Tutorial

Web Scraping Using R

Alex Bradley and Richard J. E. James
School of Psychology, University of Nottingham

Abstract
The ubiquitous use of the Internet in daily life means that there are now large reservoirs of data that can provide fresh insights into human behavior. One of the key barriers preventing more researchers from utilizing online data is that they do not have the skills to access the data. This Tutorial addresses this gap by providing a practical guide to scraping online data using the popular statistical language R. Web scraping is the process of automatically collecting information from websites. Such information can take the form of numbers, text, images, or videos. This Tutorial shows readers how to download web pages, extract information from those pages, store the extracted information, and do so across multiple pages of a website. A website has been created to assist readers in learning how to web-scrape. This website contains a series of examples that illustrate how to scrape a single web page and how to scrape multiple web pages. The examples are accompanied by videos describing the processes involved and by exercises to help readers increase their knowledge and practice their skills. Example R scripts have been made available at the Open Science Framework.

Keywords
web scraping, web crawling, reverse engineering, big data, open science, open materials

Received 1/16/19; Revision accepted 6/3/19

The vast array of behaviors and interactions that occur and are stored online presents researchers with a wealth of data to help improve understanding of human behavior. For example, these data can be used to predict people’s political preferences (Ceron, Curini, Iacus, & Porro, 2014; Malouf & Mullen, 2015), shed light on the motivations behind charitable donations to crowd-funding campaigns (Agrawal, Catalini, & Goldfarb, 2015; Kuppuswamy & Bayus, 2018), or even determine which products people tend to compare before buying (Feldman, Fresko, Goldenberg, Netzer, & Ungar, 2007). Yet psychology has been slow to utilize online data. One of the main barriers preventing psychologists from using these data is a skills gap (Adjerid & Kelley, 2018; Paxton & Griffiths, 2017). The underlying issue is that new data methods, such as web scraping, require a knowledge of programming that most psychologists do not have (Adjerid & Kelley, 2018). The aim of this Tutorial is to address this skills gap by providing a practical hands-on guide to web scraping using R.

Web scraping allows the rapid collection and processing of a large amount of data from online sources. These data can be numbers, text, or a collection of images or videos (Marres & Weltevrede, 2013). Web scraping is time efficient, allowing thousands of data points to be automatically collected, whereas previously this would have involved painstaking manual effort. Web scraping is, therefore, less labor intensive, faster, and less open to human error than the traditional copy-and-paste method (Nylen & Wallisch, 2017, Chap. 10). Web scraping also has the advantage of allowing researchers to acquire novel, untouched data sets without the need for research grants to fund the purchase of expensive equipment or the costs of compensating participants. In this Tutorial, we cover how to download a web page, how to extract information from the downloaded page, how to store the extracted information, and, finally, how to move across pages on a website.

Disclosures
A website containing the examples and accompanying exercises and videos for this Tutorial can be found at

Corresponding Author:
Alex Bradley, School of Psychology, University of Nottingham, Nottingham NG7 2RD, United Kingdom
E-mail: alexander.bradley@nottingham.ac.uk
https://practicewebscrapingsite.wordpress.com/. All the R scripts for the examples and the PowerPoint slides used in the videos can be accessed or downloaded from the Open Science Framework, at https://osf.io/6ymqg/. The website was specifically designed to help readers learn about the process of web scraping and to provide a safe environment for practicing web scraping. The introductory video provides an overview of web scraping, the web-scraping tools that we use in this Tutorial, and good web-scraping practices. Example 1 shows readers how to download, extract, and store information from a single web page. Examples 2 and 3 explain how to download, extract, and store information while using links built into a website to move across multiple web pages. Example 4 shows how to download, extract, and store information while moving across web pages by manipulating URLs. We encourage readers to watch each example video while following along with the example R script and then to take the time to complete the accompanying exercise before moving on to the next example.

Learning Objective and Assumed Knowledge

The learning objective of this Tutorial is to teach readers how to automatically collect information from a website. In particular, after completing this Tutorial, readers should be able to download a web page, should know how to extract information from a downloaded web page, should be able to store extracted information, and should understand different methods of moving from page to page while web scraping. An understanding of R and RStudio is helpful but not required. The Tutorial has been designed so that novices to web scraping and readers with little to no programming experience will find the material accessible and can begin to develop their own scraping skills. Readers who already have R and RStudio installed and have a basic understanding of the R language may wish to skip the next three sections and proceed directly to the discussion of the four steps involved in web scraping.

Installation of R, RStudio, and SelectorGadget

All the programs you will need to web-scrape are free to download and use. First, you will need to download R (R Core Team, 2019) from https://cran.rstudio.com/ and install it on your computer. Second, we recommend downloading and installing RStudio (https://www.rstudio.com/). All the code for this Tutorial will be run in the script window of RStudio. You can create new scripts in RStudio by clicking on “File,” then “New File,” and then “R Script.” Finally, you will need SelectorGadget (Cantino & Maxwell, n.d.), which can be downloaded at https://selectorgadget.com/. If you do not use Chrome as your Web browser, you will need to download it (https://www.google.com/chrome/) before downloading SelectorGadget. For more information about how to download these programs, see the introductory video on the website accompanying this Tutorial (https://practicewebscrapingsite.wordpress.com/).

Packages and Functions in R

R is an incredibly versatile programming language capable of performing many different tasks, including web scraping, statistical analysis, and data visualization. The reason for its versatility is that it has a large community of users who create software, in the form of packages, that other users can use. A package is a collection of functions designed to perform a task. For example, in this Tutorial, we use the rvest package (Wickham, 2019), which contains a variety of functions that can be used to web-scrape. A function is code that modifies or manipulates some input to produce a desired output. For example, to calculate a mean, one can use the mean function by providing it with a vector (column) of numbers (i.e., mean(numbers)). A function often takes additional instructions, known as arguments, that adjust how it modifies or manipulates the input. For example, the mean function can be modified by using the na.rm argument to specify whether missing values are to be included (true or false; e.g., mean(numbers, na.rm = TRUE)). In order to use functions contained in a package, you first need to install and load that package.

Installing and Loading R Packages

Downloading and installing packages for use in R is a two-step process. First, one needs to download the package by using the function install.packages("package name"). Next, the package must be loaded into R by using the function library(package name). For the rest of this Tutorial, you will need to download and install the rvest package by typing and running the following code in RStudio:

```r
install.packages("rvest")
library(rvest)
```

Note that once you have installed a package, you will never need to download it again. However, every time you start a new session of RStudio, you will need
to run the library function to load the packages you will be using in that session.

**Four Key Steps Involved in Web Scraping**

**Downloading a web page**

To download a web page, use the `read_html` function and supply it with the URL of that page (i.e., `read_html("address of website")`). Example 1 on the website involves collecting the titles, main text, and picture links for three articles stored on a single web page. In order to download this page, type in and run the following code:

Example1 <- read_html("https://practiceweb
scrapingsite.wordpress.com/
example-1")

The `read_html` function downloads the web page at the address given and then the less-than and hyphen notation (<-) tells the software to assign that information to an object—in this case, an object called Example1. In technical terms, Example1 is a Document Object Model because it holds all the data of the web page and preserves the structure of the information held on that web page. What this line of code does is analogous to taking a physical book and converting the information into a digital book available on an e-reader. The book on the e-reader will contain the same number of chapters, pages, and paragraphs, with the same text in each of those sections. Similarly, the `read_html` function collects and preserves the structure of the information held on the web page. This is important because it allows you to extract just the information you are interested in.

**Extracting information from a web page**

Writing the code to extract information from a web page involves two steps: specifying the location of the information to be collected and then specifying what information at that location should be extracted. A good analogy is using a textbook to obtain a famous quote by an author. First, you turn to the chapter and page where that author is mentioned, and then you find the quote so that you can copy the famous words by the author.

Step 1, involves the `html_nodes` function, to which two pieces of information must be added: the object holding the downloaded web page and the address to the information you wish to extract (i.e., `html_nodes(web page, "address to information")`). In order to generate the address to the information you want, use SelectorGadget. When SelectorGadget is installed on your computer, there is an icon in the top right of the Chrome window that looks like this: 📚. While viewing the web page you are interested in, click on this icon to open SelectorGadget and then select the information that you wish to extract. For example, to get the address to the article titles in Example 1, click on the icon for SelectorGadget and then select the titles (see Fig. 1). Look down the page and make sure that only the information you wish to extract is highlighted green or yellow. If additional information that is not required is highlighted, click on that to unselect it. When only the right information is highlighted, copy and paste the address SelectorGadget generates into the `html_nodes` function. Thus, at the end of Step 1, you have written code that indicates where the information you wish to extract is stored. The next step is to pass along this information to a function that will perform the extraction.

Step 2 involves the pipe operator (%%), which takes the output from one function and passes it to another without the need to store it. In this case, the pipe operator is added to the code in order to pass along the information in the `html_nodes` function. The pipe operator is followed by one of three commands, depending on the type of information to be extracted. If you want to extract text, use the `html_text` function (i.e., `html_text()`). If you want to extract links from the web page, use the `html_attr` function with the additional `href` argument (i.e., `html_attr("href")`). Or if you want to collect the address of images to download later, use the `html_attr` function with the additional argument `src` (i.e., `html_attr("src")`). The following code will extract the titles, text, and address to the pictures of the three articles stored on the web page in Example 1:

`html_nodes(Example 1, "strong") %>% html_text()
html_nodes(Example 1, ".Content") %>% html_text()
html_nodes(Example 1, "#post-25 img") %>% html_attr("src")`

**Storing information collected while web scraping**

There are several ways to store extracted information. The best approach will depend on the type and amount of data you are extracting. For simplicity, in this Tutorial, we describe how to store information in vectors. This process changes depending on whether you are scraping a single page or multiple pages.

We begin by explaining how to store the information from a single page. To store the information extracted...
Fig. 1. Screenshot illustrating the use of SelectorGadget to extract the titles on the web page in Example 1 (only the first title is shown here). SelectorGadget has identified “strong” as their address, and this address can then be used to extract the titles.
by the Example 1 code just presented, assign (<-) this information to vectors called Title, Text, and Images by expanding the code as follows:

Title <- html_nodes(Example 1, "strong") %>% html_text()
Text <- html_nodes(Example 1, ".Content") %>% html_text()
Images <- html_nodes(Example 1, ".post-25 img") %>% html_attr("src")

A video demonstration of how to extract and store information from a single page is available at our website (see “Example 1: Scraping a Single Webpage”).

Storing information when scraping over multiple pages is a little more complicated because as one moves over each web page, extracting information and storing it to a vector, the information captured from the previous page will be overwritten. To avoid this problem, use the following three-step process: First initialize an empty vector. Second, extract the information from a web page using the html_nodes and html_text function and store this information in a second vector. Third, add the information captured in the second vector to the initially empty first vector. The following code will extract and store the titles in Example 2:

Title <- c()
Heading <- html_nodes(Example 2, ".entry-title") %>% html_text()
Title <- c(Title, Heading)

As the web scraper goes over each new page, the information in the Heading vector will be overwritten with new information and added to all the previously extracted titles stored in the Title vector. Our website has a video demonstration of this technique (see “Example 2: Scraping Multiple Web Pages”).

**Scraping across multiple pages**

There are a variety of methods for scraping across a website, and often the way the website is designed will determine the approach to use. To keep things simple, we outline two common approaches used by web scrapers to move across webpages: following links in a webpage to other pages and manipulating the Web address.

To follow links, you need to download a webpage containing links to all the other pages to be visited and then extract and store those links. The following code from Example 2 shows how to store the titles from multiple pages of a website using the links stored on a web page:

#Initialize the empty vector
Title <- c()
#Download the web page
Example 2 <- read_html("https://practicewebscrapingsite.wordpress.com/example-2/")
#Extract and store the links
BlogPages <- html_nodes(Example2, ".Links a") %>% html_attr("href")
#Use each of the links, stored in i, to visit a web page by means of a for loop, which repeats a block of code.
for (i in BlogPages){
  Example 2 <- read_html(i)
  Heading <- html_nodes(Example2, ".entry-title") %>% html_text()
  Title <- c(Title, Heading)
}

The i in the for loop becomes each one of the links stored in BlogPages, so the indented code between the two curly brackets (()) is repeated for each link. Notice that i is passed to the read_html function to download each new web page, from which information is extracted and then stored. It is the act of passing i to the read_html function that allows the scraper to navigate over multiple pages.

To scrape webpages by manipulating URLs, you need to identify a part of the URLs that systematically changes over the web pages. You then need to artificially manipulate the URL in your code to move over the different pages. Example 4 illustrates this process for a case in which the URL changes by the page number specified (i.e., https://practicewebscrapingsite.wordpress.com/example-4-page-0/, https://practicewebscrapingsite.wordpress.com/example-4-page-1/). This example requires that you generate a sequence of numbers to represent the different page numbers. Use the seq function, which takes arguments of the first number (0), the last number (1), and the increment of change (1). This information is saved to Pages:

Pages <- seq(0,1,1)
Next, use a `for` loop to iterate over the pages. The `i` will become the page numbers 0 and 1, so the code within the `for` loop will run twice:

```r
for (i in Pages){
  Sys.sleep(2)
  WebpageURL <- paste("https://
  practicewebscrapingsite
  .wordpress.com/example-4-page-",
  i,"", sep="")
  Example4 <- read_html(WebpageURL)
}
```

The `Sys.sleep()` function in this code inserts a pause in the running of the code, to avoid putting undue stress on the website server. In this example, a 2-s pause has been inserted. The `paste` function generates the URLs by taking the part of the URL that does not change and adding that to the number stored in `i`. The `sep` argument is left blank so that the unchanging and changing parts of the URL will be joined without any separation. The generated URL is then stored in `WebpageURL` and passed to the `read_html` function, which downloads the new web page. Information from this new web page can then be extracted and stored.

### Good Practices and the Ethics of Web Scraping

Before scraping a website, it is a good idea to check if it offers an application program interface (API) that allows users to quickly collect data directly from the database behind the website. If it does offer an API that contains the information you need, it would be easier to use the API. Also, although the methods presented here should help you scrape many websites, some sites may display information in unusual formats that make them more difficult to scrape. It is worth checking whether you can download and extract information from a single page before building a complete web scraper for a website.

When web scraping, it is a good idea to insert pauses between downloading web pages, as this helps spread out the traffic to the website. Web scrapers may be banned from a website if they put undue stress on it. In Example 4, we used the `Sys.sleep` function to insert 2-s delays before downloading web pages. Before starting your own web-scraping project it would be a good idea to check your institutional review board’s policy on web scraping; you might need to make an ethics application, or your target data might be classified as archival data that do not require an ethics application. As a general rule of thumb, any information stored behind a username and password is considered private and ought not to be web scraped.

### Summary

In this Tutorial, we have introduced readers to what web scraping is and why it is a useful data-collection tool for psychologists. We have provided a basic explanation of the R environment and how to download and install R packages. Readers should feel confident in their ability to conduct the four key steps of web scraping: downloading web pages, extracting information from downloaded pages, storing that extracted information, and using Web links or manipulating URLs to navigate across multiple web pages. We strongly recommend that readers work through the examples and exercises provided on the accompanying website to further build their knowledge of web scraping and gain more experience with this method.

### Action Editor

Alex O. Holcombe served as action editor for this article.

### Author Contributions

A. Bradley is the guarantor. A. Bradley created the website and videos. Both authors drafted the manuscript, provided feedback to each other, and approved the final version of the manuscript.

### ORCID iD

Alex Bradley [https://orcid.org/0000-0003-4304-7653](https://orcid.org/0000-0003-4304-7653)

### Declaration of Conflicting Interests

The author(s) declared that there were no conflicts of interest with respect to the authorship or the publication of this article.

### Open Practices

- **Open Data:** not applicable
- **Open Materials:** [https://osf.io/6ymqg/](https://osf.io/6ymqg/)
- **Preregistration:** not applicable

All materials have been made publicly available via the Open Science Framework and can be accessed at [https://osf.io/6ymqg/](https://osf.io/6ymqg/). The complete Open Practices Disclosure for this article can be found at [http://journals.sagepub.com/doi/suppl/10.1177/2515245919859535](http://journals.sagepub.com/doi/suppl/10.1177/2515245919859535). This article has received the badge for Open Materials. More information about the Open Practices badges can be found at [http://www.psychologicalscience.org/publications/badges](http://www.psychologicalscience.org/publications/badges).
References

Adjerid, I., & Kelley, K. (2018). Big data in psychology: A framework for research advancement. *American Psychologist, 73*, 899–917. doi:10.1037/amp0000190

Agrawal, A., Catalini, C., & Goldfarb, A. (2015). Crowdfunding: Geography, social networks, and the timing of investment decisions. *Journal of Economics & Management Strategy, 24*, 253–274. doi:10.1111/jems.12093

Cantino, A., & Maxwell, K. (n.d.). *SelectorGadget: Point and click CSS selectors*. Retrieved from https://selectorgadget.com/

Ceron, A., Curini, L., Iacus, S. M., & Porro, G. (2014). Every tweet counts? How sentiment analysis of social media can improve our knowledge of citizens' political preferences with an application to Italy and France. *New Media and Society, 16*, 340–358. doi:10.1177/1461444813480466

Feldman, R., Fresko, M., Goldenberg, J., Netzer, O., & Ungar, L. (2007). Extracting product comparisons from discussion boards. In N. Ramakrishnan, O. R. Zaïane, Y. Shi, C. W. Clifton, & X. Wu (Eds.), *Proceedings of the Seventh IEEE International Conference on Data Mining (ICDM 2007)* (pp. 469–474). doi:10.1109/ICDM.2007.27

Kuppuswamy, V., & Bayus, B. L. (2018). Crowdfunding creative ideas: The dynamics of project backers. In D. Cumming & L. Hornuf (Eds.), *The economics of crowdfunding: Startups, portals, and investor behavior* (pp. 151–182). London, England: Palgrave Macmillan.

Malouf, R., & Mullen, T. (2015). Taking sides: User classification for informal online political discourse. *Internet Research, 18*, 177–190.

Marres, N., & Weltevrede, E. (2013). Scraping the social? *Journal of Cultural Economy, 6*, 313–335. doi:10.1080/17530350.2013.772070

Nylen, E. L., & Wallisch, P. (2017). *Neural data science: A primer with MATLAB® and Python™*. London, England: Academic Press.

Paxton, A., & Griffiths, T. L. (2017). Finding the traces of behavioral and cognitive processes in big data and naturally occurring datasets. *Behavior Research Methods, 49*, 1630–1638. doi:10.3758/s13428-017-0874-x

R Core Team. (2019). *R: A language and environment for statistical computing*. Retrieved from https://www.r-project.org

Wickham, H. (2019). *Package ‘rvest’*. Retrieved from https://cran.r-project.org/web/packages/rvest/rvest.pdf