Code Manual

Hello! This Code Manual has been created to explain how the scraping AOG code works.

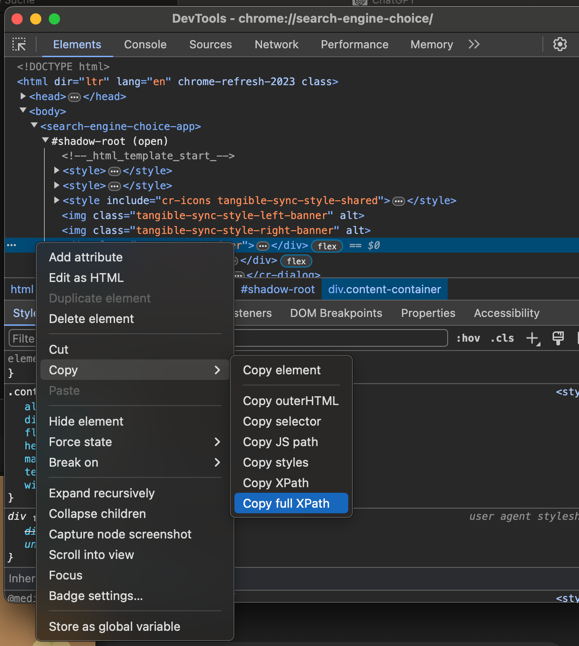
Before we start

This task is well-suited for Python, as it is more efficient for web scraping compared to R. If you don’t have Python set up on your laptop, follow the necessary installation steps. I use Visual Studio Code (VS Code) as my Integrated Development Environment (IDE).

**IDE (Integrated Development Environment)**: An IDE is a software application that provides comprehensive facilities to computer programmers for software development. It typically includes a code editor, debugger, and build automation tools. VS Code is a popular IDE that supports multiple programming languages and offers various extensions to enhance productivity.

**Selenium**: Selenium mimics user interaction with a web browser, allowing you to automate tasks that would otherwise require manual effort. It can operate a computer as if you were using it, but without needing your constant presence.

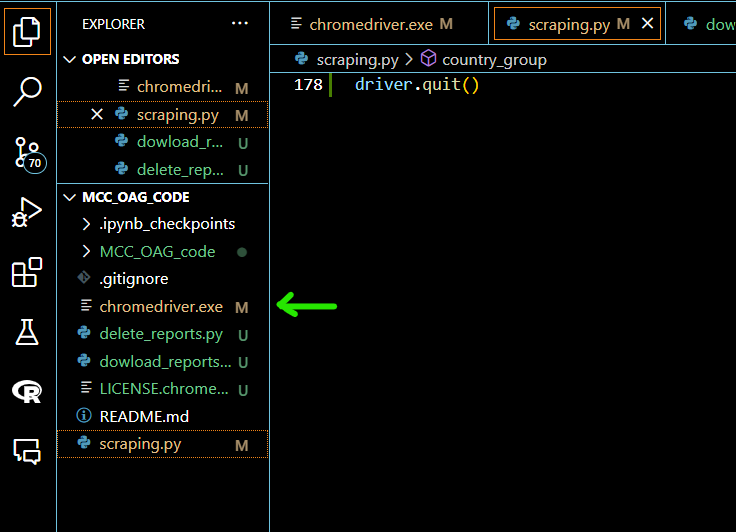
To instruct Selenium on which elements to interact with, you use the browser's inspector tool. By right-clicking on any part of the page and selecting "Inspect," you can view the HTML structure. Right-click on the specific line corresponding to the element you want to interact with and use the XPath or element identifier to connect it with Selenium. The screenshot below shows the result of using the inspector tool in this manner.



**XPath**: XPath (XML Path Language) is used to navigate through elements and attributes in an HTML document. It helps in precisely locating web elements by specifying a path to the element within the Document Object Model (DOM)

For this task, I used Google Chrome because it is the most convenient; it allows easy inspection of elements and is compatible with Chromedriver. Before using the code, there are some prerequisites to set up on your laptop. First, you need to install Chromedriver. This is where I found the link for chromedriver’s latest version: <https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://stackoverflow.com/questions/78496036/chrome-driver-version-125-cant-find-the-chromedriver-exe&ved=2ahUKEwifveTpws6GAxXdzQIHHVd2AK4QrAIoAXoECCYQAg&usg=AOvVaw2vb06WDXBlDVOMe-H6xE6W>. If you google around it should be straightforward to find. Check your current version of google - on May 2024, it should be 125 – and ensure that you download a version of Chromedriver that is compatible with your current version of Google Chrome. This is crucial; at one point, the code failed to run because of a recent update in Chrome, causing a mismatch between Chrome and Chromedriver. You then need to place the .exe version of Chromedriver in the same directory as your Python project files, so the code can access it.

It looks like this in the folders in your VScode – I don’t know how it is for Anaconda.



The project present 3 files: scraping.py, download\_reports.py and delete\_reports.py.

General Notes

* You can use your laptop while Selenium is running on your computer. I would not do things that are too intense, like coding or downloading big memory files, but you can write or go online with no issues.
* When downloading the reports, make sure you are in an environment with a good wi-fi. This way, you can ensure the reports downloads within the pauses, and there are no delays due to poor wi-fi.
* It might happen that a code "jumps" a month sometimes, meaning for example that it creates January and then March, skipping February. Honestly, with the long waiting times between reports that I set, it is highly unlikely this will happen, and it is not the case for the reports that I provided you on Job Bin, but just a word or warning. Be reassured that when the code jumps a month, the reports' names still correspond to the data of the month - for example, a report named April\_2017 will always have information on April 2017.

Creating reports

This section explains how to operate the main file, scraping.py. The code is generously commented out so for technicalities that is a good resource already, this section just provides extra support.

These are the group combinations that are being scraped:

* Group A: **AT** - Austria, **BE** - Belgium, **BG** - Bulgaria, **CY** - Cyprus, **CZ** - Czech Republic, **DE** - Germany, **DK** - Denmark, **EE** - Estonia, **FI** - Finland, **FR** - France, **GR** - Greece, **HR** - Croatia, **HU** - Hungary, **IE** - Ireland, **IT** - Italy
* Group B: **CH** - Switzerland, **ES** - Spain, **GB** - United Kingdom, **IS** - Iceland, **LT** - Lithuania, **LU** - Luxembourg, **LV** - Latvia, **MT** - Malta, **NL** - Netherlands, **NO** - Norway, **PL** - Poland, **PT** - Portugal, **RO** - Romania, **SE** - Sweden, **SI** - Slovenia, **SK** - Slovakia
* Group C: **AL** - Albania, **BA** - Bosnia and Herzegovina, **BY** - Belarus, **DZ** - Algeria, **MA** - Morocco, **ME** - Montenegro, **MK** - North Macedonia, **RS** - Serbia, **TN** - Tunisia, **TR** - Turkey, **UA** - Ukraine

And these are the combinations:

* A to B
* B to A
* A to A
* B to B
* A to C
* C to A
* C to B
* B to C

It is important to know that you can’t start the code once and then it runs, you have to start it again for every combination. The reason for this is that the code is *very*slow, basically creates the reports as fast as a person would be. For context, to create all the reports from group A to group B, the code took around 4-5 hours. Again, in the meantime you can do other things – like writing this code manual ;) – so that time is not lost, but it means that for creating all the combination it would have taken too long. I made it so slow on purpose, adding pauses as long as 1 minute, because this way it is more robust to outliers, meaning reports that take very long to download.

You need to give the reports enough time to "settle". By this I mean, the platform crashes if too many downloads are executed after the other, which explains the many pauses I put in the code.

The code creates csv files because they can be read in R, unlike XLS ones that had words.

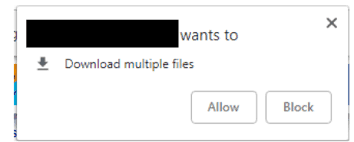
I am not completely sure why but the code does not run for 2024, probably because it is not a complete year. I created the reports differently, with a combination of scraping and manual work.

The explanation on the technicalities is in the script scraping.py.

Downloading Reports

The download\_reports script, surprise surprise, downloads the reports. It was more manageable to create two separate scripts for creating and downloading.

To download multiple files, you must allow it in your chrome settings, or enable it manually while the code runs. This is a screenshot of what the allow button you need to press might look like:



and this is the link to allow for multiple downloads  chrome://settings/content/siteDetails?site=https%3A%2F%[2Fanalytics.oag.com](http://2fanalytics.oag.com/)

Deleting Reports

I created a script delete\_reports.py that cleans the job bin, deleting the reports one after the other. I made it for myself in the testing phase, instead of having to delete each wrong report manually, and figured it might be useful for you to since I created it anyways. I made a basic version; it does not allow you to decide, for example, to clean only until A\_B\_June\_2018, but only deletes one report after the other.  If you want to interrupt the deletion at some point, it is enough to close the automated AOG page.