Lab 8: Web scraping

Solutions

September 28, 2021

About this lab

Next week we'll be working with Emily Dickinson's poetry to introduce text analysis. In order to bring her poetry into R, we can webscrape the poems from Wikipedia. (Perhaps not the most accurate source for her poetry, but a good exercise in web scraping, `for` loops, and algorithmic thinking!)

There is a separate Wikipedia page for each of Dickinson's poems—she wrote over 1,500! How can we efficiently search across all these pages to get the text from each poem into R?

We'll first web scrape a Wikipedia page that contains a *table* listing links to all her poems. Then we'll use data from that table to loop through each of the linked pages scrape the *text* of the poems.

The packages for this lab include **tidyverse**, **rvest** (for general scraping), **robotstxt** (checking `paths_allowed()`), and **purrr** (to `pluck()` a single element from a list). The **polite** package is also recommended "to promote responsible web etiquette" (but I don't quite know how to use it yet!).

Review

In the class example, we walked through the following steps to scrape and print Dickinson's *September's Baccalaureate*:

```
# 1. Identify page where poem is listed
sep_bac_url <- "https://en.wikisource.org/wiki/September%27s_Baccalaureate"

# 2. Confirm bots are allowed to access the page
paths_allowed(sep_bac_url)

# 3. Get poem text
sep_bac_text <- sep_bac_url %>%
  read_html() %>%
  # a. Get list of "div p" elements on the page
  html_elements("div p") %>%
  # b. `Pluck` poem from list and grab text
  pluck(1) %>%
  html_text()

# 4. Print poem
cat(sep_bac_text)
```

For this lab, we'll need to modify and repeat this process many times to scrape *all* of Emily Dickinson's poems available on Wikipedia!

Part 1 **Algorithmic thinking** This Wikipedia page contains a table listing Emily Dickinson's poems—she wrote over 1,500! There is a separate Wikipedia page for (almost) every one of Dickinson's poems. Our goal is ultimately to create a dataframe with the title and text of every poem linked in the table. How can we efficiently search across all these pages to get the text from each poem into R? Let's walk through the process.

1.1 Take 10 minutes or so to read Web Scraping 101, courtesy of rvest.tidyverse.org, focusing particularly on the section on Extracting Data (text, attributes, and tables). Feel free to take some notes below if desired.

1.2 Revisit the List of Emily Dickinson's poems and recall how you scraped a single poem already (*September's Baccalaureate* above). Broadly speaking, what are the things you need to do to get to our final goal? Don't worry about the specific details yet—ignore the functions and arguments you'll need in R and don't worry about the particular order of steps for now.

1. Grab the table with the list of Emily Dickinson poems.
2. Create a list of links for each poem.
3. Iterate through each of the links and scrape the poem from each page.
4. Combine the poem text with the poem titles in a dataframe.

1.3 One of your steps likely involved scraping the table that lists Emily Dickinson's poems. Use the code chunk below to do that, creating a dataframe called `poem_table`. Be sure to check that bots are allowed first, and make sure the column names of your dataframe are user-friendly or "clean" (*hint*: remember our friend `janitor::clean_names()` and/or you can rename the necessary columns yourself).

```
url_poem_list <- "https://en.wikipedia.org/wiki/List_of_Emily_Dickinson_poems"
paths_allowed(url_poem_list)
```

```
     [1] TRUE
```

```
poem_table <- url_poem_list %>%
  read_html() %>%
  html_element("#mw-content-text > div.mw-parser-output > table") %>%
  html_table() %>%
  # Clean up variable names
  janitor::clean_names() %>%
  select(title = first_line_often_used_as_title)
```

1.4 Another important step is getting a list of all the URLs so we can iterate through each linked page to scrape the poem. There are couple approaches we might take to do this: piecing the URLs together ourselves, or scraping the URLs from the table. Click through five or so poems in the table to see the pages that contain the text of each poem. What do you notice about the URLs for the pages that contain the poem text? Can you identify a pattern in the URLs? Are there links that fall outside that patter? How might we piece together the URLs ourselves? What did you learn from Web Scraping 101 that would allow us to scrape the URLs from the table directly?

Each of the URLs start with "https://en.wikisource.org/wiki/" and then has the name of the poem (or, first line of the poem) with words separated by underscores. The words seem to follow the same case (upper or lower) as displayed in the table. Sometimes the title of the poem in the table doesn't end with an em-dash, but the URL includes an em-dash (there doesn't seem to be a pattern here.) Given the unpredictable nature of the pattern breaking, using `html_attr()` to select the links directly may be more useful.

1.5 Choose one of the two approaches from Part 1.4 to create a data frame that contains all the links we need to iterate through and the link titles. Verify you only have full links (i.e., starting with "https:"), then join that dataframe with your `poem_tables` dataframe.

```
# Grab text of each URL (i.e., poem title)
url_text <- url_poem_list %>%
  read_html() %>%
  html_elements("#mw-content-text > div.mw-parser-output > table > tbody > tr > td > a") %>%
  html_text()

# Grab actual URL href
url_href <- url_poem_list %>%
  read_html() %>%
  html_elements("#mw-content-text > div.mw-parser-output > table > tbody > tr > td > a") %>%
  html_attr("href")

# Combine text and link into a dataframe
url_table <- tibble(title = url_text, href = url_href) %>%
  # Filter out non-https links
  filter(grepl("https:", url_href))

# Join with original poem table
poem_table <- poem_table %>%
  left_join(url_table)
```

Part 2 **Iterate to scrape all the poems!** We have our table of poems, we have the corresponding links available, and we know how to scrape a poem from a single webpage (see example above where we scraped *September's Baccalaureate*). The next major task is figuring out how to iterate through all the poems! Remember, the final product should be a dataframe with at least two columns: the title of the poem and the text of the poem. To do this, we will pre-allocate space in our data frame (a new column called `text`), and use a `for()` loop to iterate through the URLs and fill the `text` column with the poem text as we scrape each poem. When everything finally works in your code, replace `poem_table[seq_len(n_links), ]` with `poem_table` to run the code for all poems.

NOTES:

You should develop and test your code on a small subset of pages (e.g., one URL, 5 URLs, 20 URLs, 50 URLs) and check for errors or oddities each time before scaling up. Given the number of poems and the delay time between hits to the site, it will take a (very) long time to run the code on the full set of ~1800 links, so don't do that until you are absolutely sure your code is producing the output you expect.

Some links will not work (links don't exist or links exist but poem has been removed). Typically, when an error is encountered, the code will break and quit without producing output—not ideal! Instead, we will use `tryCatch()` to produce alternative output when a link doesn't work. This will allow us to continue on to the next link without breaking the code. For more on `tryCatch()`, check out this chapter on Handling conditions.

Some potentially problematic or wonky poems that may require code adjustment (or ignore if out of time).

- *A narrow Fellow in the Grass*
- *Alter! When the Hills do*
- *If I can stop one Heart from breaking*
- *I never lost as much but twice,*
- *The earth has many keys,*
- *The Himmaleh was known to stoop*

When you are sure everything finally works, replace `poem_table[seq_len(n_links), ]` with `poem_table` to be able to run the code for all poems (but don't run it yet!).

```
# Identify number of iterations (start with 1, 5, 20, 50, etc.)
n_links <- nrow(poem_table)

# Pre-allocate space for poem text
poem_tibble <- poem_table %>%
  mutate(text = "")

# Iterate through links to grab text
for(i in seq_len(n_links)){

  # Identify url
  link <- poem_table$href[i]

  # Fix some wonky links
  ## Redirect to link with poem
  if(!is.na(link) &
     link == "https://en.wikisource.org/wiki/I_never_lost_as_much_but_twice"){
    link <- "https://en.wikisource.org/wiki/Poems_(Dickinson)/I_never_lost_as_much_but_twice,"
  }
```

```
  ## Don't bother scraping; this is a subset of another poem
  if(!is.na(link) &
     link == "https://en.wikisource.org/wiki/The_earth_has_many_keys,"){
    link <- NA # poem already exists in Further in Summer than the birds Version 1
  }

  # Scrape poem text, using tryCatch() to handle errors
  poem_tibble$text[i] <- tryCatch(

    # Return "Missing" instead of poem text when error is thrown
    error = function(cnd) {
      return("Missing")
    },

    # Try to scrape poem text

    if(!is.na(link) &
       link == "https://en.wikisource.org/wiki/The_Himmaleh_was_known_to_stoop"){

      # Scrape div p text
      link %>%
        read_html() %>%
        html_elements("div p") %>%
        pluck(1) %>%
        html_text() %>%
        return()

    } else {

      link %>%
        read_html() %>%
        html_elements(".poem, .wst-block-center, div p") %>%
        html_text() %>%
        str_remove(fixed(".mw-parser-output .dropinitial{float:left;text-indent:0}.mw-parser-output .dr
        return()
    }
  )
}
```

Part 3 **Workflow** Yet again we've been working in an Rmd file to work through questions and answers. To practice the appropriate workflow, place your scraping code above in an R script called "scrape-poems.R". Be sure to make the file reproducible, including loading any necessary packages at the top of the script. Use `write_csv()` to output a csv file called "dickinson-poems.txt" (note the file extension!).

See associated files.