Principles of Ethical and Effective Web Scraping*

Matthew Edwards
Department of Statistical Sciences
University of Toronto
mr.edwards@utoronto.ca

September 20, 2020

Abstract

In this article, I discuss the various ethical and practical considerations for the data collection technique called **web scraping**, which is simply automated extraction of information posted online. I define the technique, and discuss its costs and benefits, including some of the recent legal and consent issues surrounding personal information. I then propose a set of principles that are designed to handle both the ethical and practical considerations of implementing a scraping algorithm. I conclude a brief discussion on secure logins. The appendix contains two very detailed examples (with python code) that follow these principles.

1 Introduction: What is Web Scraping?

In the modern digital age, the internet has become so big that it is impossible to gather all of its content in one place. The four largest online storage companies alone (Google, Amazon, Microsoft, Facebook) account for an estimated 1200 petabytes of information, which is about equal to the storage capacity of 1.2 million higher-end laptops [BBC Article 2014 Cite later]. The commercialization of online data has become a massive industry, with some businesses dedicated only to the extraction of internet data. Despite the large increase in private-sector activity, the Canadian government has been slow to adopt these techniques because of their potential to be misused and for other valid reasons we outline later. This document proposes a set of principles for ethical web scraping, the primary method for quick extraction of online data. I outline why and how it can be a valuable tool for researchers and policymakers. I also make practical suggestions for efficient scraping and provide examples that implement these suggestions.

1.1 Internet Basics

To understand web scraping, you need to understand the basics of how the internet works. The World Wide Web is essentially just a giant network of machines (one of which is your desktop computer). On the network, these machines share information using standardized communication protocols that help them understand how to format and transmit information. However, the information exchanged is designed for computers to read, not humans. It is your browser (Firefox, Chrome, IE,) that translates the files into human-readable form. Most static web pages are written in a programming language called Hypertext Markup Language (HTML), which your browser knows how to interpret. This HTML code contains all the information displayed on the website. This data can be very useful to researchers.

Web Scraping: Automatically extracting information from a web page into a local file on your computer. Also called *screen scraping*, *web harvesting*, or *web data extraction*, this is different from simply copying information from a web page by hand because the volume of information that a computer program can quickly extract is much larger.

Web Crawling: The act of automatically browsing multiple websites and web pages. Also known as *web spidering*, this is often used to scrape the data from multiple websites into one large dataset. Many search engines do this in order to find pages that are most relevant to your search parameters.

Internet Bot: A piece of software (computer code) that performs tasks over the world wide web. Sometimes called a *web bot* or just a bot, programmers often use them to do simple, repetitive things at a much faster pace than is possible for humans. Many of the today's largest bots are web crawlers.

*Aknowledgement: The original idea for this article came as a result of a brainstorm session during my time at Employment and Social Development Canada. Though I was the web-scraping project lead then, and am the sole author of every word in this document, I owe a debt of gratitude to the Research Innovation Unit at ESDC. Thank you Laurent, Caroline, Richard, Charlaine, Gray, Angelica, Thomas, Jeff, Jason, Abdou, and anyone else I might have missed. This project would not have materialized without your extremely valuable support and feedback.

When you enter the address (URL) of a website into the bar at the top of your browser, your machine sends a request to the server hosting the site. If you have permission to access the site, the server sends back the files, and your browser shows them to you. Although almost every web page is different when displayed in your browser, the language that the browser interprets is the same, regardless of which website you are trying to access. This consistency in the way we code web pages makes them ideal for extraction by a computer algorithm (program).

Table Y shows the difference between what your browser interprets, and what you see. When trying to gather information from multiple websites, it is much faster to extract the information from the HTML code than to have someone manually copy information from the rendered website. However, many companies object to this form of gathering information from their sites. This has little to do with data protection; if the information is meant to be secret, it should not be put online. The main concerns involve the consequences of the speed at which the information is collected. The next section goes into more detail about exactly why irresponsible web scraping can have a negative impact.

2 Costs and Benefits

The benefits of web scraping to organizations are plentiful. It allows analysts to gather large data quickly to inform business or policy discussion. Moreover, because the type of information found online is so varied, online data has the potential to answer a wide range of questions. Also, many of today most innovative modelling approaches (e.g. deep learning, natural language processing) require very large data to train properly. Scraping is one of the only ways to collect big data quickly.

One of the most common types of data extracted from websites is text data. This is because the internet was founded to display research, most of which was, and still is text. But in the past few decades, the internet has started to diversify its content so that it is now possible to scrape not only text, but also quantitative data, images, audio and even video.

Web scraping can cause problems because, to get access to the code of an entire website (with many different links and all their associated pages), the scraper has to send a high volume of requests in a very short period of time. The site server (machine on the network) has to keep track of all these requests individually by storing them in memory, but its memory capacity is fixed. Since these servers have to handle requests from the scraper in addition to those of millions of potential site visitors, there is a risk of the server simply running out of memory (also called *overloading*). When this happens, the server will usually stop processing new requests, and may take longer to process old ones. This means that new visitors will be unable to view the website.

Intentionally overloading a server in this way is a Denial of Service (DoS) attack, and it can have devastating consequences. For some websites, like blogs or research forums, temporary disruption of service matters little. But for e-commerce sites, social media platforms and search engines, disruption can cost companies millions of dollars, and inconvenience hundreds of thousands of people. Since the companies with the largest online platforms have the most to lose, they often deploy anti-crawling systems designed specifically to identify and block malicious bots. Unfortunately, many of them prohibit web scraping of any kind without prior permission. This is why one of the most important principles is to identify yourself and your intentions when scraping. Many companies that catch scrapers operating without permission seek legal redress, particularly if the scraper uses the extracted data for commercial purposes.

The positive aspect here is that, as researchers, we always strive to use the highest quality data possible. In the case of online collection, the highest quality tends to come from the largest, most credible websites. Since these sites also have the newest, most effective network architectures and the best anti-crawling systems, successful collection of their data is very unlikely to lead to a DoS attack. For instance, the last successful DoS attack on Twitter occurred in 2016 and required tens of millions of IP addresses to penetrate the security system. This is well beyond the capacity we use here for extraction of government research datasets. In the next section we showcase a few examples of the types of projects that have benefited from the use of small, concentrated scraping bots.

3 Consent Considerations

3.1 The Robots Exclusion Protocol(*robots.txt*)

The Robots Exclusion Protocol (also called the Robots Exclusion Standard, or just `robots.txt`) is an aggregation of standard protocols that regulate how bots behave online. Each website adhering to the standard provides a file called robots.txt, which lists their suggestions for which pages bots should and should not crawl. Often the permissions are different depending on whether the bot is a search engine because denying access to search engines can cost the website potential users. It is important to note that unlike the terms and conditions of a site, the robots.txt standard is just a suggestion for bots to follow. Failure to abide by the suggestions in the file does not constitute sufficient grounds for legal action. However, many of the most sophisticated sites, containing the most comprehensive robots.txt files also deploy anti-crawling bots of their own, so following these suggestions increases the likelihood that a scraping-bot will be able to work properly.

Note that these are just samples of the robots.txt file from each site and not the complete documents. These files can be hundreds of lines long, needing to address different permissions for dozens of users and potentially hundreds of pages. Notice that google does make some information permissible to all users (*), like the about page and how its search works. However, it disallows scraping of its search page (obviously to protect against other

companies replicating its search functionality). LinkedIn takes a much more restrictive stance, allowing some large crawlers like Google and Cincobot while forbidding crawling of any kind to all other users who do not have prior permission.

Table X shows two examples of `robot.txt` files, which users access just like any other page. The first thing to notice is that the two files have identical formatting, with simple text and new lines for each permission case. The table only displays small parts of each of these files, but it is clear that both sites allow search engines, while disallowing other users. Google, for example, does allow users to scrape the text from its `about`, `static` and `howsearchworks` pages, but disallows extraction of its search algorithm and its groups page. The reasons for selective permissions are usually valid. In the case of Google, the prohibition of scraping for search comes from the obvious fear of intellectual property theft (someone who could extract the code behind the PageRank algorithm could make millions of dollars cost-free). Permissions are disallowed on the `groups` page because they link directly to online messaging forums, so to giving bots access would allow hackers to send spam content to users.

3.2 Terms and Conditions

The Terms and Conditions of a website act as a binding contract between the site and its users. This document, usually available at or near the bottom of the web page, sets the rules and the guidelines that anyone who visits a website must agree to follow. The main reason to use such a document is to protect the site owner from liability if any users misuse the site or its content. For example, many social media platforms explicitly prohibit the posting of defamatory messages, or the spamming of other users.

If a web page collects data from its users, it will usually have a privacy policy. This informs visitors what kinds of data the site gathers, and how the data will be used. This is essential in order for visitors to give informed consent. Many of today's largest scandals involving digital companies arise because the public discovers that a website is using visitor data in ways beyond what they promised in their Ts and Cs. However, data collected from public users are often very valuable to researchers. From an ethical perspective, it is important to make sure that the scraper's intended use does not violate the conditions under which the site originally collected the data.

The Terms and Conditions also formally declare who owns the site's content. This is particularly important for commercial websites, in industries like E-commerce or social media, where intellectual property theft is a serious risk. By declaring that they are the sole owners of the content on their sites, owners ensure that anyone who tries to profit from their work (without their permission) can face legal action. As public servants, we are not likely to encounter this situation, since our research is not for commercial gain and brings a social benefit. However, it is still helpful to understand why some companies may not react in a friendly manner to unknown software browsing their pages.

By declaring ownership of the content, the site implicitly states what visitors can and cannot pass off as original information. They may do this in a separate Copyright Policy, but regardless of how it is presented, scrapers must follow it. Since it is the common practice of researchers here at ESDC to site all relevant data sources, we are unlikely to encounter any issues along this avenue. I should note that original derivations of copyrighted data are not themselves subject to copyright. This means that, as long as you site the source data properly, any original analysis belongs to the researcher (scraper). I provide more detail on the nuances of intellectual property law in the next section.

4 Legal Considerations

4.1 Copyright Law

4.2 Contract Law

5 Handling of Personal Information

6 Ethical Principles and Best Practices

In this section I present a series of principles to help guide researchers through the design and implementation of their web crawler. These principles are ordered roughly by their order within the workflow. Some address the important ethical issues described above, while others are geared toward practicality and efficiency to save time throughout the data collection process. This is not an exhaustive list (I'm sure I've missed something), however following these design principles will guarantee that your crawler is both ethical and very effective.

6.1 Before Scraping

These first items are meant to serve as solid preparation, to be completed before you write a single line of code (for your crawling algorithm). As with most complicated procedures, you can save yourself a lot of time with diligent preparation that anticipates potential problems you may encounter.

{A} **Make sure there is no other way of getting the data you need.** Although web scraping is an easy way to extract large volumes of data quickly, many site owners will likely give you the data you need if you just ask. Particularly for fellow researchers, as long as you have no malicious intent, most are very open to sharing information. Even if the terms and conditions of the site forbid it, the owner may make an exception. Web scraping should be your last resort.

{B} **Check to see if the site has an API.** An *Application Program Interface* (API) is a set of protocols or methods designed by site owners to make it easier for developers to build software. These interfaces usually contain easy-to-use building blocks that allow for data extraction without the risk of the user causing harm to the site. A good example is Twitter's command-line application twurl, designed to allow users to request tweet data through its API. Many APIs are available in standard programming languages and most contain specifications for dealing with common data structures and programming paradigms. Using a good API makes getting the information you need safer and easier.

{C} **Take only the data you will need to answer your questions.** Many users expect that, after they collect terabytes of information, interesting results are sure to stand out. However, this is the exact opposite of how good research is done. You should carefully design your hypothesis so that you know which variables are of interest before you begin constructing the crawler. Then collect only what you need. You may not need to scrape the entire origin site if an aggregator is extracting and displaying the information you want.

{D} **Explore the data flows of the sites you plan to crawl.** When feasible, it is always a good idea to examine the structure and location of the data you plan to use and the web-pages around it. This will help you identify which pages you need not collect, and help you design a more robust crawling algorithm. Obviously, when projects involve crawling a large number of pages, it is impossible to optimize for every different style and form the bot will encounter. However, knowing the structure of at least some of the web-pages will help you design your algorithm proactively.

{E} **Check the Terms and Conditions for each site.** Most sites do allow at least some web crawling, since Google's PageRank algorithm relies on crawling to classify web pages that appear in its search engine. To ensure good behavior, virtually every company has Terms and Conditions (also called *Terms of Use* or *Conditions* of Use) that tell visitors what they can legally do with data on the site. Although these vary from company to company, they have a few things in common. Most will specify that any text, graphics, logos or designs are their property under Copyright law (a notable exception to this is review/comment materials) and that they retain sole ownership of any duplications. They will also prohibit any use of their site content for commercial purposes. Virtually everything we do adheres to these conditions, so the number of infringements likely to occur when scraping many sites is small. However, it is still good practice to obtain direct written permission when possible. When the conditions prohibit scraping of online data, **DO NOT SCRAPE**.

{F} **Check the `robots.txt` file for each site.** In the continuum between unrestricted crawling and complete prohibition of all crawlers, most websites lie somewhere in the middle. Many companies allow scraping of some pages, but not others. The `robots.txt` file will specify the permissions of each page. It may also specify a preferred crawl rate (say, one request per 5 seconds), preferred time of day, and other preferences. Permissions can apply to all bots, or to individual crawlers (like Google, Yandex, etc). It is common for sites to allow search engine bots, but disallow all other crawlers. Note that the robots.txt file technically contains suggestions that bots do not have to follow. However, most major internet companies and most private users adhere to this protocol, and you should too. Table X contains more information about the specifics of `robot.txt` files.

6.2 During Scraping

At this point, I assume you have permission to scrape. This *during* section provides suggestions on how to implement your scraping algorithm. The advanced suggestions are less about ethics and more about efficiency. There is more detail on them in the technical appendix.

{G} **Do not cause harm to the website(s).** Depending on the business model, temporary suspension of online services to a company's customers could cost millions of dollars. Even if the website's servers can handle the increased volume of requests, the additional strain could leave the site more vulnerable to third party attacks. Just as we take tremendous care requesting survey data, we must not burden the owners of online information.

{H} **Identify yourself to the site owner.** Every time your browser sends an HTTP request to a website, it identifies itself through a User-Agent string contained in the header. Designed to allow the website to track which kinds of devices are requesting content, it typically includes the browser name (e.g. Mozilla) along with the version and operating system. Sites can then permit access to certain content conditional on the user. You can (and should) include additional information beyond the standard, such as your organization and an email address through which site owners can contact you. If possible, create a simple, separate webpage stating who you are, what data you are collecting and why you need it. You can then provide a link to that page in the header. Be as transparent as possible with site administrators, even if they prohibit scraping. If your use of the data could generate value for the site, they may consider excepting you from the robots.txt declarations. The technical appendix shows exactly how to do this.

{I} **Use a low crawl rate, and scrape during off-peak hours.** This is falls under not causing harm, particularly among small sites with low-capacity servers. The crawl rate is simply the number of requests per unit time (e.g. per second). A friendly crawl rate is usually one request every five to ten seconds; some sites may even recommend a specific crawl rate in their robots file. If the owners have guidelines, you should adhere to them. Scraping during off-peak hours (evenings or early mornings) reduces the likelihood that your additional requests will be burdensome to the site's servers.

{J} **Transform collected data into a standardized format.** This follows naturally from understanding exactly what data you are looking for. If you initialize a database in advance, containing places for each piece of information you are collecting, all you need to do is identify the information and it can easily be stored. After collecting your data, this makes cleaning much easier. The exact system you use will depend on your choice of programming language and the kind of data you collect (text vs numeric vs images). For instance, a traditional SQL database may be perfectly suitable for text or numeric data, but not for images or videos. These are better stored as plain files, with their name or index listed in the database. The choice is context dependent, but consistency is key.

{K} **Cache the pages you have already crawled.** This simply ensures that you will not have to reload pages you have already visited. Constantly reloading can be computationally expensive.

{L} **(Optional/Advanced) Beware of honeypots.** A *honeypot* is a link to fake content that cannot be seen when viewed in a browser, but is present in the HTML of the page. Websites use these to identify scrapers. Once the site knows that your algorithm is not human, it may ban your IP address altogether. Other sites may allow you to scrape, but will give you false information. This is called *spoofing*. Other websites can direct the link back onto itself, so that once your algorithm follows it, your bot gets stuck in an infinite loop. To combat these traps, there are libraries that can check whether a link is visible on a page. If you use a **headless browser**, you can check as you crawl. You could also check the CSS properties (e.g., `display: none;`). If you do fall into a honeypot, avoid infinite loops by setting a limit on the number of redirects on your requests.

{M} **(Optional/Advanced) Insert some randomness into your algorithm.** Some of the most effective anti-crawling systems detect fixed repetition in http requests to sites. For instance, using one request every 5 seconds is an easy pattern for anti-crawl bots (yes, they exist) to detect. Even sites that allow scraping of some pages still deploy security measures to protect against hackers, which may cause problems for you, even if the site owners have given permission. To avoid being mislabelled as spam through pattern analysis (assuming you do have permission to scrape), you can vary the time between requests using pseudorandom number generation. Just make sure the interval is site friendly (e.g., uniform distribution between 5 and 10 seconds between requests). There is a concrete example of this in the technical appendix.

{N} **(Optional/Advanced) Rotate your user-agent string.** Site security measures also signal threats when large numbers of requests come from a single user. To avoid incorrectly falling into this category, you can periodically change your user-agent string so that no one user-agent queries the site too many times. This will make it appear as though your requests are coming from different browsers. When you do this, you should also rotate your IP address; otherwise, the website will just connect those different browser requests to one source machine, rendering the positive effect of user-agent rotation ineffective. To rotate your IP address, you will have to find proxy servers to send these requests on your behalf. Some websites are dedicated to providing these servers freely, but the most secure ones can be expensive (You get what you pay for). There is an example of this in the appendix.

{O} **(Optional/Advanced) Distribute large-scale scraping across multiple servers.** If you know you will be collecting a large amount of information, you should distribute your requests across multiple servers anyway. This is because large sites funnel requests through multiple servers of their own. To avoid straining any one node on the back-end network, you should maintain a low number of requests per user agent and per IP address. In addition, if you are gathering very large data (millions/billions of pages), it is likely that the information will be stored in multiple places, so separating requests across servers can also make pipelining (automating collection and storage of data into one process) easier.

{P} **(Optional/Advanced) Use a headless browser.** Although I will not show this explicitly, this is becoming a more popular trend among scrapers. A headless browser is a browser that has no graphical interface. It shows the underlying code requested from servers, but does not render that code into visuals. This is extremely useful for testing websites under development, because you can find bugs in your code without using computational power rendering pixels on the screen. For instance, it is much faster to simulate someone following a link, then to have someone hold the mouse and test manually.

Most website servers today do not simply return HTML code, but instead inject JavaScript programs into the HTML response that only execute to display the information you want after the browser

renders the return (see the intro for more on common web languages). This means scrapers will be unable to see the information they need without the browser rendering capabilities. Headless browsers are the perfect answer to this, since those JavaScript responses translate into HTML code (containing the information you want) without the visual rendering. Many standard browsers, like Firefox and Chrome, support headless versions. You can explore the internet through site code instead of through point-and-click. You should use either Firefox or Google's browsers if possible, although there are open-source alternatives.

6.3 After Scraping

These principles will help you ensure that the data you've collected is appropriate, both for analysis and for publication.

{Q} **Check the terms and conditions for each site (again).** Since the terms and conditions outline what you can and cannot do with data displayed online, it is always good to double-check. Some companies may list information pertaining to scraping in their privacy policy, so you should also check there if possible.

{R} **Verify all licenses before publishing anything with the data.** It is possible, particularly when scraping commercial sites, that some of the data you collect will fall under Copyright, Trademark, Patent or Industrial Design protections. You should always check the licensing of the information you plan to publish to avoid legal redress from these companies. There is a more detailed discussion of this in the Legal Concerns section.

{S} **Try to benefit the website(s) in some way.** You should always try to derive some benefit for the sites you scrape. This benefit could be direct (the results of your analysis may contain obvious value to the owner of the data) or indirect (e.g., you could give publication credit or drive traffic to the site). As is the case with other types of data, people will be more likely to consent to the collection of their data if they have an obvious incentive to do so. Here in government, we do all research with the client perspective in mind. Leveraging online information should be no different.

{T} **Cite all websites you scrape.** Even if you do not formally cite in publication every site you crawl (which can be infeasible if the number is quite large), you should at least maintain a list. This is important for the reproduction of your research by others, even if the results stay within government. In addition, if you wish to monitor the sites across time to update your dataset periodically, the only way to be certain the sampling base has not changed (or to learn that it has) is to maintain a list.

{U} **(Optional/Advanced) Use quality assurance tests to verify the integrity of your data.** Sometimes your web scraper collects data that is different or worse than you expected. Ideally, you should know if this is happening before your crawler stores large numbers of pages. Most crawling algorithms test for this using *regular expressions* (sequences of characters that define search patterns), sending alerts when the structure of incoming data is unexpected.

7 A Note on Logins and Security Credentials

Many websites require that you create an account in order to view their website. This is standard on social media platforms, so that sites can monitor behavior and track the growth of their user networks. The information available behind these logins often has separate terms and conditions. Many sites that are friendly to scraping on content that is fully public may request that you avoid scraping secured information. Since the security behind these logins has become very effective with today's technology, it is almost impossible to access this information other than by actually registering for an account. When creating such accounts, it is tempting to use an existing password that you may already have in place for other accounts (like personal social media, banking, utilities, etc). **Do not use personal logins for your web scraping project.**

8 Concerns and Biases in Online Data

Much of the web is composed of pages that are either outdated or obsolete. When crawling a large number of pages, depending on the source of the information, you may come across false or duplicated information. While duplicates can be good, false information can have a negative affect on analyses. In any study with self-reported data, there will always be bias because your dataset will consist only of the subset of individuals who choose to report the information you are analyzing. Online data is no different. This is selection bias, and depending on the type of data, it can be difficult or impossible to correct. A good example of this is in prices or wages, where data analysis will only tell you the wage distribution of people with jobs, or the price distribution of those companies selling the goods or services you are examining. You will never know what the unreported observations look like. What is important here is that we recognize that only a small subset of online users actually generate information. A very small minority creates the vast majority of internet content. This phenomenon is known as the "1% rule" (van Mierlo 2014) and is especially prevalent

in social media forums, where many users join initially, but participate sparingly or not at all. In general, this ties into the Pareto Principle, also called the 80/20 rule, where 20% of people do 80% of the activity. Social desirability bias is also prevalent in social media forums. When information is fully transparent and the creators are not anonymous, content generators will tailor their posts to appear socially desirable; that is, they will say what they think the public wants to hear. This tends to limit extreme views, which is obviously a good thing in the context of digital health. However, the result is that the distribution of your data distorts towards common social norms, and may not be completely representative.

9 Conclusions

10 Technical Appendix: Python Code

10.1 Scraping of a single website

This first appendix section shows an example of scraping a single webpage. I collect historical stock price data from Yahoo finance. This code is designed to scrape just one page (tabular data for the Dow Jones Index, stock symbol DJI). I provide thorough commenting and docstrings, so you shouldn't have trouble understanding what the code is doing, but what's more important is how each section of code relates to the principles discusses earlier. I provide explicit labels to show exactly why each batch of code is relevant, in terms of either ethics or efficiency.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

class YahooScraper:
    '''Scraping Yahoo Finance Data'''
    def __init__(self, stock_symbol):
        # Store stock symbol as attribute
        self.stock_symbol = stock_symbol

    def __repr__(self):
        # Define how YahooScraper objects are printed
        print('Historical Stock Data for \%s' \% self.stock_symbol)

    def get_url(self):
        # Check for string input
        assert(isinstance(self.stock_symbol, str))
        base_url = "https://ca.finance.yahoo.com/quote/"
        # Add params as strings
        url_params = "\%5E" + self.stock_symbol + "/history?p=\%5E" + self.stock_symbol
        return base_url + url_params

    def get_data_as_list(self):
        '''Extracts tabular data as a single list of dates and variables.
        Data is listed in order, L2R, reverse chronology'''
        # No sense scraping the same data twice
        if hasattr(self, "data_list"):
            return self.data_list
        else:
            # Create URL and extract contents
            url = self.get_url()
            # Execute HTML GET request
            page = requests.get(url)
            # Parse HTML
            soup = BeautifulSoup(page.content, 'html.parser')
            # Find all table column tags with appropriate class
            data = soup.find_all('tr', class_ = "BdT Bdc(\$seperatorColor) Ta(end) Fz(s) Whs(nw)")
            # Create list of span tags containing information
            data_as_list = [row.select('span') for row in data]
            # Extract information
            data_separate = [elem.text for row in data_as_list for elem in row]
```

```
            # Remove Commas
            data_list = [t.replace(",", "") for t in data_separate]
            # Store the list as an attribute
            self.data_list = data_list
            # Return the list
            return data_list

    def data_as_pandas(self):
        '''Convert data from list to Pandas df.'''
        variables = ['date', 'open', 'high', 'low', 'close', 'adj_close', 'volume']
        vars_no_date = [var for var in variables if var != "date"]
        # No sense making the df twice
        if hasattr(self, "df"):
            return self.df
        else:
            # Extract data in list form and convert to array
            data = self.get_data_as_list()
            data_array = np.array(data)
            # Reshape array into 2D
            data_array = data_array.reshape( ( int(len(data)/len(variables)) , len(variables) ) )
            # Convert array to pandas df
            stock_df = pd.DataFrame(data_array, columns = variables)
            # Convert object dtype to numeric
            stock_df[vars_no_date] = stock_df[vars_no_date].apply(pd.to_numeric)
            # Store df as attribute
            self.df = stock_df

            return stock_df

    def plot_data(self):
        '''Plot each variable'''
        df = self.data_as_pandas()
        variables = list(df.columns)
        df.plot(x="date", y = [var for var in variables if var != "date"])
```

The code above defines a scraper class, whose only input is the stock symbol. Depending on which company's price's you want, you need only change that one argument, since the URL does not change beyond that symbol. Some programmers prefer functional style programming to object-oriented, but in my experience python excels at OOP, and if you're engineering a scraper for use by others (or even building a full-fledged API), OOP is the way to go. Having defined the class above, all that remains is to instantiate such an object, and use the methods that it inherits. The following code retrieves the data for the Dow-Jones Index (abbr. DJI).

```
# Instantiate the scraper object
dji = YahooScraper('DJI')
# Use methods defined above
print(dji.data_as_pandas())
print(dji.plot_data())
```

These three lines perform the scraping (just one request to the YF server), and return the following information:

```
           date      open      high       low     close  adj_close     volume
0   Jun. 04 2021  34618.69  34772.12  34618.69  34756.39   34756.39  270050000
1   Jun. 03 2021  34550.31  34667.41  34334.41  34577.04   34577.04  297380000
2   Jun. 02 2021  34614.62  34706.65  34545.96  34600.38   34600.38  263810000
3   Jun. 01 2021  34584.19  34849.32  34542.87  34575.31   34575.31  287700000
4    May 28 2021  34558.50  34631.11  34520.09  34529.45   34529.45  290850000
..           ...       ...       ...       ...       ...        ...        ...
95  Jan. 19 2021  30887.42  31086.62  30865.03  30930.52   30930.52  386400000
96  Jan. 15 2021  30926.77  30941.98  30612.67  30814.26   30814.26  433000000
97  Jan. 14 2021  31085.67  31223.78  30982.24  30991.52   30991.52  427810000
98  Jan. 13 2021  31084.88  31153.37  30992.05  31060.47   31060.47  413250000
99  Jan. 12 2021  31015.01  31114.56  30888.76  31068.69   31068.69  362620000
```

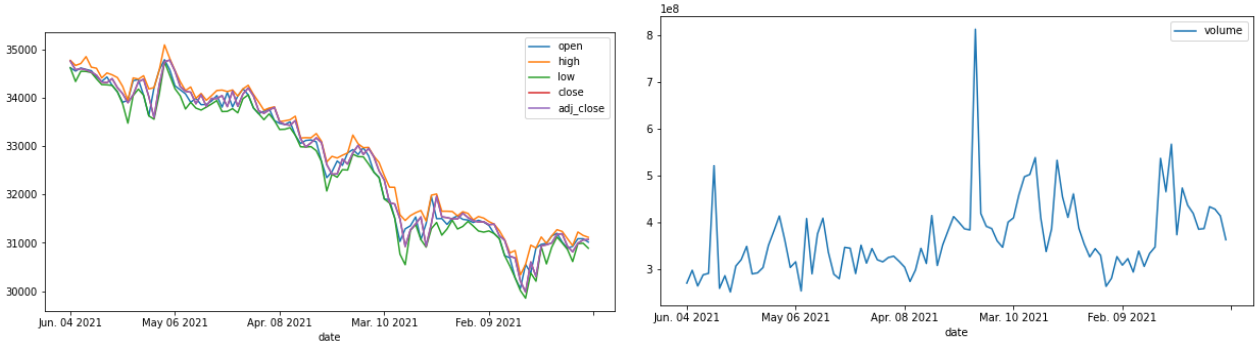

Figure 1: Line Plots of Dow-Jones Index Historical Price Scraped From Yahoo Finance.

10.2 Crawling through multiple websites