

A Web Scraping-based Framework for Dynamic E-Commerce Price Comparison using Flask, Selenium and BeautifulSoup

Swati Jadhav¹, Rohan Patil², Sagar Patil³, Sahil Patil⁴ and Swami Patil⁵

¹⁻⁵Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, India
Email: swati.jadhav@vit.edu

Abstract—This paper presents a robust framework for dynamic e-commerce price comparison leveraging web scraping techniques. The framework utilizes Flask, Selenium, and BeautifulSoup to automate the extraction of product data from multiple e-commerce platforms such as Flipkart and Amazon. By dynamically generating search URLs and employing automated content extraction, the framework retrieves and compares product prices, descriptions, and images. Error handling mechanisms mitigate challenges like network disruptions and changes in webpage structures, ensuring reliable data extraction. Integrated into a Flask web application, the system offers users an intuitive interface for querying products and viewing comparative results. The effectiveness of the framework is demonstrated through case studies and performance evaluations under varying data volumes and user loads. Future enhancements include scalability improvements and expanding scraping capabilities across diverse e-commerce domains.

Index Terms— Web scraping, e-commerce, price comparison, Flask framework, Selenium WebDriver, BeautifulSoup, dynamic content extraction, error handling, data validation, product information retrieval.

I. INTRODUCTION

In The dynamic landscape of modern e-commerce poses significant challenges for consumers seeking to compare prices across platforms such as Amazon and Flipkart [1]. Price variations for identical products often lead to missed opportunities for savings and contribute to consumer dissatisfaction. To address these issues, various price comparison systems employ diverse strategies to aggregate data from multiple sources. However, they frequently encounter challenges related to handling dynamic information, ensuring real-time updates, maintaining data accuracy, and ensuring seamless accessibility across different devices [8].

Many existing systems heavily rely on e-commerce APIs or simplistic web scraping techniques, which can result in inaccurate data due to the evolving nature of website structures and price information [3]. Moreover, scalability and performance issues often arise when these systems process large datasets, leading to delays in response times and hindering user experience [2]. Furthermore, user adoption may be hampered by unfriendly interfaces and compatibility issues across various devices [6]. These limitations underscore the pressing need for a more reliable, efficient, and universally accessible solution in the market. Recent studies highlight a clear preference among consumers for conducting price comparisons online rather than in physical retail stores, as illustrated in the pie chart Fig. 1.

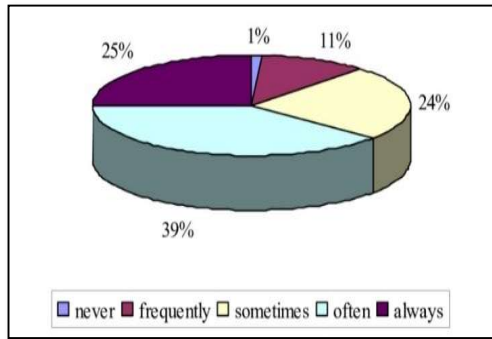


Figure 1. Consumer Preferences for Price Comparison: Retail Stores and Internet [16]

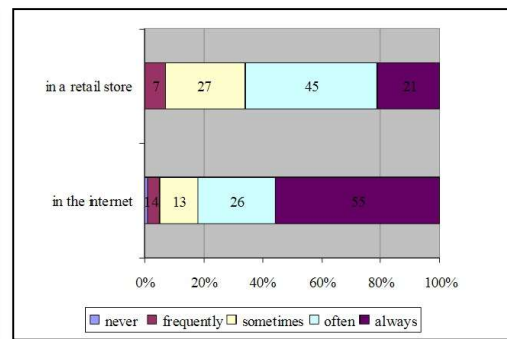


Figure 2. Consumer Preferences for Price Comparison: Retail Store vs. Internet [16]

This consumer behavior underscores the critical importance of developing effective online price comparison tools that cater to the demands of today's digitally savvy shoppers. To address these challenges, this research proposes the development of an advanced e-commerce pricing comparison platform using Flask. This platform automates the collection and comparison of prices from Flipkart and Amazon, leveraging Flask in conjunction with Selenium and BeautifulSoup for robust and scalable data extraction [10]. By adopting this approach, the platform ensures accurate real-time price comparisons while effectively managing dynamic content and adapting to changes in website structures.

The technology consolidates real-time pricing data from Flipkart and Amazon into a structured and intuitive user interface, enabling consumers to effortlessly search for products and view comparative prices in a cohesive manner. This streamlined approach significantly reduces the time and effort required for consumers to find the best deals online. Additionally, a browser extension enhances usability by providing instant access to price comparisons during online shopping sessions, thereby enhancing the overall shopping experience. The implementation of this solution has yielded positive outcomes, consistently delivering precise price comparisons that enhance the efficiency and reliability of online shopping compared to manual searches. By addressing critical gaps in the e-commerce ecosystem, this platform offers a straightforward and efficient solution that enhances consumer convenience and promotes informed decision-making in online shopping [16].

II. LITERATURE REVIEW

In recent years, the proliferation of e-commerce platforms has significantly altered consumer behavior and shopping dynamics. This section reviews several studies that explore various aspects of e-commerce, particularly focusing on price comparison tools, web scraping techniques, and consumer preferences.

Cano et al. (2023) conducted an analysis on the sustainable business models of e-marketplaces, highlighting the importance of consumer perspectives in shaping market dynamics [1]. Their study underscores the evolving nature of consumer expectations and the need for adaptable business strategies within e-commerce environments.

Troll et al. (2023) conducted a meta-analysis examining the effects of pricing strategies on consumer behavior, specifically comparing just-below versus round prices [2]. Their findings suggest that subtle pricing adjustments can influence consumer perceptions and purchasing decisions, illustrating the nuanced strategies employed by e-commerce platforms to optimize sales.

Shaikh et al. (2023) developed a price comparison tool utilizing web scraping techniques to aggregate pricing data from multiple e-commerce giants such as Amazon, eBay, and Walmart [3]. Despite its potential benefits, their study highlighted challenges related to data accuracy, real-time updates, and performance due to the dynamic nature of e-commerce platforms.

Sowmiya et al. (2023) focused on comparing product prices across various e-commerce websites, emphasizing the complexities involved in maintaining accurate and up-to-date pricing information [4]. Their research contributes to understanding the competitive landscape of online retail and the challenges faced by consumers in accessing reliable pricing data.

Rayalla et al. (2023) expanded on this theme by investigating product price comparisons on multiple e-commerce websites, highlighting the variability in pricing strategies employed by different platforms [5]. Their

findings underscore the importance of transparency and accessibility in fostering consumer trust and satisfaction in online shopping environments.

Mangio and Di Domenico (2022) discussed the implications of affiliate marketing programs in e-commerce, addressing issues related to authenticity and transparency in product endorsements [6]. Their insights contribute to understanding the ethical considerations surrounding digital marketing practices within the e-commerce sector.

Singh and Sahu (2022) conducted a feasibility assessment of e-commerce in developing countries, focusing on data collection and infrastructure challenges in markets such as Ethiopia [7]. Their study provides valuable insights into the unique barriers and opportunities for e-commerce expansion in emerging economies.

Kumar et al. (2022) explored the use of web scraping techniques for retrieving data from e-commerce websites, highlighting its applications in data aggregation and market analysis [8]. Their research underscores the technical advancements that enable efficient data extraction despite the complexities of website structures.

Rajendar et al. (2021) developed a price comparison website tailored for online shopping, emphasizing user-friendly interfaces and real-time data updates as critical features for enhancing consumer experience [9]. Their study contributes to improving accessibility and usability in online retail environments.

These studies collectively highlight the diverse methodologies and challenges inherent in e-commerce research, ranging from technical innovations in data extraction to consumer behavior analysis and market dynamics. The next section presents our proposed approach to address some of these challenges through the development of an advanced price comparison tool using Flask, Selenium, and BeautifulSoup.

III. METHODOLOGY

The primary objective of this study is to develop and implement a web scraping program capable of extracting product data from diverse e-commerce platforms such as Flipkart and Amazon. This includes retrieving information such as product names, prices, and images. The methodology outlined here details the procedures, tools, and techniques employed to achieve this objective.

A. Tools and Libraries

- Selenium WebDriver- Selenium WebDriver serves as a critical tool for automating interactions with web browsers. It enables the simulation of user actions on web pages, which is essential for extracting dynamic content not directly available in static HTML sources.
 - Browser Automation: Selenium WebDriver facilitates control over various web browsers (e.g., Chrome, Firefox), allowing actions like clicking, typing, and scrolling.
 - Headless Mode: Browsers operate in headless mode (--headless) to reduce resource consumption and enhance scraping speed.
- BeautifulSoup-The BeautifulSoup Python library is utilized for parsing HTML and XML documents, generating parse trees from parsed pages, and extracting desired data from specific HTML elements.
 - HTML Parsing: BeautifulSoup provides methods to navigate parse trees and extract data from HTML elements.
 - CSS Selectors: It supports CSS selectors, simplifying the identification of elements using class names, IDs, or other attributes.
- Requests- the Requests library manages HTTP requests, primarily for downloading product images.
 - HTTP GET Requests: Used to fetch web page content and images.
 - Response Handling: Provides functionalities to manage HTTP responses, including status codes and content.
- PIL (Python Imaging Library) - PIL, or Pillow, is employed for image data manipulation and validation.
 - Image Processing: PIL can open, manipulate, and save image files.
 - Validation: Ensures downloaded images are correctly formatted and valid.

B. Web-Scraping Process

- Configuring WebDriver- Selenium WebDriver is configured with specific options to initiate web scraping[13]:
 - Headless Mode: `options.add_argument('--headless')` runs Chrome without a GUI.

- User-Agent Rotation: Rotates user-agent strings in HTTP headers to mimic diverse browsers, reducing the risk of detection.
 - Proxy Usage: Incorporates proxies to rotate IP addresses, mitigating IP bans.
- Handling Anti-Scraping Measures- Various techniques are employed to circumvent anti-scraping mechanisms:
 - Dynamic Delays: Introduces random delays between actions to simulate human behavior.
 - CAPTCHA Detection: Identifies CAPTCHA prompts for manual or automated handling (future enhancement).

C. Web-Scraping Flowchart

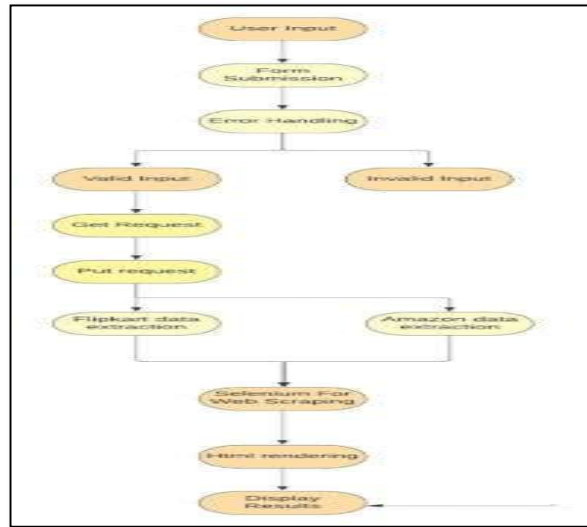


Figure 3. Web-Scraping flowchart

The flowchart shown in Fig. 3 illustrates the structured process, starting with user input submission and concluding with result display. Upon user input submission, validated data triggers the construction of search URLs for Flipkart and Amazon. Selenium WebDriver in a headless browser manages dynamic content and anti-scraping measures, parsing HTML content via BeautifulSoup to extract product titles, prices, and images. Flask formats and renders HTML for user presentation, with error handling ensuring a seamless experience. After extraction, the data is validated to ensure completeness and accuracy. This includes verifying that all required fields (title, price, image URL) are populated. The images are downloaded using the Requests library and validated using PIL. Furthermore, special attention is given to the image URL field. It is essential to verify that each product record includes a valid URL pointing to the corresponding product image. To accomplish this, the system utilizes the Requests library to download the image associated with each product from its provided URL. Once downloaded, the integrity and correctness of each image are validated using the Python Imaging Library (PIL).

D. Data Extraction

- Flipkart
 - URL Construction: Generates search URLs based on user queries.
 - Page Loading: Uses Selenium WebDriver to load URLs.
 - HTML Parsing: BeautifulSoup parses page sources.
 - Data Extraction: Extracts product details using CSS selectors.
- Amazon
 - URL Construction: Similar to Flipkart, constructs search URLs.
 - Page Loading: Loads URLs using Selenium WebDriver.
 - Handling Traffic and Anti-Scraping Measures: Uses custom headers and proxies.
 - HTML Parsing and Data Extraction: Parses page sources, extracts product data.

E. Data Validation Process

After extraction, the data is validated to ensure completeness and accuracy. This includes verifying that all required fields (title, price, image URL) are populated. The images are downloaded using the Requests library and validated using PIL.

Furthermore, special attention is given to the image URL field. It is essential to verify that each product record includes a valid URL pointing to the corresponding product image. To accomplish this, the system utilizes the Requests library to download the image associated with each product from its provided URL. Once downloaded, the integrity and correctness of each image are validated using the Python Imaging Library (PIL).

F. General Architecture of Web-Scraping

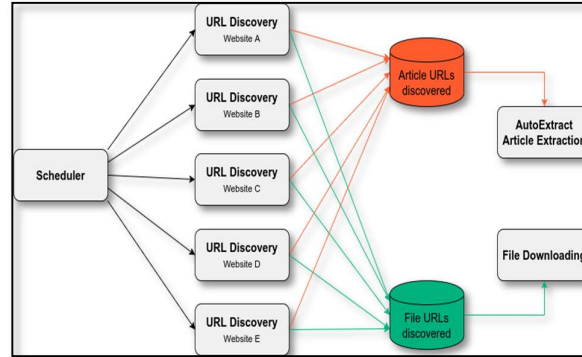


Figure 4. General Architecture of Web-Scraping

The generalized architecture adapts for scraping product data from e-commerce platforms, featuring user input triggering URL discovery, scraping product pages, downloading images, and integrating results for display.

The Fig. 4 Illustrates a generalized web scraping architecture that can be adapted for our project of scraping product data from e-commerce websites like Flipkart and Amazon[11]. Here's how each component relates to our project:

1. Scheduler: In propose method, the scheduler represents the user's action of submitting a product name. It initiates the scraping process by triggering URL discovery for the specified product on various e-commerce websites [15].
2. URL Discovery: For each website (e.g., Flipkart, Amazon), URL discovery involves generating search URLs based on the user's input. In our project, we dynamically create search URLs for the product on both Flipkart and Amazon.
3. Article URLs Discovered: This can be seen as the initial set of product URLs discovered during the search. These URLs point to the individual product pages that need to be scraped for detailed information.
4. File URLs Discovered: In our project, these are analogous to the URLs of the product images that we need to download and display in the final results.
5. AutoExtract Article Extraction: This process represents the use of Selenium and BeautifulSoup in our project to automatically extract product details such as titles, prices, and images from the discovered URLs.
6. File Downloading: This step is directly related to downloading the product images from the URLs discovered during the scraping process, allowing us to display these images in the final rendered HTML.

G. Error Handling, Integration with Flask

The system is designed to handle various errors gracefully, such as network issues, changes in webpage structure, and server-side blocks. For network issues, retry mechanisms are implemented to handle network-related errors efficiently. Changes in webpage structure are monitored through error logging, and CSS selectors are updated accordingly. To handle server-side blocks, the system employs techniques such as rotating user-agents and IP addresses to avoid detection and subsequent blocks.

Furthermore, the extracted data is seamlessly integrated into a Flask web application, providing users with an intuitive interface for querying products and viewing results. The Flask application consists of routes such as the index route for rendering the homepage and a search route for processing user queries. Upon receiving a search

query via POST request, the application retrieves data from e-commerce websites using methods like `flipkart()` and `search_amazon()`, presenting the results dynamically on the results page [12]. In conclusion, this methodology provides a robust approach to web scraping, effectively managing dynamic content, anti-scraping measures, and data validation. Utilizing Selenium WebDriver, BeautifulSoup, Requests, and PIL ensures efficient and reliable scraping. Future enhancements could encompass automated CAPTCHA handling and expanded platform scraping capabilities.

IV. RESULTS AND DISCUSSION

Promising capability has been demonstrated by the web scraping system created to gather product data from e-commerce platforms, specifically Flipkart and Amazon. Using BeautifulSoup in conjunction with Selenium WebDriver, the system successfully pulls names, prices, and photos of products from both platforms. An easy-to-use interface for product searches and result visualisation is provided to users by this data retrieval mechanism, which is smoothly incorporated into a Flask online application.



Figure 5. Product Search Result

The search results of the web-scraping project are shown in Fig. 5, the user is first prompted by the section where they are shown the product's search box and can compare prices. They will then be directed to the result page after that.



Figure 6 Output on Terminal

The following Fig. 6 illustrates the output on the Terminal when the User Fires the Search Query the Output is First Displayed on the terminal and then redirected on the Results Page.

Performance and scalability will continue to be important factors in the future. Scalability becomes critical as the system handles more user requests and data volume. For the system to remain responsive and reliable, it will be essential to assess its performance under high loads and investigate optimization techniques like distributed computing and asynchronous processing.

Reliability and maintenance are constant objectives for the sustainability of the system. To avoid data extraction errors, regular webpage structural modifications and updates to the scraping mechanism are required. Over time, the system's resilience will be increased and the troubleshooting process will be streamlined by implementing automated testing and thorough error logging tools.

Careful consideration should also be given to ethical issues. Although web scraping provides insightful information, it is crucial to follow moral standards and honor terms of service of websites. Maintaining ethical integrity requires minimizing effects on website performance and protecting against unauthorized access to sensitive data.

TABLE I. COMPARISON TABLE: EXISTING SYSTEM V/S PROPOSED SYSTEM

Existing System	Proposed System
Relies solely on BeautifulSoup for scraping static content, which may not effectively handle dynamic JavaScript-rendered content.[2]	Utilizes Selenium WebDriver for dynamic content rendering, allowing comprehensive scraping of JavaScript-rendered elements.
Limited error handling mechanisms, leading to frequent failures and disruptions in data extraction.[8]	Implements robust error-handling mechanisms, including retry mechanisms, CSS selector updates, and user-agent rotation, to ensure graceful handling of errors and minimize scraping disruptions.
Limited scalability due to reliance on static scraping methods, leading to difficulties in adapting to changes in webpage structures and scaling to larger datasets.[6]	Highly scalable architecture capable of handling large datasets and adapting to changes in webpage structures through dynamic scraping techniques and flexible error handling.
Basic integration with Flask, offering limited user interaction and data presentation capabilities.[1]	Seamlessly integrates with Flask, providing a user-friendly interface for querying products and viewing results in a structured and interactive manner.

V. CONCLUSION

This project offers a reliable and effective web scraping method that can be used to retrieve product information from e-commerce sites such as Amazon and Flipkart. The solution efficiently manages both static and JavaScript-rendered content by utilizing BeautifulSoup for HTML parsing and Selenium WebDriver for dynamic content rendering. High reliability and resilience against server-side blocks and network problems are ensured by the integration of comprehensive error-handling methods, such as retry strategies and user-agent rotation. In addition, the system's faultless integration with Flask offers an intuitive query and result presentation interface. With these improvements, the drawbacks of the current static scraping techniques are addressed, providing a scalable and reliable solution for precise and consistent data extraction from contemporary dynamic websites. Future improvements to the suggested method, include automated CAPTCHA resolution and sophisticated proxy management, in order to strengthen its resistance against anti-scraping techniques. Deeper insights and a wider range of applications can also be obtained by extending the scraping capabilities to more e-commerce platforms and including sophisticated data analysis and visualisation technologies. The suggested solution would make sure customers have access to correct and thorough information by continuously monitoring and extracting current product data from several e-commerce websites.

REFERENCES

- [1] Cano, J. A., Allec, A., & Landono-Pineda. (2023). Sustainable business model of e-marketplaces: An analysis from the consumer perspective.
- [2] Troll, E. S., Frankenbach, J., Friese, M., & Loschelder, D. D. (2023). A meta-analysis on the effects of just-below versus round prices.
- [3] Shaikh, A., Khan, R., Panokher, K., Ranjan, M. K., & Sonaje, V. (2023). E-commerce Price Comparison Website Using Web Scraping.
- [4] Sowmiya, M., Cs, S., Raja, M. M., & Kumar, S. S. (2023). Price Comparison for Products in Various Ecommerce Websites.

- [5] Rayalla, P. S., Vivek, S. V., G, S., Sanju, S., & Kumar, S. V. (2023). Product Price Comparison On Multiple E-Commerce Websites.
- [6] Mangio, F., & Di Domenico, G. (2022). All that glitters is not real affiliation: How to handle affiliate marketing programs in the era of falsity.
- [7] Singh, N., & Sahu, O. (2022). Feasibility assessment for E-commerce: A data collection from a developing country (Ethiopia).
- [8] Kumar, G. V., Hema, M. S., & Aishwarya, R. (2022). Web Scraping for E-commerce Websites.
- [9] Rajendar, S., Manikanta, K., Mahendar, M., & Madhavi, K. (2021). Price Comparison Website for Online Shopping.
- [10] Henrys, K. (2021). Importance of scraping in e-commerce and marketing.
- [11] Wang, Y. (2021). Research on E-commerce Platform of Online Shopping Consumers.
- [12] Wang, D., & Li, W. (2021). Optimization algorithm and simulation of supply chain coordination based on cross-border E-commerce network platform.
- [13] Priya, Matta, N., Sharma, D., & Pant, B. (2021). Web Scraping: Applications and Scraping Tools.
- [14] Lei, Z., Zhou, H., Ye, S., Hu, W., & Liu, G.-P. (2020). Cost-Effective Server-side Re-deployment for Web-based Online Laboratories Using NGINX Reverse Proxy.
- [15] Ambre, A., Gaikwad, P., Pawar, K., & Patil, V. (2019). Web and Android Application for Comparison of E-Commerce Products.
- [16] Andersone, I., & Gaile-Sarkane, E. (2018). Behavioral differences in consumer purchasing behavior between online and traditional shopping: Case of Latvia.
- [17] Mohite, S., Jadhav, S., Aggarwal, A., Shukla, A., Jain, D., & Jaiswal, S. (2023). Insight Now: A Cross-Platform News Application for Real-Time and Personalized News Aggregation. In IEEE International Carnahan Conference on Security Technology (ICCST), Pune, India (pp. 1-7). doi: 10.1109/ICCST59048.2023.10474234.
- [18] Jadhav, S., Bhavik, M., Barde, R., Bhosale, V., Bura, R., & Amune, A. (2023). Decentralized Query Solving Educational Website. In Second International Conference on Informatics (ICI), Noida, India (pp. 1-6). doi: 10.1109/ICI60088.2023.10421200.