Problem Set #3 - Stat 243

Todd Faulkenberry

9/24/2018

```
suppressMessages(library(tidyverse))
library(XML)
library(stringr)
library(assertthat)
```

```
##
## Attaching package: 'assertthat'
```

```
## The following object is masked from 'package:tibble':
##
##     has_name
```

Question 1

Part A

I read the article "Best Practices of Scientific Computing." I find most of these practices compelling. I'm currently taking a computer science where we've already applied many of these principles (e.g. DRY, automation of repetitive tasks, plan for mistakes.) It feels to me like these principles aretuaght as canonical for computer scientists, but optional for statisticians and other people that use computers. It raises two questions for me: 1) Should a comprehensive comp sci class (e.g. CS61a) be required for any social science discipline that uses computers; and 2) What are implications for these principles on the base R vs Tidyverse debate. Base R feels very esoteric and out of line with some of these principles (especially creating code for humans), and I don't feel like you can follow these problems perfectly while simultaneously advocating for base R.

Part D

Strengths: I think the sheer amount of materials they published for public consumption is a huge strength. Even if you don't agree with their methods or think they wrote inefficient code here and there, the fact that their even though process is outlined in detail shows a transparency and forthrightness that I think is a cornerstone of intellectual integrity. This is especially true in today's climate given the current dialogue around reproducibility the difficulties that presents. Every study should be atrongly encouraged to provide the materials and thinking behind the results, so that an engaged audience can have a clearer understanding of how the authors arrived and their conclusions. That audience can then determine what they think about the conlusions instead of having it presumed true and spoon fed to them.

Weaknesses: Though I applaud the materials provided, as we discussed in section, I do think the authors could have made these materials even more easily reproducible with a few simple steps. Firstly, I would do away with the hardcoding the filepaths that they outline at the beginning of the readme. I would personally do this by making a R Project, where the the root directory of each file would be the .rproj created. That way, anyone could download the entire github repository with the .rproj in there and immediately start reproducing without do any additional coding. I also think abstracting the hardcoding more by using arguments like file.path() would be useful because then you would be able to reproduce the study easily in any given type of computer as long as you had the same file names. Another minor thing I would change would be to give the files more intuitive names so that the order of them was more sensical. Even if this were as simple as adding numbers to the beginning of each file name, it would help the audience understand the workflow more quickly.

Question 2

To begin the problem, I used the R code provided by Chris to get the transcripts downloaded and into R (thank you, Chris!) In below Parts, I followed the parts as they were laid out in the problem set.

```
moderators <- c("LEHRER", "LEHRER", "LEHRER", "MODERATOR", "LEHRER", "HOLT")

candidates <- list(c(Dem = "CLINTON", Rep = "TRUMP"),
                   c(Dem = "OBAMA", Rep = "ROMNEY"),
                   c(Dem = "OBAMA", Rep = "MCCAIN"),
                   c(Dem = "KERRY", Rep = "BUSH"),
                   c(Dem = "GORE", Rep = "BUSH"),
                   c(Dem = "CLINTON", Rep = "DOLE"))

url <- "http://www.debates.org/index.php?page=debate-transcripts"

yrs <- seq(1996, 2012, by = 4)
type <- 'first'
main <- htmlParse(url)
listOfANodes <- getNodeSet(main, "//a[@href]")
labs <- sapply(listOfANodes, xmlValue)
inds_first <- which(str_detect(labs, "The First"))
## debates only from the specified years
inds_within <- which(str_extract(labs[inds_first], "\\d{4}")
                     %in% as.character(yrs))
inds <- inds_first[inds_within]
## add first 2016 debate, which is only in the sidebar
ind_2016 <- which(str_detect(labs, "September 26, 2016"))
inds <- c(ind_2016, inds)
debate_urls <- sapply(listOfANodes, xmlGetAttr, "href")[inds]

n <- length(debate_urls)

assert_that(n == length(yrs)+1)
```

```
## [1] TRUE
```

```
## @knitr extract

debates_html <- sapply(debate_urls, htmlParse)

get_content <- function(html) {
    # get core content containing debate text
    contentNode <- getNodeSet(html, "//div[@id = 'content-sm']")
    if(length(contentNode) > 1)
        stop("Check why there are multiple chunks of content.")
    text <- xmlValue(contentNode[[1]])
    # sanity check:
    print(xmlValue(getNodeSet(contentNode[[1]], "//h1")[[1]]))
    return(text)
}

debates_body <- sapply(debates_html, get_content)
```

```
## [1] "September 26, 2016 Debate Transcript"
```

```
## [1] "October 3, 2012 Debate Transcript"
## [1] "September 26, 2008 Debate Transcript"
## [1] "September 30. 2004 Debate Transcript"
## [1] "October 3, 2000 Transcript"
## [1] "October 6, 1996 Debate Transcript"
```

```
## sanity check
print(substring(debates_body[5], 1, 1000))
```

```
##
## "\nOctober 3, 2000 Transcript\n\nOctober 3, 2000The First Gore-Bush Presidential DebateMODERATOR: Go
```

Part A

In the code I below, I define a function that uses str_split to grab chunks from each debate and uses str_match_all to grab the names of the speakers from each debate. I then use the tibble function to combine them into a tibble. I used lapply to put all of these tibbles into a list, ultimately leading to the list of tibbles entitled 'debate_list.' I then renamed the name attribute for each tibble to be the year of the debate, and then created a function that returned the number of chunks each person spoke at each debate.

```
# Function to create a list of dataframes where each dataframe contains
# a column identifying the speaker and the column with the chunk.
debates_df <- function(debate_text) {
  chunks <- unlist(str_split(debate_text, '[A-Z]{2,}:'))[-c(1:3)]
  speakers <- unlist(str_match_all(debate_text, '[A-Z]{2,}:'))[-c(1:2)]
  speakers <- gsub(':','', speakers)

  tibble(Speaker = speakers, Chunk = chunks)
}

# Applying dataframe function above to list to create list of six data frames
debate_list <- lapply(debates_body, debates_df)


# Fixing the name attribute
years <- c('2016', '2012', '2008', '2004', '2000','1996')
names(debate_list) <- years

test <- names(debate_list)


# Returning the count of each chunk for each debate
speaker_count <- function(data) {
  data %>%
    group_by(Speaker) %>%
    summarize(count = n())
}

speaker_count(debate_list$`2016`)
```

```
## # A tibble: 3 x 2
##   Speaker count
##   <chr>   <int>
## 1 CLINTON    87
## 2 HOLT       97
```

```
## 3 TRUMP     124
```

```
speaker_count(debate_list$`2012`)
```

```
## # A tibble: 3 x 2
##   Speaker count
##   <chr>   <int>
## 1 LEHRER     82
## 2 OBAMA      56
## 3 ROMNEY     71
```

```
speaker_count(debate_list$`2008`)
```

```
## # A tibble: 4 x 2
##   Speaker              count
##   <chr>                <int>
## 1 LEHRER                 125
## 2 MCCAIN                 126
## 3 MISSISSIPPISPEAKERS      1
## 4 OBAMA                  126
```

```
speaker_count(debate_list$`2004`)
```

```
## # A tibble: 3 x 2
##   Speaker count
##   <chr>   <int>
## 1 BUSH       41
## 2 KERRY      33
## 3 LEHRER     67
```

```
speaker_count(debate_list$`2000`)
```

```
## # A tibble: 3 x 2
##   Speaker   count
##   <chr>     <int>
## 1 BUSH         56
## 2 GORE         49
## 3 MODERATOR    59
```

```
speaker_count(debate_list$`1996`)
```

```
## # A tibble: 3 x 2
##   Speaker count
##   <chr>   <int>
## 1 CLINTON    44
## 2 DOLE       46
## 3 LEHRER     52
```

One problem I had throughout this problem set was iterating over the list of dataframes, e.g. with a function. When I tried to run for loops, I had trouble accessing the columns within the tibble after I called it from the list. I would appreciate any guidance on how I could've done this more effectively without calling the same function six times in a row.

Part B

Below is the code I used to produce sentence and word character vectors for each debate.

```
# Defining the two functions
vector_words <- function(vector) {
  str_split(vector, boundary('word'))[[2]]
}

vector_senteces <- function(vector) {
  str_split(vector, boundary('sentence'))[[2]]
}


# Getting word counts
words_2016 <- vector_words(debate_list$`2016`)
words_2012 <- vector_words(debate_list$`2012`)
words_2008 <- vector_words(debate_list$`2008`)
words_2004 <- vector_words(debate_list$`2004`)
words_2000 <- vector_words(debate_list$`2000`)
words_1996 <- vector_words(debate_list$`1996`)

# Getting sentence counts
sentences_2016 <- vector_senteces(debate_list$`2016`)
sentences_2012 <- vector_senteces(debate_list$`2012`)
sentences_2008 <- vector_senteces(debate_list$`2008`)
sentences_2004 <- vector_senteces(debate_list$`2004`)
sentences_2000 <- vector_senteces(debate_list$`2000`)
sentences_1996 <- vector_senteces(debate_list$`1996`)
```

Again, because of problems with iterating over the tibbles in the list of tibbles, I had to write over the function call six separate times for both grabbing words and sentences. Also, I ran out of time figuring how to define attributes to each candidate, so the word and sentence counts here are for the entire debate, not the debate by candidate.

Part C

Below, I create a list of both the sentence and word vectors above and then loop through each, printing both the sentence count and word count from the debates starting from 1996 and going to 2016. I then print the average length of words for each debate.

```
#Creating list to iterate for sentences
sentences <- list(sentences_1996, sentences_2000, sentences_2004, sentences_2008, sentences_2012, sente

#Creating list to iterate for words
words <- list(words_1996, words_2000, words_2004, words_2008, words_2012, words_2016)


# For loop for sentences
for (sentence_vector in sentences) {
  print(length(sentence_vector))
}
```

```
## [1] 1058
## [1] 1003
## [1] 1060
## [1] 1789
## [1] 801
```

```
## [1] 1181
```

```
# For loop for words
for (word_vector in words) {
  print(length(word_vector))
}
```

```
## [1] 16179
## [1] 16131
## [1] 14273
## [1] 31570
## [1] 16875
## [1] 16757
```

```
for (word_vector in words) {
  print(sum(str_length(word_vector)) / length(word_vector))
}
```

```
## [1] 4.384449
## [1] 4.344182
## [1] 4.473972
## [1] 4.484891
## [1] 4.361185
## [1] 4.340335
```

Even though this isn't split up by candidate, a few of the findings here were very interesting. For example, in 2008, the word and sentence counts are both much larger than any other years. The word count is twice as large as ny other year. This suggests either a longer debate, a different format from the other debates, or something wrong with the scraping of the data from this debate. Another interesting immediate trend is the word and sentence count from the 2012 debate. Though 2012 had the second largest amount of words for any debate, it had to the shortest number of sentences by almost 200 sentences. This suggests much longer answers in this debate, and corroborates given the style of the two candidates (the professorial Obama and the executive and former governor Romney.)

Part D

Below, I define a function word_checks and run that function across every word vector.

```
word_checks <- function(debate_text) {
  print(paste('I:', sum(str_count(debate_text, '^I$'))))
  print(paste('we:', sum(str_count(debate_text, '^we$'))))
  print(paste('America[n]:', sum(str_count(debate_text, 'America[n]?'))))
  print(paste('democra[cy][tic]:', sum(str_count(debate_text, 'democra[cy]?[tic]?'))))
  print(paste('republic:', sum(str_count(debate_text, '^republic$'))))
  print(paste('Democrat[ic]:', sum(str_count(debate_text, 'Democrat[ic]?'))))
  print(paste('Republican:', sum(str_count(debate_text, '^Republican$'))))
  print(paste('free[dom]:', sum(str_count(debate_text, 'free[dom]?'))))
  print(paste('war:', sum(str_count(debate_text, '^war$'))))
  print(paste('God:', sum(str_count(debate_text, 'God$'))))
  print(paste('God Bless:', sum(str_count(debate_text, 'God Bless'))))
  print(paste('Jesus:', sum(str_count(debate_text, 'Jesus'))))
  print(paste('Christ:', sum(str_count(debate_text, 'Christ'))))
  print(paste('Christian:', sum(str_count(debate_text, 'Christian'))))
}
```

```
word_checks(words_1996)
```

```
## [1] "I: 406"
## [1] "we: 163"
## [1] "America[n]: 84"
## [1] "democra[cy][tic]: 6"
## [1] "republic: 0"
## [1] "Democrat[ic]: 13"
## [1] "Republican: 19"
## [1] "free[dom]: 9"
## [1] "war: 4"
## [1] "God: 1"
## [1] "God Bless: 0"
## [1] "Jesus: 0"
## [1] "Christ: 0"
## [1] "Christian: 0"
```

```
word_checks(words_2000)
```

```
## [1] "I: 374"
## [1] "we: 131"
## [1] "America[n]: 42"
## [1] "democra[cy][tic]: 3"
## [1] "republic: 0"
## [1] "Democrat[ic]: 14"
## [1] "Republican: 2"
## [1] "free[dom]: 5"
## [1] "war: 7"
## [1] "God: 0"
## [1] "God Bless: 0"
## [1] "Jesus: 0"
## [1] "Christ: 0"
## [1] "Christian: 0"
```

```
word_checks(words_2004)
```

```
## [1] "I: 258"
## [1] "we: 183"
## [1] "America[n]: 73"
## [1] "democra[cy][tic]: 7"
## [1] "republic: 0"
## [1] "Democrat[ic]: 0"
## [1] "Republican: 1"
## [1] "free[dom]: 39"
## [1] "war: 60"
## [1] "God: 2"
## [1] "God Bless: 0"
## [1] "Jesus: 0"
## [1] "Christ: 0"
## [1] "Christian: 0"
```

```
word_checks(words_2008)
```

```
## [1] "I: 551"
## [1] "we: 579"
## [1] "America[n]: 80"
```

```
## [1] "democra[cy][tic]: 14"
## [1] "republic: 0"
## [1] "Democrat[ic]: 7"
## [1] "Republican: 9"
## [1] "free[dom]: 20"
## [1] "war: 32"
## [1] "God: 0"
## [1] "God Bless: 0"
## [1] "Jesus: 0"
## [1] "Christ: 4"
## [1] "Christian: 0"
```

```
word_checks(words_2012)
```

```
## [1] "I: 243"
## [1] "we: 190"
## [1] "America[n]: 66"
## [1] "democra[cy][tic]: 1"
## [1] "republic: 0"
## [1] "Democrat[ic]: 15"
## [1] "Republican: 10"
## [1] "free[dom]: 10"
## [1] "war: 2"
## [1] "God: 0"
## [1] "God Bless: 0"
## [1] "Jesus: 0"
## [1] "Christ: 0"
## [1] "Christian: 0"
```

```
word_checks(words_2016)
```

```
## [1] "I: 382"
## [1] "we: 205"
## [1] "America[n]: 54"
## [1] "democra[cy][tic]: 1"
## [1] "republic: 0"
## [1] "Democrat[ic]: 6"
## [1] "Republican: 2"
## [1] "free[dom]: 3"
## [1] "war: 17"
## [1] "God: 0"
## [1] "God Bless: 0"
## [1] "Jesus: 0"
## [1] "Christ: 0"
## [1] "Christian: 0"
```

Part E

Ran out of time to implement unit tests.

Question 3

Ran out of time to answer this question, too. I apologize for my awful time management in this class so far.