Data Collection using Webscraping

Dr Robbie Baldock

11 November 2020

This worksheet aims to provide a practical tutorial of how webscraping can be used to collect data for further analysis using R programming and the 'rvest' package.

Firstly, make sure you have the rvest and tidyverse packages installed and load the package using the library() function:

```
install.packages("rvest")
install.packages("tidyverse")
```

Then load the package by calling the library() function. Additional details about the package can be displayed by using ?rvest.

```
library(rvest, tidyverse)
```

```
## Loading required package: xml2
```

We will now build a webscraper that collects reviews left on Trustpilot. In this example, we will collect reviews of Oakfurniture Land.

https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk

The webscraper is comprised of 3 elements: 1. The page to be read, 2. The location of the element to be collected and 3. The type of information to be returned (i.e. be it text or a url link etc.)

Create an object that is the url of the webpage to be scraped.

```
url <- "https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk"

page <- read_html(url) #1st element

page %>%
  html_node(".review-content__text") %>% #Specifies location of text to be scraped
  html_text(trim = TRUE) #Returns data as text
```

```
## [1] "Great service from friendly staff who were very patient..."
```

The trim = TRUE parameter in the html_text function removes line breaks from the responses. Try trim = FALSE to see the difference.

The SelectorGadget

".review-content__text" specifies the location of the information to be collected by the webscraper. You can easily generate these labels using a tool called 'SelectorGadget' developed by Hadley Wickham.

https://cran.r-project.org/web/packages/rvest/vignettes/selectorgadget.html

To use the selector gadget, visit the url above, drag and drop the hyperlink to the toolbar on your internet browser. Then click the SelectorGadget, click the element that you wish to collect on the page. It will then be green, all the subsequent elements that will also be collected are highlighted in yellow. If there are any additional elements that you do not wish to collect, simply click these and they will turn red and will be removed from the selection. Finally, once you have all the elements you want to collect highlighted in yellow, copy the code in the bottom right hand text box and copy it into the html_node() function. *Do not forget to include speech marks around the parameter.

What if you want to collect all the reviews on that page?

The html_node() function will only return the first element of specified by the parameter. If you want to collect all of the elements that match, simply use html_nodes() instead.

```
page %>%
  html_nodes(".review-content__text") %>%
  html_text(trim = TRUE)
```

```
##  [1] "Great service from friendly staff who were very patient..."
##  [2] "The sales assistant Carla was very helpful, did not..."
##  [3] "Knew what I wanted when arriving in store. They..."
##  [4] "NA..."
##  [5] "Ordering process difficult and not flexible re order dates..."
##  [6] "Firstly I was not happy at the hour of..."
##  [7] "Good quality well made piece of furniture.  ..."
##  [8] "I purchased the rustic oak corner TV cabinet. It..."
##  [9] "Absolutely disgusted! I bought a leather reclining chair and..."
## [10] "Excellent service,  A pleasure to seal with staff,..."
## [11] "Update: Oak Furnitureland contacted me to apologise again and..."
## [12] "Very good experience. Ordering process easy and efficient -..."
## [13] "Not actually received my furniture yet. Process for ordering..."
```

Outputs can be saved as an object by including the object name and '<-' in front of the code (in this case 'Review').

Data collected can be converted to a data frame and saved as a csv file using the following functions:

```
Review <- as.data.frame(Review)
write.csv(Review, file = "filename.csv")
```

How do I scrape reviews from multiple webpages?

Now you have managed to collect all reviews from a single web page, however as is often the case, the data you are interested in may be spread across many webpages. In this example, there are many webpages of reviews with approximately 18 reviews per page. Repeating the above code for all of the webpages would be tiresome, instead we can automate this process using a for-loop.

Notice that when you click on 'page 2' of the reviews the url of the webpage changes to 'https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk?page=2'. The '?page=2' command changes with each subsequent page of reviews (?page=3, ?page=4 etc.). We can use a function to generate a list of urls for all of the pages we want to scrape data from.

Note that scraping many web pages can not only be computationally intensive, but can also send large number of requests to a server. When creating these requests it is important to behave ethically to avoid over burdening a server with requests. As a result, we will only collect reviews from the first 5 pages.

```
urls <- sprintf("https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk?page=%d", 1:5)

urls
```

```
## [1] "https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk?page=1"
## [2] "https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk?page=2"
## [3] "https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk?page=3"
## [4] "https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk?page=4"
## [5] "https://uk.trustpilot.com/review/www.oakfurnitureland.co.uk?page=5"
```

Notice that when using sprintf() the %d is replaced by the range of values specified by the second parameter of the function (in this case the numbers 1 to 5).

Use a for loop to iterate through each webpage and collect the reviews from each page. Before calling the for loop, we need to create an empty dataframe in which we will add the data once collected.

```
AllRevs <- data.frame() #Creates an empty dataframe to store data

for (i in urls){
  page <- read_html(i)

  Review <- page %>%
    html_nodes(".review-content__text") %>%
    html_text(trim = TRUE)

  temp <- as.data.frame(Review)
  AllRevs <- rbind(AllRevs, temp)
}

AllRevs
```

Notice the addition of the last two lines of code

```
  temp <- as.data.frame(Review)
  AllRevs <- rbind(AllRevs, temp)
```

The first line creates a temporary dataframe of the reviews on that webpage (as before for the 1 webpage example), the second line uses the rbind function to bind the rows of data for each page together into a single dataframe. This function overwrites itself, therefore as the for loop progresses through each iteration, the data collected is appended to the AllRevs object.

```
nrow(AllRevs)
```

We can count the number of rows in the AllRevs object to see how many reviews are collected. The webscraper collected 92 reviews from the 5 pages.

What if I want to collect multiple elements from each post?

Additional elements can be specified using the SelectorGadget and specifiying the name for the new object. We will then read the pages using a for loop and pass them to the map_df to run the webscraper functions. Install and load the dynutils package.

```
install.packages("dynutils")
install.packages("purr")
```

```
library(dynutils)
library(purrr)
```

```
##
## Attaching package: 'purrr'
```

```
## The following object is masked from 'package:rvest':
##
##     pluck
```

```
Reviews <- data.frame() #Empty data frame

for (i in urls){
  page <- read_html(i)

  Scrape <- page %>% html_nodes(".review") %>%
    map_df(~list(title = html_nodes(.x, ".link--dark") %>%
                   html_text() %>%
                   {if(length(.) == 0) NA else .}, #Returns NA for missing data

                 review = html_nodes(.x, ".review-content__text") %>%
                   html_text(trim = TRUE) %>%
                   {if(length(.) == 0) NA else .}))

  temp <- data.frame(Scrape)
  Reviews <- rbind(temp, Reviews)
}
```