeight*). We are pulling the scroll bar from vertically position 0 to position Y.(Y = whatever the height is)



Loading all the reviews with automated software.

Congrats! Our driver successfully scrolled to the bottom of the reviews list and loaded all the review items available. Now, we can parse them and extract the data we are looking for. (Reviewer, Review Text, Review Rate, Review Sentiment, etc.). To do that, simply find the identical class name of the outer item listed inside the scroll layout, which contains all the data about individual reviews, and extract a list of reviews with it.

Parsing HTML and Data Extraction

```
response = BeautifulSoup(driver.page_source, 'html.parser')
reviews = response.find_all('div', class_='ODSEW-ShBeI NIyLF-haAclf
gm2-body-2')
```

Now we can define a function to gather relevant data from the *reviews* result set we obtained by parsing HTML. The function below would take the response set and return a Pandas DataFrame with Review Rate, Review Time and Review Text columns containing relevant extracted data.

```
def get_review_summary(result_set):
    rev_dict = {'Review Rate': [],
        'Review Time': [],
        'Review Text' : []}
    for result in result_set:
        review_rate = result.find('span', class_='ODSEW-ShBeI-
H1e3jb')["aria-label"]
        review_time = result.find('span',class_='ODSEW-ShBeI-
RgZmSc-date').text
        review_text = result.find('span',class_='ODSEW-ShBeI-
text').text
        rev_dict['Review Rate'].append(review_rate)
        rev_dict['Review Time'].append(review_time)
        rev_dict['Review Text'].append(review_text)
    import pandas as pd
    return(pd.DataFrame(rev_dict))
```

This article was a brief introduction and a simple example of web scraping with python. Data sourcing can require applying some complex data gathering methods, and it can be a tedious and expensive job without using the right tools.

Thanks for reading. Feel free to contact me with any questions regarding this article or to have a chat about Data Science.
Sez