How to Build a Web Scraper for Social Media

Oskar Lloyd
Christoffer Nilsson

Computer Systems Developer
Bachelor
15 hp
Spring 2019
Final seminar: 2019-06-03

Supervisor: Gion Koch Svedberg
Examiner: Dimitris Paraschakis
Abstract
In recent years, the act of scraping websites for information has become increasingly relevant. However, along with this increase in interest, the internet has also grown substantially and advances and improvements to websites over the years have in fact made it more difficult to scrape. One key reason for this is that scrapers simply account for a significant portion of the traffic to many websites, and so developers often implement anti-scraping measures along with the Robots Exclusion Protocol (robots.txt) to try to stymie this traffic. The popular use of dynamically loaded content – content which loads after user interaction – poses another problem for scrapers. In this paper, we have researched what kinds of issues commonly occur when scraping and crawling websites – more specifically when scraping social media – and how to solve them. In order to understand these issues better and to test solutions, a literature review was performed and design and creation methods were used to develop a prototype scraper using the frameworks Scrapy and Selenium. We found that automating interaction with dynamic elements worked best to solve the problem of dynamically loaded content. We also theorize that having an artificial random delay when scraping and randomizing intervals between each visit to a website would counteract some of the anti-scraping measures. Another, smaller aspect of our research was the legality and ethicality of scraping. Further thoughts and comments on potential solutions to other issues have also been included.
Table of Contents

Glossary 1

1 Introduction 2
   1.1 Research Questions 3
   1.2 Scope 3

2 Method 4
   2.1 Design and Creation 4
      2.1.1 Awareness 4
      2.1.2 Suggestion 4
      2.1.3 Implementation 4
      2.1.4 Evaluation 4
      2.1.5 Conclusion 4
   2.2 Literature Review Process 5
      2.2.1 Determining Search Terms 5
      2.2.2 Literature Review Results 6
   2.3 Discussion 7

3. Literature Review Results 7
   3.1 Crawling 8
   3.2 Dynamic Web Crawling 8
   3.3 Scraping 9
   3.4 Prevention Strategies 10
   3.5 Uses 10
   3.6 Legal Aspects 10
   3.7 Robots.txt 11

4 Results 12
   4.1 Anti-Scraping Techniques 12
   4.2 Structure of Conversations on Social Media 12
   4.3 Node Selection 14
   4.4 Scraping Dynamic Content 15
   4.5 Avoiding Dynamic Content 17
   4.6 Legal and Ethical Aspects 17
   4.7 Putting It All Together 18
   4.8 Prototype Evaluation 20

5 Discussion 21
   5.1 Dynamic Content 21
   5.2 Scraper Architecture 21
   5.3 Avoiding Duplicate Work 21
   5.4 Legal and Ethical Considerations 22
Glossary

AJAX – technology that enables dynamic web content by asynchronous requests.
CAPTCHA – automatic test for determining if a user is human or machine, commonly implemented on websites to avoid spam and scraping.
CSS – Cascading Style Sheets are used for styling HTML elements.
Dynamic web content – content which is generated on user interaction rather than included directly with the original HTTP request, such as clicking a button.
HTML – standard markup language for creating web pages.
HTTP – a protocol which is used to communicate on the Internet.
Robots.txt – a plain text file included in most websites’ root directory, this defines rules for crawlers and scrapers visiting the site.
Spider – a class responsible for carrying out web crawling and defining scraping behaviour, used within the Scrapy framework and elsewhere in the literature.
Web crawler – software designed to visit web resources by navigating URLs and collecting metadata, often shortened to just crawler.
Web scraper – software designed to mine data from a website, often shortened to just scraper.
XPath – descriptive query language to select nodes in XML documents, also used for HTML documents.
1 Introduction

The web is an ever-changing, incredibly big source of different kinds of data – some useful and some not so useful. Occasionally, we want to collect and store this data for various purposes such as for research or archiving, but manually doing so would likely take more than a lifetime. To that end (and to avoid the massive amounts of traffic that scraping incur), some websites and companies provide public avenues (APIs) for interested parties to connect to and request data, e.g. Twitter’s API [1]. However, not all websites provide this, which is when scraping is useful.

A web scraper is a system which i) sends an HTTP request for a given web page, ii) hopefully receives a valid response containing the HTML, and iii) parses the contents of the HTML for specific information e.g. comments in a social media post or in a forum thread. While some might use the terms scraper and web crawler synonymously, it is important to distinguish between them. Crawlers generally tend to mostly collect metadata concerning pages and do not actually collect much raw data from each individual one – in contrast to a scraper. A web scraper however often implements the functionality of both components.

There are a number of issues that can arise when attempting to scrape a web page – some of the biggest challenges of scraping are how to handle dynamic content (content which loads on user interaction, rather than included in an initial HTTP response) and how to deal with anti-scraping techniques such as frequency analysis and interval analysis [2]. Important historical events on the web could be at risk of being forgotten unless they are archived, and if scrapers do not properly handle dynamic content and anti-scraping techniques, information could be lost forever. An example of this is when MH17 crashed in 2015. Ukrainian separatists had claimed responsibility on Twitter, and later removed the claim. Thanks to archiving efforts, this tweet was saved [2].

In this paper, we have performed a literature review on the topic of web scrapers and crawlers, with a focus on what issues can emerge when scraping and how to solve these. In order to gain a firmer grasp on these issues and to test any possible solutions a scraper was built\(^1\) using Scrapy [3], a framework for building scrapers and crawlers in Python and Selenium [4], a framework originally designed for automated browser testing. We decided to focus specifically on social media due to the prevalence of dynamic content on these platforms as well as the heavy anti-scraping policies and measures. There is also a big interest in storing this data, whether for marketing purposes, research, or perhaps political reasons.

One other topic of significance and a part of our research are the questions of legality and ethicality. Is it always legal to store the collected data, especially when it can be used to identify individuals and their interests? Is it ever morally right to ignore companies’ and websites’ wishes to not be scraped?

\(^1\) [https://github.com/cloudbells/scrapy-reddit](https://github.com/cloudbells/scrapy-reddit)
1.1 Research Questions

- How do we solve the problem of scraping dynamic content in social media?
  - What different kinds of problems exist, and how do we solve them?
- How should a scraper handle anti-scraping techniques?
  - Is it unethical or illegal to bypass these?

1.2 Scope

Our research is specific to dynamic versions of the social media websites www.reddit.com\(^2\), www.facebook.com\(^3\), and www.twitter.com\(^4\). Our implementation scrapes the dynamic version of Reddit and uses the scraping framework Scrapy and the web automation framework Selenium in a Python environment. The data we are scraping is the public conversation data on these platforms. For ethical reasons \(^7\) and to err on the side of caution, no data is saved.

---

\(^2\) [https://www.reddit.com](https://www.reddit.com)
\(^3\) [https://www.facebook.com](https://www.facebook.com)
\(^4\) [https://www.twitter.com](https://www.twitter.com)
2 Method

2.1 Design and Creation
Our research process contained several steps [15], [16]: i) awareness of the problem, ii) suggestion for solving the problem, iii) implementation of our suggestion, iv) the evaluation of our implementation, and v) concluding the results of our implementation. Our research is focused on producing design principles and identifying problems with developing a complete scraping system for dynamic social media websites as well as preliminary suggestions for solving these.

2.1.1 Awareness
The awareness step represents the initial awareness of a problem [16]. We decided to start by performing a literature review in order to understand previous research into the topic as well as to understand some of the issues that can arise when scraping websites. Most importantly, we wanted to know how other researchers had designed their web scrapers and crawlers.

2.1.2 Suggestion
Based on the results of the awareness step a proposal for a research topic is defined that seeks to solve the problems identified. The proposal aims to define the components and relations between components of the proposed system and is used as a basis for the implementation step. The results of the literature review as well as creative input from the researchers are used for this step. The creative input introduces a measure of non-repeatability [16] that the reader should be aware of.

2.1.3 Implementation
The implementation step builds on the system proposed in the suggestion step and is then used for an evaluation of the overall design of the system. The implementation process was iterative, and contained several stages [16]: i) requirements analysis, ii) architectural design, iii) coding, and iv) testing in order to develop a prototype.

2.1.4 Evaluation
The prototype implementation of the previous step is used in order to evaluate the design and see if the proposed system could be used to solve the problems identified at the start of the process [16]. The evaluation answers the question of whether the proposed design and proof-of-concept implementation solves the problems identified previously. Evaluation of the implementation may also lead to awareness of new issues which can be used in a new iteration of the research.

2.1.5 Conclusion
The research ends with a consolidation of what has been learned throughout the process as well as identifying areas that need further research in order to communicate the results to the research community [16].
2.2 Literature Review Process

We performed a literature review in order to study the current research on the topic. The literature review was carried out by searching for research articles in the online databases ACM Digital Library\(^5\) (ACM) and IEEE Xplore Digital Library\(^6\) (IEEE). In order to search we constructed search terms based on our research questions. For each term we added synonyms and placed them in a search matrix. Newer articles were prioritized and their relevance determined by first reading the titles. Assuming the title sounded relevant we next read the abstract and finally the whole article, dismissing irrelevant articles.

We performed a literature review in order to ascertain the current knowledge base in the subject of web scraping, the difficulties of scraping, and the preferred architecture of a system involving scraping social media websites such as Reddit, Facebook, and Twitter.

Another facet of our research was how to handle dynamic pages, which load content not with the original HTTP request but with user interaction with the page – such as clicking a button. The entire process was iterative in nature as we determined more and more areas which we wanted to explore in more detail – in particular how scraping social media websites is perceived from an ethical standpoint and how it relates to the somewhat newly established General Data Protection Regulation (GDPR).

Our literature review process consisted of several stages: i) finding search terms, ii) searching for relevant articles, and iii) reading the relevant articles.

2.2.1 Determining Search Terms

First, we needed to determine which search terms to use, and to do this we extracted important keywords from our research questions and put these in a table. Then, each word was given a number of synonyms. See table 1 and 2 for examples.

<table>
<thead>
<tr>
<th>web</th>
<th>crawl</th>
<th>dynamic</th>
<th>design</th>
</tr>
</thead>
<tbody>
<tr>
<td>“social media”</td>
<td>crawler</td>
<td>javascript</td>
<td>architecture</td>
</tr>
<tr>
<td>forum</td>
<td>crawling</td>
<td>js</td>
<td>pattern</td>
</tr>
<tr>
<td>internet</td>
<td>scrape</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>scraper</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>scraping</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>spider</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1. First table of search terms.

\(^5\) https://dl.acm.org
\(^6\) https://ieeexplore.ieee.org
Table 2. Second table of search terms.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>web</td>
<td>crawl</td>
<td>ethics</td>
</tr>
<tr>
<td>“social media”</td>
<td>crawler</td>
<td>morals</td>
</tr>
<tr>
<td>forum</td>
<td>crawling</td>
<td>law</td>
</tr>
<tr>
<td>internet</td>
<td>scrape</td>
<td>gdpr</td>
</tr>
<tr>
<td>scraper</td>
<td>“general data protection regulation”</td>
<td></td>
</tr>
<tr>
<td>scraping</td>
<td>spider</td>
<td></td>
</tr>
</tbody>
</table>

The original keywords extracted from the research questions are listed in the top row. For our search terms, each column represents an OR function, and each row an AND function.

2.2.2 Literature Review Results

From the first search table we constructed the following search string:

(web OR "social media" OR forum OR internet) AND (crawl OR crawler OR crawling OR scrape OR scraper OR scraping OR spider) AND (dynamic OR javascript OR js) AND (architecture OR design OR pattern)

This yielded 59 results from ACM and 12940 results from IEEE.

The second table gave us the following:

(web OR "social media" OR forum OR internet) AND (crawl OR crawler OR crawling OR scrape OR scraper OR scraping OR spider) AND (ethics OR morals OR law OR gdpr OR "general data protection regulation")

This resulted in 77 hits on ACM and 3858 hits on IEEE.

Given our limited amount of time, reading all of these articles was not feasible. Instead, we elected for an iterative approach. We ordered the articles first by relevance, and then by date published. We then read each article’s title and given that it sounded relevant we then read the article’s abstract. If the abstract sounded promising, we continued by reading the whole article, and then adding it to our knowledge base. If any particular cited works was put forward by the article as especially significant or if any citation was used in several articles we did the same for those. This was repeated until we were satisfied we had gained
an overview of the research situation and recognized shared citations and mentions of other research in the articles.

In addition to the academic texts we have also reviewed the GDPR [18] to study some of the legal and ethical aspects of scraping systems.

In total 28 articles were thoroughly analyzed. The results of the literature review is presented in sections relating to different common topics identified in the literature.

2.3 Discussion
A literature review is essential in order to identify potential gaps of knowledge and to understand the state of current research [17]. Because of the large scope and different methods involved in developing scrapers we believed a literature review was important for our research.

Because of the several different techniques and software components that are used in web scraping we thought that Design and Creation was an appropriate method for getting an overview of how a scraping system works in practice. Since our research was focused on producing design principles and identifying problems, iteration and implementation works well to this end. Design and Creation can be criticized for introducing elements of non-reproducibility [16] but is nonetheless a popular research method.

Structured experiments could also have been used to determine the cost of scraping dynamic content versus traditional static content as well as determining the efficiency of our suggestions. This would have been more appropriate if we had focused specifically on one component of a scraping system, such as comparing algorithms for parsing text, but since we chose to focus on the bigger picture and have used software components developed by others, we decided that it was not appropriate. Performance of a web scraper is also a question of scaling and requires resources we did not have access to.
3. Literature Review Results

3.1 Crawling
A common and central component found in scraping systems is crawling. Crawling can be defined as the process of automatically exploring and navigating websites by following hyperlinks [19]. The component or system responsible for crawling is commonly referred to as a crawler or web crawler. Other common names include web spider or bot [20]. Crawling is a broad term that includes many different classes of crawlers for different purposes and the literature lacks a standard classification scheme. Names referring to specific crawler classes are used interchangeably and sometimes contradictorily by different authors. There are however some distinct categories of crawlers.

A deep web crawler [21] starts from a seed URL, identifies hyperlinks in the web page retrieved from the URL, and visits the hyperlinks indiscriminately to retrieve a new web page recursively. The system does not consider relations between the web pages and usually utilizes a breadth-first crawling strategy to retrieve them.

In contrast to the deep web crawler, a focused crawler [21] only crawls web pages that satisfy a specific property. Such a property can for example be a topic, which is the case for topic crawlers or topical crawlers [8]. The term focused web crawler was introduced by Chakrabarti et al. in 1999 [22] for a crawler that seeks out pages relevant to a specific topic.

Another type of crawler is the incremental crawler [23], [24] which is concerned with maintaining the freshness of crawled content on web pages that might have new content inserted or changed. Incremental crawlers can also refer to the more common deep web crawler term [25] which is a source of confusion in the literature.

Due to the vast amounts of data a crawler needs to process, distributed crawlers are also an area of active research in order to improve the performance of crawling systems [19], [27], [12]. Distributed crawlers improve the performance of normal crawling systems [20] by using a distributed architecture. A distributed crawler can employ one or several different other classes of crawlers.

The commonality between the different crawling systems is that they retrieve web resources by sending HTTP requests to a specific URL. With the advent and proliferation of dynamic websites a new central problem in the research of crawling systems has appeared [9], which is described in the next section.

3.2 Dynamic Web Crawling
New trends and technologies in web development has led to a new set of problems for the general crawling system. Traditionally for static web pages a crawling system simply needs to extract hyperlinks from a web document. However, due to the increasing use of dynamically loaded content the traditional crawling method will not be able to load all
information required on the first request and there will be a discrepancy between the
document fetched by the crawler and the document seen by the user during normal browsing
[9], [5]. The source of this discrepancy is the use of technologies for dynamically loading
web content such as Asynchronous JavaScript and XML (AJAX). Solving this problem is the
subject of much of current research into crawling systems and has led to the development of
the dynamic web crawler, also known as event driven web crawler.

AJAX-based web applications and Single Page Application (SPA) [10] architectures are
increasingly gaining popularity for developing dynamic websites. These new architectures
provide more sophisticated client-side processing of the web document the user interacts with
compared to the traditional server-side preprocessed web document [19]. In the case of
traditional static websites, the client simply renders the document fetched from the server
which is also what is used by the crawling system to extract hyperlinks. In modern dynamic
web pages using AJAX methods, the page also includes code that can be executed by the
client to send asynchronous requests to the server or change the state of the page viewed by
the client. This is usually handled by a JavaScript engine that is able to execute JavaScript
[9]. The information fetched by asynchronous calls or the JavaScript code executed by the
browser is used to manipulate the state of the website on the fly. The state of the page is
represented by a Document Object Model (DOM) – an ordered tree [26] – which is encoded
with HTML. By this mechanism dynamic web pages inject more information into the page
than is retrieved in the initial request which leads to the crawling system missing potentially
vital information [5] or hyperlinks.

The problem solved by the dynamic web crawler is thus to reach all potential application
states of the web application in order to carry out traditional crawling tasks. The development
of crawlers for dynamic websites is an open area of research and several approaches have
been described, both automatic [19], [9], [5] by working with state-flow graphs and semi-
automatic by working with user defined extraction rules [8], [10], [13].

3.3 Scraping

In a full web scraping system, the resources fetched by the crawler component are then
passed to a scraping component that extracts meaningful content from the web page
associated with the URL [14]. Since web pages already follow the DOM – which can be
understood as an ordered tree [26] – exploiting this structure is a common approach for
scraping data. A common tool to use when scraping web pages is XPath [8], [11], which is a
descriptive querying language for querying XML documents that also works on HTML.
XPath allows the user to easily select nodes in an ordered document in a more readable
manner than by using regular expressions, however using XPath often requires knowledge
about the structure of the document. Methods based on text density analysis have been
suggested [28] in order to automatically locate relevant data.
3.4 Prevention Strategies
Scraping systems are sometimes actively prohibited from visiting certain websites, either by having it be against the terms of service or by setting up security measures – usually both. This poses a challenge for scraping systems [20] but also for the implementation of the security measures. Haque and Singh [2] have compiled a list of common prevention strategies for combating scraping systems.

3.5 Uses
Web scraping has been employed for several different purposes. Perhaps the most widely known applications of scrapers are the ones employed by search engines [24] such as Google [28]. Scraping systems employed by search engines fill the task of indexing and categorizing web pages. Scrapers and crawlers are also used for archival purposes such as the crawlers employed by the Internet Archive [5]. Another use is for sentiment analysis by businesses interested in brand opinions [14]. Extracting data and information from the web is of use in research and learning activities [14] as well.

Web scraping systems have also been shown to be able to extract information from social networks [29] and forums [21]. When extracting data on social networks and forums you often end up with problems related to handling of personal information which will be explored in the next section.

3.6 Legal Aspects
Each scraping project should research the relevant legal texts themselves, due to these being different across the world [7], and whether or not a scraping project is legal is highly contextual and depends on the project.

Landers et al. [7] have researched web scraping in the context of U.S. law. They found that there have been two major legal precedents set in recent years: eBay v. Bidder’s Edge [30], and Ticketmaster v. Tickets.com [31], both in the year 2000. In the former case, Bidder’s Edge had scraped eBay and several other online auction sites and aggregated the information on their own website. The court found that in this case, Bidder’s Edge was intentionally slowing down the access of customers to eBay, and thus ruled in the favor of eBay. In the latter case, Tickets.com had scraped and compiled a list of ticket prices from Ticketmaster. The court ruled that since ticket prices were factual, there could be no copyright issue and Tickets.com won.

Legislature can quickly become obsolete, and risks utilizing terms and language which might become outdated in the future. As it stands, case law concerning web scraping can (even to lawyers) be difficult to interpret [7], and it is thus recommended to scrape only public, unencrypted data due to the inconsistencies of the case law. It is also stated that if headers which are designed to prohibit or discourage web scraping are present on the website, then doing so could be considered illegal.
In general, most websites’ contents (within the U.S.) will fall under the Copyright Act of 1970, which prohibits original works from being copied [7]. However, exceptions are made for – amongst others – researchers under fair use.

We have also examined closely the General Data Protection Regulation (GDPR), and the results of this are presented in 4.6.

### 3.7 Robots.txt

Another aspect to consider is the amount of traffic scraping generates for any given website. In order to reduce the negative aspects of what could be a massive amount of traffic, many websites have elected to include a Robots Exclusion Protocol – robots.txt, aimed at remedying this problem.

Sun et al. [6] have researched how well crawlers follow this protocol and have in this endeavor introduced eleven honeypots (bait for scrapers) where each honeypot has a different robots.txt rule. The authors would then measure crawler activity and whether or not they followed the specified rules in each honeypot. Results showed that a non-negligent number of crawlers did in fact ignore the different rules set forth by robots.txt, such as forbidding crawling of certain directories, or asking for a certain delay when accessing a page.

In the case of Facebook, any scraping of their website has been explicitly forbidden unless sanctioned [32] and as such if one violates these terms, Facebook might ban the user doing so. It is however important to distinguish between legality and ethicality in this case; while Facebook’s terms state that violation of robots.txt rules can and will end in termination of services, this does not necessarily make such a violation illegal [33].
4 Results
During our research process we have studied the structure of the social media platforms Reddit [15], Twitter [17] and Facebook [16]. A scraping system has been developed specifically for Reddit by using the frameworks Scrapy and Selenium in a Python environment.

4.1 Anti-Scraping Techniques
There are various defense mechanisms websites can implement in order to discourage crawling and scraping of them [2]. Among the most common strategies for preventing scraping are i) regularly randomizing HTML IDs and CSS class names, ii) frequency analysis i.e. analyzing how many times per day/week/month a user visits the website, and iii) interval analysis i.e. analyzing how much time has passed between each visit.

Conceptually, a common approach is to utilize three lists: a blacklist, a graylist and a whitelist. All IPs are on the whitelist by default, and so the whitelist is abstract. However, if any IP is found suspicious, it is put on the graylist and might have to solve CAPTCHAs in order to continue browsing the site and eventually be put back on the whitelist. In contrast, if an IP fails a CAPTCHA (a reasonable number of times) it is put on the blacklist. Here one might remain forever, or for a period of time depending on the website.

In order to avoid being put on the blacklist it is important to mimic human activity. At the same time however, the scraper must be faster than a human or it would defeat the purpose of scraping. This is a difficult balance to strike.

A possible solution to this is to allow any user of the scraper to define their own randomized interval between requests in order to hopefully bypass any eventual interval analysis. Another option is to limit the number of pages the scraper will be allowed to visit in a day, in the effort to avoid any frequency analysis. The system will also have to be monitored in order to discover if the IP has been put on a blacklist. This can be observed by having HTTP requests being denied by the web server or the scraper failing to scrape URLs from a specific domain. In this case the scraper will have to switch to another IP address and likely change settings.

4.2 Structure of Conversations on Social Media
The first step when scraping a website is understanding the structure of the data being scraped. Public discussion in social media is generally handled by conversation threads. The basic unit of a thread is a message – or post – made by a user to add something to the conversation. This can be text, an image, or a URL to a web resource. These messages are generally what a social media scraper is concerned with scraping.

The terminology around conversation threads is not standardized and differs on different social media communities and even between communities sharing the same social media platform. A message is usually referred to as a reply on Facebook and post on Reddit while a thread is referred to as a status on Reddit and replies on Facebook. Nonetheless, the threaded
structure is present on the social media websites we have examined as well as the basic unit of a post. The threads themselves are usually grouped together. The grouping can be divided among specific subcommunities or topics. In the case of Facebook this can be seen as threads belonging to specific groups or user walls. Reddit groups threads in subreddits which are similar to subforums in a web forum.

What makes up a post differs between social media platforms, but most platforms share some commonalities. The two main parts of a post is the message itself, which can be text, an image, or URL etc. and a user ID of the user that made the post. Usually a timestamp is also present to indicate when the post was made. In addition, most social media platforms include a system of reactions to a post. These reactions can be a score as is the case of posts on Reddit or likes as in posts on Facebook and Twitter. Before scraping, the structure of a post on the specified domain needs to be considered and a schema of the post data needs to be generated in order to find the correct parts of a post in the scraping step and to save the posts in such a way as to not lose any information. In the case of Reddit this includes a username, a score, a timestamp, and text which is defined in a configuration file – see appendix 2, line 28.

Apart from the structure of the post itself it is also important to consider the structure of the conversation data. We believe it makes sense to borrow the terminology from the tree data structure in order to find a common terminology while developing scraping systems for different social media platforms. The conversation can be understood as a tree with the first post as the root of the tree as demonstrated in figure 1. Child nodes are posts replying to the parent post. This structure needs to be maintained while saving the scraped data in order to not lose important information about the conversation. Another positive aspect apart from a standardized vocabulary of thinking about social media conversations as trees is that the tree structure is also present in the HTML markup of the website and is used in discussions about parsing and selecting elements in HTML documents. This mirroring of conversation structure and HTML is present on all websites analyzed. Using the tree structure, we can also define the current state of comments on the web page as the conversation tree which will be important when scraping dynamically loaded comments.
4.3 Node Selection

Selecting nodes in conversations and HTML markup is essential for scraping the correct data and loading dynamic content. This requires the user of the scraping system to study the specific website in question. The simplest method is selecting by CSS class. Nodes in an HTML document often include a CSS class in order to use specific styles for the node. Both Scrapy and Selenium include functionality to select nodes by CSS class. This method of selecting nodes only works if the developer of the website has included CSS classes for nodes that do not change. On most dynamic websites the class names are dynamically generated and often unique even if the element shares styling with another element – an example of this is shown in figure 2. This makes it an unsuitable method of node selection for a scraper targeting dynamic content.

Unlike CSS class names that are inherently unstable from the perspective of the developer of a scraping system, the general tree structure of the HTML markup on the website rarely changes. A scraping system can exploit this fact by using the tree structure to select nodes. One tool that can be used for this purpose is XPath. Selenium and Scrapy both support selecting elements by XPath and using XPath allows the developer to traverse the conversation tree through common traversal methods, which in turn allows the system to easily save the tree structure of the data. These properties make XPath node selection superior to CSS class node selection for scrapers targeting dynamic content and should hence be used for the domain. See appendix 2, line 14, for examples of XPath.

Figure 1. The same conversation on Reddit and Facebook (root not visible, usernames whited out) and its tree representation.

Figure 2. Dynamically generated CSS class names for a paragraph of a post on Reddit.
4.4 Scraping Dynamic Content

The most problematic facet of scraping web pages is the overwhelming amount of dynamic content – scripts which load content based on user interaction with the page. When requesting a page, the HTML you receive and intend to scrape will almost never be the whole structure of the page.

Currently, one of the most prominent [5] (and best) ways to solve this issue is to simply imitate a human interacting with the elements of interest. One framework which enables this is Selenium – originally built for automated testing of web pages, it can be used for the purposes of scrapers as well.

Selenium requires a web driver which is a simple version of a web browser. There are several web drivers available for use (such as ChromeDriver or geckodriver). The choice of driver is however not trivial and needs to be considered during development. The ChromeDriver does not allow elements to be clicked if they are obscured by other elements, such as a popup. This can be beneficial for testing purposes in order to find obscured elements, but for scraping it is more appropriate to choose a browser capable of loading comments a human would not have been able to load. Our scraper uses geckodriver as it allows clicking partially obscured elements and being run in headless mode. Headless mode runs the browser without GUI and increases performance and decreases memory usage.

On dynamic websites the initial state of the page includes some elements that are hidden and not present in the HTML. By interacting with the page these elements can be exposed and the page changes its state until all elements are loaded [5]. The main page of Reddit contains a list of threads. This list is initially short but as the user scrolls down it is dynamically filled with more threads. This is called infinite scroll and is also present on Twitter and Facebook. In order to collect the URL of all the threads, the page has to be interacted with. This necessitates the use of Selenium in order to scroll down periodically when the scraper is ready to process more threads. Appendix 1, line 52 shows an example of a method to accomplish this by scrolling to the last element of the page.

Another issue is that of splash screens. Especially if scraping is done within the EU, a splash screen where the user has to accept the use of cookies is shown as the user first enters the website. This splash screen can sometimes block further functionality of the webdriver to interact with other elements on the site, however it does not block the scraping of HTML, as the HTML is already served. If the page has to be processed by Selenium the system should therefore anticipate the splash screens and have functionality that allows it to automatically accept cookies or interact with whatever splash screen is used on the website. This requires some domain specific knowledge about the website on the part of the user or developer of the web scraper.

In the case of Reddit’s conversation threads many comments are hidden behind dynamically loaded content. If we consider the current conversation tree structure as the state of the web
page, the goal of the scraper is to load all comments so that the conversation tree is in its final state where all comments are present as nodes in the tree. In order to load these, the system has to click on various elements one by one.

Clicking the “more replies”-element loads more comments and changes the state of the page by adding the comments to the HTML markup. An example of this can be seen in figures 3 and 4. By using Selenium or a similar web automation tool the system identifies and clicks each dynamic element in order to change the state of the page to one where all comments are shown. An example of this can be seen in appendix 1, line 141. This activity is demonstrated in figure 5.

On top of these “more replies”-elements, there were also downvoted comments which – along with all replies to that comment – automatically get hidden. These also have to be clicked with Selenium. If a comment thread reached a certain depth i.e. D replies to C replies to B replies to A, the thread continues on another page and thus would not be loadable at all without visiting the second page. This can be seen in figures 6 and 7. In this case the URL of the child comment has to be visited by the crawler component and the comments scraped will have to be saved in the correct place in the conversation tree.
4.5 Avoiding Dynamic Content

Dynamic content takes a long time to scrape compared to scraping static content, often in orders of magnitude according to our tests. This is because of the preprocessing necessary to take the page to its final state where all elements are exposed in the HTML markup. In some cases there is no way around this but knowledge about the specific website that is to be scraped can inform a user or developer how to avoid it. One avoidance strategy is the use of low bandwidth versions of a website. In the case of Facebook there is a low bandwidth version accessible at mbasic.facebook.com that is devoid of dynamically loaded content. This makes the scraping of Facebook comments much faster and should inform the development of a Facebook-specific crawler. Reddit does not offer a low bandwidth version but it does offer access to an old design of the website: old.reddit.com. This version of Reddit still includes dynamic content in conversation threads but the list of threads does not include infinite scroll. This makes the discovery of threads considerably faster.

Ideally, one does not want to scrape the same page more than once in order to not waste time and processing power, or at the very least it is important to minimize any duplicate work. In the case of Reddit – and indeed most social media platforms – it is almost always going to be impossible to avoid doing double work. In order to make absolutely sure that there are any new comments, one would have to at the very minimum use Selenium to show all comments. Once again, it is difficult to strike a balance here. On the one hand, if one visits each thread too rarely some comments might have been posted since the last visit and then deleted before the current visit. On the other hand, if one scrapes a thread too often, this might take up unnecessary processing power and time. A scraper should allow for easy scheduling so that the user can set the frequency of visits to what best suits the target website.

4.6 Legal and Ethical Aspects

The General Data Protection Regulation chapter II, article 6, section 1 [18] states that the processing of personal data belonging to any person residing in the EU is prohibited by law unless:

i. consent by the owner of the personal data has been given

ii. the processing is necessary by a contract entered into by the data owner

iii. it is necessary to process personal data of a subject according to a legal obligation
iv. processing is done in order to protect “the vital interests” of the subject
v. it is necessary to perform processing of the personal data in order to fulfill a “task carried out in the public interest”
vi. the processing of personal data is performed in necessity for the purposes of legitimate interests of the controller (defined in Chapter 1, article 4, section 7 as the natural or legal person carrying out the data processing) or of a third party, in cases where this processing does not violate any interests, rights or freedoms by the data owner

Chapter I, article 4, section 2 of [18] defines the word processing as “any operation … which is performed on personal data, … such as collection, recording, structuring…”. We have performed processing both in the form of collection as well as structuring. There is also no reliable way of telling whether a user is residing in the EU or not, and as such we have had to comply with regulations set forth by the General Data Protection Regulation. No long-term storing of the data of any kind has been done, and any data collected in any process initiated by us was deleted following termination of these processes.

However, not all subjects do reside in the EU, and as such in order to not break any laws, it is important to consider all pertaining legal texts – or at least as many as possible. We have decided to focus only on the GDPR and U.S. law, but there are likely more legal contexts to consider.

Landers et al. [7] state that while most work on websites based in the U.S. is considered to be under the Copyright Act of 1970, there are exceptions for critics, commentators, reporters, authors, artists, teachers, and researchers to reuse or repurpose this work. This is called fair use. While it is unclear if comments made by the users of websites would be considered copyrighted, as we are researchers we are in this regard allowed to reuse or repurpose the work in question either way.

There is a severe lack of legal precedence for scraping but a user of a scraper should only scrape publicly available content to be safe [14]. Even if scraping the targeted content is legal it is considered polite to limit potential overburdening of the resources of the target system.

Combining data from multiple different sites also opens up the potential to identify a user [14], this should be kept in mind so that the scraped data is not abused. In order to ensure the legality and ethicality of our system we have chosen not to save any comments and limit the system to identifying them as a proof-of-concept.

4.7 Putting It All Together
In the Scrapy framework a spider class is used to invoke crawling and scraping methods as well as pipelines for putting scraped data into a database. The spider is started by supplying a seed URL. For the Reddit scraper this is the URL of the subreddit that is to be scraped. The system uses a Selenium instance called URLDriver run as a separate process to prepare the
page for harvesting of threads’ URLs. The Selenium instance invokes methods set by the user in a config file that contains domain-specific instructions for changing the state. On Reddit this is scrolling to an element at the bottom of the page in order to make the infinite scroll functionality load more threads. The URLs are collected by selecting the correct element with XPath and the specific XPath query for selecting the element will also have to be set by a user in the config file. These URLs are then sent to the spider for crawling and while the spider is crawling, the Selenium instance can continue scrolling down the page in order to load more threads and put them in the spider’s URL queue.

The spider crawls to the URLs harvested by the Selenium instance by fetching the HTML from the web server. The spider checks the HTML for any dynamic elements defined by the user in a configuration file. If a dynamic element is found the URL is sent to a Selenium instance called ThreadDriver for processing. The ThreadDriver interacts with the user defined dynamic elements until no dynamic elements are found (see chapter 4.4). Once the final state of the page has been reached, or if it contained no dynamic elements, nodes in the conversation tree are selected by their XPath according to the schema set by the user. This data would then ideally be saved to a database. The flow of data within the system is demonstrated in figure 8.

This solution allows the system to be extended to use multiple distributed ThreadDriver Selenium instances. Since processing dynamic pages is the most performance intensive procedure by far, this allows the system to scale. As the spider discovers dynamic conversations, it can put the URL in a queue to be picked up when an active Selenium instance is available and carry on crawling. These Selenium instances are run as separate processes on the host machine and could be distributed across several machines if necessary.

The configuration of the system is critical in this approach. The user will have to study the domain in order to set it up appropriately. The configuration needs to contain XPath queries to select dynamic elements and instructions for interacting with them. On Reddit this can be the “load more comments”-buttons’ XPath, and a click instruction. A schema for which post information is desired with accompanying XPath queries are also required. If the user
anticipates that the target website uses anti-scraping techniques, security settings also need to be included in configuration. Lastly the user should also have the option to respect or ignore the robots.txt file. The configuration file (see appendix 2 for an example) will have to be maintained and updated by the user if changes are made to the markup of the website. Depending on how often the target website changes its design this can be labor intensive but considering the rapidly changing landscape of web development and the large amount of different technologies used there is currently no way around this to our knowledge.

4.8 Prototype Evaluation

We have performed an experiment in order to determine whether or not it is possible through mimicking human interaction to accurately acquire and display all desired elements (in this case comments). To this end, we have scraped 6 dynamic pages using the prototype. In each case, the number of comments as shown in the thread was noted. The scraping process was then initiated and the number of comments found were printed to a text file. The two numbers were then compared to see if any discrepancies occurred.

<table>
<thead>
<tr>
<th>#</th>
<th>Comments Found by Scraper</th>
<th>Comments According to Reddit</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>195</td>
<td>205</td>
<td>95.1%</td>
</tr>
<tr>
<td>2</td>
<td>299</td>
<td>300</td>
<td>99.7%</td>
</tr>
<tr>
<td>3</td>
<td>284</td>
<td>290</td>
<td>97.9%</td>
</tr>
<tr>
<td>4</td>
<td>369</td>
<td>388</td>
<td>95.1%</td>
</tr>
<tr>
<td>5</td>
<td>281</td>
<td>290</td>
<td>96.9%</td>
</tr>
<tr>
<td>6</td>
<td>395</td>
<td>409</td>
<td>96.6%</td>
</tr>
</tbody>
</table>

Table 3. The number of comments found by the scraper, the number of comments as reported by Reddit, and the accuracy for our scraper.

These discrepancies are discussed in section 5.5.
5 Discussion

5.1 Dynamic Content
When building our prototype scraper, we found that scraping dynamic content was indeed difficult. Among many other issues regarding dynamic content, was the pop-up asking for consent to the website storing cookies as this prevented further interactions due to limitations with the framework of choice, Selenium. This was used by all websites analyzed for the prototype. Another issue specifically in the domain of social media was the need to load more comments. We opted to use a headless browser to simulate user interactions as has been previously demonstrated to work by other research groups [5], [10]. Our prototype confirms that this method can adequately prepare a dynamic web page for scraping by dismissing pop-ups, splash screens and loading more comments in threads. Using this approach for the prototype did however increase the time needed to scrape a page considerably during the testing of the prototype. Relating back to our research question of how to solve the issue of scraping dynamic content this could be a potential solution but due to the time needed by the webdriver to prepare a web page its uses are potentially limited if a system needs to scrape a large amount of comments in a limited time frame. For this reason, the approach using a webdriver may be best suited for small-scale archiving purposes where a conversation only needs to be archived a few times and the number of conversations are limited, such as archiving the comments only regarding a specific topic or the conversations in a specific group.

5.2 Scraper Architecture
To enable scraping of dynamic content for large-scale applications more research is needed. This could be achieved either by finding more efficient ways of preparing a dynamic web page or by scaling up the system.

Originally, our design approach was that of a multi-threaded one. The idea was that one spider, running on one process would crawl a certain subreddit, and send each dynamic comment thread it found to a number of Selenium instances, each running in a different process in parallel. In turn, the spider would then scrape each processed thread for comments. However, due to the limitations of the technologies used for the prototype we were not able to examine whether a multithreaded or distributed approach would alleviate some of the time issues of the headless browser approach. By developing a custom spider or extending the Scrapy framework this could be explored in the future.

5.3 Avoiding Duplicate Work
Our research has shown that alternatives exist to scraping dynamic content – such as low bandwidth versions of websites. Other methods of limiting scraping of dynamic content should be investigated in the future. Limiting the amount of preprocessing needed to scrape dynamic content could prove to be important if scraping is to be done in a timely manner. Potential solutions that could be examined to this end is to use a focused element in a
scraping system where only conversations or subtrees of a conversation tree that meet a specific criterion are loaded.

As most conversations get comments added to them regularly, scraping the same conversation several times is often needed in order to discover new comments. Investigating the best time and interval to scrape a conversation in order to limit the amount of times scraping is carried out could also prove fruitful in order to make a scraper more efficient. This could be done by analyzing the rate of change of a conversation to figure out how many times and how often a conversation should be scraped. The interval between visits could potentially be based on a model that predicts the “success” of a given thread – the idea is that the more successful a thread is, the more comments it will receive. Conceptually, the success of a thread could be defined as the current number of comments or score it has in relation to how long the thread has existed; the higher score and/or comments per unit of time, the more often it potentially should be scraped.

5.4 Legal and Ethical Considerations

When it comes to the legal aspects of scraping, there are few precedents, and the research is sparse. We have not been able to generalize in this case, because it is up to each individual scraping project to determine whether or not what is being done is legal.

The subject of legality in a scraping or crawling context is something which merits further research. The fact that some websites implement anti-scraping techniques to combat scraping does not necessarily impact the legal standing of scraping [7], however it should serve as an ethical guide. Implementation of anti-scraping techniques and defining rules in robots.txt should be taken as a sign that a website does not wish for scraping to occur on their material. It could be argued that in this case scraping would be unethical. To prevent scraping from occurring in these cases research into novel anti-scraping techniques is an important area to examine. As new anti-scraping techniques are developed scrapers will find ways to circumvent them and there is a potential for an “arms race” between scrapers and anti-scraping techniques.

What we have been able to determine is that in our case – based on the research – it is indeed legal to perform scraping. We have also made sure to scrape as ethically as possible by not saving any comments and limiting the amount of scraping performed to reduce traffic.

It can be argued that due to the presence of a Robots.txt file specifically forbidding scraping it is unethical to scrape in our case. However, because we do so as researchers and produce positive outcomes, namely furthering public knowledge, it can also be argued that it is ethical from a consequentialist ethical framework.
5.5 Prototype Evaluation

5.5.1 Number of Comments
As reported in section 4.8 the number of comments found by the scraper was not the same as the number reported by Reddit. Generally, the scraper kept an accuracy of above 95%. The discrepancy between the numbers is however not necessarily due to inadequacies of the scraper but can be due to removed comments or the presence of comments made by a shadowbanned user. When a comment is removed by a moderator on Reddit it is replaced by a notice that a comment has been removed and in the case of a shadowbanned user it is not displayed at all (yet is still counted as a comment). It is therefore difficult to determine exactly how accurate the scraper is without knowing how many comments have been removed or hidden by moderators or admins. Due to this the true accuracy of the scraper is most likely higher than reported, likely even completely accurate.

5.5.2 Other Evaluation Metrics
The process of scraping a static web page (a page which contains no dynamic elements of interest) consists essentially of nothing more than retrieving the page via an HTTP request and then parsing the contents of the HTML. Apart from the initial HTTP request, the entire process can be performed locally and thus will depend solely on the hardware capacity of the local environment.

Unlike parsing static pages, dynamic pages (in this case) do rely on further requesting information from a server and thus will depend on many things other than local hardware capacity such as local network capacity, the network capacity of the server, and the distance between the local environment and the server.

Because of these inherent differences, an evaluation of differences in speed when scraping static and dynamic pages is of limited value. It is impossible to isolate the variables in such an experiment and as such we found no value in performing one. It is evident that parsing dynamic pages will take longer since it depends on more factors other than the local environment, but it cannot with certainty be determined how much longer this will take.
6 Conclusion

With this study, we have compiled a list of the most common issues which can arise when scraping web pages: various anti-scraping techniques, the prevalence of dynamically loaded content, as well as having to consider the legal and ethical aspects of scraping.

Our research was specifically aimed at the domain of social media. The web pages we considered were Facebook, Twitter and Reddit, while the prototype developed was made specifically for Reddit. Facebook and Twitter were mentioned and targeted by other research groups [29], [34] found in the literature study but we did not find any research aimed specifically at Reddit.

The sites chosen to analyze are both popular and have some unique attributes which is why they were picked to represent the domain of social media. The fact that only three websites were studied limits the generalizability of our research as other websites may work differently. We nonetheless think that the structure of conversations in these cases is representative of how conversations are had in comments sections of other areas in the social media landscape. New trends in social media and new web technologies do however have a potential to considerably change this, and thus conversation structures may change with time.

Ultimately, we think that web scraping is an important area of research with much work to be done still, especially when it comes to ways to circumvent dynamic content, and the subject of legality. New web technologies and anti-scraping techniques will ensure that research into efficient scrapers will always be needed, but the problems and solutions in this paper can serve as a guide for future research as well as for groups interested in implementing web scrapers for social media or other websites using dynamically loaded content.
References


Appendix

1 Source Code (thread_spider.py)

```python
import scrapy
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.keys import Keys
import logging
import json

class ThreadSpider(scrapy.Spider):
    name = "thread"
    visitedUrls = []
    config = {}  

    def __init__(self, name=None, **kwargs):
        with open('config.json') as con:
            self.config = json.load(con)
            self.start_url = self.config.get('start_url')
            self.start_url = self.get_next_url()

            # Starts the selenium uridriver and clicks cookies button
            def startUrIdriver(self):
                options_url = Options()
                # Enable headLess by removing comment below
                # options_url.add_argument("--headless")
                browser_profile = webdriver.FirefoxProfile()
                browser_profile.set_preference("dom.webnotifications.enabled", False)
                self.uridriver = webdriver.Firefox()
                self.uridriver.execute_script("window.open('https://www.reddit.com/r/letstalkmusic')")
                for xpath in self.config['xpath']['xpath']["uridriverLoad"]:  
                    page_loaded = WebDriverWait(self.uridriver, 10).until(EC.presence_of_element_located((By.XPATH, xpath)))
                    if page_loaded:
                        # Find and click the button.
                        button = self.uridriver.find_element_by_xpath(xpath)
                        button.click()  

                # Finds uris on subreddit and returns a list of uris strings
                def get_next_url(self):
                    xpath = self.config['xpath']["urisPath"]
                    threadUrls = self.uridriver.find_elements_by_xpath(xpath)
                    urls = []

                    for url in threadUrls:
                        if url.get_attribute("href") not in self.visitedUrls:
                            self.visitedUrl.append(url.get_attribute("href"))
                            urls.append(url.get_attribute("href"))
                    return urls

                def scrollUrIdriver(self):
                    xpath = self.config['xpath']["urisPath"]
```

def parse(self, response):
    if self.checkDynamic(response):  # Dynamic.
        hrefList = self.parseDynamic(response)
    else:  # Static.
        hrefList = self.parseStatic(response)
        # hrefList for continue this thread
        if len(hrefList) != 0:
            for href in hrefList:
                yield response.follow(href, callback=self.parse)
        urls = self.getNextUrl()
        for url in urls:
            yield response.follow(url, callback=self.parse)
    self.scrollUntilDriver()

    # Parses the HTML, treating it as if it contains dynamic content.

def parseDynamic(self, response):
    # This disables the browser asking for notifications.
    options = Options()
    # Enable headless by removing comment below
    # options.add_argument('--headless')
    _browser_profile = webdriver.FirefoxProfile()
    _browser_profile.set_preference('dom.webnotifications.enabled', False)
    self.driver = webdriver.Firefox()
    # firefox_profile_browser_profile, executable_path='geckodriver.exe', firefox_options=options)
    self.driver.get(response.url)
    # page_loaded will be True if it finds the element within 10 seconds, False otherwise.
    xpath = self.config['xpath']['threadDriverLoad']
    page_loaded = WebDriverWait(self.driver, 10).until(
        EC.presence_of_all_elements_located(
            (By.XPATH, xpath[0]))
        )
        hrefList = []
    if page_loaded:
        # Find and click the cookies button.
        for path in xpath:
            button = self.driver.find_element_by_xpath(path)
            button.click()
        self.clickMoreComments()
        hrefList = self.continueDynamic(response)
    self.driver.close()
    return hrefList

    # Parses the HTML, treating it as if it contains dynamic content.

def parseStatic(self, response):
    return self.continueStatic(response)
# This loop will continue until it does not find any more Continue This Thread elements.
def continueDynamic(self, response):
    continuexpath = self.config['xpath']['threadDriverContinueThread']
    continue_elements = WebDriverWait(self.driver, 10).until(
        EC.presence_of_all_elements_located(
            (By.XPATH, continuexpath)
        )
    )
    print(continue_elements)
    hreflist = []
    continuexpath = '../..
    if continue_elements:
        cont = self.driver.find_elements_by_xpath(continuexpath)
        for c in cont:
            href = c.get_attribute('href')
            hreflist.append(href)
    return hreflist

# Finds "continue this thread" elements statically.
def continueStatic(self, response):
    continuexpath = self.config['xpath']['threadDriverContinueThread']
    continuexpath = '../@href'
    href = response.xpath(continuexpath).getall()
    hlist = []
    for h in href:
        hlist.append(h)
    return hlist

# Returns true if page is dynamic, false otherwise
def checkDynamic(self, response):
    xpath = self.config['xpath']['threadDriverLoadComments'][0]
    return len(response.xpath(xpath).getall()) > 0

# This loop will continue until it does not find any more More replies to click.
def clickMoreComments(self):
    loop = True
    xpath = self.config['xpath']['threadDriverLoadComments'][0]
    while (loop):
        try:
            more_elements = WebDriverWait(self.driver, 10).until(
                EC.presence_of_all_elements_located((By.XPATH, xpath))
            )
        except TimeoutException as te:
            print(str(te))
            loop = False
            break
        if more_elements:
            elements = self.driver.find_elements_by_xpath(xpath)
            for e in elements:
                try:
                    e.click()
```python
except Exception as ex:
    print(str(ex))
self.clickDownvoted()
else:
    loop = False

# Attempts to click downvoted comments.
def clickDownvoted(self):
    xpath = self.config['xpath']['threadDriverLoadComments'][1]
    downvoted = self.driver.find_elements_by_xpath(xpath)
    for d in downvoted:
        try:
            d.click()
        except Exception as ex:
            print(str(ex))
```

All source code is available at [https://github.com/cloudbells/scrapy-reddit](https://github.com/cloudbells/scrapy-reddit).
2 Configuration File

```json
{
    "saveDownvoted": true,
    "visitInterval": 300,
    "requestInterval": 5,
    "subreddits": [
        "askreddit",
        "worldnews"
    ],
    "keywords": [
        "work",
        "china",
        "bad"
    ],
    "xpath": {
        "urlDriverLoad": {
            "//button[@type='submit'][contains(text(), 'I Agree')]"
        },
        "urlXPath": "//a[contains(@data-click-id, 'comments')]",
        "threadDriverLoad": {
            "//button[@type='submit'][contains(text(), 'I Agree')]",
            "//button[contains(text(), 'View all')]"
        },
        "threadDriverLoadComments": {
            "//*[contains(@id, 'moreComments')]",
            "//div/button[1[contains(@class, 'icon-expand')]]"
        },
        "threadDriverContinueThread": "//span[text()='Continue this thread']",
        "content": {
            "username": "//div[contains(@class, 'Comment')]//div[1]/div[1]/div/a/text()",
            "score": "//div[contains(@class, 'Comment')]//div[2]/div[1]/span[1]/text()",
            "timestamp": "//div[contains(@class, 'Comment')]//div[2]/div[1]/a/span/text()",
            "quote": "//div[contains(@class, 'Comment')]//div[2]/div[2]/div/p/text()"
        }
    }
}
```

An example for a configuration file targeted at Reddit. This configuration file is available at https://github.com/cloudbells/scrapy-reddit.
3 Instructions for Scrapy and Selenium

To install Scrapy and Selenium, the easiest way is to first download Anaconda. The download link for this can be found at https://www.anaconda.com/.

Start Anaconda and navigate to Environments. Here you can create new or edit existing Python environments. To install Scrapy and Selenium, search for these in the list and install them.

After this is done, it is recommended to install an editor which supports Python development and ideally different Python environments (such as Visual Studio Code).

To get started developing Scrapy, see https://docs.scrapy.org/en/latest/intro/tutorial.html.

Start by creating a new folder for Scrapy projects. In Anaconda, navigate to Environments again and select the environment with Scrapy and Selenium, then start it in the terminal. Navigate to the newly created directory in this terminal and type:

```
scrapy startproject tutorial
```

From here, a spider is needed in order to start scraping. It is recommended to view the above linked tutorial for Scrapy to get started.

This will be enough for static web pages, however not all pages are static as we have discussed previously. In order to scrape dynamic content, one will need a framework similar to Selenium.

To get started with Selenium, it is recommended to view this documentation: https://selenium-python.readthedocs.io/. In order to utilize Selenium, one will also need a webdriver. In our prototype, we have used geckodriver. It can be downloaded here: https://github.com/mozilla/geckodriver/releases.