Web scraping in R using CSS selectors

Aka how to get more data when a client shorts you

Namrata Date · Jan 20 · 5 min read

Web scraping in R is easier than you'd imagine- tools like CSS selectors and scraping libraries make quick work of it. Indeed the "toughest" part is identifying the CSS path of an element, but I'll show you how to get there easily!

I once had a client (a commercial insurance broker in the US) who wanted a data science solution to improve their sales and conversion rates. Seemingly straightforward? The client gave us very little data to work with- just the customers' unique license number, the result of the sale (status of the policy), and the price of the premium.

I started looking around online for other sources of relevant data. The customer's unique license numbers were DOT Numbers issued by the Department of Transportation (DOT) of the US government. The Federal Motor Carrier Safety Administration (FMCSA) does routine inspections of vehicles and drivers associated with a DOT number and publish the report on their website. This is public information, and does not require a login/password for access.

Partial screenshot of the FMCSA safety report for a DOT number.

Data like reported number of crashes, vehicle/driver hazard rates could help optimize the policy premium to increase the odds of the customer purchasing the policy.

Getting this is a 3-step process:

1. Identify the elements of interest, and get their CSS path.
2. Condense the scraped elements into a dataframe.
3. Clean the output, and then join it to the existing data (given by the client).

Packages and tools

I used the rvest library to scrape the webpages, along with the CSS selector SelectorGadget. The tool can be downloaded as an extension to the browser, which makes it easier to determine the CSS path of each element on the webpage.

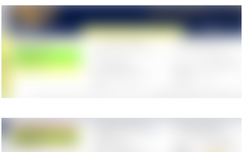
Define the URL and webpage

I started off using a single DOT Number report. The URL for the report contains the DOT Number itself, which would prove very useful to iterate the scraper through the list of DOT Numbers later.

Inspect elements of interest

To start off, I wanted to scrape the registered address of each DOT number, as well as the number of drivers they have. I need the CSS path for these elements- this tells the scraper the precise location of the elements to scrape. An address for the address, if you will.

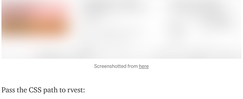
Turn on the SelectorGadget extension- a box at the bottom of the browser will appear. Select the area on the screen where the address is listed. The tool will highlight many other elements on the screen- at this point it has selected 67 other elements. Simply click on the yellow elements, until only the green address portion remains. (This only takes 3 to 4 clicks). Each "unselection" of extra elements updates the CSS path to the precise location of the address.



Now the CSS path for the address element is available. Screenshotted from here.

Now "address" is the only element highlighted. The seemingly gibberish text in the red box is the **CSS path** of this element, and this is what we will pass to rvest to scrape.

Similarly, to get the CSS path for the number of drivers:



Screenshotted from here

Pass the CSS path to rvest:

We have output! It has to be cleaned, a lot, but it worked!

Now I will scrape many more elements, for all the DOT numbers.

Iterate through all DOT numbers

I take advantage of the fact that the URL reflects the DOT number- https://ai.fmcsa.dot.gov/SMS/Carrier/2844805/CompleteProfile.aspx

I create a vector containing the URL for every DOT number.

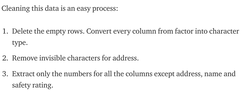
Now, scrape!

The last step combines all the different elements into one data frame called fmcsa_data. This will take some time depending on how many iterations you have. I suggest keeping the computer plugged in and making a cup of tea for yourself.

Cleaning the dataframe

The resulting dataframe is not pretty- lots of extraneous text hides the relevant information. There is also some missing data- the FMCSA website did not have reports for some DOT numbers yet, so nothing was scraped.

A closer inspection into the data reveals a lot of invisible characters too.



Cleaning this data is an easy process:

1. Delete the empty rows. Convert every column from factor into character type.
2. Remove invisible characters for address.
3. Extract only the numbers for all the columns except address, name and safety rating.

The cleaned dataframe is much more sensible!

And there we have it- a cleaned, concise dataframe from tens of thousands of reports.

I'm a data scientist at ScoreData in Palo Alto. I am very enthusiastic about effective communication- I nerd out on translating statistical output into business-friendly, actionable insight. I want this blog to be a space to share some things I've learned along the way, so come along for the ride!

https://www.linkedin.com/in/namrata-date/