Overview of Approaches for Collecting Data from Online Platforms

FELIX SOLDNER

GESIS – Leibniz Institute for the Social Sciences

Publication date: April 24, 2024; Version 1.0

In this guide, we discuss why collecting data from online platforms that are accessible through the internet is important to researchers. This is followed by an overview of web data collection methods, explaining how these methods differ and pointing to their advantages and disadvantages. We also provide criteria on how to decide which data collection method should be used for a certain task, and briefly touch on data quality aspects and legal considerations, which are essential to be clarified before collecting online data.

This guide is written for readers who want to get a first overview of possible data collection approaches and have to decide which methods might be best suited for their project.

Keywords: data collection, online data, platform data, web scraping, web API, static websites, dynamic websites

1 Why collect data from online platforms?

With the emergence of the internet, the way people communicate, socialize, and interact has changed and significantly shifted to digital spaces such as online platforms, including social media, e-commerce websites, and online forums. As a result, social scientists have increasingly turned to online platforms as a data source for studying human behavior (Edelmann et al., 2020; Lukito et al., 2023). One reason online platforms are important for social and other scientists is the accessibility to vast amounts of data – also called **digital traces** or **digital behavioral data** (DBD) – which were previously difficult to obtain at scale. Such user-generated data on online platforms can be leveraged for large-scale analyses to gain insights into various social phenomena, including political attitudes, personal or group communication behavior, consumer preferences, social network structures or societal issues.

Online platforms also provide a unique opportunity for researchers to study human behavior in natural settings. Compared to laboratory settings, which may suffer from artificiality or lack ecological validity, online platforms offer a virtual environment for studying social interactions on the web. Thus, digital behavioral data can be valuable for studying complex social phenomena that may be difficult to replicate in controlled settings.

Furthermore, online platforms offer the possibility of real-time data collection and analysis. Since digital technologies allow for the continuous monitoring of user behavior, researchers can quickly and easily collect data on changes in attitudes or behaviors over time and capture unexpected or unanticipated events as they unfold. Such data capture and analyses can be particularly valuable for studying fast-moving phenomena, e.g., social movements or online reactions to events, which may be challenging to capture through traditional research methods.

While data from online platforms are highly valuable for researchers interested in investigating human behavior, interactions, and communication dynamics, obtaining such data requires technical methods that can be challenging. We will overview those methods and discuss their usefulness.

2 Finding existing datasets or collecting your own data?

Before starting any data collection, existing datasets should be examined and explored to determine if it is necessary at all to collect new data. Often, datasets of the platforms or phenomena intended to be studied already exist. By exploring already collected data, time-consuming (and sometimes costly) data collection procedures can be avoided. Besides performing traditional web searches, datasets specifically related to social science research can be found, e.g., at GESIS search (search.gesis.org) or through:

- Google dataset search (datasetsearch.research.google.com) which is a search engine used to find publicly available data for research.
- The open science framework (OSF; osf.io), Zenodo (zenodo.org), Kaggle (kaggle.com) and Figshare (figshare.com) are platforms supporting scientists to provide openly accessible research, including papers, data, and methods (e.g., code).
- The Inter-university Consortium for Political and Social Research (ICPSR; icpsr.umich.edu).
- Organizations, such as the Pew Research Center (pewresearch.org) or the International Social Survey Program (ISSP; issp.org).

Previously collected data may not always be suitable for the individual use case that a researcher has in mind, e.g., the previous data cover the wrong temporal scope, do not include the needed variables, or they do not contain the intended platform of interest, so new data is indeed needed.

For these cases, and when online data must be newly collected, two approaches are widely adopted:

1. **Web application programming interfaces (APIs).** With web APIs, platform data can be quickly accessed automatically and structured. APIs are mostly provided by the platforms themselves and access is often tiered, with free and paid options for which user often must go through a vetting process. How the API functions can also be opaque (e.g., how the data from the API is sampled or moderated).
2. **Web scraping.** Web scraping is conducted via programs that capture the content of websites ('scrapers'), which can be complex due to the various and continuously changing structures of websites. Thus, scrapers often require more time to be implemented and maintained but leave researchers with a large degree of freedom on what and how to collect data. With a custom scraper, the data collection procedure is also very transparent.

The following sections will describe these two data collection approaches and discuss their advantages and disadvantages.

3 Online data collection approach 1: Web APIs

Unlike a **user interface** intended to facilitate communication between a person and a computer, an **application programming interface** (API) facilitates communication between computers. Initially, APIs were not intended for data collection but rather for facilitating communication between computers more efficiently through reusable pre-written code. APIs often have written documentation about the code they provide and their functions (e.g., what they do and how to use them). For example, programming packages in Python (e.g., Pandas, NumPy) or in R (e.g., Caret, Shiny) are APIs, which help programmers speed up their coding by providing a set of pre-programmed functions that perform commonly needed operations; they save the individual researcher from the need to write everything from scratch.

Similarly, web APIs extend functionality across the internet by providing functions to clients (computers accessing information from a server) and the user to interact with a server more efficiently. **Figure 1** provides a relational overview of an API development and usage as an example. APIs exist for many social media platforms (e.g., TikTok, Twitter, YouTube), news outlets (e.g., Guardian, BBC, Financial Times, New York Times) or other platforms (e.g., Google search, Imgur, Shazam).

A quick web search will help determine if APIs exist for the platform of interest. However, APIs are also often not intended for research, but for facilitating the interaction with a platform. For example, eBay and Amazon provide APIs that allow an easier integration of their services into other products (platforms, Apps, etc.). With their APIs, external platforms could show the latest offer on specific products on Amazon or integrate an eBay product search into a shopping App. In these cases, Amazon and eBay benefit from the use of their API, which is not always true when data is collected for research purposes. Similarly, YouTube provides an API, which can also be used to organize content for an account (e.g., uploading videos).

If an API exists most often the platforms themselves provide them, to ensure what and how data is shared. Although APIs are more stable than custom scrapers, they also change over time in how they can be accessed (e.g., paid tiered options), and what information they provide.

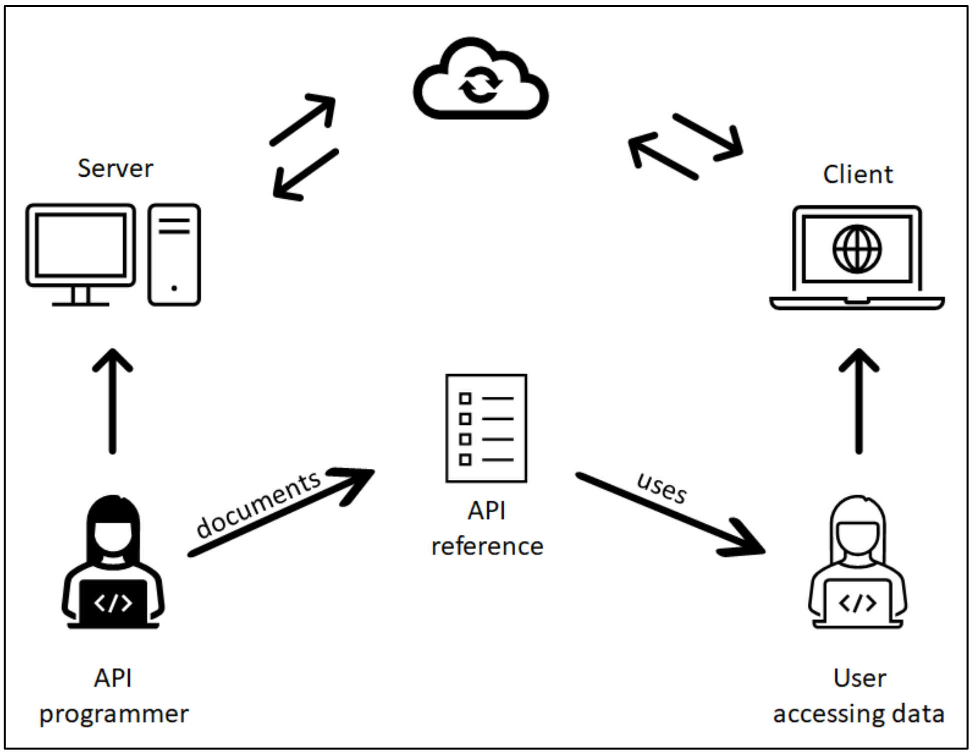


Figure 1: Relational setup of using an API

API functionalities

Some APIs can be accessed through traditional web browsers by manually entering the API call in the URL (i.e., sending a request to a server for information). Since each web API call requires a command, manually using a web browser would be too inefficient for collecting large amounts of data; it is mostly used for smaller projects or demonstration purposes only.

Web APIs are usually accessed through **wrappers** that facilitate the interaction with the API through more common programming languages, such as Python or R. Essentially, API wrappers are built on top of the APIs to further simplify the communication with the API and are convenient for the users because they easily implement the automation of requests (e.g., through for- or while-loops) and handle the received information within a programming environment.

Most APIs have restrictions on data types and how much data they provide. Such restrictions are often tiered, with free access providing the least amount of data, while higher tiers provide a wider variety of data types and larger amounts of data. Higher tiers then can be used through making payments, which can be a one-time payment or for a recurring period (e.g., monthly), or will be based on the amount of data requested (e.g., number of API calls, tokens received). Thus, determining what type of data is available and how much is essential before collecting your data. Before switching to a paid option, free access should be used to test the data collection pipeline in order to reduce costs.

Web APIs that do not have free versions often provide virtual environments (i.e., sand-boxes), which enable users to test their code before paying for any services. However, before being allowed to use API access, users must often undergo a vetting procedure in which the project's goals and procedures must be described and submitted to the API providers. The duration of receiving approval depends on the vetting procedure and the API provider and should be considered during the project planning. How to apply for API access depends on the platform and is mostly described on their website. However, users mostly have to apply for a developer account by filling out an online form, which will be manually reviewed (i.e., vetted). Once the application is approved, the user receives access to a dashboard for API settings, in which an API key can be generated. The key needs to be specified when interacting with the API and serves as an identifier of the user and the associated project. The dashboard is often used for payments and tracking the API usage (i.e., for specifying costs).

4 Online data collection approach 2: Web scraping

The terms **web scraping**, **web harvesting**, or **screen capture** all describe the procedure of collecting content from web pages. The idea is that all the content which is visible when visiting the website is collectible. While manually collecting the information from a web page is possible (e.g., downloading images, copy-paste text, saving the entire HTML site), most scraping procedures are automated with the help of stand-alone software or through custom programmed scrapers in Python, R, or other programming languages.

In most cases, not all the information on the webpage is of interest for the task, and only some parts should be collected. Thus, a webpage must be **parsed**, entailing a systematic capture of the content, which can be saved in a structured format (e.g., in a data frame). For example, if the intention is to collect product reviews, their exact placement within the webpage must be determined for the implemented program, allowing for systematic storage (i.e., assigning a variable in the data frame with only the review text).

Saving and parsing information from a webpage are two steps that are interchangeable in order. While some prefer to parse the webpage before saving any information, others prefer to save the entire webpage and then parse its content offline. Both approaches have their merits. Parsing a webpage directly without saving the entire page will require less local storage capacity and is often preferred when only specific content is needed, and the scraping procedure is expected to be short. However, parsing webpages directly can be disrupted by site maintenance, which can change the underlying structure (e.g., HTML layout) of the web pages. Such changes occur frequently, and they are not always apparent on the website. Structural changes in the sites' code can break the defined parsing procedure and lead to data gaps.

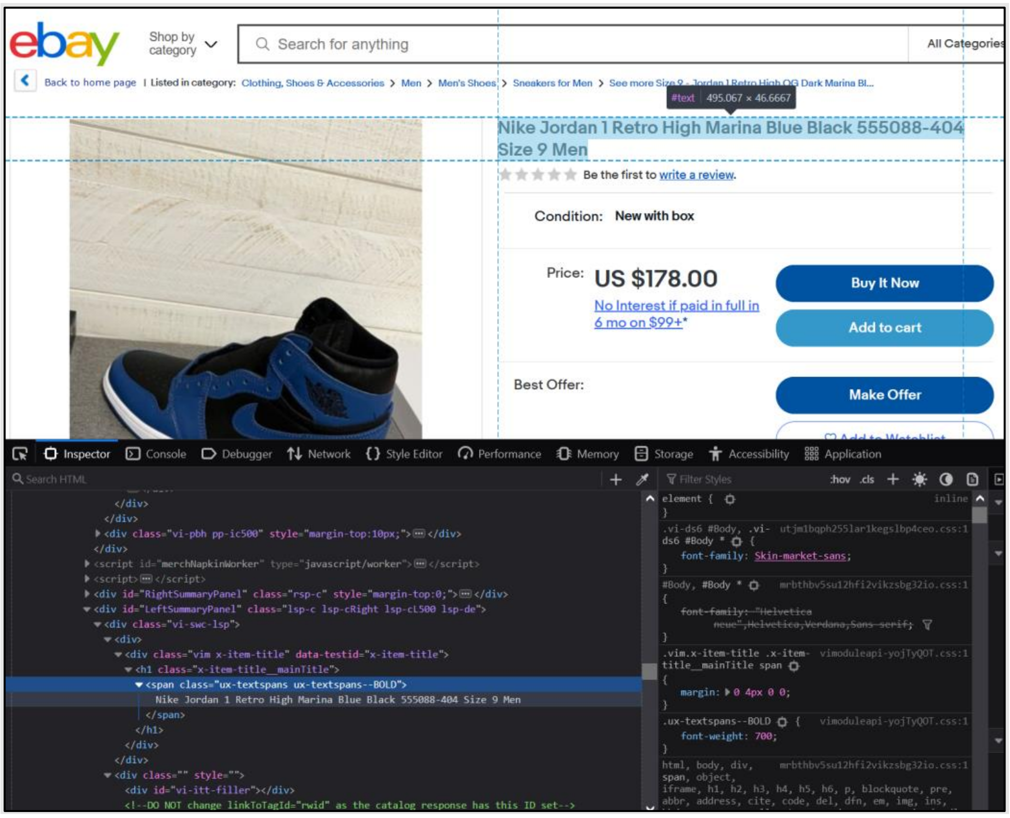
Therefore, saving the entire webpage before parsing is often preferred since parsing issues can be resolved retroactively without the threat of missing information that was deleted or changed on the webpage. However, saving an entire webpage, including images, will require more local storage space. A small-scale scraping test should be conducted before starting the main scraping to calculate the storage space needed for the entire project.

Web scraping implementations

Web scraping can also be accomplished through stand-alone software or online services. However, most tools require payments which can vary depending on the scale of the scraping project. Utilizing such tools is often achieved through point-and-click operations by navigating to the websites from which information should be collected and defining which elements of the websites should be scraped (titles, texts, images, etc.). For example, a browser plugin might be implemented to facilitate such an approach. In some cases, services offer scrapers accessible with custom APIs (often associated with increased costs), making data better accessible.

Alternatively, a **custom scraper** can be implemented in Python, R, or another programming language. Although building a scraper requires programming knowledge, all the necessary tools are publicly available without fees. The advantage of building your own scraper is that the programmer has complete control of what is collected and how, making changes to the collection procedure easier to implement. Moreover, a custom scraper allows for high transparency, which is essential within a research setting. Furthermore, many programming packages for scraping web content facilitate the programming of scrapers and can be found with a quick web search. Some helpful packages are Beautiful Soup (crummy.com/software/BeautifulSoup) or Selectorlib (selectorlib.com).

Building a web scraper, a) requires an understanding of webpage structures (i.e., HTML structure), and b) the target webpage should be inspected before beginning to build a scraper. Understanding the webpage structure is necessary when specifying the information location that should be collected by the scraper. By right-clicking with the mouse anywhere on a webpage and selecting "inspect", the structure of the web page can be made visible through a menu, as shown in **Figure 2**. By using the “inspector” function, it is possible to hover over any web element (title text, price, vendor name, etc.) and locate the information within the HTML structure.


ebay
Shop by category
Search for anything
All Categories
Back to home page | Listed in category: Clothing, Shoes & Accessories > Men > Men's Shoes > Sneakers for Men > See more Size 9 - Jordan 1 Retro High OG Dark Marina Bl...
#text 495.067 × 46.6667
Nike Jordan 1 Retro High Marina Blue Black 555088-404 Size 9 Men
Be the first to write a review.
Condition: New with box
Price: US $178.00
No Interest if paid in full in 6 mo on $99+*
Buy It Now
Add to cart
Best Offer:
Make Offer
Add to Watchlist
Inspector
Console
Debugger
Network
Style Editor
Performance
Memory
Storage
Accessibility
Application
Search HTML
Filter Styles
:hov
.cls

```
      </div>
    </div>
  <div class="vi-pbh pp-ic500" style="margin-top:10px;">…</div>
  </div>
<script id="merchNapkinWorker" type="javascript/worker">…</script>
<script>…</script>
<div id="RightSummaryPanel" class="rsp-c" style="margin-top:0;">…</div>
<div id="LeftSummaryPanel" class="lsp-c lsp-cRight lsp-cL500 lsp-de">
  <div class="vi-swc-lsp">
    <div>
      <div class="vim x-item-title" data-testid="x-item-title">
        <h1 class="x-item-title__mainTitle">
          <span class="ux-textspans ux-textspans--BOLD">
            Nike Jordan 1 Retro High Marina Blue Black 555088-404 Size 9 Men
          </span>
        </h1>
      </div>
    </div>
    <div class="" style="">
      <div id="vi-itt-filler"></div>
      <!--DO NOT change linkToTagId="rwid" as the catalog response has this ID set-->
```

```
element {                                                  inline
}
.vi-ds6 #Body, .vi-  utjm1bqph255lar1kegs1bp4ceo.css:1
ds6 #Body * {
   font-family: Skin-market-sans;
}
#Body, #Body *        mrbthbv5su12hfi2vikzsbg32io.css:1
{
   ~~font-family: "Helvetica
         neue",Helvetica,Verdana,Sans-serif;~~
}
.vim.x-item-title .x-item-   vimoduleapi-yojTyQOT.css:1
title__mainTitle span
{
   margin: 0 4px 0 0;
}
.ux-textspans--BOLD {   vimoduleapi-yojTyQOT.css:1
   font-weight: 700;
}
html, body, div,      mrbthbv5su12hfi2vikzsbg32io.css:1
span, object,
iframe, h1, h2, h3, h4, h5, h6, p, blockquote, pre,
abbr, address, cite, code, del, dfn, em, img, ins,
```

Figure 2: Screenshot of a product offer on eBay with the webpage’s underlying HTML structure shown.

After gaining a basic understanding of the web page’s structure, the next steps for programming a scraper can be taken, which are outlined in the following.

Programming a web scraper follows a structure, which can change depending on whether parsing is performed *before* or *after* downloading the web content. However, a few steps are essential, such as (i) creating a list of websites that are intended to be scraped, (ii) looping through and downloading the website’s content (with or without parsing), and (iii) saving the downloaded content in a database or data frame (e.g., SQL, CSV) if you want to analyze the data further [1].

Creating a list of websites can be done manually by copy-pasting the websites’ URLs into your programming script. However, such an approach is time-consuming and should be automated if possible. Alternatively, only the base URL should be specified (e.g., ebay.com), and the remaining URLs should be generated automatically. Similar to in-

1 In some projects, the goal is to collect only the raw data (e.g., HTML files) without bringing the content into a structure to maximize the possibilities of how the data can be used by others.

specting the website's HTML structure, understanding the URL structure is also important. Usually, the URL follows a clear pattern for pagination and other common operations (e.g., selecting the size and color of products on a shopping platform). Thus, after some testing (i.e., clicking through the webpage), a pattern most often emerges that can be used to generate the URLs automatically (e.g., looping through pages of search results). Another practice is to automatically find all links on the website of the current URL and add them to the list of URLs to be scraped. By proceeding like this, the entire webpage can be collected. However, managing duplicates becomes important in order to avoid unnecessary scraping, which can be time-intensive and might incur increased computational load to the server on which the website is hosted. Saving duplicates will also artificially increase the amount of data that is saved.

After determining how to obtain the necessary URLs, the scraper needs to collect the content of interest (including parsing the website) or the entire page. In either case, the data should be saved in a local folder structure, allowing easy (automated) access to the data in later stages of the project. Locally saving the data includes unparsed (e.g., HTML files) and parsed (e.g., CSV or Excel files) data. The data should be continuously saved while scraping the content from the website to prevent unnecessary computer memory usage and data loss if the scraping procedure unexpectedly stops (e.g., due to errors in the programming scripts, internet failure, or server issues). Once the web data collection is completed, the data can be saved into a data frame or merged if already parsed.

Scraping procedures may change depending on the type of website, and additional tools might be required to build a scraper. For instance, websites might be static or dynamic in loading displayed content. The differences between static and dynamic websites and their impacts on building a web scraper will be discussed next.

Scraping static vs. scraping dynamic websites

Content on websites can be generated statically or dynamically. Content generated statically is present whenever the requested URL is loaded, i.e., the content displayed when visiting the website without further user interaction. Content that is generated dynamically means that the website displays new content due to interactions with it, such as clicking or scrolling. For example, an endless newsfeed (e.g., on X/Twitter or Facebook), which allows for continuous scrolling, is dynamically generated. A reloading of the website is not required. Some content may appear dynamic but is already present (but hidden) on the website (e.g., a drop-down menu). A broad range of interactions can elicit dynamic content creation from the client or server side, including cookie settings, IP address, time of day, screen size, mouse scrolling, hovering, clicking, and more. Pop-ups or suggested auto-completions (e.g., in web search engines) are content that can be generated dynamically.

Potential challenges (including missing data) can be anticipated and resolved by testing if and how content is dynamically generated (i.e., interacting with the website). For example, some websites do not fully load their content without user interactions and require a different approach than traditional scraping methods can provide. In such cases, user interactions with the webpage can be augmented by automating the browser. All user interactions that elicit dynamic content creation (e.g., scrolling, clicking) can be automated with software such as Selenium (selenium.dev), which facilitates automation with common programming languages.

5 Which approach should I use?

Deciding which method to use is often heavily guided by practical reasons and is often determined by the extent to which a web API is in place and usable. First, whether an API for the intended platform is available should be checked. If no API or already created datasets exist, web scraping will likely be the only option (unless a data-sharing agreement with the platform can be negotiated). Second, if an API exists, not all necessary variables for the data collection project might be available, which should be determined. Third, the API restrictions and the amount of data needed might not be compatible, which should be assessed, and if any payments would be required.

After working out all the practical issues, any (dis-)advantages of using an API or web scraper can be considered. However, some platforms with APIs complicate the usage of custom scrapers by making the website structure overly complex, which can increase the time needed to build a scraper or completely forbid scraper usage (we will discuss some legal considerations in section 7). Thus, deciding which method to use might not always be in the hands of the data collectors. Assuming both an API and a scraper are possible, their advantages can be considered.

Advantages and disadvantages of both approaches

The big advantages of using web APIs are their ease of use and quick deployment. Since APIs are streamlined and platform-specific, they are reliable and well-suited for continuous and large-scale data collection. APIs do not tend to break easily, and it is less likely that code changes will become necessary, which allows for a prolonged usage. In turn, web APIs have data (type) limits that can be very restrictive or costly. Since API developers define how data can be collected, researchers often face issues with replicability. For example, if changes are made to the availability of data types or the amount of collectible data, replicability and continuation of projects can be complicated. APIs may also not be sufficiently documented, and the platform's sampling procedure on how they provide platform data through the API may be opaque, which complicates statistical evaluations and makes conclusions less reliable. Receiving API access requires vetting by the company the API maintains, for which an application is often required, stating the purpose of the projects. In some cases, APIs are also only available for specific users (e.g., eBay vendor API).

The big advantage of web scrapers is that they are customizable, allowing for data collection that would otherwise not be possible. With a custom scraper, data collectors have high control over what is being collected. However, a custom scraper also entails higher time investments in programming and might be further exacerbated, depending on the website structure (e.g., dynamic content), which is one of the major drawbacks of web scrapers. Furthermore, custom scrapers require continuous maintenance, assessing whether the collected data is complete and debugging any errors that arise.

The (dis-) advantages of the two approaches are summarized in **Table 1**.

	API	Custom scraper
Advantages ⬆	• easy and fast to employ • reliable (does not break easily)	• high control on data collection • allows data collection when no API is available • supports reproducibility and transparency of research
Disadvantages ⬇	• data (type) limits • pay to access data • possibly opaque data sampling • often requires vetting of users • dependent on API developers or providers	• high initial programming load • continuous maintenance needed • vulnerable to breaking when website changes • the platform's terms and conditions might prohibit scraping

Table 1: Overview of advantages and disadvantages of APIs and custom scrapers.

How to decide in practice

Re-using existing datasets is the most time and cost-efficient way of obtaining data and should be preferred before collecting new data. Contacting a platform to request data is also an option.

Whenever new data collection is needed, using a web API is most likely preferable because building and maintaining a custom scraper is time intensive.

Scrapers should also not be regarded as an option in (most) cases, when an API access application has been denied, since the platform providers already indicated that they do not want to share their data.

If no API exists or the API does not provide the correct data, a scraper is an option. There are freely available tools that support building a custom scraper.

6 Data quality considerations

The different approaches to collecting data from online platforms may influence the quality of resulting datasets and the results of downstream analyses. While it is difficult to assess the quality of a dataset as a result from a specific collection approach, there are error frameworks like the TED-On (Sen et al., 2021) that support researchers in their reflection on the data collection process, raising various potential data quality issues. In any case it is worthwhile and timesaving to consider such pitfalls early in the research process and to document your data collection process carefully. This will also support the transparency and replicability of your research.

One common issue is that the different collection methods usually have different constraints regarding the temporal availability of data. For APIs, there might be a restriction imposed directly by the platform, making only the most recent contents or activities available. While theoretically, this restriction does not apply to customized web scrapers, they might still run into difficulties with the collection of comparable content or longitudinal data when the structure of the webpage changes over time. The TES-D (Fröhling et al., 2023), an error-oriented documentation approach for online platform data, encourages researchers to systematically explore and document the consequences of their research design choices, hoping to raise awareness for such common data collection issues.

With regard to a collected dataset's quality, in particular its completeness, the direct comparison of datasets resulting from different collection approaches is helpful.

For data collected via web scraping, data quality concerns include incomplete data extractions due to changes of the website's URL pattern and HTML layout over time or due to technical difficulties with the continuous operation of the crawler, as well as the potential influence of the crawler setup (e.g., IP address location) on the displayed and collected data.

For web API data collections, the most common concerns include the opacity of the underlying sampling mechanisms that determine which data is available through the API (Tromble 2021), and the difficulties associated with properly sharing the data with other researchers, this being desirable for peer review and reproducibility or in the spirit of open science.

7 Legal considerations

Consideration should be given to the legal aspects of collecting web data. Accessing data through a web API is less likely problematic if the platform the data is collected from also maintains the API. Collecting data might not always be allowed if the API is built by others than the platform providers or if a custom web scraper is employed. Three basic checks can be performed to determine whether automated data collection is possible.

The **terms of use** should be consulted in which scraping ruls are often specified. Automated data collection might be fully allowed or disallowed, or, in some cases, specified for certain sections of the platform. For example, the capture of product information is allowed, whereas the capture of any vendor information is not allowed. Most websites contain links to their terms at the bottom of the webpage.

Most websites also contain a “**robots.txt**” file, which specifies if scraping is allowed, for whom, and on which portion of the website. The robot file can be accessed by adding the “robots.txt” file with a slash to the base URL of the website (e.g., ebay.com/robots.txt). The scraper can access this file automatically and it can be used to determine which portions of the websites are allowed to be scraped. Additional explanations of the robot.txt file can be found at robotstxt.org.

Websites might implement **anti-scraping mechanisms** such as CAPTCHAs (i.e., small tasks, that can be solved by humans to prove they are not a robot) or block the computer’s IP for a short duration if too many requests are sent. Such mechanisms are signs that the platform providers do not allow scraping, and they should, in most cases, not be circumvented. What is allowed or not allowed might also change depending on the country and individual circumstances of the data collection project and should be clarified before performing any scrapes. Additional resources about legal considerations can be found in (Altobelli et al., 2021; Krotov & Johnson, 2023; Luscombe et al., 2022).

References

Altobelli, C., Forgó, N., Johnson, E., & Napieralski, A. (2021). *To scrape or not to scrape? The lawfulness of social media crawling under the GDPR*. University of Vienna, Faculty of Law, Department of Innovation and Digitalisation in Law.

Edelmann, A., Wolff, T., Montagne, D., & Bail, C. A. (2020). Computational social science and sociology. *Annual Review of Sociology*, *46*(1), 61–81. https//doi.org/10.1146/annurev-soc-121919-054621

Fröhling, L., Sen, I., Soldner, F., Steinbrinker, L., Zens, M., & Weller, K. (2023). *Total Error Sheets for Datasets (TES-D). A critical guide to documenting online platform datasets. arXiv*. https//doi.org/10.48550/arXiv.2306.14219

Krotov, V., & Johnson, L. (2023). Big web data: Challenges related to data, technology, legality, and ethics. *Business Horizons*, *66*(4), 481–491. https//doi.org/10.1016/j.bushor.2022.10.001

Lukito, D. J., Brown, M. A., Dahlke, R., Suk, D. J., Yang, D. Y., Zhang, D. Y., Chen, B., Kim, S. J., & Soorholtz, K. (2023). *The state of digital media data research*. https//doi.org/10.26153/tsw/46177

Luscombe, A., Dick, K., & Walby, K. (2022). Algorithmic thinking in the public interest: Navigating technical, legal, and ethical hurdles to web scraping in the social sciences. *Quality & Quantity*, *56*(3), 1023–1044. https//doi.org/10.1007/s11135-021-01164-0

Sen, I., Flöck, F., Weller, K., Weiß, B., & Wagner, C. (2021). A total error framework for digital traces of human behavior on online platforms. *Public Opinion Quarterly*, *85*(S1), 399-422. https//doi.org/10.1093/poq/nfab018

Tromble, R. (2021). Where have all the data gone? A critical reflection on academic digital research in the post-API age. *Social Media + Society*, *7*(1). https//doi.org/10.1177/2056305121988929

All links in the text and the reference list were retrieved on Jan 19, 2024.

About the author

Felix Soldner is a post-doctoral researcher at GESIS – Leibniz Institute for the Social Sciences in Cologne, Germany at the Computational Social Science department. He works with data science methods, such as machine learning (ML) and natural language processing (NLP) for his research on deception detection, fraud prevention, dark web markets, or data biases impacting ML performances. For his projects he uses various data collection approaches, including custom scrapers and APIs.

Acknowledgements

Leon Fröhling contributed on the data quality part, Figure 1 is based on a draft by Roberto Ulloa, and Gizem Bacaksizlar-Turbic provided helpful comments on the manuscript.

Suggested citation

Soldner, F. (2024): *Overview of Approaches for Collecting Data from Online Platforms*
(= GESIS Guides to Digital Behavioral Data, 8). Cologne: GESIS – Leibniz Institute for the Social Sciences

Series editors

Danica Radovanović, Maria Zens, Katrin Weller, Claudia Wagner

Publisher

GESIS Leibniz Institute for the Social Sciences

License