

# Lab Assignment 4 - Web Scraping

## Instructions

Please complete the exercises below. Submit your completed assignment as a PDF, HTML or Word document outputted by knitr or compiled manually showing both the code and output. (It should be in a similar format to this document).

*Note that in this document code blocks are shown with a grey background and output from running the code blocks is displayed with ## preceding the output.*

There may be some packages used in this assignment that you have not yet installed. In many cases the instructions are to “modify” the code provided, and it is implied in all cases that you should make sure the modified code successfully runs on your computer.

---

In this assignment you will implement web scraping. Forums are good candidates for web scraping. In the first part of this assignment, we’ll work through web scraping threads on a forum about depression.

Take a look at the website we will scrape:



<https://www.mentalhealthforum.net/forum/forums/depression-forum.366/page-1>

Note what happens when you change “page-1” to “page-2” in the URL (you get another set of results). That means we can iterate over different URLs (with different page numbers) to get a lot of data.

The information shown on this site could potentially serve as a useful data source. Before we can devise the code to scrape this site, we need to get an idea of the underlying HTML.

**1. Look at the HTML that underlies each forum thread box. Identify the name of the class that the the first forum thread box belongs to.**

Below shows an example of what I mean by the “forum thread box”:

	<b>Eating when depressed</b> Tawny · Wednesday at 3:56 PM 2	Replies: 24 Views: 124	6 minutes ago Tawny 
---	--	---------------------------	--

Hint: to see the underlying HTML in the Google Chrome browser, just right-click on the element of interest and click “Inspect”. If you are not using Chrome, see [here](#) for information on how it works in other browsers if you cannot find it.

Now let's start assembling the code to scrape the thread boxes. We'll use the library *rvest*. First we construct the url:

```
library(rvest)

page <- 1 #we'll use this later to update the page number

url <- paste0("https://www.mentalhealthforum.net/forum/forums/depression-forum.366/page-",
              page)
url

## [1] "https://www.mentalhealthforum.net/forum/forums/depression-forum.366/page-1"
```

That looks good. You can always try it in your browser to check if it's functional. Next, let's make the request to the server to retrieve the HTML:

```
#request and load in the html document:
html <- read_html(url)
```

Now we have the same HTML loaded in R that the browser would receive. Our next step is to extract the information we want from the HTML. Let's start by extracting the title of the first thread:

```
#extract the threads from the html using html_nodes() by identifying the class:
threads <- html %>% html_nodes(".structItem")
threads[1]

## {xml_node} (1)}
## [1] <div class="structItem structItem--thread js-inlineModContainer js-thread ...

#now take the first thread (threads[1]) and extract the title text
##using the class of the title element:
title <- threads[1] %>% html_nodes(".structItem-title") %>% html_text()
title

## [1] "\n\t\t\t\t\t\n\t\t\t\t\t\n\t\t\t\t\tAnxiety and Depression: Which one do I have?\n\t\t\t\t\t"

#the title's are a little messy with hidden characters so let's clean them:
cleanFun <- function(htmlString) {return(gsub("\t\n", "", htmlString))}
#It's okay if you don't fully understand gsub() right now, it's a complicated function
cleanFun(title)#show the cleaned title

## [1] "Anxiety and Depression: Which one do I have?"
```

Okay, that is a good start. We successfully extracted the first title. Before we move on, let me point out where you can find in the HTML that “structItem-title” is the class for the title element. Look at the HTML shown directly below, this is the HTML for a title element. (You can find it using the Inspect feature of your web browser). You can see the class identified in the first line:

```
<div class="structItem-title">
  <a href="/forum/threads/eating-when-depressed.321117/" class data-tp-primary="on" data-xf-init="preview-tooltip"
    data-preview-url="/forum/threads/eating-when-depressed.321117/preview" id="js-XFUniqueId8">Eating when depressed
  </a>
</div>
```

Below is the R code to extract other elements from the thread box. I just use the variable *i* here so it will be easier to incorporate this code into a loop later. Try to familiarize yourself with what each line is doing, and inspect the HTML using your web browser to identify how the code is extracting each piece of content.

```
i <- 2

title <- threads[i] %>% html_nodes(".structItem-title") %>% html_text()
title <- cleanFun(title)
title

## [1] "So stressed and worried"

replies <- threads[i] %>% html_nodes(".pairs")
replies <- replies[[1]] %>% html_nodes("dd") %>% html_text()
replies

## [1] "7"

views <- threads[i] %>% html_nodes(".pairs")
views <- views[[2]] %>% html_nodes("dd") %>% html_text() #we access the second element here
##because there are two elements with the class "pairs" and views is second
views

## [1] "70"

#for timestamp we extract the "datetime" attribute rather than the text
timestamp <- threads[i] %>% html_nodes("time") %>% html_attr("datetime")
timestamp <- timestamp[1]
timestamp

## [1] "2020-10-01T19:52:45+0100"

new_row <- data.frame(title, replies, views, timestamp) #combine in a single-row data frame
new_row

##               title replies views               timestamp
## 1 So stressed and worried      7    70 2020-10-01T19:52:45+0100
```

**2. Take the code directly above and put it into a *for loop*. Make it loop over all threads that are on the first page and store the information into a new dataframe called *all\_threads*.**

This will be very similar to what you saw in the last assignment. Don't loop over pages yet, we'll do that later. *Hints:* your loop should change the value of the variable *i* on each iteration. You'll also need to include an `rbind()`. Don't forget to add `Sys.sleep(2)`, as we do not want to query a website too quickly (just like with APIs).

Now you should have some data that look like this:

	title	replies	views	timestamp
1	Anxiety and Depression: Which one do I have?	72	61K	2012-01-04T01:18:39+0000
2	So stressed and worried	7	70	2020-10-01T19:52:45+0100
3	A topic for people who deal with loneliness	21	351	2020-08-14T08:50:06+0100
4	Eating when depressed	25	150	2020-09-30T15:56:04+0100
5	tired	4	49	2020-09-30T20:06:38+0100

**3. Modify the code you created so that there is an additional higher-level loop added that iterates over the first 50 pages.**

*Hints:* You should change the *page* variable each time in this new loop, and embed the loop you created just before within in. Remember to think about where the blank data frame *all\_threads* should be created. You should have about 1,250 rows when you're done.

---

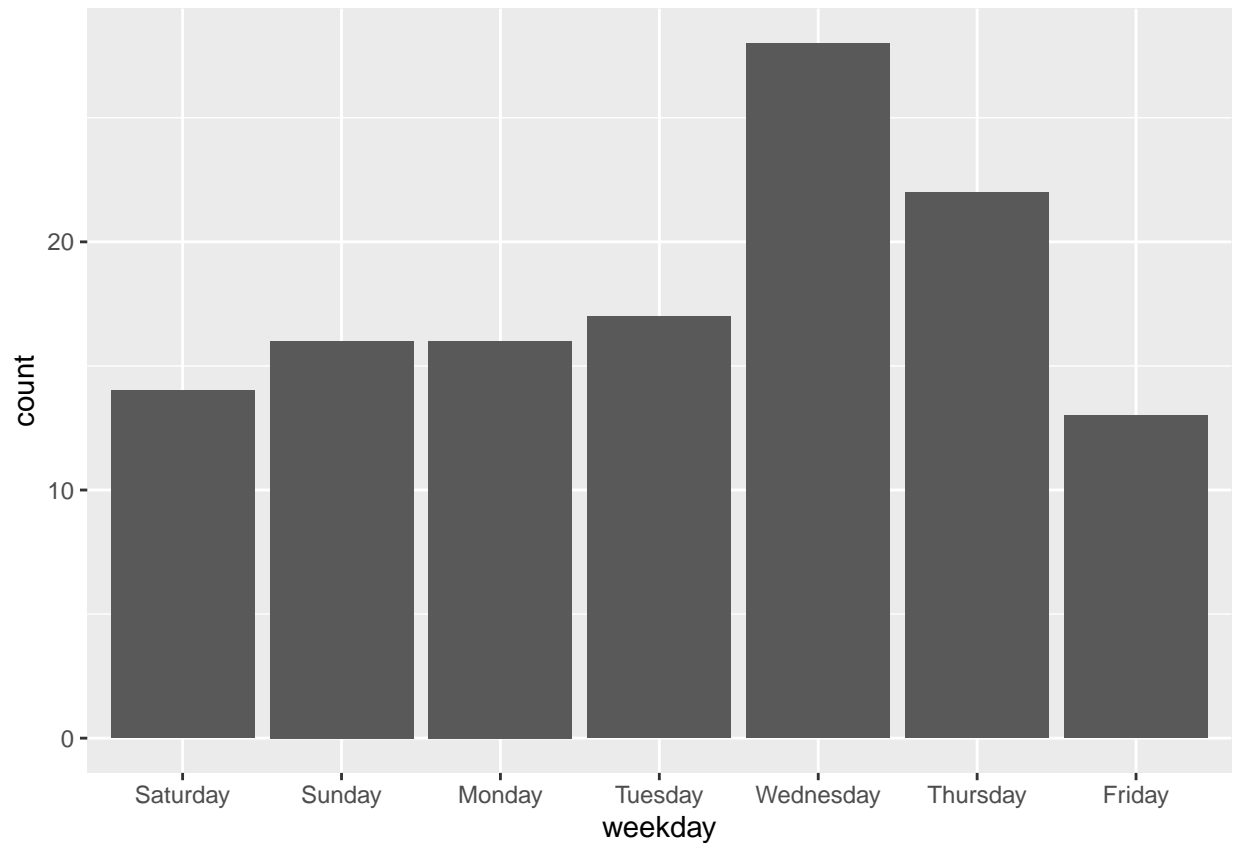
We now have a lot of data. Let's clean it up a bit and then explore it some. The code below cleans up the data frame by specifying the column classes (R class, not HTML class). It also creates a simple date variable from the timestamp, and then a day-of-the-week variable.

```
all_threads$views <- as.numeric(all_threads$views)
#note we just are ignoring that some have a "K" for now in the views numbers
all_threads$replies <- as.numeric(all_threads$replies)
all_threads$date <- as.Date(all_threads$timestamp)
all_threads$title <- as.character(all_threads$title)

all_threads$weekday <- weekdays(all_threads$date)
#indicate the correct order for the days of the week:
all_threads$weekday <- factor(all_threads$weekday,
                             levels=c("Saturday", "Sunday", "Monday", "Tuesday",
                                       "Wednesday", "Thursday", "Friday"))
```

Let's take a quick look at how the number of new threads varies by the day of the week:

```
library(ggplot2)
ggplot(all_threads, aes(weekday)) + geom_bar(stat="count")
```



It looks like there may be a “middle-of-the-week” increase in new threads, but we would have to do inferential statistics and/or collect more data to assess the robustness of this pattern.

**4. Draw a histogram of the counts of replies to each thread.**

---

**5. What does this histogram tell you about the distribution of reply counts?**

---

Later in the course, we'll go into depth on sentiment analysis. For right now, let's just run a quick and simple sentiment function in R. The function `analyzeSentiment()` runs several dictionary methods on the text that is inputted to it to perform simple sentiment analyses.

```
library(SentimentAnalysis)
sentiment <- analyzeSentiment(all_threads$title)
names(sentiment) #show the column names outputted by the sentiment function

## [1] "WordCount"          "SentimentGI"        "NegativityGI"
## [4] "PositivityGI"        "SentimentHE"        "NegativityHE"
## [7] "PositivityHE"        "SentimentLM"        "NegativityLM"
## [10] "PositivityLM"        "RatioUncertaintyLM" "SentimentQDAP"
## [13] "NegativityQDAP"     "PositivityQDAP"
```

Let's just look at the mean sentiment detected by each of the dictionary methods. Our textual data are thread titles from a depression forum, so we would expect to see all negative sentiment estimates on average.

```
mean(sentiment$SentimentGI)

## [1] -0.1702822

mean(sentiment$SentimentHE)

## [1] -0.06735009

mean(sentiment$SentimentLM)

## [1] -0.1492945

mean(sentiment$SentimentQDAP)

## [1] -0.1498394
```

The depression forum is only one of many mental health forums on this website. Take a look [here](#) at the other forums that exist there.

**6. Pick a different forum (e.g. the Anxiety Forum, Dementia Forum, etc.) from this website to scrape in the same way we scraped the depression forum. Scrape *all* of the pages that exist for it.**

**7. Find a different website that you think would be a good candidate for web scraping. Provide the link to the website.**

It should be one that has data on it you could imagine extracting into rows of data in a table. Take a look at the underlying HTML to get a sense of how you might go about extracting the meaningful content with rvest. Also consider if there is a way to page through the results by systematically changing the URL.

---

**8. Take a look on the website you found for its Terms of Use or the equivalent. Do they specify that users cannot scrape the website's data?**

Often websites do not have a statement “outlawing” web scraping, but sometimes they do. For example, in the Terms of Use for Craigslist ([here](#)), they specify: “You agree not to copy/collect CL content via robots, spiders, scripts, scrapers, crawlers, or any automated or manual equivalent (e.g., by hand).” When a website indicates that its data should not be scraped, we should abide by these terms.

---

**9. Describe the similarities and differences between collecting data through web scraping and collecting data through APIs.**

---