Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Bachelor Thesis

Web Scraping Methodologies for Image Dataset Creation: A motorcycle Imagery Case Study

Artem Gubin

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Artem Gubin

Informatics

Thesis title

Web Scraping Methodologies for Image Dataset Creation: A motorcycle Imagery Case Study

---

Objectives of thesis

The main objective of this thesis is to comprehensively analyse web scraping methodologies and techniques, culminating in the development of a motorcycle image dataset divided into classes by manufacturer, model, and years of production as a practical case study.
Partial objectives:

- Study of Web Scraping Tools and Techniques. To select and analyse various tools and techniques suitable for web scraping.
- Challenges in Web Scraping. To identify, analyse, and address the various challenges encountered in the process of web scraping.
- Legal and Ethical Considerations. To investigate the legal implications and ethical considerations associated with web scraping.
- Scalability Strategies. To explore and propose strategies for scaling web scraping processes, focusing on improving data extraction rates and managing larger data volumes.
- Evaluation of Scraped Images. To evaluate the quality and comprehensiveness of scraped images for the dataset.
- Motorcycle Image Dataset Creation. To apply the studied web scraping methodologies by creating a categorized dataset detailing the make, model, and years.

Methodology

The methodology of the thesis is divided into two main parts: theoretical and practical.

The theoretical part will primarily involve a comprehensive examination and analysis of professional and scientific literature. This includes examining various web scraping tools, frameworks, and techniques, as well as understanding the legal and ethical implications of web scraping. The comparative analysis of different methodologies will be crucial to identify the most efficient tools for image dataset creation, with a special focus on scalability and handling large volumes of data.

The practical part will be focused on applying the web scraping techniques explored in the theoretical study to create a motorcycle image dataset. This process will involve setting up a scalable web scraping system to collect images from diverse online resources, ensuring that images are categorized by their manufacturer, model, and years of production. Following the data gathering, an evaluation of the dataset's quality and comprehensiveness will be conducted in order to ensure its relevance and utility for potential applications.

The proposed extent of the thesis

30-40 pages

Keywords

Keywords: Web Scraping, Image Dataset, Vehicle Make and Model Recognition (VMMR), Motorcycle dataset, Selenium, Beautiful Soup, YOLO, Data Filtration.

Recommended information sources

Heydt, Michael. Python Web Scraping Cookbook: Over 90 Proven Recipes to Get You Scraping with Python, Microservices, Docker, and AWS. Packt Publishing, 2018. ISBN 978-1787285217

Mitchell, Ryan. Web Scraping with Python: Collecting More Data from the Modern Web. 2nd edition. O'Reilly Media, Inc., 2018. ISBN 978-1491985571

Vanden Broucke, Seppe; Baesens, Bart. Practical Web Scraping for Data Science: Best Practices and Examples with Python. 1st edition. Apress Berkeley, CA, 2018. ISBN 978-1-4842-3581-2

Expected date of thesis defence

2023/24 SS – PEF

The Bachelor Thesis Supervisor

Ing. Martin Pelikán, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 16. 2. 2024	Electronic approval: 16. 2. 2024
Ing. Martin Pelikán, Ph.D.	**doc. Ing. Tomáš Šubrt, Ph.D.**
Head of department	Dean

Prague on 15. 03. 2024

Declaration

I declare that I have worked on my bachelor thesis titled "Web Scraping Methodologies for Image Dataset Creation: A motorcycle Imagery Case Study" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break any copyrights.

In Prague on March 15, 2024 ______________________________

Acknowledgement

I would like to thank my supervisor, Ing. Martin Pelikán, Ph.D., for his guidance and support, and extend my special heartfelt appreciation to my consultant, Ing. Martin Čejka, for his exceptional mentorship, dedicated time, insightful instructions, and constant encouragement. His support was a key factor in the successful completion of this bachelor thesis.

Web Scraping Methodologies for Image Dataset Creation: A motorcycle Imagery Case Study

Abstract

This thesis investigates the methods and techniques of web scraping and subsequently applies them in the creation of a structured dataset of images for future computer vision tasks. The work is divided into theoretical and practical parts.

The theoretical part describes the objectives of web scraping and provides a thorough analysis of its tools and techniques. Furthermore, it addresses the current challenges of web scraping, including its technical, ethical, and legal aspects, and provides an in-depth discussion of these issues.

The practical part demonstrates the application of these methodologies, supported by tools like Selenium and Beautiful Soup, for collecting and processing an image dataset of motorcycles. This process involves developing scrapers, integrating them into a general scraping system, scaling the system, and using neural network filtering to ensure data quality. The final compiled dataset, categorized by manufacturer, model, and years of production, showcases the practical use of web scraping tools for Vehicle Make and Model Recognition (VMMR) tasks. Nevertheless, in summary, the thesis emphasizes the significant potential of employing web scraping methods for the creation of datasets in the field of computer vision tasks.

Keywords: Web Scraping, Image Dataset, Vehicle Make and Model Recognition (VMMR), Motorcycle dataset, Selenium, Beautiful Soup, YOLO, Data Filtration.

Metodiky web scrapingu pro tvorbu obrazových datasetů: Případová studie motocyklových obrázků

Abstrakt

Tato práce zkoumá metody a techniky web scrapingu, které následně aplikuje při tvorbě strukturovaného datasetu obrázků pro budoucí úlohy počítačového vidění. Práce je rozdělena na teoretickou a praktickou část.

Teoretická část popisuje cíle web scrapingu a poskytuje důkladnou analýzu jeho nástrojů a technik. Dále teoretická část otevírá téma aktuálních výzev web scrapingu, zahrnující technické, etické a právní aspekty, a podrobně popisuje cíle a specifické výzvy této oblasti.

Praktická část demonstruje aplikaci těchto metod s podporou nástrojů Selenium a Beautiful Soup pro sběr a zpracování obrazového datasetu motocyklů. Proces zahrnuje vývoj scrapovacích nástrojů, jejich integraci do obecného systému scrapingu, škálování systému a použití filtrace neuronových sítí pro zajištění kvality dat. Finální zkompilovaný dataset, kategorizovaný podle výrobce, modelu a roků výroby, prezentuje praktické využití nástrojů web scraping pro úlohy Rozpoznávání značek a modelů vozidel (VMMR). Obecně však práce poukazuje na potenciál využití metod web scrapingu při tvorbě datasetu pro úlohu počítačového vidění.

Klíčová slova: Web Scraping, Obrazový Dataset, Rozpoznávání Značek a Modelů Vozidel (VMMR), Dataset Motocyklů, Selenium, Beautiful Soup, YOLO, Filtrace Dat.

Table of Contents

1 Introduction

In the era of digitalization, the huge volume of data available online offers unparalleled opportunities and challenges. Among these, the extraction and utilization of visual content, particularly images, are crucial in numerous domains, from academic research to commercial applications. Web scraping, a technique used for extracting data from websites, has become an essential tool for gathering information in an efficient and automated manner. The utility of web scraping extends across a diverse range of sectors, encompassing banking and finance, marketing, socio-political analysis, research and academics, cyber security, healthcare, and many more [29][47][20].

The advent of web scraping has completely transformed the methods we use to gather and analyse visual data. Targeted image datasets have gained significant value in specialized domains like automotive analysis. These datasets are used for numerous purposes, including make & model recognition [28], market trend analysis, and consumers experience analysis [21]. The creation of a specialized motorcycle image dataset exemplifies this trend, showcasing the effectiveness of web scraping in collecting and classifying images based on specific criteria like manufacturer, model, and years.

However, the process of web scraping, specifically for creating a comprehensive and organized image dataset, is filled with challenges. These encompass not only the technical aspects of extracting and managing large volumes of data, but also navigating the complex legal and ethical landscape related to digital information. The delicate equilibrium between the accessibility of data and respect for intellectual property and privacy rights is a crucial factor to consider in the web scraping process.

This thesis aims to explore the methodologies and techniques of web scraping, with a specific focus on its application in building a structured image dataset. This study aims to provide significant insights into the effective utilization of web scraping in modern data collection and analysis. This will be accomplished by studying a variety of tools and frameworks, as well as addressing the legal, ethical, and technical problems that are involved.

2 Objectives and Methodology

2.1 Objectives

The main objective of this thesis is to comprehensively analyse web scraping methodologies and techniques, culminating in the development of a motorcycle image dataset divided into classes by manufacturer, model, and years of production as a practical case study.

Partial objectives:

- Study of Web Scraping Tools and Techniques. To select and analyse various tools and techniques suitable for web scraping.
- Challenges in Web Scraping. To identify, analyse, and address the various challenges encountered in the process of web scraping.
- Legal and Ethical Considerations. To investigate the legal implications and ethical considerations associated with web scraping.
- Scalability Strategies. To explore and propose strategies for scaling web scraping processes, focusing on improving data extraction rates and managing larger data volumes.
- Evaluation of Scraped Images. To evaluate the quality and comprehensiveness of scraped images for the dataset.
- Motorcycle Image Dataset Creation. To apply the studied web scraping methodologies by creating a categorized dataset detailing the make, model, and years.

2.2 Methodology

The methodology of the thesis is divided into two main parts: theoretical and practical.

The theoretical part will primarily involve a comprehensive examination and analysis of professional and scientific literature. This includes examining various web scraping tools, frameworks, and techniques, as well as understanding the legal and ethical implications of web scraping. The comparative analysis of different methodologies will be crucial to identify the most efficient tools for image dataset creation, with a special focus on scalability and handling large volumes of data.

The practical part will be focused on applying the web scraping techniques explored in the theoretical study to create a motorcycle image dataset. This process will involve setting up a scalable web scraping system to collect images from diverse online resources, ensuring that images are categorized by their manufacturer, model, and years of production. Following the data gathering, an evaluation of the dataset’s quality and comprehensiveness will be conducted in order to ensure its relevance and utility for potential applications.

3 Literature Review

3.1 Overview of Web Scraping

Web scraping, also known as web harvesting or web data extraction, is the process of using automated tools to extract large volumes of data from websites [29][47][39]. This process involves an automated program querying a web server, requesting data in the form of HTML (HyperText Markup Language) or other files that comprise a web page, and then parsing this data to extract the necessary information [29]. Web scraping is more than just data retrieval; it’s a complex process of automating interactions with web browser, mimicking human browsing patterns but on a significantly accelerated and more effective level.

This automated process is crucial in various domains, allowing for the rapid and systematic collection of data. Web scrapers navigate through web pages, identify needed content, and extract it in a structured format. This approach is especially effective for collecting data that is often updated or distributed across multiple web pages.

The scope of web scraping extends to numerous applications, ranging from business intelligence and market research to academic research and journalism. For instance, in the business sector, companies use web scraping to continuously analyse competitors pricing strategies and collect consumer feedback from social media and review platforms [20].

To summarize, web scraping is a vital and complex process that has an important impact in diverse fields. This section of the thesis aims to explore the myriad challenges it presents, encompassing ethical, legal, and technical aspects. Understanding these complexities is essential for grasping the technique’s full potential and implications.

3.1.1 History of Web Scraping

Web scraping has evolved significantly alongside the web itself. This progress reflects the advancements in web technologies, addressing the complexities of data management and privacy. In this section, we'll look at a brief history of web scraping from its early days to the present and how this practice has changed.

3.1.1.1 Early Development (Late 1990s – Early 2000s)

Web scraping emerged alongside the early stages of the World Wide Web. Three key features of the World Wide Web played a crucial role in web scraping process development:

- URLs (Uniform Resource Locators). URLs are the addresses used to access web pages. URLs provide a standardized way to find resources on the internet, such as images, files, and web pages. In web scraping, URLs play a key role as they define specific locations from which the necessary data needs to be retrieved.
- Hyperlinks. Hyperlinks are references or links leading from one page to another. In the context of web scraping, hyperlinks are crucial for navigating through various pages to systematically collect data from multiple sources.
- Web pages. A web pages are the documents written in HTML, containing various types of data like text, images, and multimedia. Web scrapers analyse these pages to extract relevant data, highlighting the ability to process and interpret the content of web pages fundamental to web scraping.

In the early days, data extraction techniques were basic and typically required manual processing and simple scrips for parsing HTML. During this period, the groundwork for automated web scraping was established through the creation of early automatising tools. These technologies were fundamental yet essential for laying the foundation for future innovations, with a primary focus on extracting text from static web pages.

3.1.1.2 Technological Advancements (Mid-2000s)

The mid-2000s marked a turning point for standard web scraping techniques. This period was characterized by the significant growth of technologies such as JavaScript and AJAX [50], which significantly changed the way content was loaded and displayed on web pages, making them more dynamic and introducing a more complex architecture. Thus, basic parsing tools capable of parsing static HTML have become less effective. Despite this, it was during this time that tools such as Beautiful Soup and Scrapy emerged. Beautiful Soup is a library for the Python programming language that was created to simplify the parsing of HTML and XML documents. Similarly, Scrapy, an open-source Python framework, was created for data extraction from websites. While both tools excelled at handling static content, they were not inherently equipped to manage dynamic content. A deeper examination of these tools, focusing on their roles and limitations in web scraping, will be presented in subsequent sections of this thesis.

As the web continued to evolve, particularly with the increasing use of JavaScript and AJAX, it became clear that more advanced techniques for scraping dynamic content were required. This led to the adoption of tools like Selenium, which could automate web browsers to interact with dynamic web elements, enabling the extraction of content that traditional scraping tools couldn't access.

3.1.1.3 Integration with Big Data and AI (2010s – Present)

The exponential growth of data in recent years has significantly impacted the field of web scraping. According to Statista [42], the amount of data generated annually has been consistently growing since 2010. From 2 zettabytes in 2010 (1 zettabyte is equal to 1,000,000,000 terabytes), this figure has grown approximately by 60 times, reaching 120 zettabytes in 2023. Furthermore, as indicated in **Figure 1**, this rapid expansion in data generation is expected to continue with an expected 181 zettabytes in 2025.

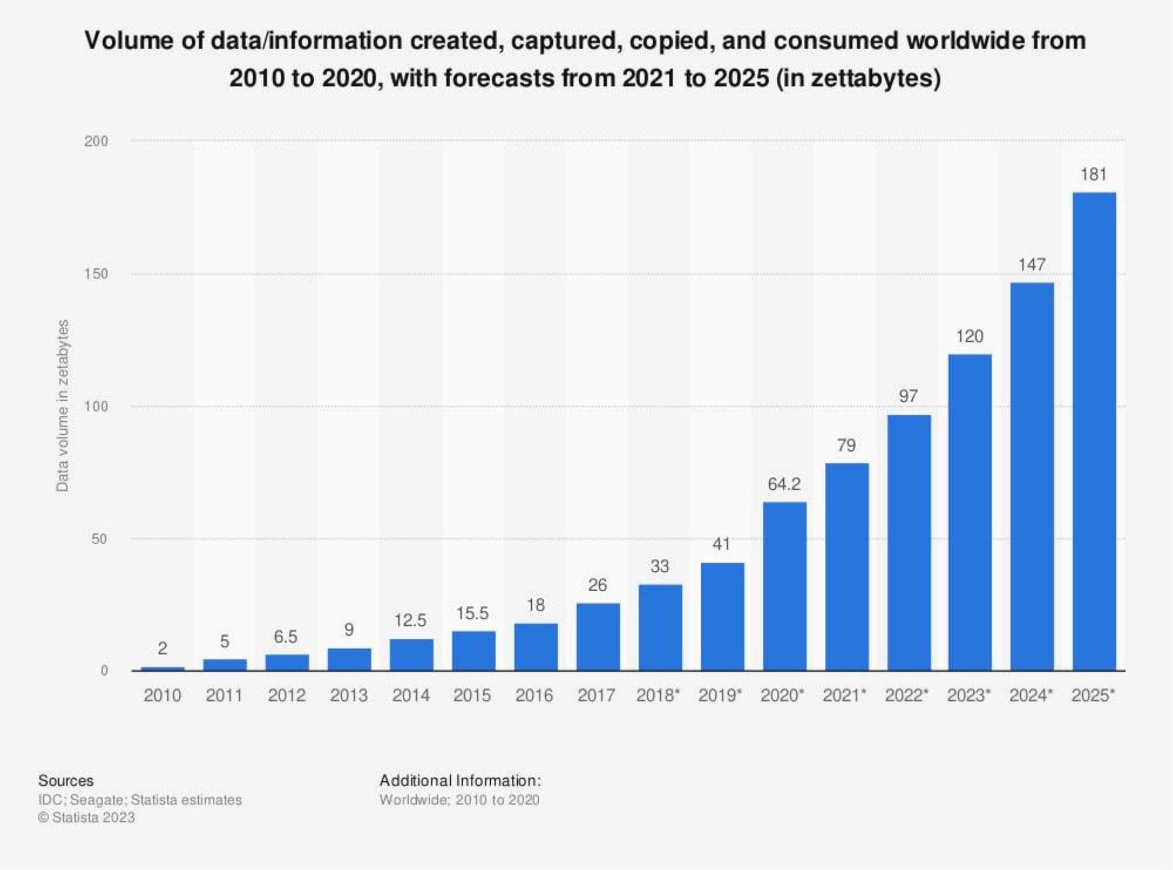


Figure 1. Amount of data generated worldwide. Source: [42]

With the integration of big data and AI, the field of web scraping experienced changes, entering a new era. AI, especially through the use of machine learning techniques, has enhanced the capabilities of web scraping tools, enabling them to extract valuable data more easily. Machine learning algorithms have improved the precision of data extraction from various web sources, including those with unstructured formats [13].

Alongside these advancements, the incorporation of Natural Language Processing (NLP) may serve as another example of machine learning integration into web scraping [13][29]. Using NLP can help when pages are not only structurally complex but also semantically rich. In such cases, this method can play an important role in extracting and interpreting the multi-level meanings and contexts found in text data. This approach can be useful in a wide range of applications. For example, NLP can be used for the extraction of data from speech transcriptions in forums, email messages, newspapers, articles, resumes, etc. [13]

3.2 Web Scraping Tools and Techniques

This section of the thesis is dedicated to exploring the fundamental aspects of web scraping. The primary objectives of this section are twofold: firstly, to conduct an extensive analysis of various web scraping techniques, including manual parsing, HTML parsing, DOM parsing, XPath, and the use of APIs. Secondly, to thoroughly examine key tools for web scraping such as Selenium, Beautiful Soup, and Scrapy. By examining these techniques and tools, the sections seek to provide a thorough understanding of the complexities and efficiencies in the web scraping process.

3.2.1 Web Scraping Techniques

3.2.1.1 Manual Scraping

Although the term “web scraping” is most commonly defined as an automated process for data collection without user manual interaction [29][47][39], a number of studies designate the manual copy-pasting method as a valid web scraping technique [26][20].

Manual scraping is typically executed by copying and pasting data into a local file, such as a spreadsheet. This technique is simple and straightforward, but it requires attention to the details and a user who must manually navigate through web pages, identify relevant data, and manually extract it. This technique is usually used in cases where:

- The amount of necessary data to collect is minimal.
- On websites where automated tools struggle due to the complex design of the website or security measures.
- In cases where a high level of accuracy is required and there is a risk of errors with automated processes.

3.2.1.2 HTML parsing

HTML parsing in web scraping is an essential technique used for extracting information from web sites. HyperText Markup Language serves as a standard markup language for creating and structuring web pages and web applications. A webpage consists of a sequence of elements, indicated by tags, that determine the structure and styling of the content [47]. Knowledge of HTML parsing is crucial for web scraping since it allows the extraction and manipulation of data from these elements.

The process usually starts by fetching the target webpage’s HTML code, often using HTTP requests. The next step involves using specialized libraries, which will be explored in details later in this chapter of the thesis. These libraries parse HTML, converting it into a structured, navigable format, enabling precise navigation and extraction of data based on specific tags, attributes, and their hierarchical relationships.

However, HTML parsing has its own limitations, especially when dealing with asynchronous dynamic content. In such cases, HTML retrieved via standard HTTP requests might not contain all the data visible after rendering in a web browser. Such complexities highlight the need for a good understanding of both static and dynamic content handling.

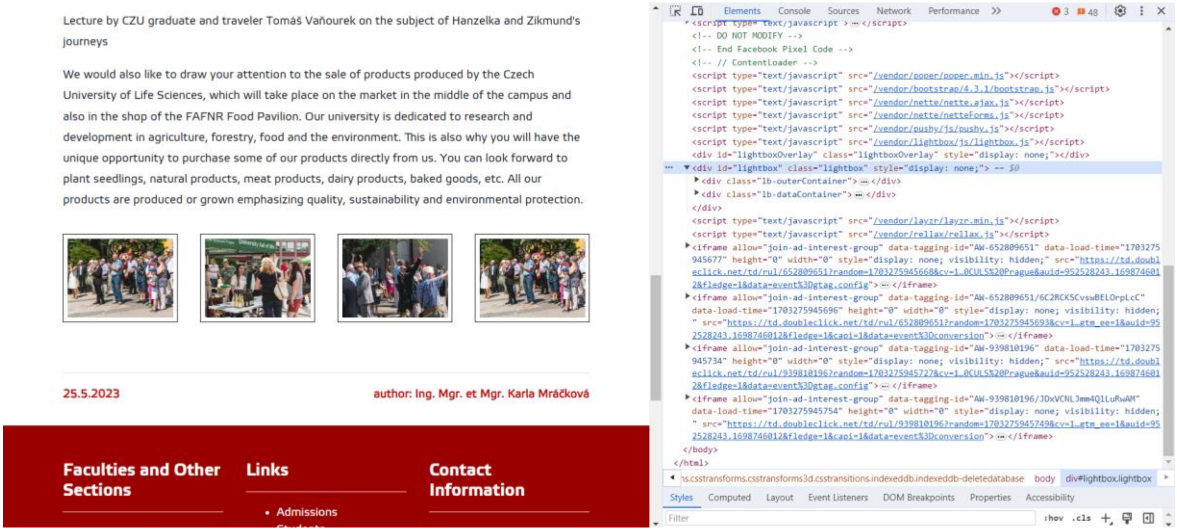
3.2.1.3 DOM parsing

The Document Object Model (DOM) is the concept developed and standardized by the World Wide Web Consortium (W3C). DOM presents a web page as a hierarchical tree structure, and this structure is central to understanding and manipulating the content of a web page, especially when dealing with dynamic content.

The key component of DOM parsing lies in its approach to web page interpretation. Unlike traditional methods that rely on the static HTML, DOM parsing involves the use of a functional web browser or browser-like environment [26][20][16]. Such a setup allows for the execution of client-side scripts, which are usually responsible for generating dynamic content on the webpage. Therefore, the resultant DOM tree includes not only the basic HTML elements, but also any content generated or modified by these scripts.

Central to this technique are tools like Selenium or Puppeteer, which simulate the functionality of web browsers. These tools are designed to render web page and run embedded scripts, thereby creating a DOM tree that represents the final content as it appears to end-users. This approach allows for the retrieval of data that would otherwise be unattainable through the basic HTML parsing described in **3.2.1.2**. Although these tools are briefly mentioned here, a more comprehensive examination of their capabilities and applications will be presented later in this chapter, providing a more detailed overview of their roles in the web scraping process.

Building upon this, the role of browser developer tools becomes clear. These integrated into browser tools provide developers with a real-time view of the DOM [16] and how it changes in response to user interactions and script executions. **Figure 2**, viewed through developer tools in a web browser Chrome, exemplify the use of such tools. On the figure, we can see a series of thumbnail images within a gallery on the ČZU Faculty of Economics and Management news article web page. Upon clicking one of these thumbnails, **Figure 3** illustrates the changes made to the DOM: a lightbox component has been added to the layout, including an enlarged image that is positioned in the middle of the viewport. The new element is dynamically inserted and accompanied by style alterations, including adjustments to the *z-index* and *visibility* values. These alterations demonstrate the interactive features of DOM, facilitated by JavaScript, and illustrate the kind of dynamic content manipulation that must be considered in the web scraping process.



Lecture by CZU graduate and traveler Tomáš Vaňourek on the subject of Hanzelka and Zikmund's journeys

We would also like to draw your attention to the sale of products produced by the Czech University of Life Sciences, which will take place on the market in the middle of the campus and also in the shop of the FAFNR Food Pavilion. Our university is dedicated to research and development in agriculture, forestry, food and the environment. This is also why you will have the unique opportunity to purchase some of our products directly from us. You can look forward to plant seedlings, natural products, meat products, dairy products, baked goods, etc. All our products are produced or grown emphasizing quality, sustainability and environmental protection.

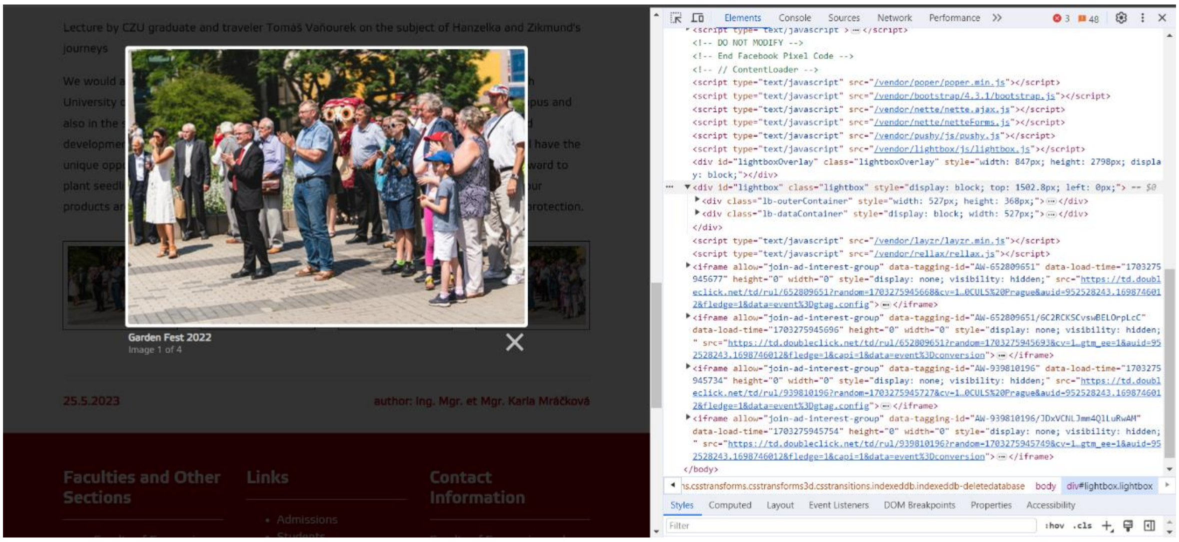
25.5.2023 author: Ing. Mgr. et Mgr. Karla Mráčková

Faculties and Other Sections | Links | Contact Information

- Admissions

```
<script type="text/javascript">…</script>
<!-- DO NOT MODIFY -->
<!-- End Facebook Pixel Code -->
<!-- // ContentLoader -->
<script type="text/javascript" src="/vendor/poper/poper.min.js"></script>
<script type="text/javascript" src="/vendor/bootstrap/4.3.1/bootstrap.js"></script>
<script type="text/javascript" src="/vendor/nette/nette.ajax.js"></script>
<script type="text/javascript" src="/vendor/nette/netteForms.js"></script>
<script type="text/javascript" src="/vendor/pushy/js/pushy.js"></script>
<script type="text/javascript" src="/vendor/lightbox/js/lightbox.js"></script>
<div id="lightboxOverlay" class="lightboxOverlay" style="display: none;"></div>
▼<div id="lightbox" class="lightbox" style="display: none;"> == $0
  ▶<div class="lb-outerContainer">…</div>
  ▶<div class="lb-dataContainer">…</div>
</div>
<script type="text/javascript" src="/vendor/layzr/layzr.min.js"></script>
<script type="text/javascript" src="/vendor/rellax/rellax.js"></script>
▶<iframe allow="join-ad-interest-group" data-tagging-id="AW-652809651" data-load-time="1703275
945677" height="0" width="0" style="display: none; visibility: hidden;" src="https://td.doubl
eclick.net/td/rul/652809651?random=1703275945668&cv=1…0CULS%20Prague&auid=952528243.169874601
2&fledge=1&data=event%3Dgtag.config">…</iframe>
▶<iframe allow="join-ad-interest-group" data-tagging-id="AW-652809651/6C2RCKSCvswBELOrpLcC"
data-load-time="1703275945696" height="0" width="0" style="display: none; visibility: hidden;
" src="https://td.doubleclick.net/td/rul/652809651?random=1703275945693&cv=1…gtm_ee=1&auid=95
2528243.1698746012&fledge=1&capi=1&data=event%3Dconversion">…</iframe>
▶<iframe allow="join-ad-interest-group" data-tagging-id="AW-939810196" data-load-time="1703275
945734" height="0" width="0" style="display: none; visibility: hidden;" src="https://td.doubl
eclick.net/td/rul/939810196?random=1703275945727&cv=1…0CULS%20Prague&auid=952528243.169874601
2&fledge=1&data=event%3Dgtag.config">…</iframe>
▶<iframe allow="join-ad-interest-group" data-tagging-id="AW-939810196/JDxVCNLJmm4Q1LuRwAM"
data-load-time="1703275945754" height="0" width="0" style="display: none; visibility: hidden;
" src="https://td.doubleclick.net/td/rul/939810196?random=1703275945749&cv=1…gtm_ee=1&auid=95
2528243.1698746012&fledge=1&capi=1&data=event%3Dconversion">…</iframe>
</body>
</html>
```

Figure 2. DOM Tree of ČZU website before user interaction. Source: own.



Lecture by CZU graduate and traveler Tomáš Vaňourek on the subject of Hanzelka and Zikmund's journeys

We would a… University o… pus and also in the… developmen… have the unique opp… ward to plant seedli… ur products ar… protection.

Garden Fest 2022
Image 1 of 4

25.5.2023 author: Ing. Mgr. et Mgr. Karla Mráčková

Faculties and Other Sections | Links | Contact Information

- Admissions

```
<script type="text/javascript">…</script>
<!-- DO NOT MODIFY -->
<!-- End Facebook Pixel Code -->
<!-- // ContentLoader -->
<script type="text/javascript" src="/vendor/poper/poper.min.js"></script>
<script type="text/javascript" src="/vendor/bootstrap/4.3.1/bootstrap.js"></script>
<script type="text/javascript" src="/vendor/nette/nette.ajax.js"></script>
<script type="text/javascript" src="/vendor/nette/netteForms.js"></script>
<script type="text/javascript" src="/vendor/pushy/js/pushy.js"></script>
<script type="text/javascript" src="/vendor/lightbox/js/lightbox.js"></script>
<div id="lightboxOverlay" class="lightboxOverlay" style="width: 847px; height: 2798px; displa
y: block;"></div>
▼<div id="lightbox" class="lightbox" style="display: block; top: 1502.8px; left: 0px;"> == $0
  ▶<div class="lb-outerContainer" style="width: 527px; height: 368px;">…</div>
  ▶<div class="lb-dataContainer" style="display: block; width: 527px;">…</div>
</div>
<script type="text/javascript" src="/vendor/layzr/layzr.min.js"></script>
<script type="text/javascript" src="/vendor/rellax/rellax.js"></script>
▶<iframe allow="join-ad-interest-group" data-tagging-id="AW-652809651" data-load-time="1703275
945677" height="0" width="0" style="display: none; visibility: hidden;" src="https://td.doubl
eclick.net/td/rul/652809651?random=1703275945668&cv=1…0CULS%20Prague&auid=952528243.169874601
2&fledge=1&data=event%3Dgtag.config">…</iframe>
▶<iframe allow="join-ad-interest-group" data-tagging-id="AW-652809651/6C2RCKSCvswBELOrpLcC"
data-load-time="1703275945696" height="0" width="0" style="display: none; visibility: hidden;
" src="https://td.doubleclick.net/td/rul/652809651?random=1703275945693&cv=1…gtm_ee=1&auid=95
2528243.1698746012&fledge=1&capi=1&data=event%3Dconversion">…</iframe>
▶<iframe allow="join-ad-interest-group" data-tagging-id="AW-939810196" data-load-time="1703275
945734" height="0" width="0" style="display: none; visibility: hidden;" src="https://td.doubl
eclick.net/td/rul/939810196?random=1703275945727&cv=1…0CULS%20Prague&auid=952528243.169874601
2&fledge=1&data=event%3Dgtag.config">…</iframe>
▶<iframe allow="join-ad-interest-group" data-tagging-id="AW-939810196/JDxVCNLJmm4Q1LuRwAM"
data-load-time="1703275945754" height="0" width="0" style="display: none; visibility: hidden;
" src="https://td.doubleclick.net/td/rul/939810196?random=1703275945749&cv=1…gtm_ee=1&auid=95
2528243.1698746012&fledge=1&capi=1&data=event%3Dconversion">…</iframe>
</body>
```

Figure 3. DOM Tree of ČZU website after user interaction. Source: own

3.2.1.4 XPath parsing

"XPath (short for XML Path) is a query language used for navigating and selecting portions of an XML document. Founded by the W3C in 1999, it is occasionally used in languages such as Python, Java, and C# when dealing with XML documents." [29]

Looking at the name, one can assume that this is a technique for XML documents, however, it is also applicable to HTML documents [26]. Fundamentally, XPath operates on four core concepts [29]:

- Node selection. Node is an individual element or attribute in the XML or HTML document hierarchy, identifiable by tags or properties, and XPath differentiates between root and non-root nodes. As an example, '*/div*' targets a div node at the document's root, but '*//div*' selects all div nodes regardless of their position.

- Attribute selection. XPath allows to isolate nodes based on their attributes, such as selecting nodes with '*src*' attribute, which specifies the path of the resource file.
- Positional selection. XPath allows to select nodes according to their position in the document.
- Wildcard Selection. Using the asterisk XPath enables the selection of various characters or nodes, enhancing the flexibility in targeting elements.

XPath parsing shares similarities with DOM parsing in terms of navigation a document's structure. However, while DOM parsing engages with the entire document's tree structure through a browser or browser-like environment, XPath takes a more targeted approach, querying specific elements within a tree for greater precision.

This parsing technique is conducted by creating an expression that precisely locates and extracts elements within HTML or XML and such a process is typically executed via web scraping tools like Selenium or Scrapy [47].

As a practical example, it is demonstrated how the names of products in the ČZU shop, presented in **Figure 4**, can be extracted using an XPath query. In the figure, it can be seen that each product is contained within a '*<div>*' element classed as '*product*' and the product name itself is nested within a '*<span>*' tag tagged with a *'data-micro='name''* attribute. In order to extract the names of products, the XPath expression *'//div[@class='product']//span[@data-micro='name']/text()'* can be utilized, where

- '//' indicates that the search should consider entire document.
- *'div[@class='product']'* targets '*div*' elements with class having '*product*' as value.
- '*span[@data-micro='name']*' similar to previous targets '*span*' elements with attribute '*data-micro*' having '*name*' value.
- *'/text()'* selects the text nodes of the '*span*' elements children.

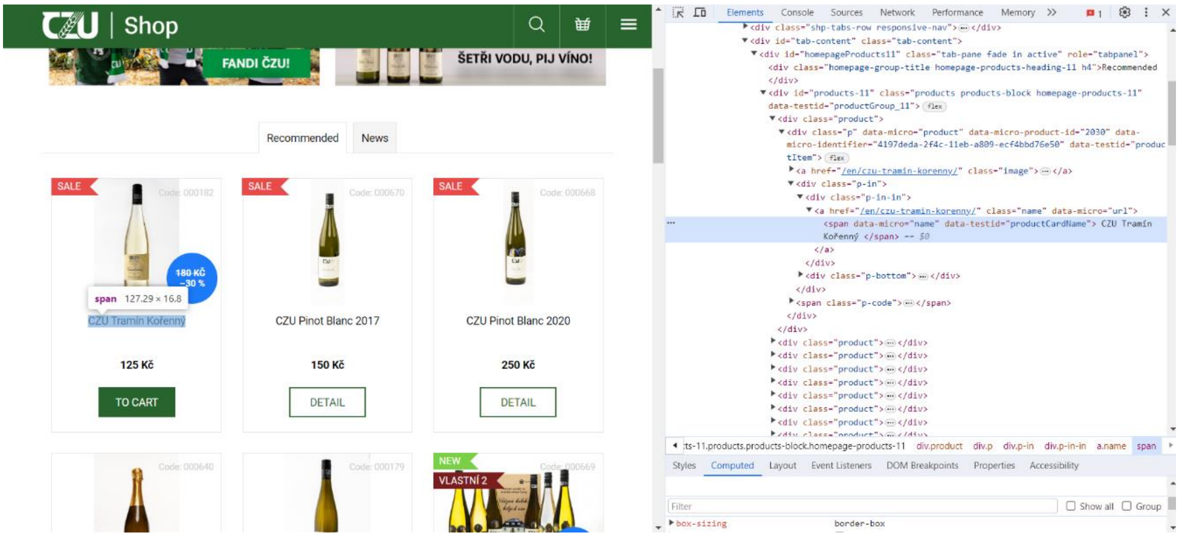
However, executing this query directly in a browser's console will result in a NodeList consisting of text node objects, where each object represents a product name but is accompanied by other node-related data. Therefore, for illustrative purposes and to present product names in a more user-friendly format, JavaScript can be used to execute this query and display results in a more readable format. The code snippet to perform this action is as follows:

Source Code 1. XPath Query with JS to Extract Product Names from ČZU shop. Source: own

```
// XPath query to find text within spans under divs with 'product' class
var xpathResult = document.evaluate("//div[@class='product']//span[@data-
micro='name']/text()", document, null, XPathResult.ORDERED_NODE_SNAPSHOT_TYPE,
null);
// Iterate through the results and log context
for (var i = 0; i < xpathResult.snapshotLength; i++) {
// Trim and log the text of each node
console.log(xpathResult.snapshotItem(i).nodeValue.trim());
}
```

Upon executing the JavaScript snippet in the browser's console, the result is an organized array of product names, as shown in **Figure 5**. This array is a direct result of the code iterating over each '<*span*>' element identified by the XPath and extracting the clean context.

Although XPath can be considered as powerful web scraping technique, it is not without its drawbacks and challenges. Since XPath queries are based on the DOM structure, this means that any changes in layout, classes, or attributes can break the parser. Also, this method itself is quite complex and may not be entirely practical in cases with web pages that use dynamic content generation, and therefore this technique implies the need for a deep understanding of DOM and the creation of more complex queries.

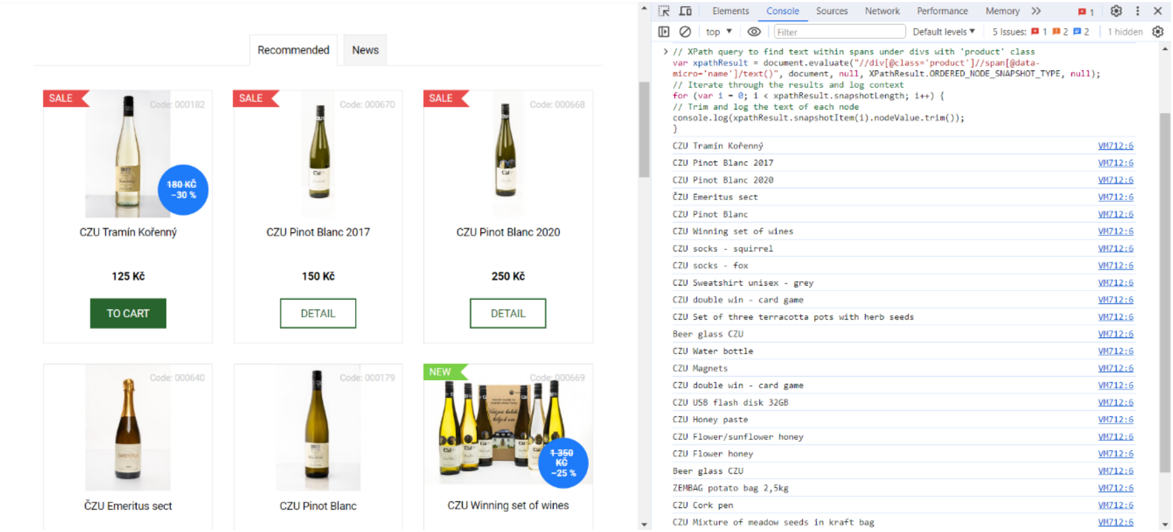


```
▶<div class="shp-tabs-row responsive-nav">…</div>
▼<div id="tab-content" class="tab-content">
  ▼<div id="homepageProducts11" class="tab-pane fade in active" role="tabpanel">
    <div class="homepage-group-title homepage-products-heading-11 h4">Recommended
    </div>
   ▼<div id="products-11" class="products products-block homepage-products-11"
    data-testid="productGroup_11"> flex
     ▼<div class="product">
       ▼<div class="p" data-micro="product" data-micro-product-id="2030" data-
        micro-identifier="4197deda-2f4c-11eb-a809-ecf4bbd76e50" data-testid="produc
        tItem"> flex
         ▶<a href="/en/czu-tramin-korenny/" class="image">…</a>
         ▼<div class="p-in">
           ▼<div class="p-in-in">
             ▼<a href="/en/czu-tramin-korenny/" class="name" data-micro="url">
                <span data-micro="name" data-testid="productCardName"> CZU Tramín
                Kořenný </span> == $0
              </a>
            </div>
           ▶<div class="p-bottom">…</div>
          </div>
         ▶<span class="p-code">…</span>
        </div>
      </div>
     ▶<div class="product">…</div>
     ▶<div class="product">…</div>
     ▶<div class="product">…</div>
     ▶<div class="product">…</div>
     ▶<div class="product">…</div>
     ▶<div class="product">…</div>
     ▶<div class="product">…</div>
```

```
…ts-11.products.products-block.homepage-products-11  div.product  div.p  div.p-in  div.p-in-in  a.name  span
```

```
▶ box-sizing                border-box
```

Figure 4. Elements containing the names of products in the ČZU shop. Source: own



```
> // XPath query to find text within spans under divs with 'product' class
  var xpathResult = document.evaluate("//div[@class='product']//span[@data-
  micro='name']/text()", document, null, XPathResult.ORDERED_NODE_SNAPSHOT_TYPE, null);
  // Iterate through the results and log context
  for (var i = 0; i < xpathResult.snapshotLength; i++) {
  // Trim and log the text of each node
  console.log(xpathResult.snapshotItem(i).nodeValue.trim());
  }
  CZU Tramín Kořenný                                          VM712:6
  CZU Pinot Blanc 2017                                        VM712:6
  CZU Pinot Blanc 2020                                        VM712:6
  ČZU Emeritus sect                                           VM712:6
  CZU Pinot Blanc                                             VM712:6
  CZU Winning set of wines                                    VM712:6
  CZU socks - squirrel                                        VM712:6
  CZU socks - fox                                             VM712:6
  CZU Sweatshirt unisex - grey                                VM712:6
  CZU double win - card game                                  VM712:6
  CZU Set of three terracotta pots with herb seeds            VM712:6
  Beer glass CZU                                              VM712:6
  CZU Water bottle                                            VM712:6
  CZU Magnets                                                 VM712:6
  CZU double win - card game                                  VM712:6
  CZU USB flash disk 32GB                                     VM712:6
  CZU Honey paste                                             VM712:6
  CZU Flower/sunflower honey                                  VM712:6
  CZU Flower honey                                            VM712:6
  Beer glass CZU                                              VM712:6
  ZEMBAG potato bag 2,5kg                                     VM712:6
  CZU Cork pen                                                VM712:6
  CZU Mixture of meadow seeds in kraft bag                    VM712:6
```

Figure 5. Product names array resulted from executing an XPath query. Source: own

3.2.1.5 API scraping

Mitchell Ryan in his book “Web Scraping with Python: Collecting More Data from the Modern Web” defines web scraping as the practice of obtaining data by any means other than interacting with the Application Programming Interface (API) [29]. This definition positions traditional web scraping techniques discussed previously as distinctly separate from API usage. While API interaction is not a traditional web scraping technique, it is crucial for efficient web data retrieval nowadays.

Fundamentally, an API is a set of rules and protocols for building and interacting with software applications. It allows different software programs to communicate with each other by defining methods of requesting and receiving data. This ease of use and structured access that the API provides make it an integral part of modern data retrieval techniques and a very important aspect in the field of web scraping. As indicated in a number of studies, APIs provide the easiest and most direct way to access data from the websites that provide API, such as Twitter (X), Facebook, LinkedIn and Google [47][20][10].

However, the use of APIs for data retrieval has its limitations since not all websites provide API, and those that do may have restrictions. Key challenges that may be listed [47]:

- Limited Availability. The targeted website does not provide API.
- Cost and Access Restrictions. Some APIs are not free, and rate limited which means limited access time.

- Partial Data Exposure. APIs may limit the scope of information that can be retrieved.

In the cases listed above, the traditional web scraping methods discussed in this chapter become dominant. They allow data to be retrieved and manipulated without relying on APIs, so this flexibility makes them an important tool when working with websites that do not provide or have restrictive API policies.

3.2.2 Web Scraping Tools

After examining various web scraping techniques, it becomes essential to discuss the tools that make these techniques possible. This section aims to provide a comprehensive overview of the most notable tools in the web scraping field, highlighting their features, capabilities, limitations, and applications. By investigating these tools, we can understand what functionality they have and how they are used, which will play a crucial role in creating a web scraper prototype in the practical part of this thesis.

3.2.2.1 Beautiful Soup

"*Beautiful Soup is a Python library for pulling data out of HTML and XML files. It works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree*" [4]. This succinct definition provided in the official documentation of this library encapsulates the essence of Beautiful Soup as a tool that simplifies the process of data extraction from web documents.

At its core, Beautiful Soup serves as a foundational tool for HTML and XML analytics. It was designed to make the process of web scraping more accessible and efficient, particularly in parsing and extracting data from webpage, and the library achieves this by converting markup documents content into a navigable tree of Python objects [4]. For instance, every tag in the HTML is converted into a Beautiful Soup object, which can be manipulated as any other Python object. Therefore, library allow precise selection based on tags, attributes, and their hierarchies. This allows for easy searching and manipulation of the DOM tree, which is crucial for extracting specific targeted data from web pages.

Another key strength of library is its compatibility with various parsers [29][52][46], such as *lxml* [46], known for speed and efficiency, or *html5lib* for handling malformed HTML [29]. Such a choice of parser can provide more flexibility in scraping different web content, ensuring optimal parsing performance across diverse web environments.

As a practical example of Beautiful Soup's use, its ability and functionality will be demonstrated to scrape product names, prices, and links to products from the ČZU merchandise shop. As it can be seen in **Figure 6**, the webpage is organized into product cards, each within a '*div'* tag with '*class "products"'*. Inside the cards located a structured data, which can be targeted for extraction. The product name is wrapped in a '*span'* tag with '*data-micro="name"'* attribute, link to a product located in a '*a'* tag with '*class="name"'* attribute and the price is in a '*div'* with '*data-micro="offer"'* where the numerical price is stored in the '*data-micro-price'* attribute. To extract targeted data, a Python script, as demonstrated in **Source Code 2,** can be used.

The script starts by importing the necessary modules: *requests* to make web requests, Beautiful Soup from bs4, *json* for formatting and saving data as JSON, and *urljoin* from *urllib.parse* for handling URL paths correctly.

Inside '*scrape_products'* function, which takes URL as its parameter, the *request.get* fetches the content of the webpage at the given URL and Beautiful Soup parses the fetched HTML content (response.txt) using Python's built-in HTML parser. Then the script starts to iterate over each product card. For each product it extracts:

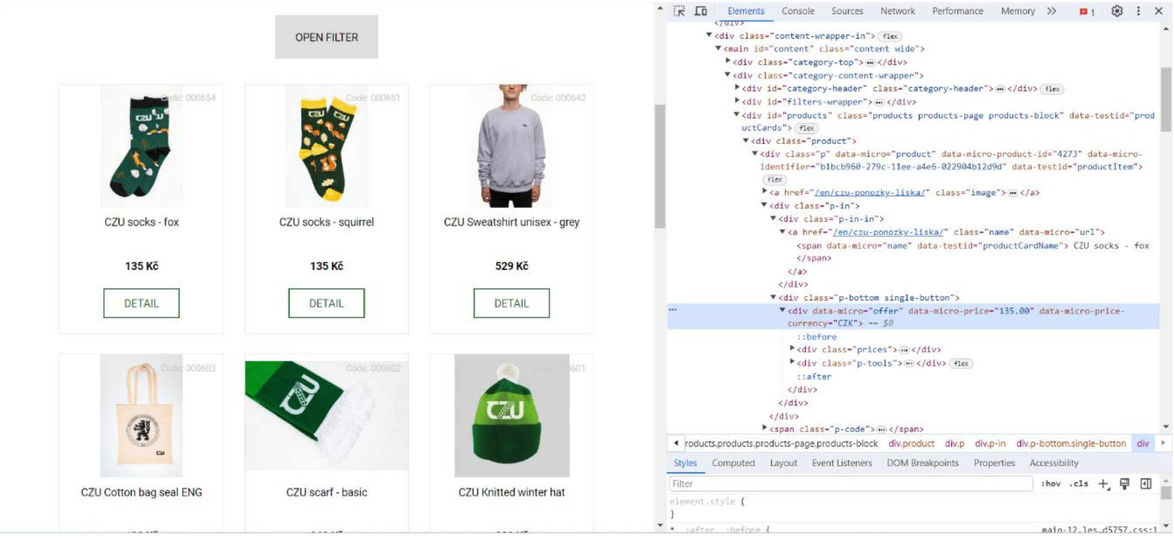
- Name from a '*span'* tag with '*data-micro="name"'*
- Absolute URL by joining the URL with relative link from '*href'* attribute in '*a'* tag with class '*name'*
- Price from '*data-micro-price'* attribute within a '*div'* tagged with '*data-micro="offer"'*

Then data is stored into the list and saved into *products.json*. As a result, the extracted data is compiled into a JSON structure, providing a clear and organized view, as demonstrated in **Figure 7**.

While Beautiful Soup is an effective tool, it has its own challenges and limitations. Since it relies on the static structure of HTML, it has a drawback while working with dynamic content employed by JavaScript. Therefore, it is usually required tools like Selenium for dynamic content. Also, for full-scale web scraping processes, it often needs to be supplemented with other libraries. For instance, in a practical example, it was combined with *request* for web requests and *json* for data storage. However, despite these limitations, Beautiful Soup is still a great asset for data extraction tasks.

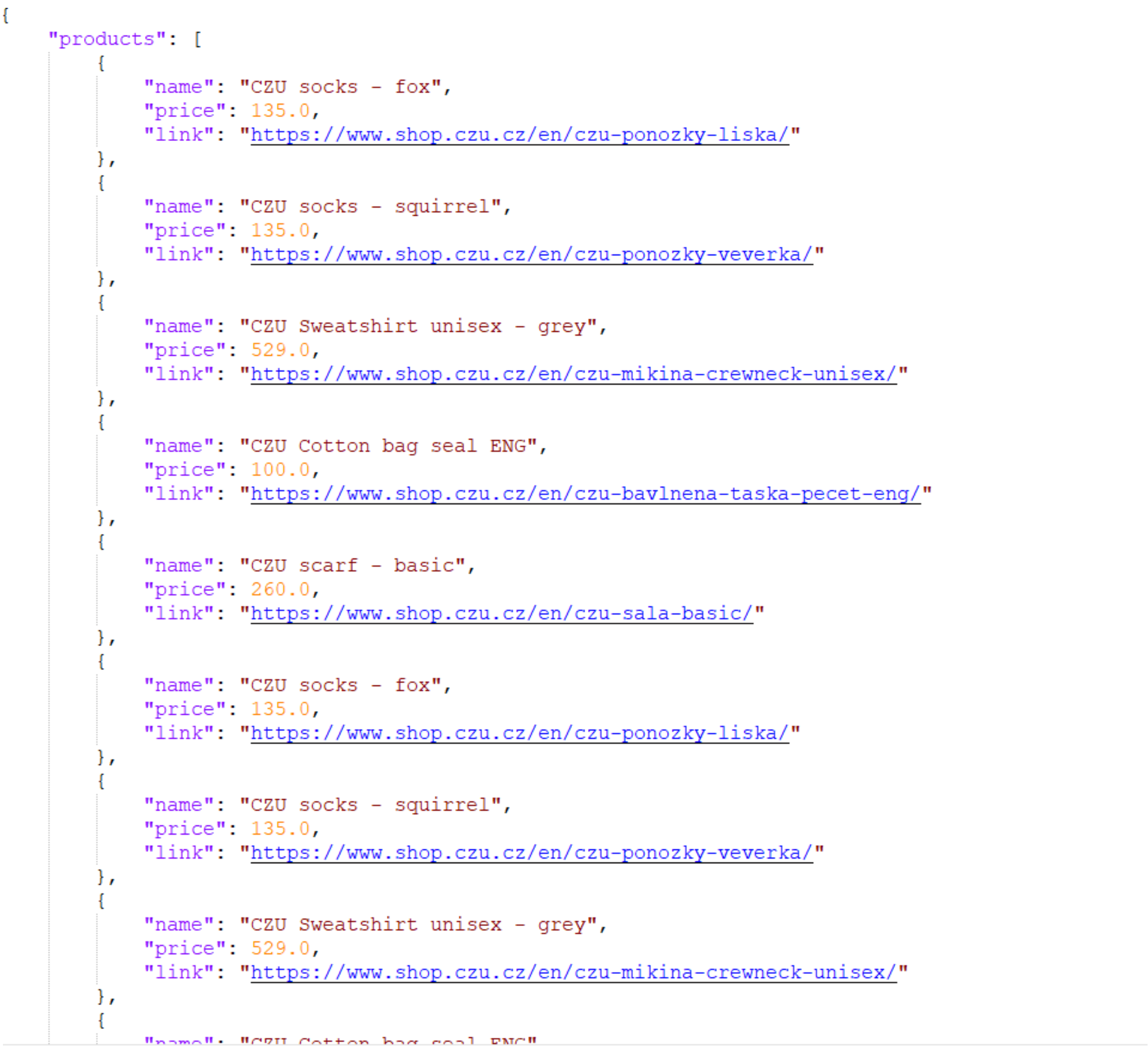
Source Code 2. ČZU Merchandise Shop Scraper Using Python and Beautiful Soup. Source: own

```
import requests
from bs4 import BeautifulSoup
import json
from urllib.parse import urljoin


def scrape_products(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')

    products = []
    for product_div in soup.find_all('div', class_='product'):
        # Extract product name
        name = product_div.find('span', {'data-micro': 'name'}).text.strip()

        # Extract product link
        full_link = urljoin(url, product_div.find('a', {'class':
'name'}).get('href'))

        # Extract product price
        price = float(product_div.find('div', {'data-micro':
'offer'}).get('data-micro-price'))

        products.append({
            'name': name,
            'price': price,
            'link': full_link
        })

    return products

# Base URL
url = 'https://www.shop.czu.cz/en/merch/'

# Scrape products
products_data = scrape_products(url)

# Save the data to a JSON file
with open('products.json', 'w') as json_file:
    json.dump({'products': products_data}, json_file, indent=4)

print('Products data saved to products.json')
```



```
▼<div class="content-wrapper-in"> flex
  ▼<main id="content" class="content wide">
    ▶<div class="category-top"> ⋯ </div>
    ▼<div class="category-content-wrapper">
      ▶<div id="category-header" class="category-header"> ⋯ </div> flex
      ▶<div id="filters-wrapper"> ⋯ </div>
      ▼<div id="products" class="products products-page products-block" data-testid="prod
      uctCards"> flex
        ▼<div class="product">
          ▼<div class="p" data-micro="product" data-micro-product-id="4273" data-micro-
          identifier="b1bcb960-279c-11ee-a4e6-022904b12d9d" data-testid="productItem">
          flex
            ▶<a href="/en/czu-ponozky-liska/" class="image"> ⋯ </a>
            ▼<div class="p-in">
              ▼<div class="p-in-in">
                ▼<a href="/en/czu-ponozky-liska/" class="name" data-micro="url">
                  <span data-micro="name" data-testid="productCardName"> CZU socks - fox
                  </span>
                </a>
              </div>
              ▼<div class="p-bottom single-button">
⋯               ▼<div data-micro="offer" data-micro-price="135.00" data-micro-price-
                currency="CZK"> == $0
                  ::before
                  ▶<div class="prices"> ⋯ </div>
                  ▶<div class="p-tools"> ⋯ </div> flex
                  ::after
                </div>
              </div>
            </div>
            ▶<span class="p-code"> ⋯ </span>
roducts.products.products-page.products-block  div.product  div.p  div.p-in  div.p-bottom.single-button  div
element.style {
}
*, ::after, ::before {                                              main-12.les…d5757.css:1
```

Figure 6. ČZU merchandise shop website structure. Source: own.



```
{
    "products": [
        {
            "name": "CZU socks - fox",
            "price": 135.0,
            "link": "https://www.shop.czu.cz/en/czu-ponozky-liska/"
        },
        {
            "name": "CZU socks - squirrel",
            "price": 135.0,
            "link": "https://www.shop.czu.cz/en/czu-ponozky-veverka/"
        },
        {
            "name": "CZU Sweatshirt unisex - grey",
            "price": 529.0,
            "link": "https://www.shop.czu.cz/en/czu-mikina-crewneck-unisex/"
        },
        {
            "name": "CZU Cotton bag seal ENG",
            "price": 100.0,
            "link": "https://www.shop.czu.cz/en/czu-bavlnena-taska-pecet-eng/"
        },
        {
            "name": "CZU scarf - basic",
            "price": 260.0,
            "link": "https://www.shop.czu.cz/en/czu-sala-basic/"
        },
        {
            "name": "CZU socks - fox",
            "price": 135.0,
            "link": "https://www.shop.czu.cz/en/czu-ponozky-liska/"
        },
        {
            "name": "CZU socks - squirrel",
            "price": 135.0,
            "link": "https://www.shop.czu.cz/en/czu-ponozky-veverka/"
        },
        {
            "name": "CZU Sweatshirt unisex - grey",
            "price": 529.0,
            "link": "https://www.shop.czu.cz/en/czu-mikina-crewneck-unisex/"
        },
        {
            "name": "CZU Cotton bag seal ENG"
```

Figure 7. Extracted data from the ČZU merchandise shop in JSON. Source: own

3.2.2.2 Selenium

"*Selenium is a powerful web scraping tool that was originally developed for the purpose of automated website testing. Selenium works by automating browsers to load a website, retrieve its contents, and perform actions like a user would when using the browser. As such, it's also a powerful tool for web scraping. Selenium can be controlled from various programming languages, such as Java, C#, PHP, and of course, Python.*" [47]. The above quote from the book by Seppe Vanden Broucke Bart Baesens gives an excellent insight into how this framework can play an important role in the field of web scraping. Selenium, originally designed for automated website testing [29][47][13], has transformed into a powerful web scraping tool, capable of accurately imitating website as they appear in web browsers. Due to the fact that the framework was originally created to automate websites testing, it provides many key features to automate the web scraping process, allowing to execute user-like actions such as clicking buttons, filling out forms, waiting for dynamic content to be load, and much more [13].

Central to Selenium's functionality is its reliance on third-party browsers, facilitated by WebDrivers [29][47]. According to the official Selenium documentation [48], WebDriver is an API and protocol that establishes a language-neutral interface for controlling the behaviour of web browsers, where each browser is supported by a specific WebDriver implementation, known as a driver. The driver is the key component that delegates the command to the browser and manages the communication between the Selenium and the browser. When user utilize Selenium, the typical process involves opening a browser window where it is possible to visually observe the navigation and interaction as the script executes. However, while it might be useful for debugging and development, it comes with certain drawbacks. Running a browser in its full graphical mode can be resource-intensive, and this might not be ideal for environments where resources are limited, such as server setups without display [47].

To address these challenges, Selenium offers support for "headless" browsers. Such browsers operate without the traditional graphical user interface, running "invisibly" in the background. This headless mode is partially valuable since it still allows to perform all necessary tasks – such as rendering HTML, handling cookies, and executing JavaScript [47]. The primary literature used for this thesis, specifically books from Mitchell Ryan [29] and Seppe Vanden Broucke Bart Baesens [47], as well as other sources [16], most often use or describe tools like PhantomJS for this feature. PhantomJS is a headless web browser scriptable with JavaScript which could be used as a driver in Selenium [37]. However, as of March 2024, PhantomJS is no longer in development and is not actively maintained [37]. This shift has resulted in an increased focus on alternative headless browser choices, particularly headless versions of popular browsers like Chrome and Firefox and tools like Puppeteer. Vitaly Slobodin, the previous maintainer of PhantomJS, has highlighted the unavoidable transition to headless Chrome, emphasizing its advantages over PhantomJS [37]. And in March 2018 PhantomJS's developers ceased updating, urging users to switch to more advanced and sustainable headless browsing solutions.

However, despite the advancements and versatility of technologies discussed previously, they face various challenges and limitations. A study by Jonker et al. [18] illustrates how properties unique to these tools, such as '*window.navigator.webdriver'* or '*document.$cdc_asdjflasutopfhvcZLmcfl_'* in ChromeDriver, the WebDriver implementation for the Google Chrome browser, can be detected by bot detection mechanisms. These properties can lead to blocking or other countermeasures from web sites against automated scraping activities.

Researchers from the same study [18] conducted a scan across the top 1 million websites ranked by Alexa, uncovering those 127,799 sites had scripts matching bot detection patterns. 93.76% of these scripts were focused on targeting PhantomJS. Other patterns identified included WebDriver on 1.31% of sites, Selenium on 1.34%, and Chrome in headless mode on 0.99%. However, the majority of web sites only triggered a single detection pattern, predominantly '*PhantomJS(?![a-zA-z-])*'. The study also discovered that the most complex site had 23 different detection patterns.

These findings highlight that while Selenium and headless browsers are powerful tools, they are also detectable. This underscores the need for continuous adaptation in the web scraping process to stay ahead of detection mechanisms.

3.2.2.3 Scrapy

Scrapy is an open-source and collaborative web crawling framework for Python, designed to extract data from various web sites efficiently and systematically. Scrapy operates on Python and utilizes Twisted [12], an event-driven networking framework, which enhances the capabilities of handling massive amounts of data dynamically. The architecture of the framework is meticulously designed, consisting of several key components such as "Engine", "Scheduler", "Downloader", "Spiders" and the "Item

Pipeline” as shown in **Figure 8**. The “Engine” serves as the central component of the system, directing data flows and triggering events. The “Scheduler” is responsible for querying requests, the “Downloader” is in charge of retrieving data, and “Spiders”, which are user-defined classes, are where specific scraping rules are set. After the extraction, the “Item Pipeline” performs crucial tasks like data cleansing, validation, and storage.

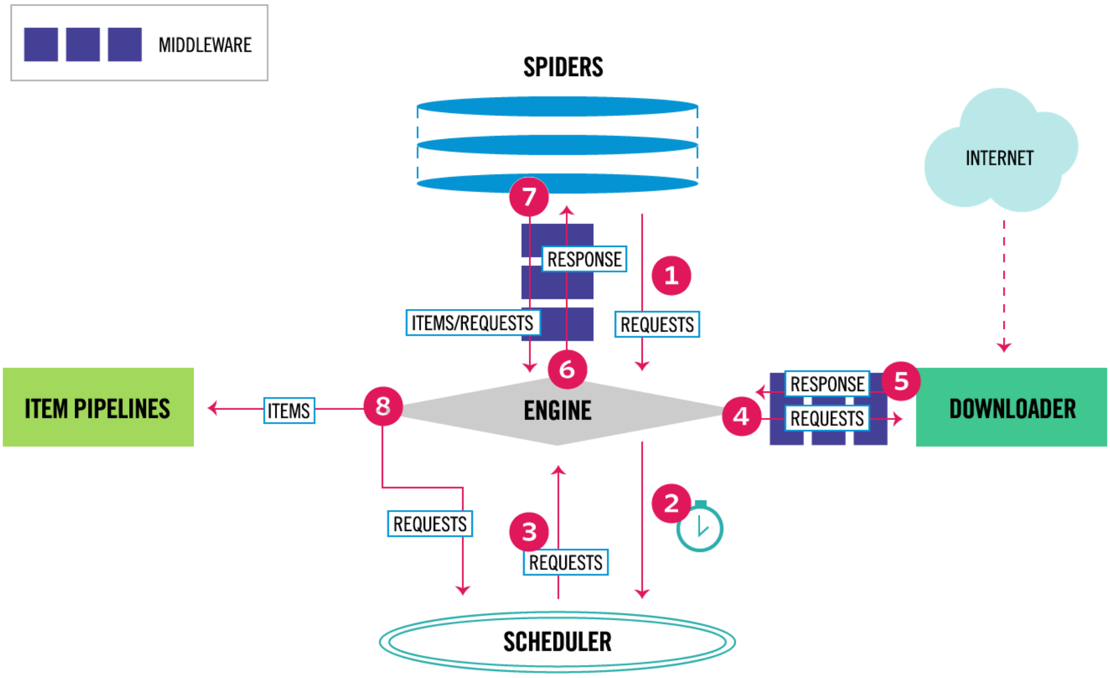


Figure 8. Scrapy architecture. Source: [3]

Furthermore, Scrapy, which was developed with Twisted as mentioned before, employs non-blocking, asynchronous code to facilitate concurrency [3]. This feature allows this tool to handle multiple requests simultaneously, significantly reducing the time required for data extraction [12].

In addition, the extensibility of Scrapy is another point of strength. It provides support for a wide range of middleware and plugins, therefore providing a customisation to specific project requirements. This flexibility provides a way for dealing with complex web scraping tasks, such as rotating proxies, handling CAPTHAs, or integrating with various data storage solutions [12].

The study by Lotfi et al. [26] compared Scrapy with other tools like Beautiful Soup and Selenium, which were described in sections **3.2.2.1** and **3.2.2.2**, highlighting Scrapy as a superior tool for large-scale and complex scraping tasks between Python tools.

However, although Scrapy possesses powerful features, it also presents certain challenges. Its complexity and asynchronous nature, while advantageous for large-scale tasks, can pose a steep learning curve for novices. In addition, managing web sites that heavily rely on JavaScript might require additional setups, like integration with headless browsers.

3.3 Web Scraping Challenges

In this chapter of the thesis, a comprehensive examination of various challenges inherent in the practice of web scraping will be conducted. This chapter aims to examine the complexities of scraping dynamic content and the continual adaptation to changing web structures. A pivotal point of the discussion will be on anti-scraping measures implemented by websites, and strategies and techniques to work around these limitations will be examined. Through this exploration, the chapter aims to provide a thorough understanding of various multifaceted challenges, outlining the technical and operational strategies to overcome them.

3.3.1 Dynamic Content

The evolution of web scraping has progressed in tandem with dynamic nature of web content, presenting unique challenges. Dynamic content on webpages is usually generated through client-side programming languages like JavaScript, which alter the webpage's structure or content in response to user interaction or asynchronous events. According to the World Wide Web Consortium (W3C), JavaScript is used by 98.8% of all websites, underscoring its pivotal role [45]. Another key component of dynamism is Asynchronous JavaScript and XML (AJAX). AJAX allows web pages to communicate with servers and update content without needing to reload a page. Many modern web pages nowadays use this technology, for example, to fetch new e-mails, notifications, update a live news feed, and all this without page refresh [47]. Therefore, this technology poses another challenge for the traditional web scraping stack of technologies that rely on the static HTML content of a page.

To address these challenges, tools like Selenium, described in **3.2.2.2**, or Puppeteer can be employed. As already mentioned in the previous sections of the thesis, such tools can automate the process of interaction with a web page, for example, by clicking on specific buttons in order to update the DOM Tree and load the necessary data to extract into it.

Furthermore, another approach to this topic involves intercepting AJAX calls. This method involves monitoring and analysing the network traffic generated by the web browser. This can be accomplished using browser developers tools or other network sniffing tools, which allows to observe the details of each AJAX request, including its URL, parameters, and response data. Such replication is usually achieved by creating HTTP requests that mimic the original AJAX call. This approach bypasses the need to load or interact with the entire web page, leading to more efficient data extraction.

3.3.2 Targeted Anti-Scraping Measures

A significant challenge that arises in the constantly evolving field of web scraping is the implementation of targeted anti-scraping measures by web sites. The purpose of these countermeasures is to prevent automated access to the data, safeguard against unauthorised data extraction, and preserve website integrity. This section aims to analyse the specifics of such measures, their influence, and viable techniques for overcoming these challenges.

3.3.2.1 CAPTCHA

"CAPTCHA stands for Completely Automated Public Turing Test to tell Computers and Humans Apart. As the acronym suggests, it is a test to determine whether the user is human or not. A typical CAPTCHA consists of distorted text, which a computer program will find difficult to interpret but a human can (hopefully) still read. Many websites use CAPTCHA to try and prevent bots from interacting with their website." [24]

CAPTCHAs are one of the most common anti-scraping techniques, typically it requires the user to perform specific task, such as recognizing distorted text or identifying object in an image. However, this approach for bot detection is considered not the most convenient for the end user and shows that constantly forcing the user to undergo such tests significantly spoils the web experience [24][40]. However, CAPTCHA systems have evolved significantly beyond their initial designs. A notable development in this arena is Google's reCAPTCHA v3, which represents a significant shift in how user verification is approached.

reCAPTCHA v3 is an artificial intelligence system based on machine learning algorithms and, unlike its predecessor, operates largely in the background, assessing user interaction with a website to assign a risk score [2] between 0.0 and 1.0, where a score close to 1.0 means that the user is more likely human.

Nevertheless, solutions for solving CAPTCHA systems have been found and can be used to successfully bypass such a system. For example, in studies conducted by Bursztein et al. [7] and Bock et al. [5], it was demonstrated how ML-based systems can bypass distorted text, image-based, and audio-based CAPTCHAs, such as the first and second versions of reCAPTCHA. Furthermore, a study by Akrout et al. [2] successfully employed the proposed Reinforcement Learning (RL) formulation, which proved remarkably effective, achieving a success rate of over 90% in defeating reCAPTCHAv3.

Another noteworthy aspect of the study by Bock et al. [5] was their use of Selenium for automating interaction with web page. The researchers discovered that the reCAPTCHA system was initially able to detect Selenium-driven interactions. However, the authors successfully modified the Selenium Web Driver for Mac OS X, making it undetectable.

Similarly, the study by Akrout et al. [2] also proves that reCAPTCHA is able to detect Selenium. Upon investigation, it was discovered that reCAPTCHA consistently returned low scores for interactions driven by Selenium and further analysis of the HTTP queries revealed the presence of automated headers in Web Driver, along with other different variables that are not typically found in a regular browser. However, both studies also demonstrate that this can be circumvented in various ways, such as by creating proxies, using third-party libraries, and other sophisticated methods. Taking all the above-mentioned aspects into account, we can say that this outcome underscores a crucial point: systems like reCAPTCHA are still foolproof against common web scraping tools.

3.3.2.2 IP Rate Limiting & IP Rotation

Internet Protocol (IP) Rate Limiting is a network-level security mechanism used by web sites to prevent internet threats like web scraping. The mechanism is straightforward: it allows a specific number of requests from a single IP address within a specified timeframe. For instance, a web site might permit only 100 requests per hour from an IP address. Going beyond this limit usually leads to the servers rejecting more requests from the restricted IP for a certain time, often indicated by an HTTP 429 status code (Too many requests). However, while this mechanism can be effective against basic scraping activities, its efficiency is low against more sophisticated methods involving rotational IP addresses.

The strategy of IP rotation through the use of proxy or Virtual Private Network (VPN) emerges as a crucial technique not only for bypassing rate limits but also for overcoming various different web scraping problems, such as website bans. Even after taking all precautions, the risk of being detected as a bot and being blocked still remains, and sometimes it can happen due to the fact that the web scraping process occurs from one IP address [16]. In such cases, the person responsible for maintaining a targeted web site will take measures to include this IP address in a blacklist, i.e., block it.

In response to this risk, a scraper can periodically change the IP address from which requests are made. This process involves routing traffic through different intermediaries, such as a proxy or VPN, each providing a unique IP and potentially varying the geographical origin of requests.

Automation of IP rotation can be also achieved through web the scraping tools discussed before. For example, the Scrapy framework supports middleware for HTTP proxies [11] such as "*scrapy-proxies"*, which streamlines the process of changing proxies and thus minimizes the risk of scraper detection and blacklisting [16].

However, while the use of proxies offers a good advantage, it's also important to consider the reliability of these resources. For example, free proxies available on the web may look like a cost-effective solution, but their stability and availability are often questionable. A study by Achsan and Wibowo [1] highlights this concern. Authors in their study analysed the availability of 5,043 proxy servers and discovered that only 1,931 (38.23%) of these servers were available. Furthermore, only after two days, the number dropped to 1,576 (31.26%). Additionally, 4.83% of the proxies were reported to be very slow.

Thus, while proxies and VPNs are invaluable for web scraping purposes, achieving effective IP rotation often requires additional investments in stable proxy lists and paid VPNs, however, given the above, such an investment can be very profitable since it allows to bypass possible blocks and restrictions from websites.

3.3.2.3 Honeypot Traps

Honeypot traps in terms of web scraping are invisible decoy traps designed to be hidden from normal users but detectable by scraping bots, therefore, interaction with these traps flags the bot as a scraper [32]. Honeypot traps can be applied to almost any element presented on a website, including links, images, files, and more [29]. Such traps are usually styled with CSS to evade visual detection, meaning that they are hidden from a typical browser user but detectable by bots parsing HTML. However, tools like Selenium can still distinguish visible and hidden elements. For example, the *'is_displayed()'* function in Selenium is particularly useful in this context, allowing to identify and avoid interaction with hidden elements.

However, the challenge with honeypot traps lies in their dual nature. While it is recommended to avoid hidden elements, it can be counterproductive [29] and lead to data loss. A scraper must strike a delicate balance between recognizing and interacting with hidden elements in order to make scraping effective but also avoid detection. Nevertheless, it is safe to say that by using more advanced web scraping techniques that mimic human behaviour, it is possible to avoid falling into such traps.

3.3.2.4 Behavioural Patterns

Behavioural analysis is becoming a crucial tool in detecting web bots, as it differentiates between humans and automated scripts by examining user actions such as mouse movement, keystroke dynamics, and even browsing patterns. The most common methods for web bot detection based on behavioural patterns are analysis of mouse movements and keystroke dynamics [8][32]. Keystroke analysis involves the calculation of the duration at which speed text was typed and at what speed different keys were pressed. The dynamic of the mouse is usually calculated by its movement. Many studies on biometric behaviour have found that human behaviour is often much more unpredictable and complex than the behaviour of an automated program. This is manifested by the fact that web bots tends to move the cursor in a straight line and at constant speed, and also press keys at certain intervals.

One exemplary study in this area is presented in research conducted by Chu et al. [8]. The authors of this study presented a method for identifying and blocking bots by analysing Human Observational Behaviours (HOBs), exploiting previously mentioned mouse and keystroke behavioural biometrics. The authors of the study developed a prototype for an automatic classification system consisting of a client-side JavaScript logger and a server-side detector. The logger records a user’s input data and streams it to the server-side detector. The detector analyses the given data and extracts biometric-related features utilizing the C4.5 machine learning classification algorithm. The results of such a system demonstrated 99% detection accuracy with minor overhead.

However, the efficacy of behaviour analysis is constantly challenged by the advancement of tools and techniques capable of mimicking human behaviour. Notably, in the previously mentioned study by Akrout et al. [2] about reCAPTCHA v3, the authors also set a goal to mimic human mouse movement since the system from Google also relies on behavioural analysis. The authors assumed that an average user would not move the mouse pixel by pixel, thus defining a cell size for mouse movement steps and random mouse allocating on a web page. In order to adapt to various screen resolutions, they proposed a divide-and-conquer technique. This involved breaking the grid into small sub-grids and applying their trained agent to these to find an optimal path. This research underscores the race between web security measures and the evolving sophistication of automated systems designed to bypass them.

3.3.2.5 Technical indicators

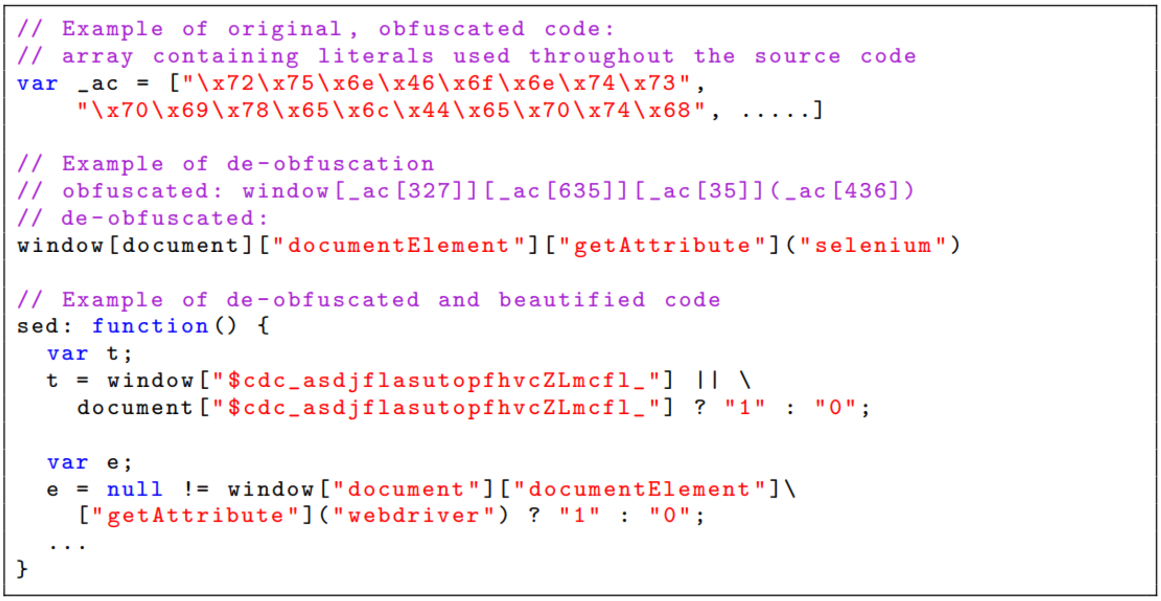
Technical indicators are another approach in web bot detection used by web sites to distinguish between humans and automated bots.

One of the key elements in this detection mechanism is the browser fingerprint, which encompasses an array of browser characteristics such as user agent strings, JavaScript engine speed, screen resolution, and installed fonts [18]. Integral to this part are HTTP headers, which are part of the HTTP request from browser to server, providing information such as browser type and language preferences. HTTP headers are not directly related to fingerprint, however, they contribute significantly in identifying profile of the user and bot. Since the information from HTTP headers is easy to acquire in real-time, this method can be implemented into web server software without any challenges [40].

One of the most pertinent examples of sophisticated bot detection based on technical indicators can be drawn from the study conducted by Jonker et al. [18] where the authors encountered a web site which can detect and block Selenium-based visitors. The authors found the use of specific scripts and the presence of HTTP headers related to a company specialized in web bot detection. After manual deobfuscation, authors discovered that scripts provided three main functionalities:

- Behaviour-based detection, which was described previously in this section.
- Code injection routines, allowing to include CAPTCHA to the page.
- DOM properties-based detection where multiple built-in objects and functions were accessed through JavaScript, as demonstrated in **Figure 9**.

In **Figure 9** it can be observed that the script uses ‘*window[document]["documentElement"]["getAttribute"]("selenium")*’ to check if the Selenium attribute is present in the DOM tree. Additionally, it identifies unique bot properties like ‘*document.$cdc_asdjflasutopfhvcZLmcfl*’ specific to ChromeDriver and gathers browser data such as supported MIME types through ‘*navigator.MimeTypes*’. This is a very important part for understanding and developing ways to bypass such detection, since the practical part of the thesis will also use Selenium as one of the tools.



```
// Example of original, obfuscated code:
// array containing literals used throughout the source code
var _ac = ["\x72\x75\x6e\x46\x6f\x6e\x74\x73",
    "\x70\x69\x78\x65\x6c\x44\x65\x70\x74\x68", .....]

// Example of de-obfuscation
// obfuscated: window[_ac[327]][_ac[635]][_ac[35]](_ac[436])
// de-obfuscated:
window[document]["documentElement"]["getAttribute"]("selenium")

// Example of de-obfuscated and beautified code
sed: function() {
  var t;
  t = window["$cdc_asdjflasutopfhvcZLmcfl_"] || \
    document["$cdc_asdjflasutopfhvcZLmcfl_"] ? "1" : "0";

  var e;
  e = null != window["document"]["documentElement"]\
    ["getAttribute"]("webdriver") ? "1" : "0";
  ...
}
```

Figure 9. Example from bot-detection script. Source: [18]

Luckily, such challenges can be overcome. In the same study by Jonker et al. [18] authors were able to bypass detection by changing only one property. Furthermore, the integration of external libraries and tools alongside previously discussed tools like Selenium can enhance the ability to manipulate various browser attributes, including the user-agent [29][47]. User-agent is a string that the browser sends to websites, identifying itself and providing details like the browser type, operating system, and version. By altering the user-agent along with other parameters, it’s possible to create a more convincing human-like browsing profile.

By skilfully integrating these techniques, scrapers may evade detection in such cases, ensuring more successful data collection.

3.4 Legal & Ethical Considerations

This chapter of the thesis shifts the focus from the technical aspects of web scraping to its legal and ethical dimensions. The objective of this chapter is to examine the delicate equilibrium between efficient data collection and compliance with legal and ethical norms. Here we will elucidate the fundamental laws and ethical standards that govern web scraping practices, underscoring the significance of data gathering within these boundaries to uphold both legal and ethical responsibility.

3.4.1 Legal Aspects of Web Scraping:

It is worth noting that web scraping is a fairly new area that is currently actively developing, and its legal aspects are quite extensive and complex. However, key aspects related to this topic will be considered.

At the time of writing this bachelor's thesis, no direct legislation has been found directly related to web scraping, however, web scraping is guided by a set of fundamental legislative practices such as copyright infringement, breach of contract, and trespass to chattels [23][22]. Next, each of these practises will be examined to determine their effect on web scraping.

3.4.1.1 Terms of Use

One of the key factors surrounding the debate about the legality of web scraping is the website’s Terms of Use (ToU). ToU defines a list of rules for the use of a web site and also often has information about what information is collected on the website and how this information is used. Such an agreement is often established through mechanisms where the website provides the user with a “checkbox” and the opportunity to read the terms of this agreement. The agreement between the user and the website is concluded after the user clicks on the “checkbox”, proving that he has read the rules, and a violation of these rules may lead to a “breach of contract” on the side of the website’s user [23]. This is due to the fact that the presence of clauses in the ToU that explicitly prohibit web scraping is not sufficient on its own to legally preclude such activities; what is crucial is the explicit consent of the user to these terms.

The method by which the ToU are presented to users also may play a significant role. For example, the method described above for providing conditions using the clickwrap checkbox requires active actions from the user, unlike cases when the site may simply contain a link to ToU [20][17]. The last example may raise questions regarding the user's awareness of these terms and may not have strong legal affects.

3.4.1.2 Copyrighted Material

If data is scraped and republished without permission, particularly if the content is clearly protected by copyright law, it might lead to legal proceedings for copyright infringement. However, it is important to note that copyright laws do not automatically grant ownership of user-generated content to web site owners [23]. For instance, a website featuring user reviews does not inherently own the copyright to these reviews.

Furthermore, the "fair use" principle allows for the use of copyrighted material on a limited scale for specific purposes such as criticism, comments, news reporting, and research [47].

3.4.1.3 Trespass to chattels

"Trespass to chattels" is a legal concept that can be invoked when scraping activities interfere with a website's functioning, therefore, overloading or damaging a web site or web server. However, the damage should be material and easy to prove in order for the owner to be eligible for financial compensation [23].

According to Mitchell Ryan [29], for a web scraping activity to be considered trespass to chattels, three key criteria need to be satisfied:

- Lack of consent. This clause means a violation of ToU, which was described in more details previously.
- Actual harm. The scraping activity must cause material harm to the website owner.
- Intentionality. The act of scraping must be intentional. If a person develops a scraper, he is generally aware of its potential impact on a website's server.

3.4.2 Ethical Aspect of Web Scraping

While considering the ethical side of the question, it is important to distinguish between what is legally permissible and what is justified from an ethical point of view. Since the rapid evolution of digital data, technologies, and methods, which frequently outpaces the development of legal frameworks and ethical principles, there is a continuous need for innovative solutions to address emerging ethical challenges [27].

3.4.2.1 Robots.txt

Robots Exclusion Protocol, or robots.txt, is a file located in the root directory of most websites (e.g., www.google.com/robots.txt) and serves as a set of guidelines for web crawlers defining the specific sections of the website that should not be accessed or extracted by scraping. From an ethical perspective, it is considered a fundamental courtesy since it represents a web site's explicit wishes about how their content should be accessed and used [29][27].

However, the robots.txt file is not legally binding and serves more as a sign saying, "Please don't go to these parts of the site." [29]. Therefore, while the law might not enforce adherence to robots.txt, ethical web scraping practices demand a higher standard of conduct that respects the intentions and constraints established by web site administration.

3.4.2.2 Potential damage

Web Scraping activity can unintentionally harm website functionality. For example, an excessively large number of requests to a web server from a scraper can cause a "denial of service" (DoS) attack [27], which overloads the server's network and thereby interferes with the normal operations of the web site [47]. This not only affects the site's performance but also impacts its users, underscoring the need for responsible scraping practices.

3.4.2.3 Privacy Concerns

The capability of web scraping to efficiently amass vast amounts of information poses another ethical concern regarding privacy. For example, the data such as user interactions, posts, and personal views gathered using web scraping techniques on social network platforms has the potential to expose sensitive information such as political or religious beliefs. The implications of such data collection are profound, especially when utilized by government agencies or law enforcement for surveillance, potentially impacting human rights related to freedom of speech, association, and protection against unreasonable searches [14].

4 Practical Part

The previous chapters of this bachelor thesis have laid the foundation for understanding various aspects of web scraping, including its tools and techniques, challenges, legality, and ethical considerations. This obtained theoretical foundation opens the way to the practical application of the acquired knowledge, and in this part, the beginning of the practical implementation of the acquired skills is initiated.

4.1 Motivation and Goals

The primary motivation behind this practical case study is to illustrate the versatility and efficiency of web scraping as a data collection tool, expanding its use to fields such as research and development. The primary intention is to demonstrate that scraping is not limited to its use in its most known areas, such as business intelligence [13] for gaining insights into price data, trends, competitors' prices, and market dynamics [15] but also may make significant contributions to scientific and technological advancements. This is particularly relevant in the growing field of computer vision, where large and varied datasets are essential for training and refining neural networks.

To exemplify this, the practical goal of the case study will be to focus on the creation of a comprehensive image dataset of motorcycles, categorized by manufacturer, model, and year of production. This dataset may serve as a resource for vehicle make and model recognition tasks, which is an important area in modern systems for monitoring road traffic and activity, analysing traffic behaviour [25], security access control systems in parking lots, buildings, and other restricted areas [36]. In addition, it can also complement license plate recognition systems, providing a higher level of security against fraudulent use of license plates in cases of traffic crimes [36].

However, the practical part of this work hopes to not only demonstrate the process of creating a dataset for such an industry but also demonstrate the potential of the methods and practices which can be useful for the entire field of artificial intelligence and machine learning. The applicable methods and techniques in this part can show a general idea of how such a practice of web scraping can help collect any other dataset for further research and development, for example, in the field of computer vision, providing a valuable resource for training neural networks. By showcasing the creation of this dataset, the thesis aims to expand the understanding of scraping activities as a versatile tool for technological research and development, emphasizing its role in fostering innovation and progress in the current digital era.

4.2 Data Analysis & Data Source

4.2.1 Data analysis

In this segment of the practical part, the project sets a clear and structured goal: the compilation of an image dataset encompassing 15 distinct motorcycle models from a variety of manufacturers. This number was chosen strategically to balance comprehensiveness with feasibility. The task of collecting a detailed dataset, even for a single brand alone, could potentially require the efforts of an entire team of data analysts. This complexity arises from the fact that each manufacturer of motorcycles may have a multitude of models, and each model can span several generations, usually separated by years of manufacturing, each having its own visual features. Consequently, the significant challenge arising if the objective is to precisely identify a particular model from a manufacturer utilizing computer vision techniques, it necessitates analysis of each generation of the respective model. A potential solution to this challenge may involve combining visually similar models into a single class for neural network training. This approach, while feasible, is notably labour-intensive and time-consuming.

However, considering that one of the primary tasks of this part is to showcase the technical application of web scraping in dataset creation rather than the preparation of an initial data list, a more focused approach has been adopted. This involves the compilation of a list of popular models based on various publicly available online sources [19][49][31]. This method ensures that the selected models retain popularity and relevance in the current market, thereby also increasing the practical value of the dataset.

4.2.2 Data list preparation

This section of the thesis is dedicated to the creation of an initial input list for which images will be collected, consisting of manufacturer, model, and production years.

At the moment of writing this thesis, no specific studies categorizing motorcycle models based on sales popularity were found. Therefore, as previously mentioned, the compilation of this list was primarily driven by information sourced from a variety of online sources.

An important aspect of the data list is the inclusion of the model years of production. As highlighted in section **4.2.1**, motorcycle models often undergo generational changes, each usually distinguished by unique visual characteristics. Thus, for example, one model may have dozens of generations starting from 1960 to the present day. Due to this reason, the list prioritizes the latest generations of specific models that are currently in production at the moment of writing this thesis. The selection process involved a detailed review of information from motorcycle manufacturers and online resources [30] that provides insights into model evolution and current market relevance. This approach ensured the accuracy and relevance of the data for the practical part, and the result list is demonstrated in **Table 1**.

Manufacturer	Model	Production years
Aprilia	RS660	2020-2024
Aprilia	RSV4	2017-2024
BMW	F900XR	2019-2024
BMW	M1000R	2020-2024
BMW	R1250GS	2018-2024
BMW	R18	2020-2024
Ducati	Scrambler 800	2016-2024
Harley-Davidson	Street Glide	2016-2024
Honda	CB500F	2017-2024
Honda	PCX125	2017-2024
Kawasaki	Ninja 400	2018-2024
Kawasaki	Z900	2018-2024
KTM	Super Duke 1290	2021-2024
KTM	Duke 390	2017-2024
Yamaha	MT-07	2016-2024

Table 1. Data list of manufacturers, models, years of production. Source: own

4.2.3 Data source

In order to compile a dataset, it is necessary to identify an initial source from which data can be extracted. In a study similar to the practical part of this thesis, Tafazzoli et al. [41] embarked on compiling a car dataset also specifically aimed at vehicle make and model recognition (VMMR), which was categorized by manufacturer, model, and years of production. Central to their approach was the utilization of online platforms related to vehicle sales as the initial source of data. A significant benefit of this approach is the inherent diversity and richness of the dataset. As the authors of the study [41] noted, images on these platforms are typically uploaded by a variety of different users, therefore, it is ensured that the images are taken from various view angles and with different devices.

Such variability of images adds a level of uniqueness and realism, which are crucial for developing robust and accurate neural networks in VMMR tasks.

Therefore, for the initial data resource, websites related to motorbike sales were selected, specifically websites autoscout24.com and motohunt.com. These websites were selected for their comprehensive list of different manufacturers and models, as well as their wealth of data. However, it is crucial to understand the significant disadvantage of this method. Images from such websites frequently contain irrelevant data, such as pictures of buildings, specific motorcycle components, people, and so on.

Nevertheless, such challenges, as well as solutions and techniques for data filtering, will be discussed in subsequent sections of the thesis.

4.3 Web Scraping System Planning

As the delve into the practical part continues, it becomes crucial to define the technological stack that will facilitate the creation of a dataset. As the core of a future scraping system was chosen Python due to the fact that most of the literature on which this thesis is based uses it as the main programming language, and many other studies identify it as one of the best languages specifically for web scraping [47][16].

In conjunction with Python, it is also planned to use the tools described earlier in the theoretical part, specifically Selenium for automating web browsers, which is important for sites with dynamic content, as well as Beautiful Soup for data extraction. Initially, the Scrapy framework described in **3.2.2.3** was considered for use, however, after further evaluation, this framework was considered unsuitable for the practical part of this thesis. Scrapy is known for its effectiveness in large-scale projects [12], especially those requiring advanced features such as proxy rotation and captcha solving, which are beyond the scope of this study.

Following the technological stack establishment, the next stage is to delve into the conceptual architecture of the web scraping system, as illustrated in **Figure 10**.

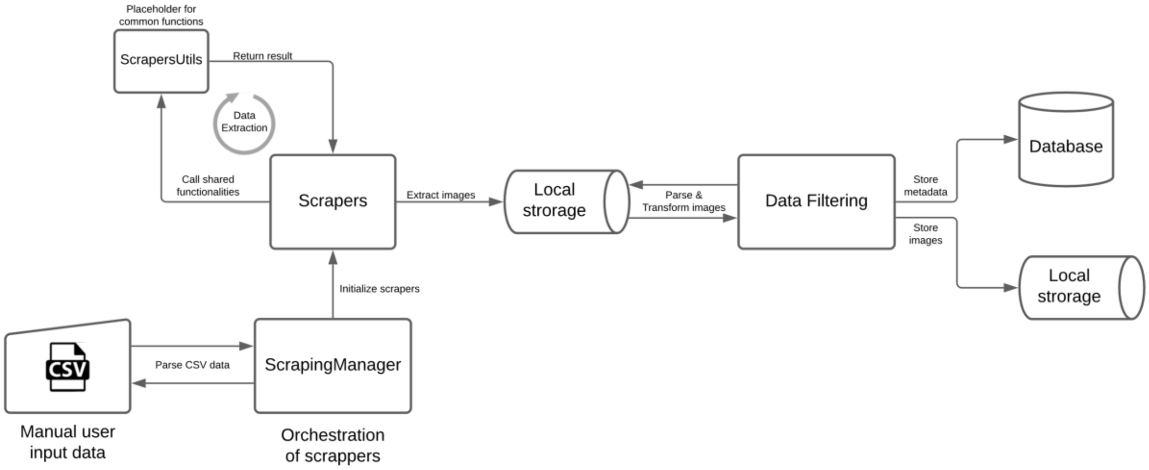


Figure 10. Conceptual architecture of web scraping system. Source: own.

The planned process initiates with a standalone CSV file containing links to search results for the needed motorcycle model on the targeted website. This file serves as a starting point for the scraping process. Subsequently, a specialized management module is utilized with the primary purpose of orchestrating scrapers, effectively distributing the URLs from the CSV file to them. The scrapers themselves are designed to extract URLs leading to individual pages containing images from the search results, and then retrieve images from these individual motorcycle pages.

Upon data extraction, the system employs a specialized filtering module to ensure the quality of the data, guaranteeing that only relevant and high-quality images are retained. Following the filtration process, the images are stored on a filesystem, adhering to a particular naming convention and file architecture. Concurrently, the metadata of images is stored in a database, facilitating enhanced data analysis capabilities.

The subsequent sections of the practical part will provide a comprehensive explanation of each stage in this process. This will include detailed information about how the management module orchestrates scrapers, the specific methodologies used for data extraction and filtering, as well as the reasons for the chosen approaches. These discussions will offer a more profound understanding of the operational dynamics of each component, emphasizing their importance in the overall efficiency of the web scraping system.

4.4 First scrapers

The approach to scrape motorcycle sales websites involves an analysis of their common structural patterns. Such websites typically allow for searches based on specific parameters like manufacturer, model, and years. The search result leads to individual sales pages, each containing information and images needed to extract. In order to effectively extract data, it was chosen to employ a two-module strategy for each website. The first scraper will be designed to collects URLs from the search result, which will be manually provided by inputting a URL leading to the desired motorcycle's search result on the website. This method can be classified as manual scraping [26][20], although it is not the most efficient way, this approach was found satisfactory due to the manageable size of the motorcycle data list. The second scraper will then proceed to retrieve images from the individual motorcycle listings. Another crucial parameter is the nature of the websites, which can be categorized into two main types: static and dynamic. For websites with a static nature, Beautiful Soup will be used primarily, whereas for the handling of dynamic websites, Selenium will be utilized.

4.4.1 Static nature scrapers

As was mentioned earlier, scrapers can be categorized as static and dynamic. In the case of **motohunt.com**, an analysis was conducted to determine its nature. The analysis entailed evaluating the website's response patterns, understanding the method of content loading, and determining if the website utilize client-side scripts for content rendering. As a result, it was discovered that this targeted website is more static, and most of the content is directly embedded in the HTML code delivered by the server without requiring additional client-side programming.

4.4.1.1 Links Scraper

As previously noted, a search on sales sites gives us a list of links leading to individual sales pages. This way allows for the extraction of these links, which can then be passed to another module for scraping images. As a result, a script was developed for extracting links, as illustrated in **Source Code 3**.

The '*MotoHuntLinksScraper*' utilizes BeautifulSoup for parsing HTML and extracting relevant links and is initialized with a '*base_url'* parameter, defaulting to:

https://motohunt.com/. This URL serves as the starting point for link scraping activities. The other functional components are responsible for:

- Link Identification. When the method ‘*extract_links’* is called, it employs BeautifulSoup to parse the HTML content, searching for ‘*div’* elements with the ‘*class card-body nolinkcolor’*. Within these elements, it locates ‘*a’* tags containing the ‘*href*’ attribute, the values of which are the links to individual motorcycle sale pages. These links are then processed through the ‘*ScrapingUtils.get_full_url’* method which helps to construct the full URL. This method of the utility class ensures that relative links from ‘*href’* are correctly converted to absolute URLs, considering the base URL of the website.
- Pagination handling. To navigate through multi-page results, the method ‘*find_next_page_link*’ identifies the link to the next page in the pagination sequence by searching for specific elements utilizing BeautifulSoup.
- Links Scraping. The ‘*scrape_all_links*’ method is designed to extract links across multiple pages, beginning from a given start URL. It also uses the ‘*ScrapingUtils.get_random_user_agent*’ method to vary user-agent strings in HTTP requests, thereby reducing the risk of scraper detection [29][47]. The method continues to accumulate links until there are no more pages left to scrape.

Source Code 3. Links Scraper. Source: own

```
from bs4 import BeautifulSoup
from utils.scraping_utils import ScrapingUtils

class MotoHuntLinksScraper:
    def __init__(self, base_url="https://motohunt.com"):
        self.base_url = base_url

    def extract_links(self, html_content):
        soup = BeautifulSoup(html_content, 'html.parser')
        links = []
        for card_body in soup.find_all("div", class_="card-body nolinkcolor"):
            a_tag = card_body.find("a", href=True)
            if a_tag:
                full_link = ScrapingUtils.get_full_url(self.base_url,
a_tag['href'])
                links.append(full_link)
        return links

    # Pagination handling
    def find_next_page_link(self, html_content):
        soup = BeautifulSoup(html_content, 'html.parser')
        next_link_tag = soup.find("span", id="results-text").find("a",
id="next", href=True)
        if next_link_tag:
            return ScrapingUtils.get_full_url(self.base_url,
next_link_tag['href'])
        return None

    def scrape_all_links(self, start_url):
        all_links = []
        next_page = start_url
        while next_page:
            # Requests via random user agents
            headers = {'User-Agent': ScrapingUtils.get_random_user_agent()}
            page_content = ScrapingUtils.get_page_content(next_page,
headers=headers)
            if page_content:
                links = self.extract_links(page_content)
                all_links.extend(links)
                next_page = self.find_next_page_link(page_content)
            else:
                break
        return all_links
```

4.4.1.2 Image Scraper

After successfully gathering URLs, the next step involves the extraction of images presented on individual sales pages. For this task, '*MotoHuntImageScraper*' was developed and illustrated in **Source Code 4**.

Source Code 4. Image Scraper for a Static Nature Website. Source: own

```
import time
from bs4 import BeautifulSoup
from utils.scraping_utils import ScrapingUtils

class MotoHuntImageScraper:
    def __init__(self, base_url="https://motohunt.com"):
        self.base_url = base_url

    def extract_and_download_images(self, page_url, subfolder_name):
        headers = {'User-Agent': ScrapingUtils.get_random_user_agent()}
        page_content = ScrapingUtils.get_page_content(page_url,
headers=headers)

        if not page_content:
            print(f"Error while processing page {page_url}")
            return

        soup = BeautifulSoup(page_content, 'html.parser')
        carousel_inner = soup.find("div", class_="carousel-inner")
        if carousel_inner is None:
            print("No images found on page.")
            return

        for carousel_item in carousel_inner.find_all("div", class_=["carousel-
item", "carousel-item active"]):
```

```
            img_tag = carousel_item.find("img")
            if img_tag and img_tag.get("data-src"):
                image_url = img_tag["data-src"]
                file_path = 
ScrapingUtils.download_image_with_unique_name(image_url, subfolder_name, 
headers)
                time.sleep(0.3)
                if file_path:
                    print("Image downloaded successfully:", file_path)
                else:
                    print(f"Failed to download image {image_url}")
```

This class was specifically designed to navigate through each previously gathered URL, parse the HTML content of the page, identify needed images, and download them.

The ‘*extract_and_download_images*’ method starts by sending a web request with previously described user-agents to mimic different browsers. Then it retrieves the HTML content of the page using ‘*ScrapingUtils.get_page_content*’. Upon receiving HTML content, the method employs Beautiful Soup to parse it and look for the ‘*div*’ with ‘*carousel-inner class*’. Then scraper iterates through each ‘*carousel-item*’ within the carousel, extracting the URLs of the images and passing them to ‘*ScrapingUtils.download_image_with_unique_name*’, a method designed to download and save images with unique filenames in a specified subfolder, thus preventing file name collision and ensuring organized data storage. In addition, the method also respects ethical standards by using a time delay [29] between image downloads in order to avoid overloading the web server.

4.4.2 Dynamic nature scrapers

Prior to the development of a scrapers for second targeted website, specifically for **autoscout24.com,** a comprehensive analysis of the site was conducted, following the same methodology of identifying common response patterns, analysis of client-side scripts and methods of content loading. This analysis revealed distinct dynamic features that significantly influence the scraping strategy. Such discovered dynamic features include:

- Cookie acceptance modal window. One notable difference from the previous targeted website is the presence of a modal window for accepting cookies. This window obstructs access to the main content of the site, necessitating the automatic acceptance of cookies.
- Lazy loading of images. Another notable feature is the implementation of lazy loading for images in galleries. This technique delays the loading of images until they are needed, specifically until the user interacts with the gallery. For instance,

on the motorcycle sale page, only two first photos are present in the DOM tree of the website. To upload other images, user must scroll through the gallery.

Considering these factors, Selenium was chosen as the primary tool to automate and address the dynamic challenges. A two-module strategy of scraping links and images, similar to the approach in **4.3.1**, was chosen. However, dynamic features did not have a major impact on the creation of the link scraper, therefore, the scraper for links extraction has the similar logic as the scraper illustrated in **Source Code 3** but has modified selector logic for Beautiful Soup. The primary challenge lay in the image scraper, however, this was effectively overcome using Selenium, as demonstrated in **Source Code 5.**

Source Code 5. Image Scraper for a Dynamic Nature Website. Source: own

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.common.exceptions import NoSuchElementException, TimeoutException
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.chrome.options import Options
from utils.scraping_utils import ScrapingUtils


class ScoutImageScraper:
    def __init__(self):
        chrome_options = Options()
        user_agent = ScrapingUtils.get_random_user_agent()
        chrome_options.add_argument(f"user-agent={user_agent}")

        self.driver = webdriver.Chrome()

    # Cookie modal window handler
    def handle_cookies(self):
        try:
            WebDriverWait(self.driver, 10).until(
                EC.visibility_of_element_located((By.CSS_SELECTOR, 'div[data-
testid="as24-cmp-container"]'))
            )
            accept_button = self.driver.find_element(By.CSS_SELECTOR,
                                                     'button[data-testid="as24-
cmp-accept-all-button"]')
            accept_button.click()
            time.sleep(2)
        except TimeoutException:
            print("Cookie modal not found or already accepted.")
        except Exception as e:
            print(f"Error handling cookies: {e}")

    def extract_and_download_images(self, url, subfolder_name):
        print(f"Processing URL: {url}")
        self.driver.get(url)
        self.handle_cookies()
        slide_number = 1
        while True:
            try:
                image_element = self.driver.find_element(By.CSS_SELECTOR,
                                                         f'div[aria-label="Go
to Slide {slide_number}"] picture img')
                image_url = image_element.get_attribute('src')
                if image_url:
                    # Make request with random user agent and save with uuid
                    file_path =
ScrapingUtils.download_image_with_unique_name(image_url, subfolder_name,
headers={
                        'User-Agent': ScrapingUtils.get_random_user_agent()})
                    if file_path:
                        print(f"Image saved successfully to {file_path}")
                    else:
                        print(f"Failed to download image from {image_url}")
                # Gallery sliding logic
                next_button = self.driver.find_element(By.CSS_SELECTOR,
'button[aria-label="Next Slide"]')
                next_button.click()
                slide_number += 1
                time.sleep(0.6)
            except NoSuchElementException:
                print("No more slides found. Moving to the next URL...")
                break

    def close_driver(self):
        self.driver.quit()
```

The scraper starts by initializing Selenium Chrome WebDriver with Chrome options, then random user-agent is assigned for each session. The other functionalities include:

- Cookie handling. WebDriver through the method '*handle_cookie*' automatically detects and interacts with the cookie acceptance modal window. Utilizing Selenium's '*WebDriverWait'* and '*EC.visibility_of_element_located'*, scraper waits for the presence of specific CSS elements related to the cookie window and then simulates a click on the acceptance button granting access to the main content of the web page.
- Dynamic Image Extraction. The method '*extract_and_download_images*' demonstrates Selenium's capabilities to handle dynamic content. Scraper identifies elements within the gallery using a CSS selector specific to each slide, then retrieves the image URL from the '*src'* attribute of the '*img*' element. For each image URL, the method from the utility class downloads the image, saving it with a

unique filename. Further, the method uses CSS selectors to find a button for moving to the next slide in the gallery and simulates a click on this button. This ensures that the next image is loaded into the site's DOM tree and the entire extraction process is repeated until there are no more images in the gallery. Thus, this approach allows to handle the lazy loading and extract all the images.

4.4.3 Evaluation of scrapers

During the course of this practical part, two fully functional scrapers consisting of two modules each were developed. The scrapers work sequentially, with the Link Scraper module first handling a URL that leads to a search result page specific to the desired motorcycle model on the designated website. Subsequently, it extracts links that lead to individual motorcycle pages. Next, the Image Scraper module retrieves images from these pages.

Furthermore, an evaluation of these scrapers was conducted to evaluate their efficiency. During a 25-minute test run, the scraper designed for **motohunt.com** successfully extracted 1,293 images. On the other hand, the scraper for **autoscout24.com** extracted 687 images within the same timeframe. The variance in output results can be explained by the differences in the scraping techniques employed for each website. A scraper designed for a static website does not utilize Selenium, resulting in much better performance by avoiding the need to automate interactions with the website for handling dynamic content challenges.

At this stage, it is evident that the initial scrapers are effectively performing their main functions, providing a strong foundation for further development. However, there is a significant opportunity to refine and optimize the existing scraping strategy. The subsequent part of this thesis will explore scalable strategies to increase the volume of data extraction while maintaining the quality and integrity of the acquired images.

4.5 Scalability and Optimization

This section of the thesis focuses on enhancing the current web scraping system with the aim of scaling up the amount of data extracted. The main focus will be on refining the current setup to increase its efficiency.

4.5.1 Scrapers orchestration

In the progression of scraping system development, an important step is the orchestration of the developed scrapers. The orchestration involves strategically coordinating various developed scraping scripts to operate simultaneously, therefore optimizing and scaling the data collection process. The main motivation behind this approach arises from the need to enhance efficiency and manage the scraping of different websites concurrently.

Choosing the appropriate mechanism for parallel execution in this context is crucial. Thereby, multithreading and multiprocessing emerge as two notable practices in parallel programming, each offering distinct advantages and applicability [29][34]. Multithreading involves running multiple threads within a single process. It’s a technique ideal for Input/Output operations, such as waiting for data from external sources like network responses. However, multithreading in Python is constrained by the Global Interpreter Lock (GIL), which allows only one thread to execute Python bytecode at a time, potentially limiting performance gains in CPU-bound tasks and creating bottlenecks [29].

On the other hand, multiprocessing is a form of true parallelism and involves utilizing multiple process units, each potentially operating on separate CPU cores, or even utilizing a GPU to execute different sections of a program concurrently [34]. Processes in multiprocessing setup bypass GIL, which allows execution of the same code lines and modifying separate instantiations of the same object, thereby eliminating the bottlenecks in multithreading [29].

Considering the aspects outlined, multiprocessing is identified as the most suitable choice for the scraping orchestration core. Currently, only two scrapers for two websites have been developed, one for a website with a more static nature and the other for a website with a more dynamic nature. However, the previously adopted modular design approach facilitates the seamless integration of additional scrapers for other websites. As indicated in the section **3.3.1**, JavaScript is employed on 98.8% of websites according to the W3C [45], therefore it's likely that possible future targeted websites will also be dynamic in nature. Consequently, the scraping of such sites often may require the use of Selenium, which, as mentioned in **3.2.2.2**, known for its resource-intensive nature, particularly in CPU usage. This highlights the logic behind selecting multiprocessing, culminating in the creation of a scraper orchestration manager, as illustrated in **Source Code 6**.

Source Code 6. Scraper Orchestration Manager. Source: own

```
import csv
import os
import time
import multiprocessing
from collections import defaultdict
from urllib.parse import urlparse
from scripts.motohunt_links import MotoHuntLinksScraper
from scripts.motohunt_images import MotoHuntImageScraper
from scripts.autoscout_links import ScoutLinksScraper
from scripts.autoscout_images import ScoutImageScraper
from utils.log_urls import LogInvalidUrls


class ScraperManager:
    # Mapping of domains for scrapers
    SCRAPER_MAPPING = {
        "motohunt.com": {
            "link_scraper": MotoHuntLinksScraper,
            "image_scraper": MotoHuntImageScraper
        },
        "www.autoscout24.com": {
            "link_scraper": ScoutLinksScraper,
            "image_scraper": ScoutImageScraper
        }
    }

    def __init__(self, csv_file, max_concurrent_tasks_per_site):
        self.csv_file = csv_file
        self.max_concurrent_tasks_per_site = max_concurrent_tasks_per_site

        parent_directory =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
        self.csv_file = os.path.join(parent_directory, 'data', 'bLinks.csv')
        self.urlLogger = LogInvalidUrls(
os.path.join(os.path.dirname(os.path.dirname(os.path.abspath(__file__))),
'bLinksLogs'))

    # Check for csv
    def wait_for_csv(self):
        while not os.path.exists(self.csv_file):
            print(f"bLinks.csv not found in 'data' folder. Please add the file
to continue... Recheck in 10 seconds")
            time.sleep(10)

    # Read and parse CSV, return dict of sites with their URLs and folder names
    def read_csv(self):
        self.wait_for_csv()
        sites = defaultdict(list)
        invalid_urls = []

        with open(self.csv_file, 'r') as file:
            reader = csv.reader(file, delimiter='|')
            for row in reader:
                if len(row) != 2:
                    invalid_urls.append((' | '.join(row), "Incorrect format:
expected URL | FolderName"))
                    print("Incorrect CSV line")
                    continue

                url, folder_name = row[0].strip(), row[1].strip()
                parsed_url = urlparse(url)
                if not parsed_url.scheme or not parsed_url.netloc:
                    invalid_urls.append((url, "Invalid URL"))
                    continue

                domain = parsed_url.netloc
                sites[domain].append((url, folder_name))

        self.urlLogger.log(invalid_urls)
        return sites

    # List of async tasks
    def scraping_for_domain(self, domain, url_pairs):
        print(f"Starting scraping for domain: {domain}")
        # Create a pool of worker processes for concurrent execution
        with multiprocessing.Pool(self.max_concurrent_tasks_per_site) as pool:
            # Async execute tasks for each URL pair in parallel
            results = [pool.apply_async(ScraperWorker.run_scrapers,
args=(url_pair,)) for url_pair in url_pairs]
            # Wait for all results to complete before proceeding
            for result in results:
                result.get()

    def start(self):
        sites = self.read_csv()
        processes = []

        for domain, url_pairs in sites.items():
            # Start a separate process for each domain
            p = multiprocessing.Process(target=self.start_scraping_for_domain,
args=(domain, url_pairs))
            processes.append(p)
            p.start()

        for process in processes:
            process.join()


class ScraperWorker:
    @staticmethod
    def run_scrapers(url_folder_pair):
        url, subfolder_name = url_folder_pair
        domain = urlparse(url).netloc
        # Get the corresponding scraper classes for the domain
        scraper_classes = ScraperManager.SCRAPER_MAPPING.get(domain)

        if not scraper_classes:
            print(f"No scraper found for {url}")
            return
        # Initialize the scrapers
        link_scraper = scraper_classes["link_scraper"]()
        image_scraper = scraper_classes["image_scraper"]()
        # Perform scraping
        all_links = link_scraper.scrape_all_links(url)
        print(f"Scraped {len(all_links)} links from {url}")
        # For each link, call the method to extract and download images
        for link in all_links:
            image_scraper.extract_and_download_images(link, subfolder_name)
        print(f"Completed image scraping for {url}")

if __name__ == '__main__':
    csv_file = 'bLinks.csv'
    max_concurrent_tasks_per_site = 3  # Specify if needed
    manager = ScraperManager(csv_file, max_concurrent_tasks_per_site)
    manager.start()
```

The '*ScraperManager*' class begins with a declaration of the '*SCRAPER_MAPPING*' dictionary. This mapping associates each domain with its respective scraping classes: one for links ('*link_scraper*') and another for images ('*image_scraper*'). For example, **motohunt.com** is mapped to '*MotoHuntLinksScraper*' and '*MotoHuntImageScraper*', both illustrated in **4.3.1**. This design allows a flexible choice of suitable scrapers depending on the domain being proceeded, hence enabling a modular and scalable system. The other functionality includes:

- Configuration and initialization. The class is initialized with two parameters: '*csv_file*' and '*max_concurrent_tasks_per_site'*. The '*csv_file*' specifies the path to the '*bLinks.csv*' file, which contains the URLs leading to the search result page of specific needed motorcycle on targeted websites. As described previously, these links are collected manually, and the format of each line in the *'bLinks.csv'* file should be "URL | FolderNameToSaveImages". Lastly, the '*max_concurrent_tasks_per_site*' sets a limit on the number of processes that can be run per website.
- CSV file processing. The '*wait_for_csv*' method checks for the presence of a '*bLinks.csv*' file. Once the CSV file is located, the '*read_csv*' method parses it to extract URLs and corresponding folder name for image storage. Method organize URLs by their domains, ensuring tasks are allocated correctly. Additionally, the method validates each line in CSV, checking for proper format. Invalid formats are logged using the '*LogInvalidUrls*' utility class, therefore providing a mechanism to track and address input data errors.

- Domain-based scraping. The method ‘*scraping_for_domain*’ creates a pool of worker processes based on the value of ‘*max_concurrent_tasks_per_site*’. This pool allows simultaneous execution of multiple scraping processes across different URLs within the same domain.
- Orchestration of scraping tasks. The ‘*start*’ method serves as the orchestration for the entire scraping activity. Method iterates over each domain obtained from the ‘*read_csv*’ method and initiates a separate process for each domain’s scraping tasks.

Another component integral to this scraping system is the ’*ScraperWorker*’ class. This class operates in tandem with ‘*ScraperManager*’, focusing on the execution of specific scraping tasks. Using the static ‘*run_scrapers*’ methods, it dynamically selects and initialize the appropriate scraper class based on the URL’s domain.

4.5.2 Orchestrated scraping system evaluation

After creating an orchestrator for scrapers, the next stage involves testing the system to assess its efficiency. The method adopted for this evaluation mirrored the approach described in **4.3.3**, involving a controlled 25-minutes test run. The primary objective of this test was to determine the quantity of data that could be extracted within this timeframe, utilizing the enhanced scraping system. As a result, the current scraping system successfully downloaded 6,551 images within 25 minutes. This output was distributed across two targeted websites, with 4,594 images from **motohunt.com** and 1,957 images from **autoscout24.com**. This increase in data retrieval, as compared to the initial tests, demonstrates the efficiency of the orchestrated system in scaling up the data extraction process.

However, during the test, it was observed that both targeted websites have limits on the number of advertisements displayed for specific motorcycle search. One website limited its display to 400 advertising, while the second one to 216. Such a limitation directly influences the number of extracted links. Consequently, the overall number of extractable images decreases. To address this restriction, the website requires users to use filters for more precise searches, such as filtering by specific years or price ranges. Therefore, by using website filters throughout the link collection process for search result pages, it will be possible to gather all links of individual motorcycle sales pages and extract a larger amount of images. However, given the amount of data obtained from the test, it was deemed unnecessary to implement such measures at this stage.

4.5.3 “Raw” dataset creation

After evaluating the orchestrated scraping system, the following stage is to collect the initial “raw” dataset of motorcycle images based on the predefined list of motorcycle models in **Table 1**. The process starts by manually collecting URLs from two targeted websites, which are directed to pages displaying search results of specific needed motorcycle. After manually collecting links and placing them into a specified CSV file, the scraping manager was initiated, and within a couple of hours, a total of 38,551 images were extracted. On average, from 1000 to 4000 images for each motorcycle model. The exception was one model that was not present on one of the targeted websites.

However, it is crucial to acknowledge that the dataset compiled at this stage is “raw” in nature. Although it contains a large number of images, as previously mentioned in **4.2.3**, some of them are irrelevant, therefore, such images require filtering to separate them from motorcycle images.

4.6 Data Quality & Filtration

As previously mentioned in the study conducted by Tafazzoli et al. [41], while the chosen method of collecting data from online sales platform is beneficial for computer vision tasks due to the diversity of data, it also has its disadvantages in the form of irrelevant images. These may include images of buildings, people, specific parts of motorcycles such as steering wheels or wheels, or images captured at angles that do not fully show the entire motorcycle. An example of such images is shown in **Figure 11**.

The dataset currently contains multiple irrelevant images, which might significantly impact the main possible application of the dataset, specifically the performance and accuracy of neural networks models to recognize the manufacturer and model of motorcycle. Therefore, the challenge at this stage is to ensure the quality of scraped images and their relevance.

To address this challenge, this section of the thesis sets a primary objective to choose and implement a methodology that can effectively distinguish and separate full motorcycle images from those that are irrelevant or unsuitable for the dataset intended use.

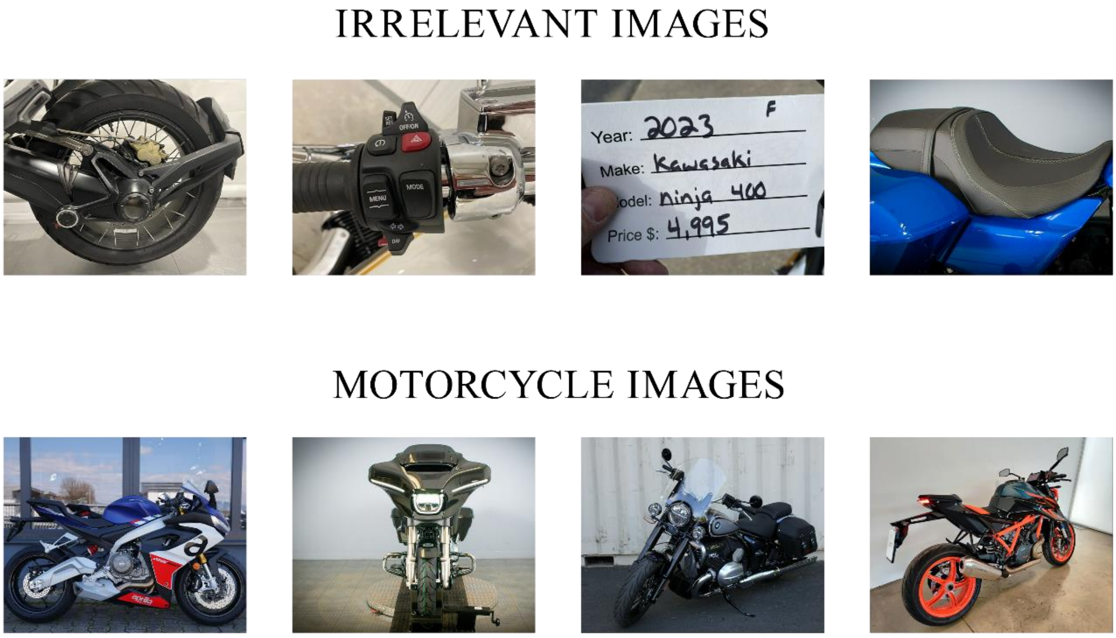


Figure 11. Irrelevant & Motorcycle images. Source: own

4.6.1 Manual filtration

One straightforward approach is manual filtration. This method involves analysing and categorizing each image individually to keep only the relevant images in the dataset and remove others. The strength of this method in its accuracy, as it relies on human judgement to identify nuances in images that an automated system might overlook.

However, the practical value of this approach decreases as the dataset grows. With the web scraping system potentially incorporating more scrapers and a boarder range of motorcycles to scrape, the volume of data quickly increases. Therefore, manual filtration rapidly becomes very time-consuming and the potential for human error and inconsistency grows. As such, while manual filtration may be effective for small datasets, its applicability is limited for larger, more complex datasets, therefore, more automated, scalable solutions for data filtration are becoming essential.

4.6.2 Automated filtration

The utility of an automated system for filtering large datasets was effectively showcased in a study conducted by Lyu et al. [28], where authors successfully used the YOLO (You Only Look Once) object detection algorithm to remove irrelevant images, such as noisy photos and car interiors from their car dataset. To filter the motorcycle dataset, this study took a similar approach but utilizing YOLOv8 developed by Ultralytics. This enhanced version of YOLO will be used to precisely identify and separate images that do not align with the main objective of the dataset, focusing on retaining images that fully portray motorcycle.

However, images classified as irrelevant will not be removed but will be kept for possible future research value. This decision is based on a study conducted by Yang et al. [51], where image filtering was also applied to compile a dataset of car images. As highlighted in the study, specific images of car components, such as, for example, headlights, were found to have a potential for further research and use. Therefore, retaining irrelevant images that may contain individual parts of a motorcycle is viewed as a strategic step that lays the foundation for future advancements and analysis.

4.6.2.1 Classify model creation

In the initial phase of creating a model for image separation, the pre-trained YOLOv8 model for object detection was selected, and the first step involved annotating objects within images, with the aim of training the model to distinguish between relevant and redundant images in the dataset. However, the initial approach of utilising an object detection model did not completely reflect the main objective of data filtration. The primary requirement was not to identify and locate objects but rather to classify each image as a whole.

Subsequently, a more suitable solution was found. The focus has shifted to the utilization of a pre-trained model specifically for image classification. According to Ultralytics documentation [43], classifier models are more suited for filtration tasks, as they are designed to identify the class to which an image belongs without needing to identify the object’s position or precise shape. Accordingly, a dataset for training a neural network was prepared in accordance with Ultralytics’ dataset preparation guidelines [44] for classifier models. For this purpose, 3057 images of random, unique, and diverse motorcycles were selected utilizing a previously developed scraping system and divided into two classes. The first class labelled as “motorcycle”, comprised of 2000 clear images of motorcycles and the second class labelled as "redundant”, consisted of 1057 images that were previously discussed as irrelevant. Each class of dataset was further divided according to the 70/30 rule, with 70% allocated for training and 30% for validation. This is a common approach that has been observed in many other studies [9][35].

After compiling the training dataset, the neural network was trained for 30 epochs, where each epoch represents a complete cycle through the entire training dataset [35].

Following the training, performance metrics were recorded and analysed, as illustrated in **Figure 12**.

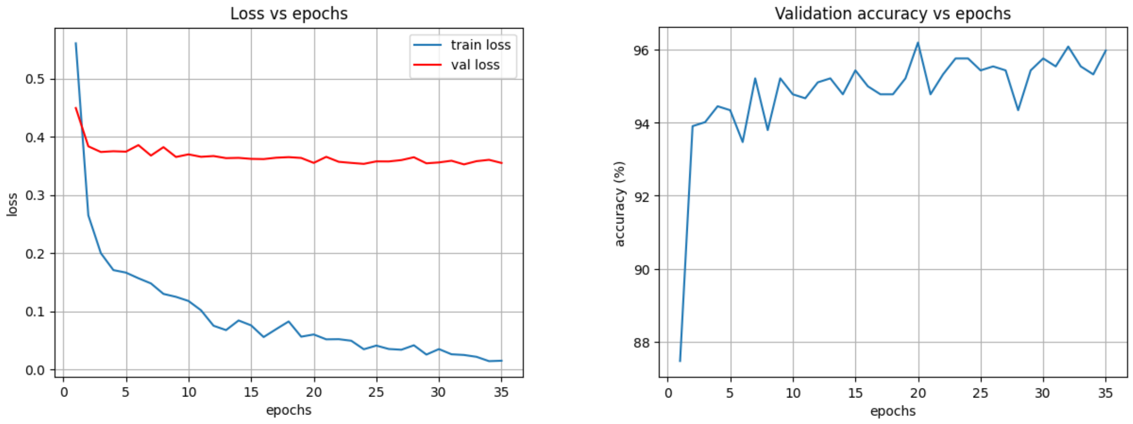


Figure 12. Results of training neural network model for classification. Source: own

The first graph, labelled as “Loss vs epochs”, demonstrates the model’s learning evolution across 30 epochs. The training loss, represented by the blue line, decreases from the initial epochs and stabilizes as the number of epochs progresses. This indicates an enhancement in the model’s image classification accuracy [35]. The validation loss, represented by the red line, decreases at a slower rate and shows minor fluctuations, which may indicate the moments where the model is learning from the new validation data.

The second graph, titled “Validation accuracy vs epochs”, displays the model’s accuracy on the validation set over epochs.

In order to calculate neural network model accuracy, it is necessary to analyse the confusion matrix, a matrix that precisely illustrates the performance of the learning algorithm [35], which is illustrated in **Figure 13**.

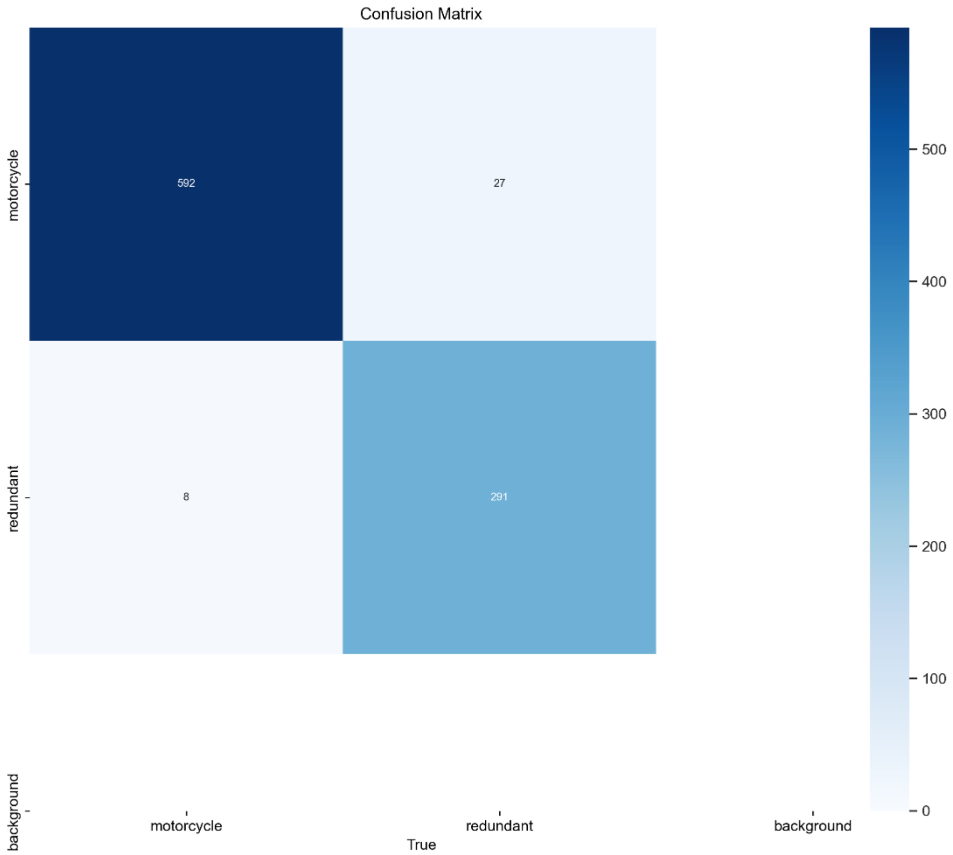


Figure 13. Confusion matrix of a trained model. Source: own

The equation for accuracy, represented in **Equation 1**, is given by:

$$ACC = \frac{TP + TN}{FP + FN + TP + TN}$$

Equation 1

The values of TP, TN, FN, and FP are directly derived from the confusion matrix illustrated in **Figure 13**, where:

- TP (True Positive). Correctly identified motorcycle images, shown in the top left cell of the confusion matrix.
- TN (True Negative). Correctly identified images as not being motorcycle (redundant class), shown in the bottom right cell.
- FN (False Negative). Motorcycle images, incorrectly classified as redundant images, shown in the bottom left cell.
- FP (False Positive). Redundant images, incorrectly classified as motorcycle, displayed in the top right cell of the matrix.

After calculating the values, it was determined that the accuracy of the neural network model is approximately 96.19%. Such a level of accuracy indicates that the neural network has effectively learned patterns from the training data and is able to apply them to validation data. This is crucial for the model's ability to perform well on unseen images, demonstrating its reliability and effectiveness in classifying and filtering motorcycle images within the dataset.

4.6.2.2 Wrapping model

Following the creation of the classification model, a specific separate module was created to use a trained model on scraped images, as demonstrated in **Source Code 7**. The ‘*ImageFilterer*’ is designed to interact with the filesystem, where images are stored and organized into folders based on manufacturer, model, and years of production. Upon module execution, each image is classified as either a “motorcycle” or “redundant” and then automatically placed in subfolders with their respective name inside the parent folder where the “raw” images were originally stored.

Source Code 7. Image Filtration Module. Source: own

```
import shutil
from pathlib import Path
from PIL import Image
from ultralytics import YOLO
from concurrent.futures import ThreadPoolExecutor

class ImageFilterer:
    def init(self, weights_path, image_extensions=None):
        self.model = YOLO(weights_path)
        self.image_extensions = image_extensions or ['*.jpg', '*.jpeg',
'*.png', '*.webp']

    def process_image(self, image_path, motorcycle_folder,
redundant_folder):
        try:
            result = self.model(Image.open(image_path))[0]
            if result.probs.top1 == 0 and result.probs.top1conf > 0.50:
# Class 0 is 'motorcycle'
                shutil.move(image_path, motorcycle_folder /
image_path.name)
            else:
                shutil.move(image_path, redundant_folder /
image_path.name)
        except Exception as e:
            print(f"Error processing {image_path}: {e}")

    def process_folder(self, folder_path):
        try:
            motorcycle_folder = folder_path / 'motorcycle'
            redundant_folder = folder_path / 'redundant'

            os.makedirs(motorcycle_folder, exist_ok=True)
            os.makedirs(redundant_folder, exist_ok=True)
            for ext in self.image_extensions:
                for image_path in folder_path.glob(ext):
                    self.process_image(image_path, motorcycle_folder, 
redundant_folder)

            # Rename the processed folder
            if "FILTERED" not in folder_path.name:
                processed_folder_name = folder_path.name + "__FILTERED"
                processed_folder_path = folder_path.parent / 
processed_folder_name
                folder_path.rename(processed_folder_path)
        except Exception as e:
            print(f"Error processing folder {folder_path}: {e}")

if __name == "main":
    script_directory = os.path.dirname(os.path.abspath(file))

    # Initialize the ImageFilterer
    weights_path = os.path.join(script_directory, 'weights', 
'moto_filterer.pt')
    if not os.path.exists(weights_path):
        print("Weights file not found. Please check the weights 
directory.")
        exit()

    image_filterer = ImageFilterer(weights_path)

    # Ask user for parent folder and number of threads
    parent_folder_path = input("Please enter the path to the parent 
folder: ")
    parent_folder = Path(parent_folder_path)
    max_workers = int(input("Please specify number of threads: "))

    # Process each folder in parallel
    with ThreadPoolExecutor(max_workers) as executor:
        futures = []
        for subfolder in parent_folder.iterdir():
            if subfolder.is_dir() and "__FILTERED" not in 
subfolder.name:

futures.append(executor.submit(image_filterer.process_folder, 
subfolder))
        for future in futures:
            future.result()  # Wait for all threads to complete
```

The module starts by initializing a trained model for classification and the optional parameter ‘*image_extensions*’ to specify the types of images to process. Other functionalities of the module include:

- Image processing. The method ‘*process_image*’ takes an image file path and the paths for subfolders, then it uses a trained model to classify each image and place them to corresponding subfolders based on classification. If the model classified the image as a motorcycle with confidence above 50%, the image moved to the "motorcycle" subfolder, otherwise it goes to the "redundant" subfolder.

- Folder-level processing. The method '*process_folder*' creates a "motorcycle" and "redundant" subfolders and iterates over all images, applying the '*process_image*' method to each.

To enhance the efficiency of handling big volumes of data, the module employs multithreading. As a result, this parallel processing reduces the time required for filtering images, making the module more robust and scalable.

Subsequently to the quantitative analysis in **4.5.2.1**, a qualitative evaluation was conducted. The trained model, encapsulated within this module, underwent a classification test on approximately 1200 random images of motorcycles, and personal observation was employed to monitor the performance of the model. The results showed a high level of accuracy, with the model accurately classifying most of the images. Despite occasional misclassifications, the overall performance can be regarded as satisfactory. This demonstrates the model's readiness for practical application in filtering the "raw" dataset created in **4.4.3**.

4.7 Data Storage & Final Dataset

This final section of the practical part focuses on establishing the optimal methodologies for storing and analysing the dataset after the filtration step. Furthermore, this section marks the creation of the final dataset, achieving one of the main objectives of this thesis.

4.7.1 Cloud storage

Effective data storage is crucial for handling large datasets. And one of such effective solutions can be cloud storage, which offers scalability and reliability. For example, Amazon Simple Storage Service (S3) is a good candidate, offering a scalable cloud-based storage with variable object sizes, ranging from a minimum of 1 byte to a maximum of 5 gigabytes [6].

The architecture of S3 is designed to manage an almost "infinity" number of objects, each organized within a system known as "buckets". Buckets are the primary containers that store data objects, functioning similar to the directories in a file system. Each object within S3 is encapsulated in a byte-stream format and may be uniquely identified by a Uniform Resource Identifier (URI), which is associated with a specific bucket. Moreover, S3's architecture, by utilizing a SOAP or REST-based interface, allows seamless data retrieval ('*get(uri)*') and updating mechanisms ('*put(uri, bytestream)*') [6].

Given these attributes, S3 could serve as great storage for a motorcycle dataset, especially in the case of the possible further scalability of the scraping system. In fact, the book "Python Web Scraping Cookbook" [16] by Michael Heydt, which helped in the development of scrapers, provides practical insights into how images can be effectively loaded into S3 buckets, thus, considering all the aspects above, initially it was planned to use a cloud storage approach.

However, it is crucial to clarify that this service is not free. The costs associated with data storage were important factors to consider, especially since this work is an academic rather than commercial project. Consequently, a different storage solution was selected. Nonetheless, in the context of large-scale projects, cloud-based storage solutions can be the optimal choice for extensive datasets.

4.7.2 Relational Database Approach

In considering alternative storage solutions, the option of utilizing relational databases was also explored, particularly for image storing. Nevertheless, this approach is generally not regarded as good practice due to several inherent limitations.

Images in relational databases are typically stored as binary large objects (BLOBs), and a large number of BLOB objects can significantly increase the size of the database. Such an increase in size often leads to performance issues, meaning that database will require more resources and time for operations such as backups, replication, data retrieval, and upload.

Despite these challenges, the storage of smaller binary objects, such as image thumbnails, may be acceptable. As an example, research conducted by Sears et al. [38] indicates that relational databases are more efficient for objects smaller than 256 kilobytes, whereas filesystems are better suited for handling objects larger than 1 megabyte.

Therefore, considering the specifics of scraped images in the motorcycle dataset, where images typically exceed 1 megabyte, it was decided not to use the database as primary storage. Nevertheless, it was decided not to completely remove the use of database. The database can effectively store metadata for each image, including details such as manufacturer, model, year of production, and classification results from a previously trained neural network model. By utilizing a database in this manner, it may simplify data organization and facilitate data analysis, making the relational database a great tool for the overall data management strategy and results analysis.

4.7.3 File System Storage

The final decision in terms of this bachelor thesis was to leave the existing logic for storing the images inside the file system after scraping and filtration procedures.

A file system is an essential component of any computer, designed to specifically store, organize, and manage files on storage devices. As indicated in the previously mentioned study by Sears et al. [38], this approach can be considered optimal for images, which typically exceed 1 megabyte in size, and most of the images in the current dataset fall under these criteria. Furthermore, all scraped images are organized according to a particular file structure and naming conventions. As a result, it makes it easier to handle and access vast amounts of data.

However, as previously mentioned, to enhance data analysis capabilities, the metadata of each image will be stored in a relational database. Such an approach effectively combines the advantages of filesystem and database, optimizing overall management. The methodology and implementation of this technique will be provided in the following sections of this chapter.

4.7.4 Database Implementation

For the efficient storage and management of image metadata, the relational database management system MySQL was chosen. The first step involves the creation of a database called "*Motostorage*", within this database, a table named "images" was established with a structure illustrated in **Source Code 8.**

Source Code 8. Schema for Storing Image Metadata. Source: own

```
CREATE TABLE images (
    id INT AUTO_INCREMENT PRIMARY KEY,
    manufacturer VARCHAR(255),
    model VARCHAR(255),
    year_start INT,
    year_end INT,
    image_type ENUM('motorcycle', 'redundant'),
    file_path VARCHAR(800),
    file_name VARCHAR(255),
    date_added TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE (file_name)
);
```

Each field within the table serves the following purposes:

- ‘*id*’. A primary key, unique auto-incremented identifier for each record.
- ‘*manufacturer*’, ‘*model*’, ‘*year_start*’, ‘*year_end*’, ‘*image_type*’. Fields for storing specific details of the image, reflecting the manufacturer, model, years of production, and category of image as either “motorcycle” or “redundant”.
- ‘*file_path*’, ‘*file_name*’. Fields for storing information about the image name and location.
- ‘*date_added*’. Timestamp to record a date and time when a record was inserted.
- ‘*UNIQUE*’. Constraint for *f’'ile_name*’ to avoid duplicates.

After initial database preparation, the next step involves developing a module responsible for placeholding the logic of uploading metadata. Consequently, the module represented in **Source Code 9** was developed to facilitate this process.

Source Code 9. Database Uploader Module. Source: own

```
import mysql.connector
from mysql.connector import Error

class DatabaseUploader:
    def __init__(self, config):
        self.config = config
        self.connection = None
        self.connect()

    def connect(self):
        # Establish connection through config file
        try:
            self.connection = mysql.connector.connect(**self.config)
            self.connection.autocommit = True
        except Error as e:
            print(f"Error connecting to MySQL database via config file: {e}")

    def insert_metadata(self, manufacturer, model, year_start, year_end,
image_type, file_path, file_name):
        if self.connection.is_connected():
            cursor = self.connection.cursor()
            # Ignore clause in case of duplicates, MySQL 5.7.5 and higher
            add_image_query = (
                "INSERT IGNORE INTO images "
                "(manufacturer, model, year_start, year_end, image_type,
file_path, file_name) "
                "VALUES (%s, %s, %s, %s, %s, %s, %s)"
            )
            image_data = (manufacturer, model, year_start, year_end,
image_type, file_path, file_name)
            cursor.execute(add_image_query, image_data)
            cursor.close()

    def close(self):
        # Close db connection
        if self.connection.is_connected():
            self.connection.close()
```

The ‘*DatabaseUploader*’ initializes with a configuration file, which contains the database connection credentials. Other functionalities include:

- Connection establishment. The ‘*connect*’ method establishes a connection to the MySQL database using the provided configuration from the config file.
- Metadata insertion. Method ‘*insert_metadata*’ takes multiple parameters about image metadata and inserts this data into the table using a SQL query.
- Connection close. The ‘*close*’ method ensures that the database connection is properly closed after operations are completed.

After the creation of the ‘*DatabaseUploader*’, the filtration module, referenced in **Source Code 7**, was modified. The main modifications include new methods which utilise the data upload logic. Now, due to modifications made, after the filtration step, the user is prompted to decide whether to enter the metadata of the filtered images into the database.

4.7.5 Final Dataset Creation

This section represents the achievement of one of the main objectives of this thesis: the creation of the final motorcycle dataset. To start this final phase, the “raw” dataset obtained in **4.4.3** underwent the filtration procedure utilizing the modified filtration module. Subsequently, the images were sorted and classified, then their metadata was loaded into the database for further analysis.

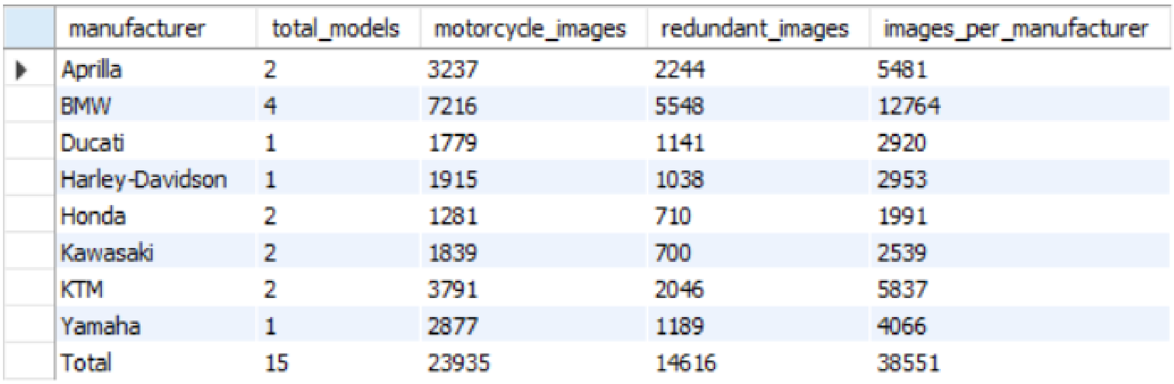
Consequently, once metadata has been loaded, it is possible to conduct analysis of the results obtained. For this purpose, a particular SQL query demonstrated in **Source Code 10** was utilized.

Source Code 10. SQL Query for Manufacturer Image Summary. Source: own

```
SELECT
    IFNULL(manufacturer, 'Total') AS manufacturer,
    COUNT(DISTINCT model) AS total_models,
    SUM(CASE WHEN image_type = 'motorcycle' THEN 1 ELSE 0 END) AS
motorcycle_images,
    SUM(CASE WHEN image_type = 'redundant' THEN 1 ELSE 0 END) AS
redundant_images,
    COUNT(*) AS images_per_manufacturer
FROM
    images
GROUP BY
    manufacturer WITH ROLLUP;
```

The query is designed to display the list of manufacturers, including the number of unique models they have, the counts of "motorcycle" and "redundant" images, and the total number of images for each manufacturer. In addition, query display a summarized row labeled "Total", which represents the sum of all images and models in the table.

The output of this query is visually represented in **Figure 14**. An analysis of the query output revealed that the database now contains 15 distinct motorcycle models and a total of 38,551 images. This result is in perfect alignment with the data that was previously collected and described in **4.4.3**, meaning that all 38,551 images from the "raw" dataset were filtered and successfully loaded.



manufacturer	total_models	motorcycle_images	redundant_images	images_per_manufacturer
Aprilla	2	3237	2244	5481
BMW	4	7216	5548	12764
Ducati	1	1779	1141	2920
Harley-Davidson	1	1915	1038	2953
Honda	2	1281	710	1991
Kawasaki	2	1839	700	2539
KTM	2	3791	2046	5837
Yamaha	1	2877	1189	4066
Total	15	23935	14616	38551

Figure 14. Visual Summary of Image Metadata by Manufacturer. Source: own.

To further contextualize the dataset, an analysis was performed according to the initial data list provided in **Table 1**, which outlined the targeted manufacturers and models and served as the foundation for dataset creation.

In order to show results achieved, the SQL query from **Source Code 10** was modified, as illustrated in **Source Code 11**.

Source Code 11. SQL Query for Image Count by Attributes. Source: own

```
SELECT
    manufacturer, model, year_start, year_end,
    SUM(CASE WHEN image_type = 'motorcycle' THEN 1 ELSE 0 END) AS
motorcycle_images,
    SUM(CASE WHEN image_type = 'redundant' THEN 1 ELSE 0 END) AS
redundant_images,
    COUNT(*) AS images_total
FROM
    images
GROUP BY
    manufacturer, model, year_start, year_end;
```

The new query provides the count of images by each manufacturer and model, including the start and end years. And the results of this query have been placed in **Table 2** for better visualization.

Manufacturer	Model	Production years	Motorcycle images	Redundant images	Total images
Aprilia	RS660	2020-2024	1219	709	1928
Aprilia	RSV4	2017-2024	2018	1535	3553
BMW	F900XR	2019-2024	2003	1350	3353
BMW	M1000R	2020-2024	593	441	1034
BMW	R1250GS	2018-2024	2098	1650	3748
BMW	R18	2020-2024	2522	2107	4629
Ducati	Scrambler 800	2016-2024	1779	1141	2920
Harley-Davidson	Street Glide	2016-2024	1915	1038	2953
Honda	CB500F	2017-2024	999	540	1539
Honda	PCX125	2017-2024	282	170	452
Kawasaki	Ninja 400	2018-2024	775	273	1048
Kawasaki	Z900	2018-2024	1064	427	1491
KTM	Super Duke 1290	2021-2024	1718	835	2553
KTM	Duke 390	2017-2024	2073	1211	3284
Yamaha	MT-07	2016-2024	2877	1189	4066

Table 2. Final dataset results overview. Source: own.

Upon reviewing the results, it is evident that the main goal of gathering data on the selected models has been successfully achieved. The dataset contains mostly models with more than 1000 high-quality motorcycle images. Notable exceptions include certain recently released models with a lower marketplace presence at the moment and one model (PCX15) potentially listed under a different name on one of the targeted websites.

Nevertheless, the developed scraping system and filtration module have demonstrated effectiveness and scalability, meaning that the existing system allows for potential expansion in terms of image quantity. For the current scope of the thesis, the quantity and quality of the scraped images are considered satisfactory. This accomplishment represents the completion of the practical part of the thesis, achieving the primary objective of compiling a comprehensive and well-organized motorcycle dataset.

5 Results and Discussion

5.1 Results

5.1.1 Web Scraping Methodology and Implementation

The initial phase of this study involved a thorough analysis of web scraping methodologies and tools. This investigation played a crucial role in understanding how web scraping, as an automated method of data extraction, may play a significant role in obtaining large quantities of data for various different applications.

5.1.2 Developed Scraping & Filtration System

In the practical part of the thesis, the techniques and tools described in the theoretical part were successfully employed in the form of a scraping system capable of acquiring a large volume of images. The system was designed with a focus on modularity and scalability, consisting of a specifically structured module for each targeted website and an orchestrator for these modules.

Each scraping module in a system consisted of two closely linked scrapers, one for gathering links and the other for collecting images. The link collection scraper within each module was designed to extract URLs that leads to specific needed motorcycle page. Once the URLs are gathered, a second scraper, designed for image collection, retrieves images from these pages.

Further, to simplify and scale a scraping process, the specialized orchestrator was developed to manage the operations of scraping modules, allowing simultaneous multiprocessing data acquiring. The developed orchestrator maximized the efficiency of the scraping process, showcasing the system’s capability to manage multiple data extraction activities concurrently.

Furthermore, to enhance the quality and relevance of the scraped data, an automated filtering process utilizing the YOLO algorithm was developed. By utilizing this algorithm to train a specialized neural network model, the process of filtering data to separate motorcycle images from unrelated or irrelevant content was successfully automated, ensuring the dataset’s quality and relevance for further use in the field of VMMR. Additionally, the system incorporated a feature for appending images metadata to the database, facilitating more efficient analysis and interpretation of the scraped data.

In summary, the scraping system developed in this thesis demonstrates a good degree of modularity, efficiency, and adaptability. Through a dual-script architecture within each scraping module, coupled with an orchestrator and advanced filtering, the system may serve as a sophisticated tool for motorcycle dataset compilation. In addition, the approach demonstrated through the development of this system can be used as a foundation not only for collecting images of motorcycles but also for gathering images to compile any other dataset for computer vision tasks.

5.1.3 Compiled dataset

The creation of the motorcycle dataset represents the culmination of the practical part and the accomplishment of the main objective of this thesis. Featuring 15 unique models from various manufacturers and encompassing a total of 38,551 images and 23,935 high-quality, diverse clean images of motorcycles, the compiled dataset, detailing the make, model, and production years, serves as an excellent starting point for Vehicle Make and Model Recognition (VMMR) applications.

It is important to note that a key aspect of Vehicle Make and Model Recognition (VMMR) and other machine learning endeavours is having an adequate number of images per class to ensure effective training. In this context, considering the compiled dataset, the amount of images collected for each motorcycle model seems to be sufficient for effective neural network training. However, it is important to note that the quantity of images for each class is abstract and relies entirely on the specific of project and the architecture on which the training of the model for object recognition and classification will be based.

Another important aspect to mention is that the current dataset focuses on only 15, primarily recent and popular, models. Nevertheless, as previously stated in the first chapter of the practical part, the aim was not only to compile a dataset but also to demonstrate the underlaying architecture patterns to accomplish this and similar tasks. And the developed scraped system, with its modular and scalable design, enables straightforward expansion of the dataset. Therefore, if the objective is to create a dataset not only for 15 unique models but, for example, for more than 100 models, the current system will allow this to be accomplished.

5.2 Discussions

5.2.1 Insights on the Developed Scraping System

One notable aspect of the developed system is its reliance on manual user input. As outlined in the practical part, the user is required to manually search for a needed motorcycle on a targeted website, then manually retrieve the URL leading to the search result on the page and paste it into a specialized CSV file in a specific format.

An alternative method could involve automating the scraping process by removing dependence on the user. As an example, the system may automatically scrape each model from a specific manufacturer, save the results, and then repeat the process for another manufacturer. However, this automatic approach has its own disadvantages, primarily due to the complexity and variety within motorcycle models. As previously mentioned, different generations of a single model can have differences in their visual appearance. Therefore, scraping each model without considering their generations might result in a dataset with reduced utility for tasks requiring precise make, model, and generation identification.

Furthermore, such an automatic approach may have another disadvantage. During scraping, it was discovered that not all models may be present on the targeted website, or they may have different name. For instance, only a limited number of images were obtained for the model "Honda PCX125" since this particular model was not listed on one of the targeted websites, which is focused on sales for the North American region. Upon analysing publicly available sources, it was discovered that this model is mainly produced for the Asia-Pacific and European markets. Consequently, the number of advertisements of this specific model was limited on the website for the North American market.

These factors once again demonstrate that the process of gathering the information for the required motorcycles to scrape can be resource-intensive and time-consuming. Therefore, due to the factors described above, a decision was made to reach a compromise and leave the system's reliance on the user.

5.2.2 Data Quality and Filtering

Data quality was a crucial factor to consider during the course of this thesis. The context of this study primarily deals with media files, particularly with images, which required a specialized approach. Since the method of scraping vehicle sales websites inherently introduce the challenge of extracting images with irrelevant content [41], to address this, a specialized neural network model based on the YOLO algorithm was introduced for content filtering. This approach aligns with many other studies in similar domains, confirming its suitability and effectiveness as the primary solution.

However, an area for potential optimization lies in the current standalone nature of the filtering process. Currently, filtering can only be done after the scraping process, and this introduces an extra step before the data becomes clean and usable. For instance, implementing real-time filtering synchronized with the scraping process, wherein data is automatically sorted in the directory where images are saved, could speed up the overall acquisition of the final dataset and remove the dependence on manual intervention for starting the filtering process.

In summary, while the current method successfully separates relevant from irrelevant images, thereby ensuring the quality of the data, the merging of scraping and filtering processes represents a great potential for further improvements.

5.2.3 Reflections on the Compiled Dataset

The main application of the compiled dataset is in Vehicle Make and Model Recognition (VMMR) tasks within the field of computer vision. The current dataset contains a solid amount of high-quality images for 15 different motorcycle models, primarily selected based on their newness and growing popularity in the market. This choice ensures that the dataset is relevant and up-to-date, reflecting current trends in the motorcycle industry.

However, a key consideration regarding the dataset is its size. Although the choice of selecting exactly 15 models was justified due to complexness in initial data analysis and data list preparation, considering the unique visual characteristics of different models generations, this dataset may not be sufficient for more complex, extensive VMMR projects. Nevertheless, at the time of writing this thesis, no substantial studies were found that specifically focus on the motorcycle model recognition. This factor indicates an unexplored area in the field of VMMR, therefore, the current dataset, despite its limitations in size, can be considered the foundation and starting point for studies dealing with motorcycle recognition. If there is a need to expand a dataset by obtaining more data for further models, the developed scraping system will effectively handle this operation, allowing growth with minimum challenges.

In summary, the compiled dataset provides a fundamental basis for VMMR applications, particularly in the context of motorcycles. The current scope, emphasizing recent and gaining popularity models, provides a solid starting point for future research in this field.

6 Conclusion

In conclusion, the research conducted in this bachelor thesis has showcased the synthesis of theoretical insights and practical implementations in the field of web scraping. The theoretical part of this study involved a thorough analysis of web scraping methodologies and tools, providing essential insights into the complexities and potentials of automated data extraction for various applications. This analysis also addressed various challenges, ethical and legal nuances, providing excellent foundation for the subsequent practical work.

The practical part of this thesis successfully applied knowledges gained from theoretical study. As a result, a scraping system was developed to efficiently extract a wide array of motorcycle images. Advanced filtration techniques were integrated to ensure the data quality, and a database system was implemented for effective data analysis. The result of these efforts was the creation of a comprehensive motorcycle image dataset, categorized by make, model, and production years, serving as a valuable resource for Vehicle Make and Model Recognition tasks in computer vision field.

Overall, this thesis not only fulfils its primary goal of developing a motorcycle image dataset but also contributes to a deeper understanding of web scraping methodologies, emphasizing their versatility as a tool for various applications. This study highlights that while web scraping can be highly effective in computer vision, its utility may cover a broad range of other various fields, demonstrating its potential as a universal solution for data collection challenges.

7 References

[1] ACHSAN, Harry T. Yani and WIBOWO, Wahyu Catur. A Fast Distributed Focused-web Crawling. Procedia Engineering, [s.l.], v. 69, 2014, p. 492-499. ISSN 1877-7058. DOI 10.1016/j.proeng.2014.03.017.

[2] AKROUT, Ismail, FERIANI, Amal, and AKROUT, Mohamed. Hacking Google reCAPTCHA v3 using Reinforcement Learning. 2019 Conference on Reinforcement Learning and Decision Making (RLDM). 2019. DOI 10.48550/arXiv.1903.01003

[3] Architecture overview. Architecture overview - Scrapy 2.11.0 documentation [online]. 1 January 2024. [Accessed 9 January 2024]. Available from: https://docs.scrapy.org/en/latest/topics/architecture.html

[4] Beautiful Soup documentation. Beautiful Soup Documentation - Beautiful Soup 4.12.0 documentation [online]. [Accessed 29 November 2023]. Available from: https://www.crummy.com/software/BeautifulSoup/bs4/doc/

[5] BOCK, Kevin, PATEL, Daven, HUGHEY, George, and LEVIN, Dave. uncaptcha: A low-resource defeat of recaptcha's audio challenge. In: USENIX Workshop on Offensive Technologies. 2017.

[6] BRANTNER, Matthias, FLORESCU, Daniela, GRAF, David, KOSSMANN, Donald and KRASKA, Tim. Building a database on S3. Proceedings of the 2008 ACM SIGMOD international conference on Management of data. 9 June 2008. DOI 10.1145/1376616.1376645.

[7] BURSZTEIN, Elie, AIGRAIN, Jérôme, MOSCICKI, Angelika, and MITCHELL, John C. The End is Nigh: Generic Solving of Text-based CAPTCHAs. In: Workshop on Offensive Technologies. 2014.

[8] CHU, Zi, et al. Blog or block: detecting blog bots through behavioral biometrics. Computer Networks [online]. 2013, 57(3), 634–646 [viewed 19 January 2024]. ISSN 1389-1286. Available from: doi:10.1016/j.comnet.2012.10.005

[9] DEQUITO, C J, DICHAVES, I J, JUAN, R J, MINAGA, M Y, ILAO, J P, M O CORDEL, II and DEL GALLEGO, N P. Vision-based bicycle and motorcycle detection using a YOLO-based network. Journal of Physics: Conference Series. 1 May 2021. Vol. 1922, no. 1, p. 012003. DOI 10.1088/1742-6596/1922/1/012003.

[10] DONGO, Irvin, CADINALE, Yudith, AGUILERA, Ana, MARTÍNEZ, Fabiola, QUINTERO, Yuni and BARRIOS, Sergio. Web scraping versus Twitter API. Proceedings of the 22nd International Conference on Information Integration and Web-based Applications &amp; Services. 2020. DOI 10.1145/3428757.3429104.

[11] Downloader middleware. Downloader Middleware - Scrapy 2.11.0 documentation [online]. 15 January 2024. [Accessed 17 January 2024]. Available from: https://docs.scrapy.org/en/latest/topics/downloader-middleware.html#scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware

[12] FAN, Yuhao. Design and implementation of distributed crawler system based on Scrapy. IOP Conference Series: Earth and Environmental Science. 2018. Vol. 108, p. 042086. DOI 10.1088/1755-1315/108/4/042086.

[13] FERRARA, Emilio, DE MEO, Pasquale, FIUMARA, Giacomo and BAUMGARTNER, Robert. Web data extraction, applications and techniques: A survey. Knowledge-Based Systems. November 2014. Vol. 70, p. 5–30. DOI 10.1016/j.knosys.2014.07.007.

[14] GOLD, Zachary and LATONERO, Mark. Robots Welcome? Ethical and Legal Considerations for Web Crawling and Scraping. Washington Journal of Law, Technology & Arts, vol. 13, no. 3, 2018, p. 275. Available at: https://digitalcommons.law.uw.edu/wjlta/vol13/iss3/4.

[15] HENRYS, Kasereka. Importance of web scraping in e-commerce and e-marketing. SSRN Electronic Journal. January 2021. DOI 10.2139/ssrn.3769593.

[16] HEYDT, Michael. Python Web Scraping Cookbook: Over 90 proven recipes to get you scraping with Python, micro services, Docker and AWS. Packt Publishing, 2018. ISBN 9781787285217.

[17] Internet “data scraping”: A Primer for counseling clients. New York Law Journal [online]. 15 July 2013. [Accessed 23 January 2024]. Available from: https://www.law.com/newyorklawjournal/almID/1202610687621/

[18] Jonker, H., Krumnow, B., Vlot, G., 2019. Fingerprint Surface-Based Detection of Web Bot Detectors. In: K. Sako, S. Schneider, P. Ryan, eds. Computer Security – ESORICS 2019. Lecture Notes in Computer Science, vol 11736. Cham: Springer. Available from: https://doi.org/10.1007/978-3-030-29962-0_28

[19] JOSELYN, Raven. Europe’s top 10 motorcycle thrills: 2023 best-sellers unleashed. Cars & Bikes [online]. 22 December 2023. [Accessed 13 February 2024]. Available from: https://www.automobilesnext.com/2023/12/europes-top-10-motorcycle-thrills-2023-best-sellers-unleashed/

[20] KHDER, Moaiad. Web scraping or web crawling: State of Art, Techniques, approaches and application. International Journal of Advances in Soft Computing and its Applications. 2021. Vol. 13, no. 3, p. 145–168. DOI 10.15849/ijasca.211128.11.

[21] Kirjazovas, V. (2020, November 5). Web scraping automotive industry data. Web Scraping Automotive Industry Data. https://oxylabs.io/blog/web-scraping-in-automotive-industry

[22] Krotov, V., & Johnson, L. (2022). Big web data: Challenges related to data, technology, legality, and ethics. Business Horizons. https://doi.org/10.1016/j.bushor.2022.10.001

[23] Krotov, V., Johnson, L., & Silva, L. (2020). Legality and Ethics of Web Scraping. Communications of the Association for Information Systems, 47, 539–563. https://doi.org/10.17705/1cais.04724

[24] LAWSON, Richard. Web Scraping with Python: Successfully scrape data from any website with the power of Python. Packt Publishing, 2015. ISBN 9781782164364.

[25] LEE, Hyo, ULLAH, Ihsan, WAN, Weiguo, GAO, Yongbin and FANG, Zhijun. Real-time vehicle make and model recognition with the residual SqueezeNet architecture. Sensors. 26 February 2019. Vol. 19, no. 5, p. 982. DOI 10.3390/s19050982.

[26] LOTFI, Chaimaa, SRINIVASAN, Swetha, ERTZ, Myriam and LATROUS, Imen. Web scraping techniques and applications: A literature review. SCRS CONFERENCE PROCEEDINGS ON INTELLIGENT SYSTEMS. January 2021. P. 381–394. DOI 10.52458/978-93-91842-08-6-38.

[27] Luscombe, A., Dick, K., & Walby, K. (2021). Algorithmic thinking in the public interest: navigating technical, legal, and ethical hurdles to web scraping in the social sciences. Quality & Quantity. https://doi.org/10.1007/s11135-021-01164-0

[28] LYU, Y., SCHIOPU, I., CORNELIS, B., and MUNTEANU, A. Framework for Vehicle Make and Model Recognition—A New Large-Scale Dataset and an Efficient Two-Branch–Two-Stage Deep Learning Architecture. Sensors, 2022, vol. 22, no. 21, article 8439. Available at: https://doi.org/10.3390/s22218439.

[29] Mitchell, Ryan. Web Scraping with Python: Collecting More Data from the Modern Web. 2nd edition. O'Reilly Media, Inc., 2018. ISBN 978-1491985571

[30] Motorcycles Database. autoevolution [online]. [Accessed 13 February 2024]. Available from: https://www.autoevolution.com/moto/

[31] NASH, Carole. Top 10... best selling motorcycles and scooters of 2023: Carole Nash. Carole Nash UK [online]. 15 January 2024. [Accessed 13 February 2024]. Available from: https://www.carolenash.com/news/blogs/jake-dixon/detail/top-10-best-selling-motorcycles-and-scooters-of-2023

[32] PARK, K.S., PAI, V.S., LEE, K.-W., and CALO, S.B. Securing web service by automatic robot detection. In: Proceedings of the 2006 USENIX Annual Technical Conference. Boston, MA, USA, 30 May–3 June 2006, p. 255-260.

[33] Patel, H. (2018, December 31). How web scraping is transforming the world with its applications. Medium. https://towardsdatascience.com/https-medium-com-hiren787-patel-web-scraping-applications-a6f370d316f4

[34] PRUDENTE, Norlan. Web Crawler Optimization. [online] Accessed on 25 February 2024 from: https://nprudente.com/files/WhitePaper.pdf.

[35] RASCHKA, Sebastian, and Vahid MIRJALILI. Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition. Packt Publishing, 2019. ISBN 9781789955750

[36] SARFRAZ, M. Saquib, SAEED, Ahmed, KHAN, M. Haris and RIAZ, Zahid. Bayesian prior models for vehicle make and model recognition. Proceedings of the 7th International Conference on Frontiers of Information Technology. 16 December 2009. DOI 10.1145/1838002.1838041.

[37] Scriptable headless browser. PhantomJS [online]. [Accessed 14 December 2023]. Available from: https://phantomjs.org/

[38] SEARS, R., INGEN, C.V., and GRAY, J. To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem? 2007. Available at: https://doi.org/10.48550/arXiv.cs/0701168.

[39] SINGRODIA, Vidhi, MITRA, Anirban and PAUL, Subrata. A review on web scrapping and its applications. 2019 International Conference on Computer Communication and Informatics (ICCCI). 2019. DOI 10.1109/iccci.2019.8821809.

[40] SUCHACKA, Grażyna, et al. Efficient on-the-fly Web bot detection. Knowledge-Based Systems [online]. 2021, 223, 107074 [viewed 15 January 2024]. ISSN 0950-7051. Available from: doi:10.1016/j.knosys.2021.107074

[41] TAFAZZOLI, Faezeh, FRIGUI, Hichem and NISHIYAMA, Keishin. A large and diverse dataset for improved vehicle make and model recognition. 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW). July 2017. DOI 10.1109/cvprw.2017.121.

[42] TAYLOR, Petroc. Data Growth Worldwide 2010-2025. Statista [online]. 16 November 2023. [Accessed 18 November 2023]. Available from: https://www.statista.com/statistics/871513/worldwide-data-created/

[43] ULTRALYTICS. Classify. Classify - Ultralytics YOLOv8 Docs [online]. 3 February 2024. [Accessed 23 February 2024]. Available from: https://docs.ultralytics.com/tasks/classify/

[44] ULTRALYTICS. Image classification datasets overview. Ultralytics YOLOv8 Docs [online]. 7 January 2024. [Accessed 24 February 2024]. Available from: https://docs.ultralytics.com/datasets/classify/

[45] Usage statistics of client-side programming languages for websites. W3Techs [online]. [Accessed 14 January 2024]. Available from: https://w3techs.com/technologies/overview/client_side_language

[46] UZUN, E., YERLİKAYA, T. and KIRAT, O., 2018. Comparison of python libraries used for web data extraction. Journal of the Technical University-Sofia Plovdiv Branch, Bulgaria. [online], 24, pp.87-92. Available at: https://erdincuzun.com/wp-content/uploads/download/plovdiv_2018_01.pdf [Accessed 3 December 2023].

[47] Vanden Broucke, Seppe; Baesens, Bart. Practical Web Scraping for Data Science: Best Practices and Examples with Python. 1st edition. Apress Berkeley, CA, 2018. ISBN 978-1-4842-3581-2

[48] WebDriver. Selenium [online]. [Accessed 14 December 2023]. Available from: https://www.selenium.dev/documentation/webdriver/

[49] WHITE, Walter. Europe: Most selling motorcycles in 2023. Auto User Guide [online]. 21 October 2023. [Accessed 13 February 2024]. Available from: https://www.autouserguide.com/blogs/europe-most-selling-motorcycles-in-2023/

[50] WIRFS-BROCK, Allen and EICH, Brendan. JavaScript: The first 20 years. Proceedings of the ACM on Programming Languages. 2020. Vol. 4, no. HOPL, p. 1–189. DOI 10.1145/3386327.

[51] YANG, Linjie, LUO, Ping, LOY, Chen Change and TANG, Xiaoou. A large-scale car dataset for fine-grained categorization and verification. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 24 September 2015. DOI 10.1109/cvpr.2015.7299023.

[52] ZHENG, Chunmei, HE, Guomei and PENG, Zuojie. A study of web information extraction technology based on Beautiful soup. Journal of Computers. 30 July 2015. Vol. 10, no. 6, p. 381–387. DOI 10.17706/jcp.10.6.381-387.

8 List of pictures, tables, source codes, equations, and abbreviations

8.1 List of figures

8.2 List of tables

8.3 List of source codes

8.4 List of equations

8.5 List of abbreviations

AJAX - Asynchronous JavaScript and XML

Amazon S3 - Amazon Simple Storage Service

API - Application programming interface

BLOB - Binary Large Object

CAPTCHA - Completely Automated Public Turing test to tell Computers and Humans Apart

CSS - Cascading Style Sheets

DOM - Document Object Model

HTML - Hypertext Markup Language

HTTP - Hypertext Transfer Protocol

JSON - JavaScript Object Notation

REST - Representational State Transfer

SOAP - Simple Object Access Protocol

SQL - Structured Query Language

URI - Uniform Resource Identifier

URL - Uniform Resource Locator

VMMR - Vehicle Make and Model Recognition

VPN - Virtual private network

XML - Extensible Markup Language

XPath - XML Path Language

YOLO - You Only Look Once