IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 23, Issue 3, Ser. II (May – June 2021), PP 01-05
www.iosrjournals.org

Web Scraping with Python and Selenium

Sarah Fatima[1], Shaik Luqmaan[2] , Nuha Abdul Rasheed[3]
[1](*Department of Computer Science and Engineering, Muffakham Jah College of Engineering and Technology / Osmania University, India*)
[2](*Department of Computer Science and Engineering, Muffakham Jah College of Engineering and Technology / Osmania University, India*)
[3](*Department of Computer Science and Engineering, Muffakham Jah College of Engineering and Technology / Osmania University, India*)

Abstract:
In this paper, we have designed a method for retrieving web information using selenium and python script. Selenium is used to automate web browser interaction. The block-based structure is obtained by using a python script. Web information is mostly unstructured, the proposed work helps to organize the unstructured data and make it useful for various data analysis techniques. It also focuses on ways in which data can be persisted and used from various websites for which APIs are not available.
Key Word*: Web scraping; Selenium; Information Extraction; HTML Parsing; Web data retrieval.*

Date of Submission: 02-06-2021 Date of Acceptance: 15-06-2021

I. Introduction

Web scraping is a process of information extraction from the world wide web (www), accomplished by writing automated script routines that request data by querying the desired web server and retrieving the data by using different parsing techniques [1]. Scraping helps in transforming unstructured HTML data into various structured data formats like CSV, spreadsheets. As it is known, the nature of web data is changing frequently, using an easy-to-use language like python which accepts dynamic inputs can be highly productive, as code changes are easily done to keep up with the speed of web updates. Using the wide collection of python libraries, such as requests, pandas, csv, webdriver can ease the process of fetching URLs and pulling out information from web pages, building scrapers that can hop from one domain to another, gather information, and store that information for later use. To automate web browser interaction, the single interface open-source tool Selenium is used that can mimic human browsing behaviors. Besides, numpy and pandas are used to process the data [2]. By using this implementation, web data is transformed into structured blocks. The block-based structure is obtained by using a python script with Selenium. The proposed experimental work shows, parsing the HTML code, installation of python and selenium, python scripting and interpretation, and structural extraction of web information. The evolving needs of internet and social media services require various techniques for the extraction of web data. Web information is mostly unstructured, the proposed work helps to organize the unstructured data and make it useful for various data analysis techniques.

Web Scraping and its applications:

Web Scraping is the practice of gathering information automatically from any website using an application that simulates human web-behavior. This is achieved by writing automated scripts that query the web server, request data, and transform the data in various structured formats like CSV, spreadsheets, and JSON. This technique is highly used to persist the data from various websites for which APIs are not available. In practice, web scraping uses a wide variety of programming techniques and technologies, such as data analysis, information retrieval and security, Cyber Security, HTML parsing techniques. Web scraping has various applications across many domains. Some of them are

- For collecting data from a collection of sites that do not have a warranted API. With web scraping, even a small, finite amount of data can be viewed and accessed via a Python script and stored in a database for further processing.
- Analysis of product data from social media platforms like Twitter and e-commerce sites like amazon i.e., big data and sentiment analysis.
- To use the raw extracted data like texts, images to refine machine learning models and to develop datasets.

- Observing customer sentiment by scraping customer feedback and reviews of different businesses as visiting various websites can be cumbersome.

Scraping with Python and Selenium:

Web Scraping is all about dealing with huge amounts of data, Python is one of the most favorable options to handle it, as it has a relatively easy learning curve and has a vast set of libraries and frameworks like NumPy, CSV, Webdriver, etc. Using Python-based web scraping tools such as Selenium has its benefits. Selenium is an automation testing framework for websites that takes control of the browser and mimics actual human behavior using a web-driver package. With the majority of the websites being JavaScript-heavy, Selenium provides an easy way to extract data using Scrapy selectors to grab HTML code.

II. Related Work

Before the evolution of technology, people used to manually copy the data from various websites and paste it in a local file to analyze the data. But in today's day and age with technology increasing at a rapid pace the old method of extracting the data can be overwhelming and time consuming. This is where web scraping can be useful. It can automate the data extraction process and transform the data into desired structured format.

Web Scraping Tools and Techniques

In this section various tools as well as techniques used for web scraping are presented. They have been found through searching the web or having heard about them due to their popularity.

1 Scrapy

Scrapy is an open-source Python framework initially outlined exclusively for web scraping and also supports web crawling and extricate data via APIs. Data extraction can be done using Xpath or CSS which is the built-in way as well as by using external libraries such as BeautifulSoup and xml. It allows for storing data within the cloud.

2 HtmlUnit

HtmlUnit is a typical headless Java browser used for testing web applications. It allows commonly used browser functionality such as following links, filling out forms, etc.

Moreover, it is used for web scraping purposes. The objective is to mimic a "real" browser, and thus HtmlUnit incorporates support for JavaScript, AJAX and usage of cookies 13.

3 Text pattern matching

A basic yet effective approach to extract data from web pages can be based on the UNIX grep command or regular expression - coordinating facilities of programming languages (for example, Perl or Python).

4 HTTP programming

HTTP requests posted to the remote web server using socket programming are used to retrieve static and dynamic web pages.

5 HTML parsing

Data of the same category are typically encoded into similar pages (generated from database) by a common script or template. In data mining, a wrapper is a program that extracts contents from such templates in a particular information source and translates it into a relational form. HTML pages can be parsed, and content can be retrieved using semi structured data query languages like Xquery and the HTQL.

6 DOM parsing

Browsers such as Mozilla browser or Internet Explorer can parse web pages into a DOM tree using which parts of pages can be retrieved by programs. The resulting DOM tree is parsed using languages like Xpath.

Web scraping using BeautifulSoup

BeautifulSoup module is a web scraping framework of Python which is used to query and organize all the parsed HTML into data using different parsers which the module has to offer. BeautifulSoup uses the HTML parser that's native to Python's standard library known as html parser.

To get started, import Beautiful Soup and need to create beautiful soup objects called 'soup'. There are lots of ways to navigate the HTML using BeautifulSoup objects, one of the more common methods is find all methods. Here we can enter a single tag name and it will return either the first or all of the tags of that kind. We could also provide a list of tag names and the method will provide anything matching either tag. Alternatively, we can specify the attributes, such as ID or class and it will return based on that criterion.

III. Proposed Methodology and Implementation

The proposed work focuses on analyzing a web page and extracting required visual blocks which can be lists or unstructured tables and store these datasets in various already available structured formats such as CSV, spreadsheets or SQL databases using respective Python libraries. Selenium web drivers are used to mimic

human behavior and ease the extraction of large data sets and images, we have created one script to perform required scraping.

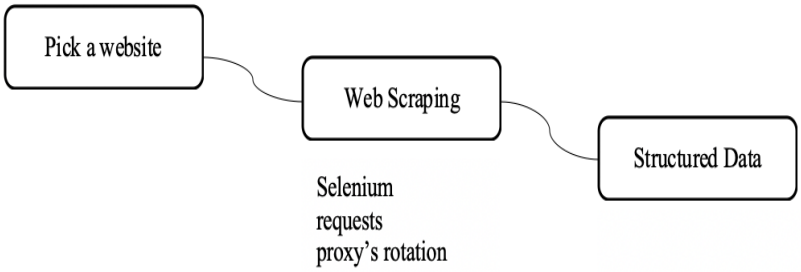

Fig.1. Web Scraping Timeline

Main tools used

1. Python (3.5)
2. *Selenium library*: for handling text extraction from a web page's source code using element id, XPath expressions or CSS selectors.
3. *requests library*: for handling the interaction with the web page (Using HTTP requests).
4. *csv library*: for storing extracted data.
5. *Proxy header rotations*: generating random headers and getting free proxy IPs in order to avoid IP blocks.

Description of work

The research work is developed in Python using HTML parsing running on Anaconda Platform. Script is supported by Selenium library [5]. The site used for scraping instances of unstructured data with and without pagination. Simulation of experimented work:

A. Installation of Python
B. Importing selenium web drivers, requests and csv library
C. Execution of script using Python
D. Persisting the generated structured data in the database.

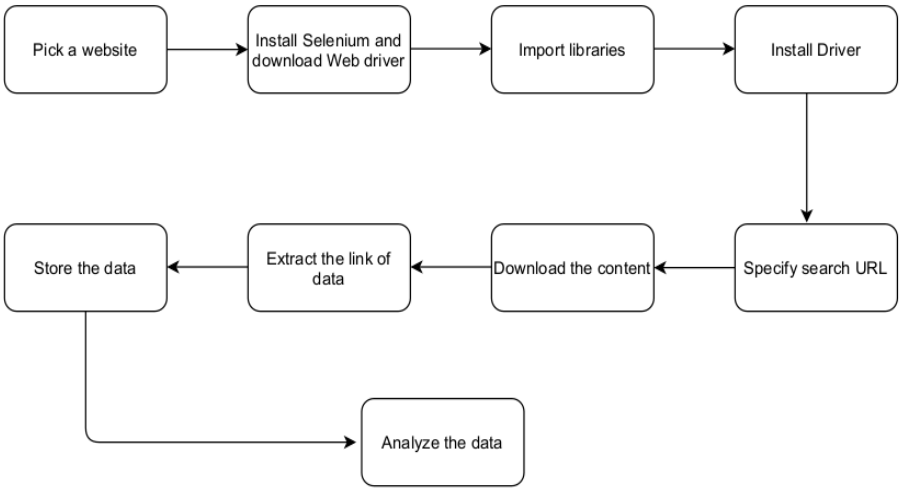

Fig.2. Complete flow of web scraping

When the script is run, an instance of chrome web driver is initiated, and the required output CSV files are initialized. The scraper then parses the data from the aforementioned URL using element id or XPath and starts collecting the data, which then will be written in output files using writer headers. The script is designed to throw errors in case of time out or if the element id or XPath is missing.

```
def __init__(self, keyword, output_filename1, output_filename2, output_filename3):
    self.chrome_options = webdriver.ChromeOptions()
    # self.chrome_options.add_argument('--headless')
    self.chrome_options.add_argument("--no-sandbox")
    self.driver = webdriver.Chrome(executable_path=r'C:\chromedriver_win32\chromedriver.exe'
                                   ,chrome_options=self.chrome_options)
    self.session_requests = requests.session()
    self.keyword = keyword
    self.output_filename1 = output_filename1
    self.output_filename2 = output_filename2
    self.output_filename3 = output_filename3
    self.parse()
```

Fig.3. Initialization of selenium web driver

```
self.driver.find_element_by_id("companyID").clear()
self.driver.find_element_by_id("companyID").send_keys(self.keyword)
```

Fig.4. Extraction of data using element id

```
next_link = self.driver.find_element_by_xpath("//*[@id='charges_next']")
next_link.click()
sleep(5)
```

Fig.5. Extraction of next links using Xpath when paginated

Results from the data files obtained from web scraping can further be used for machine learning and data analytics techniques.

IV. Discussion

When we start web scraping, we will notice how much we value the simple things that browsers perform for us. The web browser is a very handy tool for generating and sending the information packets from our computer, and interpreting the data we receive as text, images, etc. back from the server. The internet includes many types of interesting data sources that serve as a goldmine of interesting stuff. Unfortunately, the current unstructured nature of the web makes it extremely difficult to easily access or export this data. Modern browsers are brilliant at showcasing visuals, displaying motions, and arranging out websites in a pleasing style, but they do not offer a capacitance to export their data, at least not in most situations. So, web scraping, instead of seeing the website page through the interface of your browser, gathers the data from the browser. Now-a-days many websites provide a service called an API, which gives access to the data, but most of the website doesn't provide an API to interact with or doesn't expose an API required for our functionality. In such cases building a web scraper to gather the data can be handy.

Applications of Web Scraping:

- E-Commerce
- Finance
- Research
- Data Science
- Social Media
- Sales

V. Conclusion

Web scraping can be useful to gather different types of data from websites either for business or personal purposes and there are many ways to do it. But it is also necessary to be mindful of the burden that your web scraper is putting on the website and there can be consequences of irresponsible web scraping. Consider running a script throughout the first 100 pages; this would be an aggressive scraper, and we would be placing an unreasonably enormous strain on the website servers, perhaps disrupting their operation and web scraping is a violation of certain websites' terms and conditions in such cases the website is likely to take action against you.

Web Scraping Code of Conduct:

1. Don't unlawfully distribute downloaded materials.
2. Downloading copies of documents that are plainly not public is not permitted.
3. Check if the data which is required is already available.
4. Don't try aggressively scrapping, introduce delays in the scripts.
5. Check state's law before scraping any website.

References

[1]. Anand V. Saurkar, Kedar G. Pathare, Shweta A. Gode, "An overview of web scraping techniques and tools" International Journal on Future Revolution in Computer Science and Communication Engineering, April 2018.
[2]. Jiahao Wu, "Web Scraping using Python: Step by step guide" ResearchGate publications (2019).
[3]. Matthew Russell, "Using python for web scraping," No Starch Press, 2012.
[4]. Ryan Mitchell, "Web scraping with Python," O'Reilly Media, 2015.
[5]. Selenium Library : https://pypi.org/project/selenium/.

Sarah Fatima, et. al. "Web Scraping with Python and Selenium." *IOSR Journal of Computer Engineering (IOSR-JCE)*, 23(3), 2021, pp. 01-05.