



# Web Scrapping Using Selenium

Sourav Singh

Computer Science Engineering  
Panipat Institute of Engineering and Technology  
Haryana, India  
[souravsingh8917@gmail.com](mailto:souravsingh8917@gmail.com)

Anurag Shukla

Computer Science Engineering  
Panipat Institute of Engineering and Technology  
Haryana, India  
[anuragshukla9000@gmail.com](mailto:anuragshukla9000@gmail.com)

Devanshu

Computer Science Engineering  
Panipat Institute of Engineering and Technology  
Haryana, India  
[devanshu8220@gmail.com](mailto:devanshu8220@gmail.com)

Dr Anju Bhandari Gandhi

Professor, Department of Computer Science and Engineering,  
Panipat Institute of Engineering and Technology  
Haryana, India  
[anjugandhi.cse@piet.co.in](mailto:anjugandhi.cse@piet.co.in)

**Abstract—** Web pages are constructed using the text based markup languages like HTML and XML, and often include the wealth of valuable information in the text form. Still, most of websites are designed for being en- users not for simplicity of automated usage. Because of this the tool kits that scrape the web content were made. The web scraper is an Application Programming Interface (API) to extract the information from the web pages.

Web scraping can be done manually by the software, this term typically refers to the automated processes implemented through bot or web crawler. It is a form of repetition in which particular information is collected and copied from the web pages typically into a local database or spreadsheets for analysis or something else.

## I. INTRODUCTION

Web scraping is a process of extracting the information from a webpage or set of webpages. This can be useful for extracting the information from website which is not easily accessible through structured datasets or APIs,

While APIs are intended to be used programmatically, web scraping involves using computers to ingest data that was intended for humans to see and understand. Whereas API data is delivered in machine-readable formats such as XML or JSON, web pages will generally render data in a human-readable format using HTML. Thus, web scraping often involves the ability to extract the structured data from HTML text element.

The exact process of web scraping will depend on the tool you're using, the specific web page, and the desired content. Web scraping can have negative consequences if done unethically, the ability to programmatically request and parse information from websites can have very positive consequences as well.

Web scraping is used for web indexing, online cost change monitoring and price comparison, web mining and data mining, product review scraping, weather data monitoring, web data integration, web mashup, collecting real-estate lists and many more.

We also theorize that having an artificial random delay when scraping and randomizing intervals between each visit to a website would counteract some of the anti-scraping measures.

Another, smaller aspect of our research was the legality and ethicality of scraping. Further thoughts and comments on potential solutions to other issues have also been included.

## II. LITERATURE REVIEW

### Crawling

A common and central component found in scraping systems is crawling. Crawling can be defined as the process of automatically exploring and navigating websites by following hyperlinks.

The component or system responsible for crawling is commonly referred to as a crawler or web crawler. Other common names include web spider or both. Crawling is a broad term that includes many different classes of crawlers for different purposes and the literature lacks a standard classification scheme. Names referring to specific crawler classes are used interchangeably and sometimes contradictorily by different authors. There are however some distinct categories of crawlers.

A deep web crawler starts from a seed URL, identifies hyperlinks in the web page retrieved from the URL, and visits the hyperlinks indiscriminately to retrieve a new web page recursively. The system does not consider relations between the web pages and usually utilizes a breadth-first crawling strategy to retrieve them.

### Dynamic Web Crawling

New trends and technologies in web development has led to a new set of problems for the general crawling system. Traditionally for static web pages a crawling system simply needs to extract hyperlinks from a web document. However, due to the increasing use of dynamically loaded content the traditional crawling method will not be able to load all information required on the first request and there will be a discrepancy between the document fetched by the crawler and the document seen by the user during normal browsing. The source of this discrepancy is the use of technologies for dynamically loading web content such as Asynchronous JavaScript and XML (AJAX). Solving this problem is the subject of much of current research into crawling systems and has led to the development of the dynamic web crawler, also known as event driven web crawler.

AJAX-based web applications and Single Page Application (SPA) architectures are increasingly gaining popularity for developing dynamic websites. These new architectures provide more sophisticated client-side processing of the web document the user interacts with compared to the traditional server-side preprocessed web document. In the case of traditional static websites, the client simply renders the document fetched from the server which is also what is used by the crawling system to extract hyperlinks. In modern dynamic web pages using AJAX methods, the page also includes code that can be executed by the client to send asynchronous requests to the server or change the state of the page viewed by the client. This is usually handled by a JavaScript engine that is able to execute JavaScript. The information fetched by asynchronous calls or the JavaScript code executed by the browser is used to manipulate the state of the website on the fly. The state of the page is represented by a Document Object Model (DOM) – an ordered tree – which is encoded with HTML. By this mechanism dynamic web pages inject more information into the page than is retrieved in the initial request which leads to the crawling system missing potentially vital information or hyperlinks.

## III. WHY SELENIUM?

There are plenty of techniques for web scraping. But selenium is all weather scraping techniques. The problem of other scraper techniques like BeautifulSoup, auto scraper, html parsing etc. they are not feasible in all aspects.

The first problem with BeautifulSoup is if we encounter any HTTP error then we have to deal with it first and then we can go for scraping.

The most common problem with BeautifulSoup and other scraping techniques is that if a website has not allowed or permission of web crawling then it will not be going to scrap the data (only that data that has not allowed crawling). Then for scraping this problem, Selenium is used.

Selenium is quite popular in automation. It is used in JAVA and Python both. Selenium uses its own alias web browser. When we pass the link of any websites, it will open those websites in its own web browser.

Selenium pretend that website is opening on a web browser, so by this it can solve crawling problem which is happening in other web scraping techniques. So, that's why it is all weather scraping technique and it is invincible.

The only problem with selenium is that is very outdated and that's why it is very slow.

#### IV. IMPLEMENTATION

In implementing the web scrapper through any scraping technique, the first compulsory step is to know what do you want? what is your required data? etc are the things we should make clear in our mind. The next step is to inspect your website i.e., search in which HTML tag your required data is present. These two steps are very common step in all scrapping techniques.

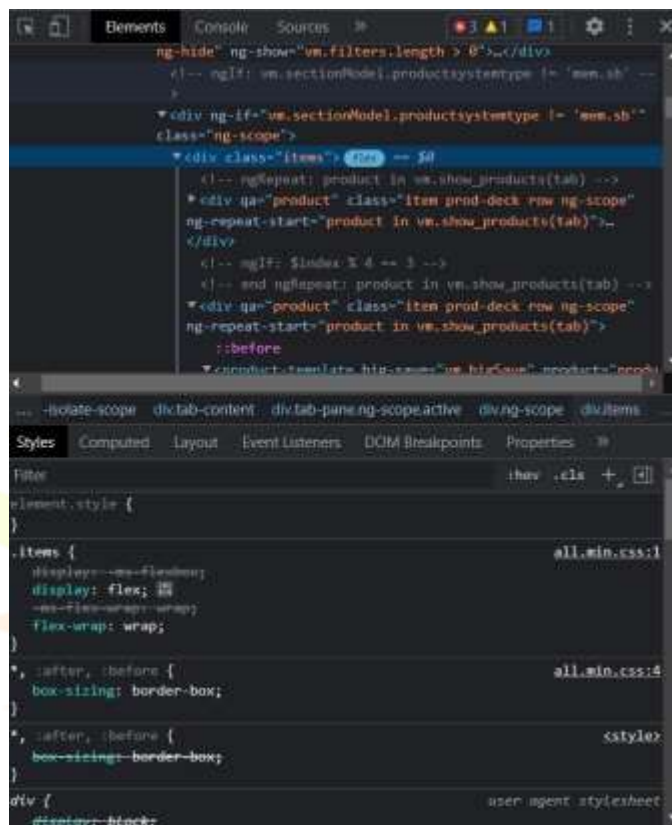


Fig-1 – Required tag after inspection

Now, in every scrapping technique we should pass the URL of our website (which data you want to scrap). We are using PYTHON language to implement our code. And the reason for using this language is that it is very easy to use and it have many in-built functions.

So, first thing is to import selenium library and we have to install chromium driver in case of chrome and in case of Firefox we have to install Firefox driver. After this we pass our URL and that URL will open in chromium driver.

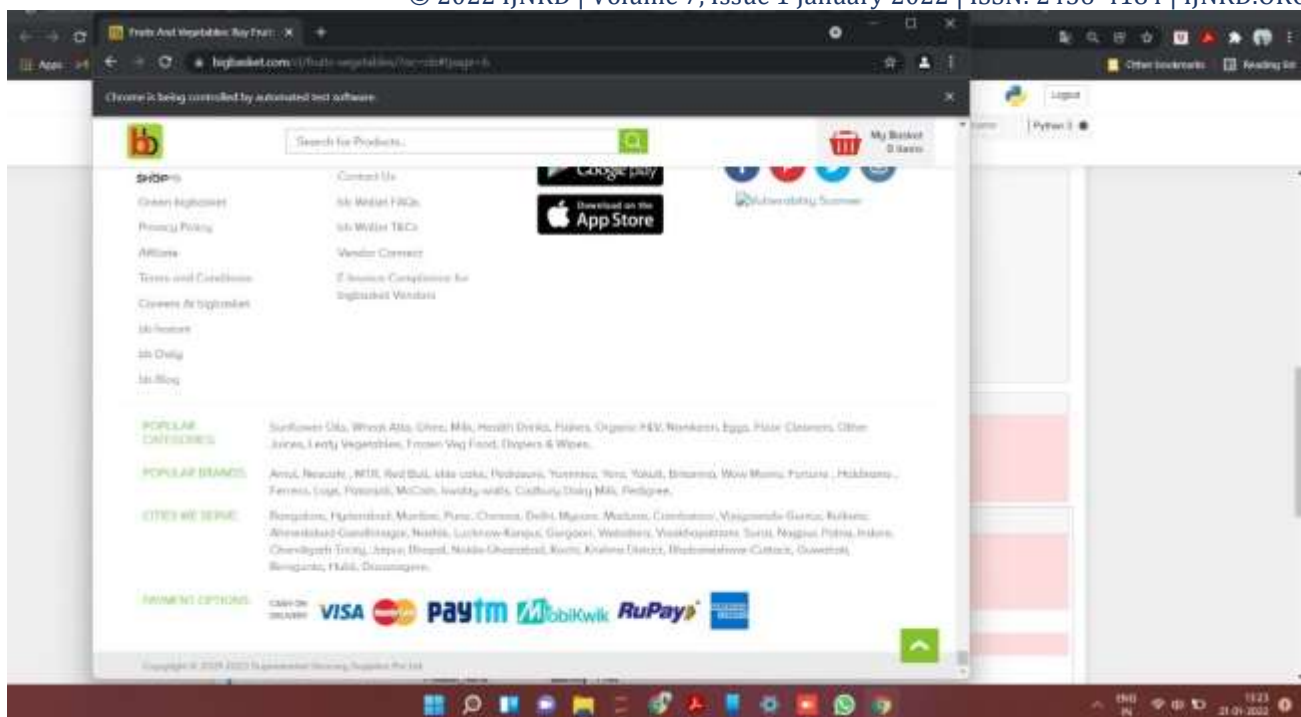


Fig-2 – Website opening in chrome driver.

In above figure, we can see that chrome driver is exactly similar to original chrome. Selenium uses chrome driver because it pretends that the website is opening in web browser. So, with this it prevents from HTTP error i.e., website is only open in web browser.

Now, the last step is scrapping, so in this we pass the class name of tag which we have find in our inspection step and the rest step will automatically perform by selenium.

In below figure, the python code of our scrapper is shown. In this code there is function of big scrapper (), which will going to scrap the data of fruits and vegetables of big-basket website. And the scrapped data will show in pandas data frame and we can even store that data in .csv file also.

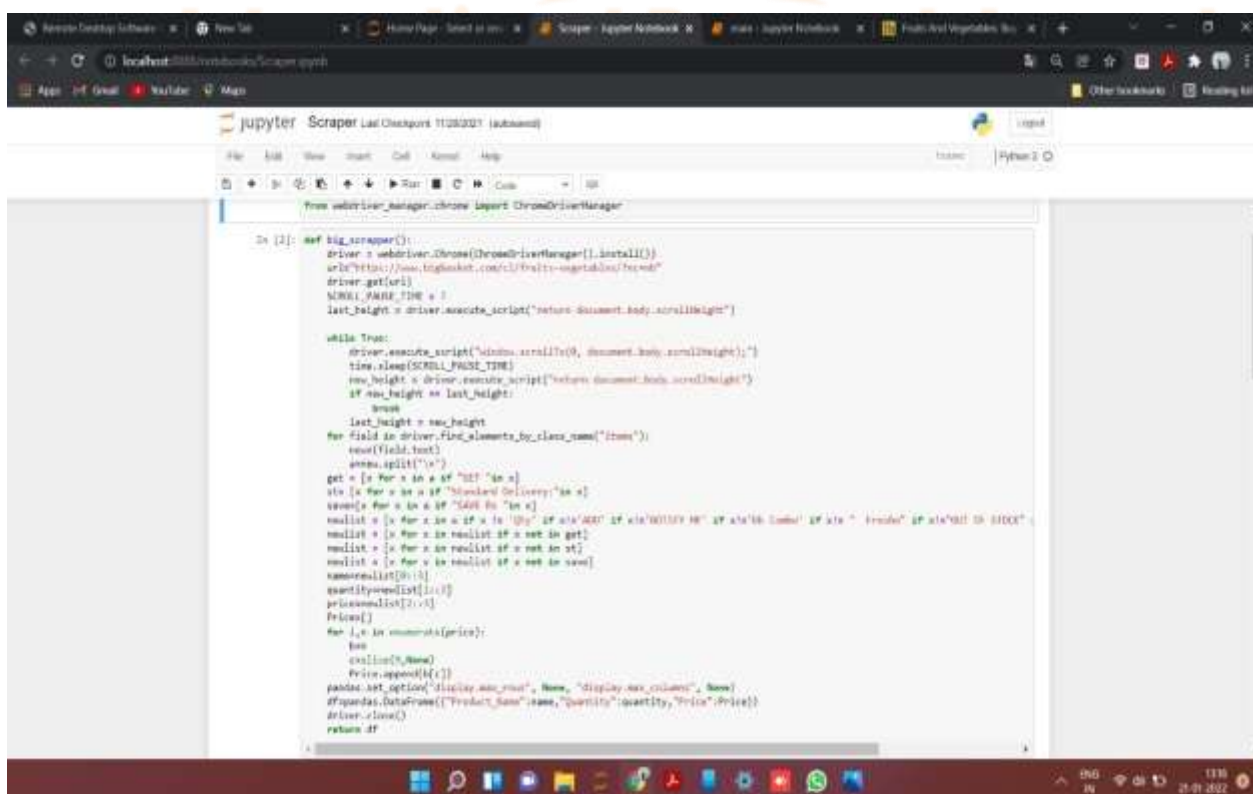
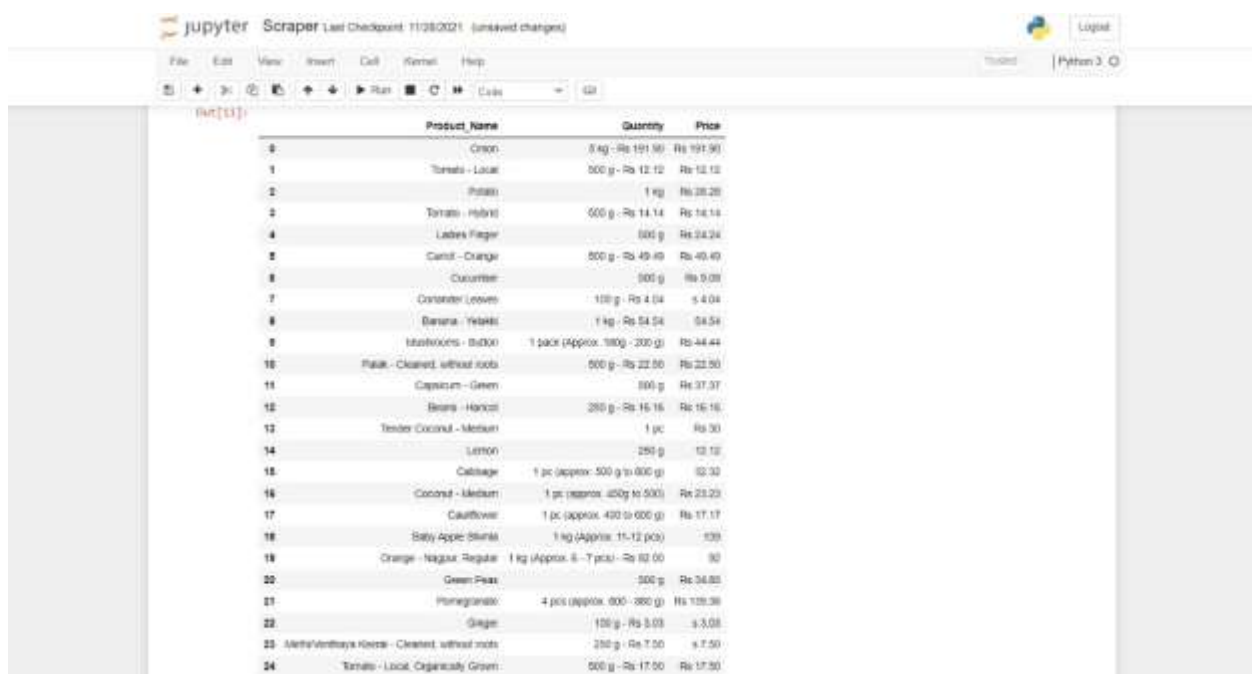


Fig-3- python code





Product Name      Quantity      Price

0	Onion	5 kg - Rs 191.50 - Rs 191.50
1	Tomato - Local	500 g - Rs 12.12 - Rs 12.12
2	Potato	1 kg - Rs 28.28
3	Tomato - Hybrid	500 g - Rs 14.14 - Rs 14.14
4	Ladies Finger	500 g - Rs 24.24
5	Carrot - Orange	500 g - Rs 49.49 - Rs 49.49
6	Cucumber	500 g - Rs 9.09
7	Coriander (leaves)	100 g - Rs 4.04 - Rs 4.04
8	Banana - Hybrid	1 kg - Rs 54.54 - Rs 54.54
9	InstantNoodles - ButDoi	1 pack (approx. 180g - 200 g)
10	Maize - Cleaned, without roots	500 g - Rs 22.50 - Rs 22.50
11	Capasium - Green	500 g - Rs 37.37
12	Beans - Haricot	250 g - Rs 16.16 - Rs 16.16
13	Tender Coconut - Mature	1 pc - Rs 30
14	Lemon	250 g - Rs 12.12
15	Cabbage	1 pc (approx. 500 g to 600 g)
16	Coconut - Medium	1 pc (approx. 450g to 500)
17	Cauliflower	1 pc (approx. 400 to 600 g)
18	Baby Apple (Small)	1 kg (Approx. 15-12 pcs)
19	Orange - Nagpur - Regular	1 kg (Approx. 8 - 7 pcs) - Rs 82.00 - 90
20	Green Peas	500 g - Rs 34.85
21	Pomegranate	4 pcs (approx. 600 - 800 g)
22	Ginger	100 g - Rs 3.03 - Rs 3.03
23	Aethiopian Kales - Cleaned, without roots	250 g - Rs 7.50 - Rs 7.50
24	Tomato - Local, Organically Grown	500 g - Rs 17.50 - Rs 17.50

Fig-4- Scrapped Data

## V. CONCLUSION

The algorithm proposed, although having a limited number of websites with which to work, may be useful in niche situations. For certain websites with certain characteristics the algorithm can be implemented to create a scraper which is unaffected by small changes in the HTML.

A robust web scraper. It is quite simple yet provides decent results. There are a lot of different issues that can be run into when constructing this type of algorithm, but primarily the HTML-code is the main factor that determines the performance. The primary use case for this type of algorithm is for websites with repetitive content in which a business model relies on being able to consistently scrape a website.

The purpose of this research paper is to evaluate the art web scraping tools based on established software evaluation metrics. A recommendation based on state-of-the-art web scraping tools can then be presented, with these different software metrics in mind. As a reminder, the research question is defined as follows

With respect to established metrics of software evaluation, what is the best state of the art web scraping tool?

Regardless of how specific the methodology would have been, there is no way to conclude a definite 'best' tool. Many aspects are highly subjective, and can be interpreted in different ways.

However, some recommendations can be made. These are based on bias in terms of desired programming language, whether the user has more experience or preference within a certain language. For example, in one of the papers presented in the previous work section, psychologists used web scraping to gather data for research.

## REFERENCES

1. John C. Mitchell Adam Barth, Collin Jackson. Robust defenses for cross- site request forgery. pages 75–88, 01 2008.
2. Karan Aggarwal, Abram Hindle, and Eleni Stroulia. Co-evolution of project documentation and popularity within GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 360–363, New York, NY, USA, 2014. ACM.
3. Alex Galea. The Applied Data Science Workshop book of Packt Publications.

4. H. Borges, A. Hora, and M. T. Valente. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344, Oct 2016.
5. Hudson Borges and Marco Tulio Valente. What's in a GitHub star? understanding repository starring practices in a social coding platform. *CoRR*, abs/1811.07643, 2018.
6. Osmar Castrillo-Fernández. Web scraping: applications and tools. European Public Sector Information Platform, 2015.
7. Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in GitHub: Transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, CSCW '12*, pages 1277–1286, New York, NY, USA, 2012. ACM.
8. Fazal e Amin and Alan Oxley Ahmad Kamil Mahmood. Reusability assessment of open-source components for software product lines. 2011

