Scraping Google Maps reviews in Python

Scraping latest reviews using BeautifulSoup and Selenium

Mattia Gasparini Follow
Apr 23 · 6 min read ★

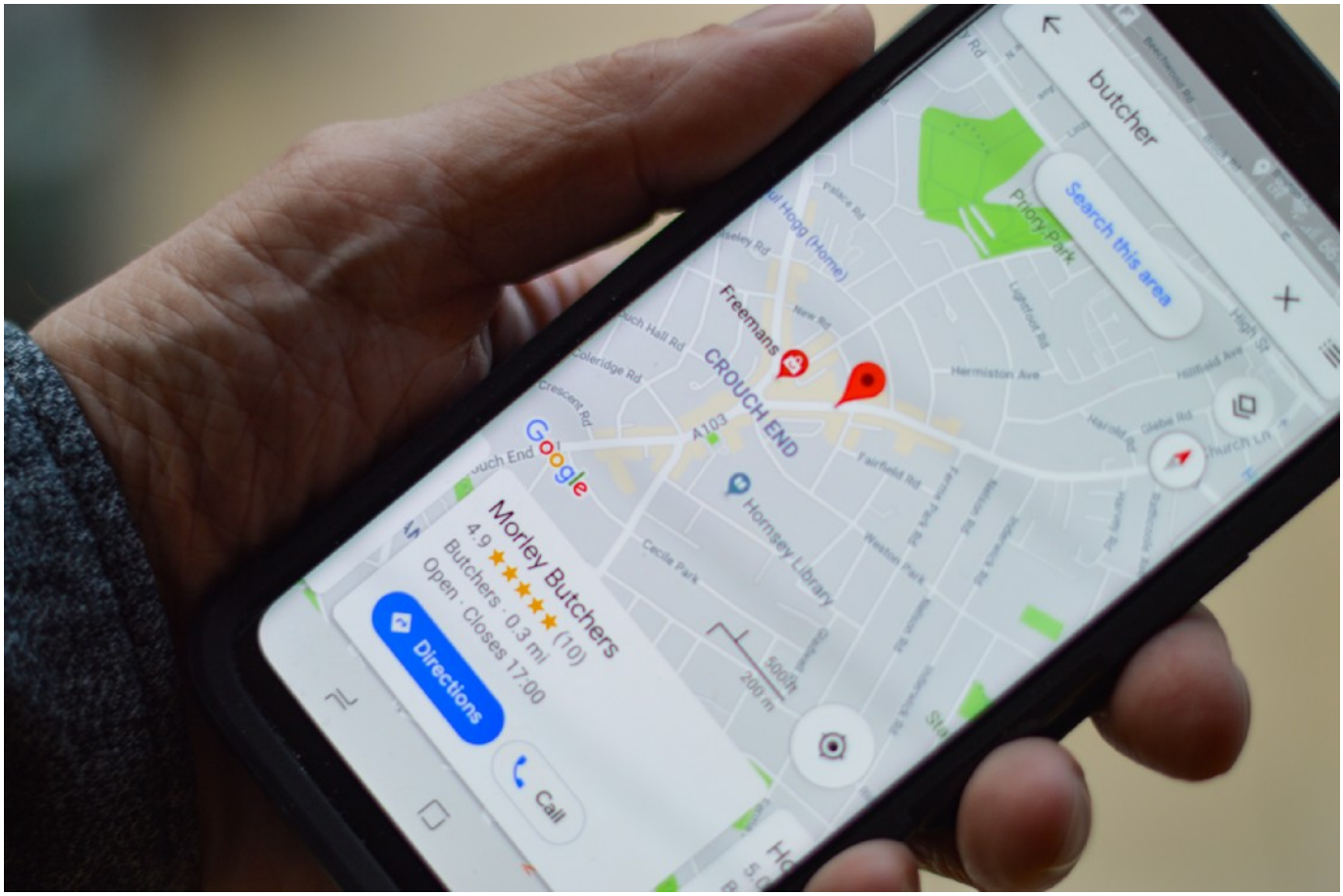


Photo by henry perks on Unsplash

In this article, I would like to share with you some knowledge about how to apply data scraping using **Python Selenium** and **BeautifulSoup** libraries: the combination of these two tools in the correct manner allows to define a set of API to collect data from almost any website.

Note: any data collected from websites can be subject to copyright, thus meaning that should not be reused without owner consent and that should not be definitely used for commercial purpose.

The main objective of this article is to show how to collect data as a coding exercise, as well as a way to build datasets for research and/or personal projects.

After this short disclaimer, shall we begin?

. . .

Let's start from the tools.

Anaconda

Anaconda is a framework that helps to maintain Python library dependencies for each project in a transparent way, using the concept of virtual environments: you create an environment, install each library inside it, and you activate/deactivate the environment based on your needs, without interfering with other projects' dependencies. You can download it here.

This tool is not mandatory, it can be replaced by any virtual environment library (e.g.: virtualenv), but it could be useful if you want to add further steps such as data cleansing, data analysis and machine learning at the end of the pipeline presented here. Also, you could install the described libraries through pip as well.

Anyway, to create our scraping environment, run the following code:

```
conda create --name scraping python=3.6
conda activate scraping
```

Selenium

Selenium is a framework built for web app automatic testing: its API allows us to simulate clicks, scrolls and any other interaction that happens on a website. For this

reason, it is extremely useful for scraping web sites, too: clicks and scrolls trigger changes in the page, that loads more data (or other types of data).

The library is written in Java, Python, C#, JavaScript and many other languages: in this article, we will use the Python implementation to load the target web page and generate all the interactions needed to retrieve additional information.

To install Selenium, run command:

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

We need also to install the *webdriver* of the browser we want to use. The webdriver is the piece of software that automatically runs a browser instance, over which Selenium will work.

I decided to go for **Google Chromedriver**, which you can download from here, but any driver should work fine.

Attention: manual tests we will see later must be run using the browser we choose at this step.

BeautifulSoup

BeautifulSoup is a native Python library that parses HTML and XML files: it helps navigating nodes of the tree, accessing attributes and properties in a very intuitive way.

The main use for us will be to parse the HTML page after being processed by Selenium, extracting information as raw text and sending it to further processing.

To install the library in our conda environment, run command:

```
conda install -c anaconda beautifulsoup4
```

. . .

Well, now we should be all set up to start defining our scraping module!

The target example will show how to collect **the latest Google Maps reviews**: we will define a scraper that navigates to a specific point of interest (POI from now on) and retrieves its associated latest reviews.

1. Initialization

As first step, we need to initialize our webdriver. The driver can be configured with several options, but for now we set only English as browser language:

```
options = Options()
options.add_argument("--lang=en")
driver = webdriver.Chrome(chrome_options=options)
```

2. URL Input

Then, we need to provide our target page: as we want to scrape Google Maps reviews, we choose a POI and get the url that points directly to the reviews. In this step, the driver simply open the page.

Example target page for GM reviews

URL of this type is quite complex: we need to manually copy it from the browser into a variable, and pass it to the driver.

```
url =
https://www.google.it/maps/place/Pantheon/@41.8986108,12.4746842,17z/
data=!3m1!4b1!4m7!3m6!1s0x132f604f678640a9:0xcad165fa2036ce2c!8m2!3d4
1.8986108!4d12.4768729!9m1!1b1

driver.get(url)
```

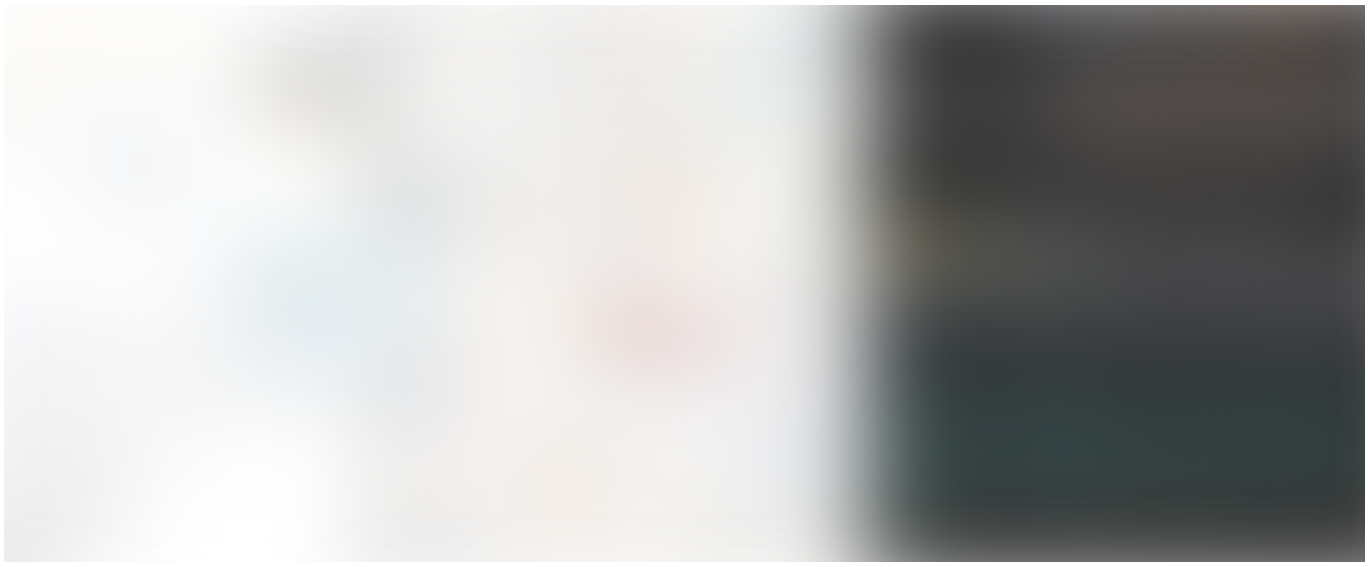
3. Click Menu Buttons

Now, we want to extract *the latest* reviews, while the page default settings presents *the most relevant* ones. We need to click the “Sort” menu and, then, the “Newest” tab.

Here is where Selenium (and coding skills) comes really into play: we need to look for the button, click it, and then click the second button. For this purpose, I used the XPath search methods provided by Selenium: it can be easier than CSS search, thanks also to Chropath, a browser extension that adds an XPath interpreter into browser developer tools.

In this way, we *inspect* the page to test expressions until we highlight the desired elements:

XPath expression test for the first button



XPath expression test for the second button

Unfortunately, this is not enough. Google Maps website (as many other modern websites) is mainly implemented using AJAX: many parts of the site are loaded **asynchronously**, meaning that buttons may not be loaded if Selenium looks for them immediately after loading the page.

But we have a solution in that situation, too. Selenium implements *wait functions*: the click is performed after certain conditions have been verified or after a maximum timeout has passed. Waiting until the element is present and clickable on the page resolves the previous issue.

The code for that part is:

```
wait = WebDriverWait(driver, 10)
menu_bt = wait.until(EC.element_to_be_clickable(
                        (By.XPATH, '//button[@data-value=\'Sort\']'))
                    )
menu_bt.click()
recent_rating_bt = driver.find_elements_by_xpath(
                                        '//div[@role=\'menuitem\']')[1]
recent_rating_bt.click()
time.sleep(5)
```

Sleep function is added at the end of this block because the click triggers an AJAX call to reload reviews, thus we need to wait before moving to next step…

4. Review Data Extraction

And now, finally, we got to the target, **reviews data**. We send the page to BeautifulSoup parser which helps to find the correct HTML tags, divs and properties.

As first, we identify the wrapper div of the review: the find_all method creates a list of div elements that respect specific properties. In our case, the list contains the div of the reviews present on the page.

```
response = BeautifulSoup(driver.page_source, 'html.parser')
rlist = response.find_all('div', class_='section-review-content')
```

For each review, we parse its information: rating stars, text content (if available), date of review, name of the reviewer, id of the review.

```
id_r = r.find('button',
              class_='section-review-action-menu')['data-review-id']
username = r.find('div',
                  class_='section-review-title').find('span').text

try:
    review_text = r.find('span', class_='section-review-text').text
except Exception:
    review_text = None

rating = r.find('span', class_='section-review-stars')['aria-label']
rel_date = r.find('span', class_='section-review-publish-date').text
```

5. Scrolling

Last but not least: the page has already loaded 20 reviews, but the others are not available without scrolling down the page.

Scrolling is not directly implemented in Selenium API, but they allow to execute JavaScript code and apply it to a specific element of the page: we identify the div object that can be scrolled and we run a simple JS line of code to scroll to the bottom of the page.

```
scrollable_div = driver.find_element_by_css_selector(
 'div.section-layout.section-scrollbox.scrollable-y.scrollable-show'
                     )
driver.execute_script(
               'arguments[0].scrollTop = arguments[0].scrollHeight',
                scrollable_div
               )
```

. . .

As you can see, in this brief tutorial I tried to explain the core elements to scrape reviews from Google Maps, but the same concepts are applicable to many modern websites. The complexity is high and I did not consider all the details here: if you want to have a complete example, check my repository on Github here.

Thank you for reading. Let me know what you think about it and if you have any question!

Data Science Scraping Python