World Academics Journal of Engineering Sciences
Vol.**11**, Issue.**4**, pp.**01-07**, December **2024**
E-ISSN: 2348-635X
Available online at: www.isroset.org

Research Article

From Web to Insights: Automating and Optimizing Job Data Collection with Selenium

Pramiti Tewari[1*], Utkarsh Gupta[2], Samriddhi Tripathi[3], Ajay Kumar[4]

[1,2,3,4]CSE, Jaypee University of Engineering and Technology, Guna, India

**Corresponding Author: pramiti0309@gmail.com*

Received: 25/Oct/2024; **Accepted**: 27/Nov/2024; **Published**: 31/Dec/2024

Abstract— The research explores web scraping as an efficient method for data extraction, focusing on job postings from LinkedIn using Selenium. By automating the interactions with dynamic web elements, the study extracts job data such as job titles, companies, and addresses from the recent job postings. It also addresses the challenges such as dynamic content handling, anti-bot mechanisms while also keeping the legal norms and ethical considerations of data mining. Python modules such as BeautifulSoup, Scrapy and Selenium are reviewed as choices for the automated script while emphasis is given on Selenium's scalability, robustness, efficiency and adaptability to real-world scenarios such as multi-page navigation, error handling and regular updates. The approach highlights web scraping's potential in leveraging data mining and potential for effective analysis, offering an ethical solution for data-driven approach.

Keywords— Web Scraping, Selenium, Python Automation, Dynamic Content Handling, Pagination, WebDriver

1. Introduction

With the rapid growth in the need to collect and study data, data extraction and collection has become a vital requirement for industries to make decisions, solve issues and generate insights in order to develop the latest software or tools in general. Data collection can be done in various ways namely surveys and manual entry, extraction of data using web scraping, accessing data via Application Programming Interfaces from websites, using query languages such as SQL or NoSQL to extract data from databases and techniques as file parsing. While selecting any of the given techniques, the context and aim of the research must be accounted for [1]. The challenges of getting relevant data from social media sites arises due to limited accessibility and continuous updates in the websites. Despite, the challenges posed systematic extraction is often seen as a cornerstone and the most relevant process to collect data and generate insights or facilitate studies.

Web scraping is one of the commonly used techniques to extract first-hand information from the websites or vivid platforms using an automated script. Libraries such as Scrapy, BeautifulSoup and Selenium are used to write the automated scripts in order to scrape data. However, some challenges are incurred while using these solutions such as ethical concerns related to the privacy of data, the limitations that such websites impose and risks of violating the site's policies.

The article explores various web scraping techniques and the challenges presented by them along with the probable solutions. It focuses on leveraging the Selenium module, using Chrome WebDriver to scrape job postings from LinkedIn. Python and its modules are used to automate data extraction, navigate through the pages and manage elements dynamically. The article aims to highlight the aspects of web scraping using Selenium and discuss the various challenges faced and the various solutions to optimize the scraping process. By organizing the data collected, the approach is utilized to study the latest trends in the job market with the primary benefit of being an efficient tool to scrape the recent data from the various sites.

2. Background

The job market is constantly evolving with new trends in skill sets, the task force and the tech stack in use. Social media postings on the recent hirings are at the dispense of the organizations to hire employees. Manually analyzing this data to study the recent trends in extremely inefficient which in turn leads to a need of automated methods to collect and analyze data. Web scraping offers a method to systematically collect this data and store it in the form of CSV files.

2.1 Existing Work

Web Scraping involves bots or crawlers to extract specific data from websites and store in database. The targeted data retrieving techniques are useful to analyze the data [2].

Modern frameworks account for an ease of setting up pipelines for web scraping reducing the efforts and is an important alternative when APIs are unavailable [3]. It is an aid for the analysis of unstructured data and involves various technologies such as spidering and pattern matching [4]. Python's ease of use and the vast development community make it a natural choice for web scraping. It not only fosters advancements in the diverse applications such as open government data, big data analytics, business intelligence etc. but also promotes development [5]. Another commonly used programming language is R which statistically integrates programming tools for the datasets [6]. Research has also been conducted on integrating traditional statistical methods with modern computational techniques to leverage the field of data mining [7]. Various approaches such as using DOM tree for parsing or using UzunExt which extracts the contents without forming a DOM tree but by using strings [8]. The article utilizes different approaches for successful data extraction.

2.2 Tools and Technologies for Data Scraping

Web scraping is one of the commonly used techniques to extract first-hand information from the websites or vivid platforms using an automated script. Web pages can be categorized as static and dynamic. Static pages, often, are easier to scrape since an HTTP request is sent to the server while dynamic pages rely on interacting with the page elements and server-side scripting languages such as JavaScript, PHP etc. [9]. Various libraries are utilized for this purpose:

2.2.1 BeautifulSoup

BeautifulSoup is a python library widely used for parsing the web documents, particularly HTML and XML documents. It facilitates structured data extraction by travelling the DOM tree of the document [10]. It converts the document to Unicode and UTF-8 and works with a parser to navigate through the pages [11]. This feature is particularly important for scraping web pages with diverse and non-standard character sets. It has been tailored to deal with specific parts of the web pages and prioritizes speed over efficiency, which could be a drawback for more intricate scraping tasks requiring comprehensive data extraction, leaving some data unprocessed [12]. It is designed to quickly parse and navigate documents, but for more complex HTML pages, it might not be as thorough as other parsers. This trade-off makes it an excellent choice for straightforward scraping tasks where speed is a priority.

2.2.2 Scrapy

Scrapy allows cascade operations and accounts for data needed for listings. It provides selectors necessary to handle broken HTML codes [13]. Scrapy comes with features such as creating a spider class, responsible for defining how a website must be scrapped and then saves it. However, it also handles static elements with more efficiency.

2.2.3 Selenium

Selenium is a powerful, versatile and robust module used to dynamically interact with the elements on a web page and emulate user actions such as clicking buttons, entering text, scrolling, and navigating through different pages. It also provides an easy and seamless integration with API's and an easy access to web drivers [14] to control different browsers like Chrome, Firefox, Safari, Edge. These drivers translate Selenium commands into browser-specific actions, making it a versatile tool for cross-browser testing and automation. Class labels or unique identities such as IDs, names and XPath are used for efficient data extraction [15]. This precise targeting of elements is necessary for reliable data extraction, particularly from complex web pages with nested elements or dynamically generated content.

Due to the dynamic nature of Selenium and seamless handling of AJAX (Asynchronous JavaScript and XML) calls, it ensures that interaction with elements is performed once they are fully loaded and ready, thus it is one of the most sought-after modules for web scraping. Selenium works exceptionally well in automating complex workflows that involve multiple steps and interactions across different web pages. For example, it can log into websites, navigate through several pages, fill out forms, and extract the necessary information, all while handling dynamic content and interactions seamlessly.

2.3 Role of Web Scraping in Data Analytics

Web scraping plays a critical role in data analytics by providing a powerful means to collect large volumes of data from the internet. Web Scraping enables data collection by facilitating professionals with data collection by tracking emerging trends in the desired fields and optimizing the results for any sort of analysis predictive results. This process involves automated extraction of information from websites to derive valuable insights and support various business decisions. It is also insightful for market dynamics, economics related data, competitive data and prevailing trends which could be used by firms to form an action plan for their product building and execution [16].

By leveraging machine learning algorithms and statistical models, we can analyse this data to uncover patterns, predict future trends, and make data-driven decisions. This helps in improving the accuracy and reliability of predictive analytics. The data collected through web scraping can be used to formulate actionable plans for various aspects. Individuals can use this data to make long-term plans and roadmap to learn about new growth opportunities.

3. Data Extraction

Data extraction in this study is done using an automated script written using the Selenium module of Python using Jupyter notebook as a computing platform as it supports programming documents that seamlessly integrate code bases and results, efficient ode debugging and offers effective visualizations for the input [17]. With its support for markdown cells, code cells, and visual outputs, Jupyter Notebook allows researchers to write, test, and debug code while simultaneously viewing the results of their executions, thus enhances readability and presentation. The flowchart shown in Figure 1 illustrates the sequential steps in the automated process. It outlines the sequential steps in the automated job data extraction process, capturing the intricate interactions between web elements and the dynamic nature of LinkedIn's website. These steps include logging into LinkedIn, navigating through different web pages, identifying and interacting with web elements, and extracting the required data.

The script systematically collects data from various sections of the LinkedIn job postings, which includes job titles, company names, locations, and other relevant details. The flowchart also showcases implementation of exception handling. This is crucial for managing the dynamic changes on LinkedIn's website, such as content loading delays, changes in web elements, etc. Each step of the workflow is designed to ensure systematic extraction, minimizing the risk of data loss. Automating the data extraction process with Selenium significantly increases efficiency compared to manual data collection. This is essential for conducting reliable data analysis and making informed decisions based on the extracted data.

This flowchart not only visualizes the overall process but also highlights the logic behind navigation, data collection, and exception handling. The use of Selenium allows for real-time interaction with web elements, enabling accurate and efficient data retrieval. It systematically represents each step, from launching the web driver to processing the extracted data, ensuring a clear understanding of the workflow.

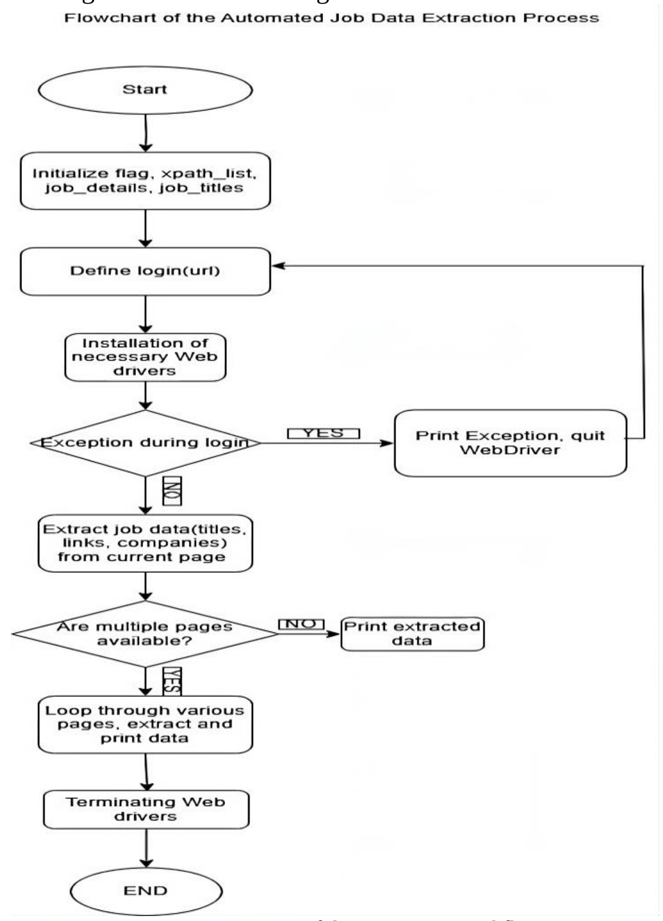
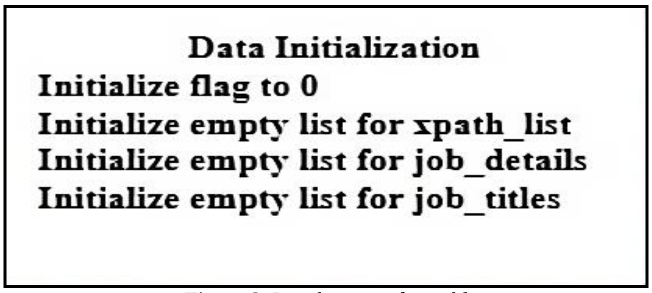

Flowchart of the Automated Job Data Extraction Process
Start
Initialize flag, xpath_list, job_details, job_titles
Define login(url)
Installation of necessary Web drivers
Exception during login
YES
Print Exception, quit WebDriver
NO
Extract job data(titles, links, companies) from current page
Are multiple pages available?
NO
Print extracted data
YES
Loop through various pages, extract and print data
Terminating Web drivers
END

Figure 1. Overview of data extraction workflow

4. Methodology

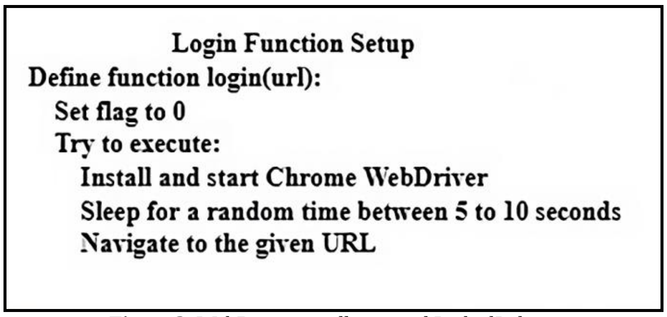
Here the author presents the automated script using a pseudocode:



```
Data Initialization
Initialize flag to 0
Initialize empty list for xpath_list
Initialize empty list for job_details
Initialize empty list for job_titles
```

Figure 2. Initialization of variables

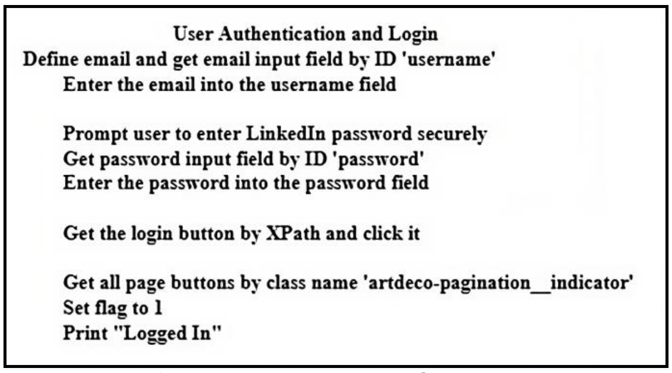
The author begins with the initialization of a flag to 0, which shall be used to indicate whether the login was successful or not, while three empty lists are initialized: 'xpath_list' for storing XPaths of page buttons, 'job_details' for entering the details of jobs like company name that posted the job, location of posting, and 'job_titles' to keep track of job titles.



```
Login Function Setup
Define function login(url):
  Set flag to 0
  Try to execute:
    Install and start Chrome WebDriver
    Sleep for a random time between 5 to 10 seconds
    Navigate to the given URL
```

Figure 3. WebDriver installation and LinkedIn login

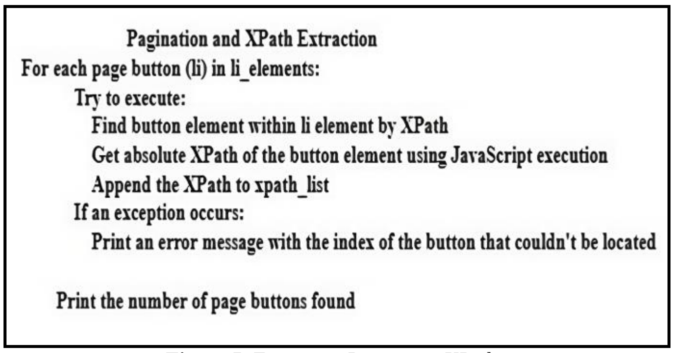
The `login` function contains several steps: installing and starting the Chrome WebDriver to set up the web scraping environment, introducing a delay by sleeping for a random time between 5 to 10 seconds to mimic human interaction, and navigating to the given URL, which hits the specified LinkedIn login page.



```
User Authentication and Login
Define email and get email input field by ID 'username'
    Enter the email into the username field

    Prompt user to enter LinkedIn password securely
    Get password input field by ID 'password'
    Enter the password into the password field

    Get the login button by XPath and click it

    Get all page buttons by class name 'artdeco-pagination__indicator'
    Set flag to 1
    Print "Logged In"
```

Figure 4. Automating Login and Navigation

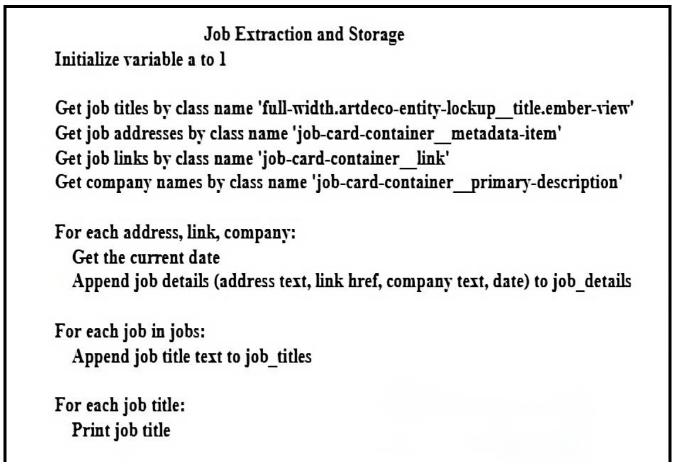
The login process then proceeds by defining the email and identifying the email input field by its ID 'username'. The email is then entered into this field. The user is prompted to enter their LinkedIn password securely using a secure input method such as masking the password input with dots and asterisks, and the password input field is identified by its ID 'password'. The password is entered into this field. Finally, the login button is identified by its XPath and clicked to submit the form. Once signed in, all pagination buttons on the webpage are located using their class name ‘artdeco-pagination_indicator’. Flag variable is updated to 1 indicating successful login process.



```
            Pagination and XPath Extraction
For each page button (li) in li_elements:
    Try to execute:
        Find button element within li element by XPath
        Get absolute XPath of the button element using JavaScript execution
        Append the XPath to xpath_list
    If an exception occurs:
        Print an error message with the index of the button that couldn't be located

    Print the number of page buttons found
```

Figure 5. Extracting Pagination XPaths

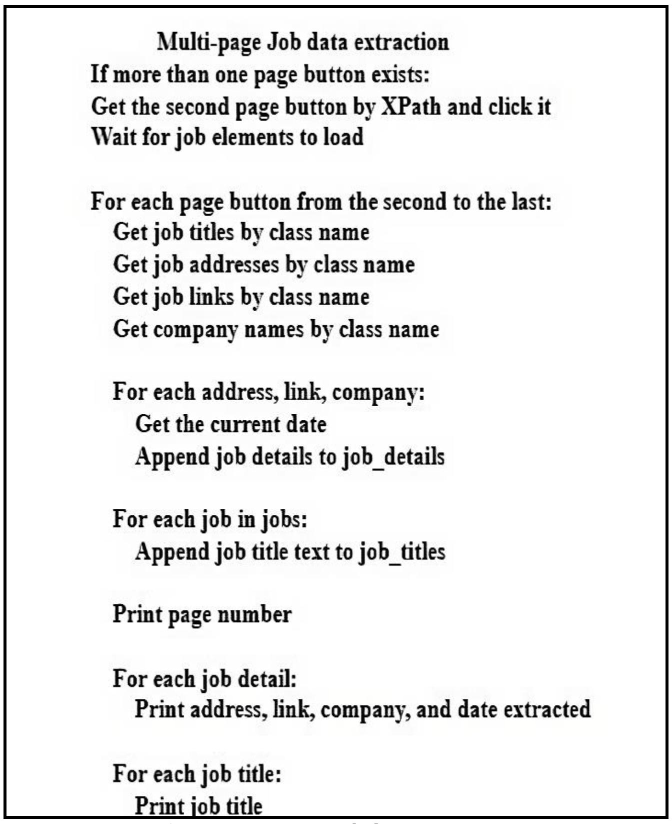
After logging in, the author also displays a message “logged in”. The process then moves to extracting the XPaths of these page buttons. For each page button in the `li_elements`, the author locates the button element within the list item by XPath, extract the button's absolute XPath using JavaScript execution. Then the author appends this XPath to the `xpath_list`. If an exception occurs, an error message is printed ensuring smooth execution of the program even if some elements are not found. The number of page buttons found is then printed, which confirms the number of buttons extracted.



```
                Job Extraction and Storage
Initialize variable a to 1

Get job titles by class name 'full-width.artdeco-entity-lockup__title.ember-view'
Get job addresses by class name 'job-card-container__metadata-item'
Get job links by class name 'job-card-container__link'
Get company names by class name 'job-card-container__primary-description'

For each address, link, company:
    Get the current date
    Append job details (address text, link href, company text, date) to job_details

For each job in jobs:
    Append job title text to job_titles

For each job title:
    Print job title

For each job detail:
    Print address, link, company, and date extracted
```

Figure 6. Extracting and storing job details

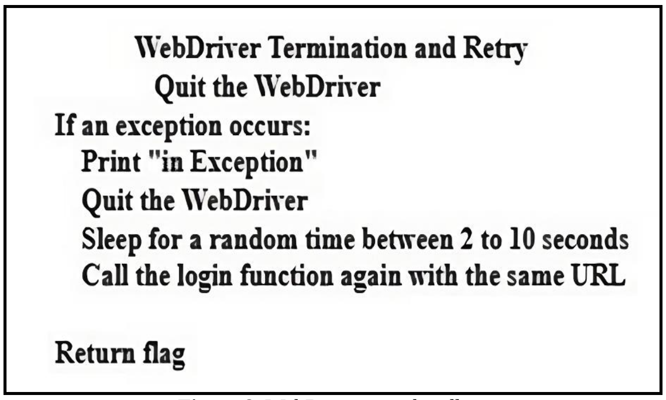
A variable `a` is initialized to 1, which is used to keep a count of number of jobs extracted. The next process involves extracting job details. Job titles are retrieved by their class name 'full-width.artdeco-entity-lockup__title.ember-view', job addresses by 'job-card-container__metadata-item', job links by 'job-card-container__link', and company names by 'job-card-container__primary-description'. For each address, link, and company, the current date is captured. The extracted job details, including the address text, href link, company text, and date of extraction, are appended to the `job_details` list. For each job, the job title text is appended to the `job_titles` list. Following this, job details consisting address, link and company name and job titles are printed along with date extracted.



```
        Multi-page Job data extraction
If more than one page button exists:
Get the second page button by XPath and click it
Wait for job elements to load

For each page button from the second to the last:
    Get job titles by class name
    Get job addresses by class name
    Get job links by class name
    Get company names by class name

    For each address, link, company:
        Get the current date
        Append job details to job_details

    For each job in jobs:
        Append job title text to job_titles

    Print page number

    For each job detail:
        Print address, link, company, and date extracted

    For each job title:
        Print job title
```

Figure 7. Extracting job data across pages

To handle multiple pages, the second page button is located by its XPath and clicked to navigate to the second page. The program then waits for the job elements to load, ensuring the new web page is fully loaded. For each page button from the second to the last, the extraction and printing process for job titles, addresses, links, and company names is repeated. The next page button is located by its XPath and clicked to navigate to the next page, and the program waits for all the job elements to load to ensure the smooth and efficient execution of the code. The loop terminates after the extraction is completed till the last page.



```
        WebDriver Termination and Retry
            Quit the WebDriver
If an exception occurs:
    Print "in Exception"
    Quit the WebDriver
    Sleep for a random time between 2 to 10 seconds
    Call the login function again with the same URL

Return flag
```

Figure 8. WebDriver error handling

The final step is to quit the WebDriver, effectively closing the browser. In terms of exception handling, if an exception occurs, the program indicates the occurrence by printing "in Exception" and ensures the browser is closed by quitting the WebDriver before the script re-attempts the login function. It then sleeps for a random time between 2 to 10 seconds to introduce a delay before retrying the process. After this delay, the `login` function is called again to retry the sign in process. Eventually, the function returns the flag to indicate the status of the login attempt which helps in identifying if the code needs to retry or not.

This script not only handles the multiple exceptions and ensures smooth execution but also accounts for the updates in the application.

5. Challenges in Development

With the ever-evolving trends, manually extracting and analyzing vast datasets is a cumbersome, time-consuming and an inefficient process. Therefore, automation script is a more viable and effective solution when dealing with such large volume of data. However, it comes with a fair share of challenges. One major issue is that data is generally not static, so the results might change even for the same data. This means that even if you extract data from the same source at different times, the results can vary. Moreover, websites change their structure time to time which pose a challenge while timely updating the script. Generally, non-sequential URL parameters for handling multiple pages are utilized by websites to obstruct data extraction, which makes it extremely dynamic in nature, posing another challenge [18]. Usually, websites also have strict terms and privacy measures to avoid any breach in data privacy, which can be concerning especially if there is no accountability of the data or if it is outdated [19]. Selective reporting or lack of control over the mining of data can also lead to many limitations across the internet [20]. However, with the author's work a generic dataset is utilized to contribute to a job recommendation tool, which has its own scope in the future, thus contributing in the field of technology. The author's proposed approach encountered many challenges in the development stage:

5.1 ElementClickInterceptedException

The error 'ElementClickInterceptedException' is faced when script tries to click a button, but it could not locate that element. For example, the button labeled 'Page 5' could not be clicked because a <div> with the class 'scaffold-layout_list' was overlapping it. This problem occurs when the page layout changes or dynamic content loads, causing elements to overlap the target element.

To address the ElementClickInterceptedException, it is important to ensure that the page has fully loaded and all elements are in their final positions before attempting to interact with them. To fix this issue, author implied Implicit wait technique. Implicit waits tell the web driver to wait for a certain amount of time when trying to find an element if it is not immediately available. If the problem persists, as a last resort, author can use JavaScript to click the element. This bypasses the normal Selenium click mechanism and can sometimes avoid the above-mentioned problem.

5.2 LinkedIn's Anti-Bot Detection

LinkedIn has sophisticated systems to detect automated tools and block or redirect them. These systems are designed to maintain the integrity of the platform by ensuring that all users abide by LinkedIn's terms of service. It happens if LinkedIn notices unusual browsing patterns that suggest a bot is being used. Bots often perform repetitive actions at a speed and consistency that is unusual for human users.

As a result, when LinkedIn detects suspicious activity, the user might experience redirects to login pages or different URLs to verify their identity. Another possible result is an HTTP Error 429. This error means LinkedIn has blocked the IP address because it made too many requests in a short period. This mechanism is called rate limiting, which restricts the number of requests that can be made from a single IP address within a specified timeframe.

To avoid violating LinkedIn's anti-bot detection mechanisms, it is essential to mimic human browsing behaviour more closely and reduce the frequency of requests. Implementing delays between requests can help in mimicking human browsing patterns. Human interactions are generally irregular, so incorporating randomness in your script can help avoid detection. Avoid Continuous Scraping, schedule it to run at intervals.

5.3 MaxRetryError

The 'MaxRetryError' implies that a script repeatedly tried and failed to connect to a server. In this case, it was trying to reach localhost on port 55063 but could not establish a connection. The displayed message, [WinError 10061] No connection could be made because the target machine actively refused it, means that the server on localhost is not accepting connections on the specified port. This issue can arise from various underlying causes.

This can happen due to many reasons such as if the server is not running, script may be trying to connect to an incorrect port, firewalls or other security software might be blocking the connection. If the server is not active, it cannot accept any connections, leading to a refusal.

The issue can be fixed by ensuring that the server is up and running. Confirm that the port specified in the script matches the port on which the server is listening. Ensure that your firewall or any security software is not blocking the port, if any, then adjust the firewall settings to allow traffic on the specified port.

6. Results and Discussion

Here, the results of web scraping process using Selenium are presented in a tabular manner. This table 6.1 summarizes the extracted job postings, and presents the details like job title, company name, location and the date of extraction.

Table 6.1 Summary of Job Data Extracted from LinkedIn

S. No	Title	Company	Location	Date
1.	Data Scientist	Recro	India (Remote)	13-10-24
2.	SEO Intern	SEO Inventiv	India (Remote)	16-10-24
3.	Data Analyst	After Passout Pvt Ltd	Noida, India (On-site)	18-10-24
4.	ML Engineer	Weekday (YC W21)	Noida, India (On-site)	29-10-24
5.	AI Engineer	IBM	India (Remote)	13-11-24
6.	SDE intern	Contlo	India (Remote)	07-10-24
7.	Web Developer	Outscal	India (Remote)	07-10-24

The script's effectiveness in gathering relevant job data is quite evident. Each entry reflects to a unique job posting, providing a snapshot of the current job market trends. This data can be further analysed to identify patterns and emerging skills for various job roles. Date of extraction ensures that the data is relevant and up-to-date. This is crucial for accurate and reliable analysis.

7. Conclusion

In this study, the author explored the vital role of web scraping in data analytics. Web scraping and techniques confronting numerous difficulties as the extraction of the information are not excessively simple [21]. The author’s emphasis was on its application in extracting job postings from LinkedIn using Selenium. The automated approach showed us the limitations and inefficiencies of manual data collection, providing a more efficient and scalable solution to analyze evolving job market trends. By leveraging Python’s varied libraries, particularly Selenium, the research showcased how dynamic interactions with web elements and navigating multiple pages, can be effectively automated.

Nowadays, websites often use JavaScript to load data dynamically which poses a significant challenge for conventional web scraping techniques. Selenium, however, can interact with browser just like a human user, waiting for elements to load and then extracting the necessary information.

User authentication was another critical challenge addressed by the author. The author showed how Selenium could be used to automate sign in process securely, ensuring that the user credentials are kept private and that the scraping process remains within the bounds of LinkedIn’s terms of data security policy.

Job postings on LinkedIn are typically spread across multiple pages. The author displayed how this process could be automated, systematically handled pagination to gather a comprehensive dataset of job postings. Traversing multiple pages of job listings was another highlight of the study.

The proposed technique highlights the flexibility and power of Selenium in handling real-world challenges such as dynamic content loading, user authentication, and traversing data in multiple pages. During the extraction process, the author followed all the privacy and data security guidelines stated by LinkedIn. This ensured that the data collection was conducted ethically and legally, respecting user privacy and the platform's terms of service.

Through the systematic handling of errors, and successfully extracting the job details through web scraping one can demonstrate its immense value as a tool for data-driven decision-making in an increasingly digital world. By automating the extraction of job postings from LinkedIn, the author highlighted how such techniques could provide deeper insights into job market trends, thereby helping in more informed decision-making [22].
The research highlighted the power and flexibility of Selenium in overcoming real-world challenges, making it a reliable tool for modern data analytics.

Conflict of Interest
The authors declare that they do not have any conflict of interest.

Funding Source
The present work has no funding source in any manner from any organization.

Authors’ Contributions
Author 1 conceived the study and performed conducive literary research, critical review, perceiving the conceptual analysis taken up. Through this rigorous review process, Author 1 was able to perceive and articulate the conceptual analysis that formed the basis of the research.
Author 2 contributed in the technical aspect of the research involving development of the code for data extraction and analysis and conceptualized the execution process by designing algorithms and writing scripts.
Author 3 processed the data, handled the collection, and played an instrumental role in the literature review.
Author 4 directed the overall progress of the study providing guidance and oversight throughout the research process. and reviewed and finalized the article.

Acknowledgements
A special mention to the open-source documents and tools for their invaluable support, enabling seamless implementation and execution of the project. We extend our heartfelt gratitude to the creators and maintainers of the open-source documents and tools. Their accessibility and comprehensive guidance greatly contributed to the development and optimization of the automated scraping script. These resources offered detailed instructions examples and advanced techniques that enhanced the efficiency and reliability of our script.

The open-source community's commitment to sharing knowledge and improving tools facilitated our ability to navigate challenges, such as handling dynamic content and

ensuring data accuracy. The support from these open-source documents and tools reflected the importance of community-driven development in advancing technological capabilities.

We acknowledge and appreciate the significant impact of these resources and the dedicated individuals behind them, whose efforts have empowered countless projects like ours to achieve their goals.

References

[1] T. C. Lethbridge, S. E. Sim, and J. Singer, "Studying software engineers: Data collection techniques for software field studies," Empir. Softw. Eng., Vol.**10**, No.**3**, pp.**311–341, 2005.** doi: 10.1007/s10664-005-1290-x.

[2] H. Chaib and K. Salah-ddine, "Using Web Scraping In A Knowledge Environment To Build Ontologies Using Python And Scrapy," no. October, 2020.

[3] H. Lo, M. Reboiro-jato, F. Fdez-riverola, and D. Glez-pen, "Web scraping technologies in an API world," Vol.**15**, No.**5**, pp.**788–797, 2013.** doi: 10.1093/bib/bbt026.

[4] M. A. Khder, "Web Scraping or Web Crawling : State of Art , Techniques , Approaches and Application," Vol.**13**, No.**3**, **2021.** doi: 10.15849/IJASCA.211128.11.

[5] V. Singrodia and A. Mitra, "A Review on Web Scrapping and its Applications," 2019 Int. Conf. Comput. Commun. Informatics, no. January, pp.**1–6, 2019.** doi: 10.1109/ICCCI.2019.8821809.

[6] R. J. E. James, "Web Scraping Using R," **2019**. doi: 10.1177/2515245919859535.

[7] M. Dogucu and M. Çetinkaya-rundel, "Web Scraping in the Statistics and Data Science Curriculum : Challenges and Opportunities Web Scraping in the Statistics and Data Science Curriculum : Challenges and," J. Stat. Educ., pp.**1–24, 2021**. doi: 10.1080/10691898.2020.1787116.

[8] E. Uzun, "A Novel Web Scraping Approach Using the Additional Information Obtained from Web Pages," IEEE Access, Vol.**8**, pp.**61726–61740, 2020.** doi: 10.1109/ACCESS.2020.2984503.

[9] S. Kumar, J. Thakur, D. Ekka, and I. Sahu, "Web Scraping Using Python," Int. J. Adv. Eng. Manag., Vol.**4**, No.**9**, pp.**235, 2022.** doi: 10.35629/5252-0409235237.

[10] C. Zheng, G. He, and Z. Peng, "A Study of Web Information Extraction Technology Based on Beautiful Soup," J. Comput., Vol.**10**, No.**6**, pp.**381–387, 2015.** doi: 10.17706/jcp.10.6.381-387.

[11] L. Richardson, "Beautiful Soup Documentation Release 4.4.0," Media.Readthedocs.Org, pp.**1–72, 2019.**

[12] A. Abodayeh, R. Hejazi, W. Najjar, L. Shihadeh, and R. Latif, "Web Scraping for Data Analytics: A BeautifulSoup Implementation," Proc. - 2023 6th Int. Conf. Women Data Sci. Prince Sultan Univ. WiDS-PSU 2023, no. January, pp.**65–69, 2023.** doi: 10.1109/WiDS-PSU57071.2023.00025.

[13] V. Suganthi and M. M. Varun, "INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH Automation Using Selenium," Sci. Eng. Technol. | An ISO, Vol.**9001**, No.**4**, pp**.5181, 2008.** doi: 10.15680/IJMRSET.2024.0704026.

[14] S. Mehta, P. Gayatri, and P. Jain, "An Improving Approach for Fast Web Scrapping Using Machine Learning and Selenium Automation," Vol **8**, No.**10**, pp.**434–438, 2019.**

[15] K. Henrys, "Importance of web scraping in e-commerce and e-marketing," no. January, pp.**1–10, 2021.**

[16] F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "A Large-scale Study about Quality and Reproducibility of Jupyter Notebooks," pp.**1–11, 2024.**

[17] P. Meschenmoser, N. Meuschke, M. Hotz, and B. Gipp, "Bibliographic Details BibTe X , EndNote … Authors ' Details D, Lib Magazine Scraping Scientific Web Repositories : Challenges and Solutions for Automated Content Extraction 1 Introduction 2 Related Work 3 Challenges for Scraping", doi: 10.1045/September, **2016.**

[18] V. Krotov and L. Johnson, "Big web data: Challenges related to data, technology, legality, and ethics," Bus. Horiz., no. October 2022**, 2023**. doi: 10.1016/j.bushor.2022.10.001.

[19] K. Weerasinghe, M. W. P. Maduranga, and M. M. V. T. Kawya, "Enhancing Web Scraping with Artificial Intelligence: A Review," January, **2024.**

[20] V. Srividhya and P. Megala, "Scraping and Visualization of Product Data from E-commerce Websites," Int. J. Comput. Sci. Eng., Vol.**7**, No.**5**, pp.**1403–1407, 2019.** doi: 10.26438/ijcse/v7i5.14031407.

[21] S. Kulkarni, "Web Scraping: Extracting Insights from the Digital Landscape," Int. J. Res. Appl. Sci. Eng. Technol., Vol.**11**, No.**5**, pp.**7564–7567, 2023.** doi: 10.22214/ijraset.2023.53467.

AUTHORS PROFILE

Pramiti Tewari has done her schooling from Puranchandra Vidyaniketan, Kanpur. Currently, she is pursuing B.Tech. in Computer Science and Engineering from Jaypee University of Engineering and Technology, Guna, Madhya Pradesh (India). Her interests include Data mining, statistical analytics and modelling along with a proclivity towards Applied Mathematics and Combinatorics.

Utkarsh Gupta has done his schooling from Heritage International School, Kanpur. Currently, he is pursuing B.Tech. in Computer Science and Engineering from Jaypee University of Engineering and Technology, Guna, Madhya Pradesh (India). His interests include Data Analysis, Machine Learning and Game Development.

Samriddhi Tripathi completed her schooling at Carmel Convent School, Bhopal (M.P). Presently, she is pursuing a B.Tech. degree in Computer Science at Jaypee University of Engineering and Technology, located in Guna, India. Her areas of interest span Artificial intelligence, Machine Learning, and Developmental cores, with a strong inclination towards Quantum Computing and a passion for exploring innovative technologies and practical applications.

Ajay Kumar has been working as an Assistant Professor since 2006. He completed his Ph.D. in 2017 from Jaypee University of Engineering and Technology in the Department of Computer Science and Engineering. His Ph.D. work focused on the design and analysis of an effective partition-based clustering algorithm and its applications. He earned his M.E. from M.I.T.S. Gwalior in 2005 with a thesis titled "Design and Analysis of a Data-Mining Tool." He obtained his B.Tech in Information Technology from M.I.E.T., Meerut in 2002. Additionally, he has an Advanced Diploma in "Network Planning and Administration" from C-DAC (Mohali). Prior to joining JUET, he served as a Senior Lecturer in Computer Science at S.I.E.T. Meerut. His areas of interest include Data Mining and Pattern Recognition.