Problem Set #2

Todd Faulkenberry

9/16/2018

Problem 1

I applied a handful of concepts from Unit 5 in this problem set:

- Debugging - I used traceback() during this problem set to figure out where my functions were messing up when I was constructing them. It was particularly useful for create_scholar_df(), where it helped me determine that I was giving the function the wrong type of input. In the future, I plan to use browser(), as it seems useful for more complicated functions but wasn't quite as useful for the straighforward problems I had here.
- Syntax - I followed Hadley Wickham's style guide, which I prefer to Google's because I find it more readable. Aspects I used include underscores in function/object names, spacing around operators, respecting space for curly braces in functions, and using proper assignment (<-).
- Style - I followed advice from the coding section, such as including the description of each function, making them modular, and defining variables instead of hard coding numbers.

Problem 2

Part A

Binary formats, as the name implies, are files written in a series of 0s and 1s. Examples of binary formats include zip files and .rda. As opposed to plain text files like .csv, they are not human readable. So they aren't editable by hand by humans – you need to load them into a program that can interpret it – but they can be more efficient for storing because they do not need to include things to make the text readable. That explains the file difference here - the .rda file is much smaller because it does not have to include commas to separate values, while the .csv file does. The numbers corroborate this. Because a 0 or a 1 each take up one byte (aka 8 bits), 8000000 / 8 = 1e07, which is the exact number of characters that we put into A initially without adding anything else.

CSV and other plain text formats rely on ASCII, a set of 128 characters that basically comprise what you see on a US keyboard. Like 0 and 1, which are a part of ASCII, characters of this set take up one byte (equivalent to 8 bits) in storage. Given that file.size() returns the size of the file in bits, and given the additional characters inherent in .csv files, we'd expect this .csv file to have 16,736,287 (133890295 / 8) characters, with one character taking up each byte.

Part B

In this example, we have moved all observations from 100 columns to one to avoid saving commas. Even though we have removed all the commas in this example, we have basically replaced them with new line characters that tell the file reader that each observation in the data is on a different row. Because we've just traded out one type of character for another, we haven't reduced the file size at all.

Part C

Between first comparison and second comparison, the difference is that the second comparison of read.csv specifies colClasses. Specifying colClasses means that R doesn't need to spend time figuring out what class each type of variable and it can just read them in with the format already set. The difference between the second and third comparison is that load is a function made to to specifically work on binary formats like .rda, while scan is loading in a plaintext file that has more characters and thus will take longer. Additionally, because scan's default delimiter is white space, the fact that it must search for commas in this scenario likely adds some amount of time.

Part D

tmp2.rda is a file created from a function that calls a random number from a normal distribution of mean = 1 and sd = 0, and replicates it 10,000,000. This means tmp2.rda is the same number printed many times over. In comparison, tmp.rda was created from a function that called the same normal distribution 10,000,000 times, leading to 10,000,000 different numbers. The fact that tmp.rda had to store so many different unique numbers explains the difference in file size.

Problem 3

Part A

Here, I wrote a function that takes in the name of a researcher and returns two character strings: Their Google Scholar webpage and their Google Scholar ID.

```r
get_scholar_link <- function(name) {
  # Purpose: This function helps you find the Google Scholar webpage and
  # Google Scholar ID for any given researcher that has a profile in the
  # database.
  #
  # Input: This function takes in one function, a character string of a
  # researcher's name.
  #
  # Outputs: This function has two outputs: A text string of the researcher's
  # Google Scholar webpage and a text string of their Google Scholar ID.
  assert_that(is.character(name))


  first_name <- tolower(strsplit(name, ' ')[[1]][1])
  last_name <- tolower(strsplit(name, ' ')[[1]][2])

  scholar_html <- paste0('https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=',
                         first_name, '+', last_name, '&oq=had')

  page <- read_html(scholar_html)

  links <- page %>%
    html_nodes('a') %>%
    html_attr('href')

  id_page <- links[41]
  web_page <- paste0('https://scholar.google.com', id_page)

  scholar_id <- str_extract(id_page, '(?<==)[^=]*[^=]')

  result <- c(web_page, scholar_id)
  return(result)
}

hadley_link <- get_scholar_link('hadley wickham')

hadley_link
```

```
## [1] "https://scholar.google.com/citations?user=YA43PbsAAAAJ&hl=en&oe=ASCII&oi=ao"
## [2] "YA43PbsAAAAJ&hl"
```

```
trevor_link <- get_scholar_link('trevor hastie')

trevor_link
```

```
## [1] "https://scholar.google.com/citations?user=tQVe-fAAAAAJ&hl=en&oe=ASCII&oi=ao"
## [2] "tQVe-fAAAAAJ&hl"
```

This function does work and successfully pulled the information for me of both Trevor Hastie and Hadley Wickham, but could be improved in a few ways. Firstly, you currently have to put the inputs in explicitly as characters(e.g. 'Hadley Wickham'). Secondly, it returns a string for the webpage instead of an xml document.

Part B

Though the above function returns a string, the below function handles the input by immediately reading that string as a html, allowing the webpage to be scraped and the dataframe to be created. This function worked for both Hadley Wickham and Trevor Hastie.

```
create_scholar_df <- function(name) {
  # Purpose: This function creates a dataframe of a Google Scholar's
  # information from their Google Scholar page. This df has five columns:
  # Article, Authors, Journal, Year, and Citations.
  #
  # Input: This function takes in one function, a character string of a
  # researcher's name.
  #
  # Output: This function has one output, the dataframe described in the
  # Purpose section.
  assert_that(is.character(name))


  scholar_html <- read_html(get_scholar_link(name)[1])

  titles <- scholar_html %>%
    html_nodes('.gsc_a_at') %>%
    html_text()

  authors_and_pubs <- scholar_html %>%
    html_nodes('.gs_gray') %>%
    html_text()

  authors <- authors_and_pubs[seq(1,length(authors_and_pubs),2)]
  pubs <- authors_and_pubs[seq(2,length(authors_and_pubs),2)]

  cites_and_dates <- scholar_html %>%
    html_nodes('.gs_ibl') %>%
    html_text()

  cites <- cites_and_dates[seq(7,length(cites_and_dates),2)]
  dates <- cites_and_dates[seq(8,length(cites_and_dates),2)]

  scholar_df <- data.frame('Article' = titles,
                          'Authors' = authors,
                          'Journal' = pubs,
                          'Year' = dates,
                          'Citations' = cites)

  return(scholar_df)

}



hadley_df <- create_scholar_df('hadley wickham')

head(hadley_df)
```

```
##                                               Article
## 1        ggplot2: elegant graphics for data analysis
## 2 The split-apply-combine strategy for data analysis
## 3            Reshaping data with the reshape package
## 4         ggmap: Spatial Visualization with ggplot2.
## 5                                          Tidy data
## 6           Dates and Times Made Easy with lubridate
##                    Authors
## 1                H Wickham
## 2                H Wickham
## 3                H Wickham
## 4     D Kahle, H Wickham
## 5                H Wickham
## 6 G Grolemund, H Wickham
##                                            Journal Year Citations
## 1                                    Springer, 2009 2009      9827
## 2  Journal of Statistical Software 40 (1), 1-29, 2011 2011       994
## 3 Journal of Statistical Software 21 (12), 1-20, 2007 2007       808
## 4                           R Journal 5 (1), 2013 2013       437
## 5       Journal of Statistical Software 59 (10), 2014 2014       297
## 6      Journal of Statistical Software 40, 1-25, 2011 2011       181
```

```
trevor_df <- create_scholar_df('trevor hastie')

head(trevor_df)
```

```
##
## 1                                                                                 Unsupervis
## 2                                                                           Generalized addi
## 3         Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical i
## 4                                                        Regularization and variable selection via the
## 5                                                                                Least angle
## 6 Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by t
##                                                                              Authors
## 1                                          T Hastie, R Tibshirani, J Friedman
## 2                                                                    TJ Hastie
## 3 T Sørlie, CM Perou, R Tibshirani, T Aas, S Geisler, H Johnsen, T Hastie, ...
## 4                                                              H Zou, T Hastie
## 5                                B Efron, T Hastie, I Johnstone, R Tibshirani
## 6                                          J Friedman, T Hastie, R Tibshirani
##                                                                                     Journal
## 1                                The elements of statistical learning, 485-585, 2009
## 2                                             Statistical models in S, 249-307, 2017
## 3          Proceedings of the National Academy of Sciences 98 (19), 10869-10874, 2001
## 4 Journal of the Royal Statistical Society: Series B (Statistical Methodology ..., 2005
## 5                                    The Annals of statistics 32 (2), 407-499, 2004
## 6                                    The annals of statistics 28 (2), 337-407, 2000
##   Year Citations
## 1 2009     40041
## 2 2017     15769
## 3 2001     11890
## 4 2005      8007
## 5 2004      7843
## 6 2000      6260
```

Part C

I added assertthat tests to both functions to make sure the input is a character string. I had trouble implementing a test function and ran out of time, so I ended up not using one. As discussed below in Problem 4, a good test to add if we were fleshing this out a little more would be to ensure that the returned value doesn't contain an '@' so that we know we aren't violating Google Scholar's robots.txt file.

Part D

Didn't do extra credit in interest of time!

Problem 4

In this situation where we're webscraping two specific authors (Wickham and Hastie), our scraper is acting ethically. We know this because of the robots.txt file, which has three lines of interest to us:

- Allow: /citations?user=
- Disallow: /citations?user=*%40
- Disallow: /citations?user=*@

What the robots.txt is telling us is that, yes, we can scrape the Google Scholar page generally, as long as the web address does not contain an '@' (%40 is the HTML encoding reference for @.) Neither Wickham or Hastie's Scholar page includes an '@', so we are in the the clear right now. Certainly, however, there exists Google Scholar pages that have this character. Moreoever, the above functions don't protect for this in any way, so we could accidentally but very easily violate Google Scholar's webscraping policy if we searched for the wrong scholar. One action we'd need to take before using the functions created above in any systemic or public way would be to make sure that they don't return if we call a scholar whose URL contains an '@'.