Stat 243 Problem Set 3

A.J. Torre

September 18, 2018

Problems

Comments: I worked with Huy, Andrea, RJ, and Sam on this assignment.

1. a) Read one of these four items.

I chose to read Gentzkow and Shapiro, which was an article directed towards social scientists who learned to code mainly to complete their research and never formally learned. Gentzkow and Shapiro outline some common coding issues that social science researchers run into, such as code not working a week later or realizing updating one thing means there are multiple parts of code that need to be revised. Since I'm still learning how to code in R, I think this article was helpful to read about good habits to develop, such as trying to learn from the experts (ie don't do it on your own blindly) and ideas on how to manage their data and code in an organized and repoducible way.

One question that I have from reading this article and from the code review discussion in lab is what are good test functions for when the output of your code is content? For example, for PS2 problem 3a), would be there be a way to test that your function does indeed output the correct url and html from the page using a test function and not just clicking on the google scholar page to make sure it matches with the output of the function? I understand using testing to make sure inputs are the right class or that a math function does equal what you'd expect, but I felt like for 3c), I could only think of more "superficial" tests instead of the actual html output of the function. This reading helped solidify for me the importance of using testing functions, as it's more reproducible to write test functions than to test interactively, but I think I still need practice on how to write test functions.

b) Nothing to turn in.

c) Nothing to turn in.

d) Based on our discussion in section, please list a few strengths and a few weakness of the reproducibility materials provided for the analysis in clm.pdf

In terms of reproducibility, the authors mention at the end that pehaps on issue of reproduciblity is if the code can apply to outside countries and rural cities outside of the US. Not all countries may have the pattern that younger populations move to the city, especially in more rural areas. As discussed in lab, the strengths of this code and paper are the detailed file descriptions, added documentation of workflow diagrams, and that they give both raw and clean data. In terms of weaknesses, we discussed: they could have used R project to help sort/ organize the code to avoid the step of having to set 5 different directories by hand, their code for the directories only works on their computer so this part isn't easily reproducible, they hardcoded for largest cities but that may change in some years/ this code is reproducible for exactly the same cities they worked on, possibly adding error messages if website doesn't exist so user can better understand if url moved/ data moved, modular design of code (if someone only wants to reproduce one part of code), and providing an environment summary of their packages.

While in lab, we listed more weaknesses than strengths of this paper and code, I am sure that coding at this level for this scale of project is challening it and of itself. Also, (as I realized while working on some of the codes for this hw assignment), there's some decision-making involved with reproducibility for how much work you are willing to do versus how much work you think that whomever is trying to reproduce your code is trying to do. For example, while they did hard code in the directories, they may have come to the decision that if someone was trying to reproduce their work, that person would likely know enough coding to be able to figure out how to modify their code for a different computer. Also, as mentioned at the end of lab, it's unrealistic to incorporate every single reproducibility practice into your work, and there will always have to be some judgement calls on which part of the project is most important.

2. a) Convert the text so that for each debate, the spoken words are split up into individual chunks of text spoken by each speaker (including the moderator). Make sure that any formatting and non-spoken text (eg the 'Laughter and 'Applause' tags) is stripped out. For the Laughter and Applause tags, retain information about the number of times it occurred in the debate for each candidate. Please print out or plot the number of chunks for the candidates.

This is the code that Chris provided to get the debates off the web. I also use the candidates list he made in later portions of the code.

```r
## Note that this code uses the XML package rather than xml2
## and rvest simply because I had this code sitting around
## from a previous demonstration.

## @knitr download

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"))


library(XML)
library(stringr)
library(assertthat)

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
```

```r
## @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"
```

First, to start this problem, I worked on trying to understand the patterns that the names follow within the debates text. Using regex expressions, names follow the pattern "[A-Z]+:". So, to split at the name, I worked with a classmate to follow one of the in class examples (switching from 3 May to May 3rd) using the \1 and \2 in R. The main idea is that we switched the order of the name and the colon (and gave some specifications in Regex for the name, like at least 2 upper-case letters as colons appear elsewhere in the text), and then added in tilday to split on as we couldn't find a way to get string split to let you keep the delimiter.

```r
# to separate the debates into a list for each debate to be
# one element in the list this will help us use lappy later
# on in the coding
debates_body_cleaner <- str_replace_all(debates_body, "[\r\n]",
    "") %>% str_remove_all("\\\\")
separate_debates <- list()
for (i in 1:length(debates_body_cleaner)) {
    separate_debates[i] <- debates_body_cleaner[i]
}

# this function is to clean up the script that we got from
# webscraping the text of the website it removes the newline
# chars and splits it up by speaker chunks enter in debates
# body/ specific debate
get_name_with_chunks <- function(debatesubset) {
    debatesubnew <- str_split(str_replace_all(debatesubset, "([A-Z]{2,}:)",
        "~\\1"), "~")
    debatesubnew <- unlist(debatesubnew)  #need thihs for next part
    return(debatesubnew)
}  #can assign function to variable to save this output
```

```r
# use lapply to get each debate formatted
separate_debates <- lapply(separate_debates, get_name_with_chunks)
expect_length(separate_debates, 6)
```

Next, to combine all the chunks of the same speaker, I used a for-loop and if-else statement that goes through and sees if the names (in regex) match in any consecutive lines. If so, it pastes them together. If not, it keeps them separate. All these lines get stored into a new list. I used lapply to get each debate with no duplicates.

```r
# this function is used to combine the chunks if a speaker
# speaks more than once in a row can save the output to a
# variable to hold onto to these results
get_debate_no_duplicates <- function(debatesubset) {
    debate_no_dupl <- list()
    j = 1
    debate_no_dupl[1] <- debatesubset[1]
    for (i in 2:length(debatesubset)) {
        if (identical(str_match(debatesubset[i], "^[A-Z]+:"),
            str_match(debate_no_dupl[j], "^[A-Z]+:"))) {
            debate_no_dupl[j] <- paste0(debate_no_dupl[j], debatesubset[i])
        } else {
            j = j + 1
            debate_no_dupl[j] <- debatesubset[i]
        }
    }
    return(debate_no_dupl)
}

# to apply to all our debates
separate_debates <- lapply(separate_debates, get_debate_no_duplicates)
```

To figure out the applause and laughter for a candidate, I worked with some classmates to see how the different debates used different syntax and how to use regex to account for these. Since the applause and laughter tags can be in either brackets or paranthesis and be in uppercase, lowercase, or just have the first letter capitalized. The most challenging part of writing this function was accounting for the all the different formats and taking the time to look through every debate to make sure the regex expression would be able to account for all the different cases. I used a “or” in regex and ignore case to account for the differing syntax. This function returns the candidate’s name with the number of times they got laugter and applause during the debate.

```r
# this function takes in the name of a debate file and which
# debate they want to choose from (a number from 1 to 6) and
# returns the laughs & applause the democratic candidate got
get_laughapplause_count_demcand <- function(debatesubset, x) {
    cand1name <- str_match(candidates[[x]][1], "^[A-Z]+")
    canddebate <- c()
    for (i in 1:length(debatesubset)) {
        if (identical(str_match(debatesubset[i], "^[A-Z]+"),
            cand1name)) {
            canddebate[i] <- debatesubset[i]
        }
    }
    # ignore case to account for variations of non-verbals
    # throughout the different scripts
    candlaughs <- length(grep("\\(LAUGHTER\\)|\\[laughter\\]",
        canddebate, ignore.case = TRUE))
    candapplause <- length(grep("\\(APPLAUSE\\)|\\[applause\\]",
        canddebate, ignore.case = TRUE))
    return(paste0(str_match(candidates[[x]][1], "^[A-Z]+"), "'s results:",
        " Laughs: ", candlaughs, " Applause: ", candapplause))
}

# to get it for all the dem candidates spanning debates for
# (i in 1:6){
# get_laughapplause_count_demcand(separate_debates[[i]],i) }
list6 <- 1:6
mapply(get_laughapplause_count_demcand, separate_debates, list6)
```

```
## [1] "CLINTON's results: Laughs: 4 Applause: 4"
## [2] "OBAMA's results: Laughs: 3 Applause: 0"
## [3] "OBAMA's results: Laughs: 0 Applause: 0"
## [4] "KERRY's results: Laughs: 2 Applause: 0"
## [5] "GORE's results: Laughs: 0 Applause: 0"
## [6] "CLINTON's results: Laughs: 0 Applause: 0"
```

Above are the laughter and applause counts for the democratic candidates. Below, we have a similar function that will output the laughs for the Republican candidate.

```r
# this function takes in the name of a debate file and which
# debate they want to choose from (a number from 1 to 6) and
# returns the laughs & applause the republican candidate got
get_laughapplause_count_repcand <- function(debatesubset, x) {
    cand1name <- str_match(candidates[[x]][2], "^[A-Z]+")
    canddebate <- c()
    for (i in 1:length(debatesubset)) {
        if (identical(str_match(debatesubset[i], "^[A-Z]+"),
            cand1name)) {
            canddebate[i] <- debatesubset[i]
        }
    }
    candlaughs <- length(grep("\\(LAUGHTER\\)|\\[laughter\\]",
        canddebate))
    candapplause <- length(grep("\\(APPLAUSE\\)|\\[applause\\]",
        canddebate))
    # resultstbl<-cbind(candlaughs, candapplause)
    return(paste0(str_match(candidates[[x]][2], "^[A-Z]+"), "'s results:",
        " Laughs: ", candlaughs, " Applause: ", candapplause))
}

mapply(get_laughapplause_count_repcand, separate_debates, list6)
```

```
## [1] "TRUMP's results: Laughs: 1 Applause: 5"
## [2] "ROMNEY's results: Laughs: 1 Applause: 0"
## [3] "MCCAIN's results: Laughs: 2 Applause: 0"
## [4] "BUSH's results: Laughs: 1 Applause: 0"
## [5] "BUSH's results: Laughs: 0 Applause: 0"
## [6] "DOLE's results: Laughs: 0 Applause: 0"
```

Next, in order to remove the tags from the files/ text, I used more regex. Similar to the laugh and applause function, this regex uses an "or" as the different debates are formatted a little differently so we remove words within both brackets and paranthesis.

```r
get_rid_of_tags <- function(debateinput) {
    debate_cleaned <- str_remove_all(debateinput, "\\[.*?\\]|\\(.*?\\)")
    return(debate_cleaned)
}

separate_debates_cleaned <- lapply(separate_debates, get_rid_of_tags)
```

In order to get the number of separate instances when a candidate speaks in the debate, I wrote a function to subset the data into its own character vector and then take the length of this vector and print the results. When just trying to get the sentences into its own vector, I ran into the issue that when the candidate hadn't spoken thhe vector kept those lines as NULL. To get rid of this, I ran another if statement to remove the NULL lines to get the finalized candidate chunks.

When comparing these results, we noticed some disparities in how often the candidates spoke. Looking at debate 1, we noticed that Trump spoke much more than Clinton, and that the moderator actually spoke on more separate occasions that Hilary Clinton did, which is somewhat telling of how the debate went. It can be useful to compare the number of times that the candidates spoke before even reading the debate transcripts so we get an idea if the debate was balanced or not.

```r
# this function takes 3 input, where x and y are based off
# the candidate vector provided and debatesubset is the
# debate of interest it splits up one candidate's chunks and
# prints how often they spoke
get_cand_chunks <- function(x, y, debatesubset) {
    cand_chunks <- character(0)
    removethis <- character(0)
    namecand <- str_match(candidates[[x]][y], "^[A-Z]+")
    for (i in 1:length(debatesubset)) {
        if (identical(str_match(debatesubset[i], "^[A-Z]+"),
            namecand)) {
            cand_chunks[i] <- debatesubset[i]
        }
    }
    for (i in 1:length(cand_chunks)) {
        if (is.null(cand_chunks[i])) {
            removethis[i] <- cand_chunks[i]
        }
    }
    cand_chunks <- setdiff(cand_chunks, removethis)
    cand_length <- length(cand_chunks)
    print(paste0(str_match(candidates[[x]][y], "^[A-Z]+"), " spoke this many times: ",
        cand_length))
    return(cand_chunks)
}

# here I used mapply to get all the democratic candidates
# chunks into one list of length 6
dem_chunks <- mapply(get_cand_chunks, list6, y = 1, separate_debates_cleaned)
```

```
## [1] "CLINTON spoke this many times: 88"
## [1] "OBAMA spoke this many times: 43"
## [1] "OBAMA spoke this many times: 60"
## [1] "KERRY spoke this many times: 34"
## [1] "GORE spoke this many times: 50"
## [1] "CLINTON spoke this many times: 45"
```

```r
expect_length(dem_chunks, 6)

# here I used mapply to get all the republican candidates
# chunks into one list of length 6
rep_chunks <- mapply(get_cand_chunks, list6, y = 2, separate_debates_cleaned)
```

```
## [1] "TRUMP spoke this many times: 121"
## [1] "ROMNEY spoke this many times: 54"
## [1] "MCCAIN spoke this many times: 59"
## [1] "BUSH spoke this many times: 42"
## [1] "BUSH spoke this many times: 57"
## [1] "DOLE spoke this many times: 47"
```

```r
expect_length(rep_chunks, 6)
```

b) Use regular expression processing to extract the sentences and individual words as character vectors, one element per sentence and one element per word.

This function takes one argument, which is the candidate's chunks, outputted from our function in part a) and returns a vector of sentences. To start this function, I used str_replace to get rid of the most common abbreviations (such as Mr., Mrs., U.S.A.) that use a period so that when this function splits on a period, it doesn't split these words. From the scripts, it looked like some sentences didn't have a space after a period before starting a new sentence. For ex, "I ran today.It was good." So, I chose the delimeter to be the ending punctuation of the sentence so that the function wouldn't miss any sentences with no spaces between them and used the regex for space with a star for 0 or more times.

```r
# this function takes out common ways periods are used to NOT
# end sentences and then splits into separate sentences
get_cand_sentences <- function(candchunks) {
    cand_chunks <- str_replace_all(candchunks, "Mr.", "Mr")
    cand_chunks <- str_replace_all(candchunks, "Mrs.", "Mrs")
    cand_chunks <- str_replace_all(candchunks, "Ms.", "Ms")
    cand_chhunks <- str_replace_all(candchunks, "U.S.A", "USA")
    canddebatesents <- character(0)
    canddebatesents <- unlist(strsplit(unlist(candchunks), "(?<=[\\.\\?\\!])\\s*(?=[A-Z])",
        perl = T))
    expect_that(canddebatesents, is_a("character"))
    return(canddebatesents)
}

# using lapply to our previous output to get the vector of
# words for each democratic candidate
dem_cand_sentences <- lapply(dem_chunks, get_cand_sentences)

# using lapply to our previous output to get the vector of
# words for each republican candidate
rep_cand_sentences <- lapply(rep_chunks, get_cand_sentences)
```

b) (continued.) To get the words of the candidate, I wrote a function that would take the vector of sentences and extract all the words/ numbers. Also, this function removes the candidate's name from the sentences, ex "CLINTON:", as it's not one of their spoken words. I also ended up using str_extract_all to get all the words. I originally wrote a function that split on the white space of the sentence, but I realized that there were special characters within the sentences and I wanted to find a away to avoid adding those in the vector.

```
# this function gets each individual word of the candidate by
# first removing the candidates' name from the text and then
# extracting all the characters
get_cand_words <- function(cand_sents) {
    cand_sents_tmp <- str_remove_all(cand_sents, "[A-Z]+:")
    cand_sents_tmp <- str_remove_all(cand_sents_tmp, "'")  #removes apostrophes
    # for the purpose of keeping contractions intact when
    # extracting
    cand_words <- str_extract_all(cand_sents_tmp, "[A-Za-z0-9]+") %>%
        unlist %>% str_split("//s") %>% str_extract_all("[A-Za-z0-9]+")
    return(cand_words)
}

# getting the sentences for all dem candidates
dem_cand_words <- lapply(dem_cand_sentences, get_cand_words)
expect_that(dem_cand_words, is_a("list"))

# getting the sentences for all rep candidates
rep_cand_words <- lapply(rep_cand_sentences, get_cand_words)
expect_that(rep_cand_words, is_a("list"))
```

c) For each candidate, for each debate, count the number of words and characters and compute the average word length for eachh candidate. Store this information in n R data structure and make of the word length for the candidates. Comment briefly on the results.

To get the average word length for each candidate, I used nchar to get the total length of what each candidate's words and divided this by the length of their word vector (the total number of words they spoke). Below, I used lapply to get average word length for each candidate by their political party and thhen used cbind to make a table so that I could compare the word length.

```
# this function takes input of a list of the candidates's
# words and outputs the average wordlength
get_cand_wordlength <- function(cand_words) {
    cand_letters <- nchar(cand_words, type = "chars", allowNA = FALSE,
        keepNA = FALSE)
    total_letters <- sum(cand_letters)
    cand_wordlength <- total_letters/length(cand_words)
    expect_that(cand_wordlength, is_a("numeric"))
    return(paste0("Average word length for your candidate: ",
        cand_wordlength))
}

# to get for all the dem candidates
dem_wordlength <- lapply(dem_cand_words, get_cand_wordlength)

# to gt for all the rep candidates
rep_wordlength <- lapply(rep_cand_words, get_cand_wordlength)

cands_for_tbl <- list("Clinton|Trump", "Obama|Romney", "Obama|McCain",
    "Kerry|Bush", "Gore|Bush", "Clinton|Dole")

cand_words_tbl <- cbind(cands_for_tbl, dem_wordlength, rep_wordlength)

# table to compare the average word lengths per candidate
cand_words_tbl
```

```
##      cands_for_tbl
## [1,] "Clinton|Trump"
## [2,] "Obama|Romney"
## [3,] "Obama|McCain"
## [4,] "Kerry|Bush"
## [5,] "Gore|Bush"
## [6,] "Clinton|Dole"
##      dem_wordlength
## [1,] "Average word length for your candidate: 4.27321121027086"
## [2,] "Average word length for your candidate: 4.40322580645161"
## [3,] "Average word length for your candidate: 4.33097854526426"
## [4,] "Average word length for your candidate: 4.25045499090018"
## [5,] "Average word length for your candidate: 4.31162919023314"
## [6,] "Average word length for your candidate: 4.33367969866216"
##      rep_wordlength
## [1,] "Average word length for your candidate: 4.13600464576074"
## [2,] "Average word length for your candidate: 4.2674581897102"
## [3,] "Average word length for your candidate: 4.37346540178571"
## [4,] "Average word length for your candidate: 4.27344241661422"
## [5,] "Average word length for your candidate: 4.2579010534738"
## [6,] "Average word length for your candidate: 4.25485377242566"
```

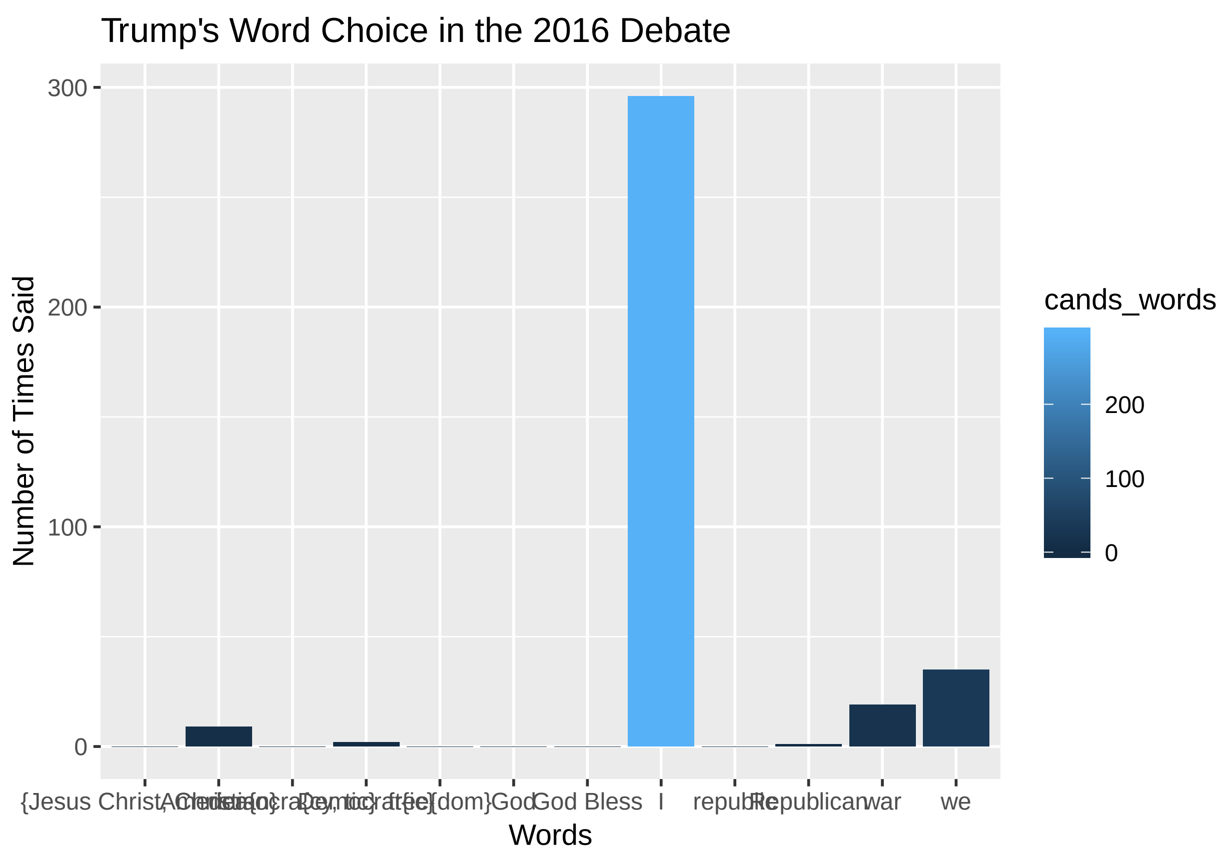
Looking at these average word lengths,it seems as though most candidates have about the same average word length. Initially, I was surprised by these results and thought that perhaps, the average word length should be higher, but once I looked through the actual words, I think the return of this function made more sense. Also, in these debates, they're addressing the American public so they're not likely to use too much jargon or many large words.

d) For each candidate, count the following words or word stems and store in an R data structure:I, we, America{,n}, democra{cy,tic}, repbulic, Democrat{,ic}, Republican, free{,dom}, war, God [not including God Bless], God Bless, {Jesus Christ, Christian}. Make a plot or two and comment briefly on the results.

```r
# this function takes an input of the candidates' sentences
# and then counts how often they use certain words returning
# a sorted data frame
get_cand_used_words <- function(cand_sents) {
    candswordsvector <- c("America{n}", "I", "we", "democra{cy, tic}",
        "republic", "Democrat{ic}", "Republican", "free{dom}",
        "war", "God", "God Bless", "{Jesus Christ, Christian")
    cands_words <- integer(0)
    cands_words[1] <- str_count(cand_sents, "America|American")
    cands_words[2] <- str_count(cand_sents, "I[[:space:]]|I'")
    cands_words[3] <- str_count(cand_sents, "\\<we\\>|we'|\\<We\\>|We'")
    cands_words[4] <- str_count(cand_sents, "democracy|democratic")
    cands_words[5] <- str_count(cand_sents, "republic")
    cands_words[6] <- str_count(cand_sents, "Democrat|Democratic")
    cands_words[7] <- str_count(cand_sents, "Republican")
    cands_words[8] <- str_count(cand_sents, "free|freedom")
    cands_words[9] <- str_count(cand_sents, "war")
    cands_words[10] <- str_count(cand_sents, "\\<God\\>")
    cands_words[11] <- str_count(cand_sents, "^God Bless$")
    cands_words[12] <- str_count(cand_sents, "^Jesus Christ$|Christian")
    expect_that(cands_words, is_a("integer"))
    cand_df <- data.frame(cands_words, candswordsvector)
    cand_df_final <- arrange(cand_df, cands_words)
    return(cand_df_final)
}


clinton_used_words <- get_cand_used_words(dem_cand_sentences[1])
trump_used_words <- get_cand_used_words(rep_cand_sentences[1])

trump_words <- ggplot(trump_used_words, aes(x = candswordsvector,
    y = cands_words, fill = cands_words)) + geom_bar(stat = "identity") +
    ggtitle("Trump's Word Choice in the 2016 Debate") + xlab("Words") +
    ylab("Number of Times Said")
trump_words + scale_color_brewer("Dark2")
```

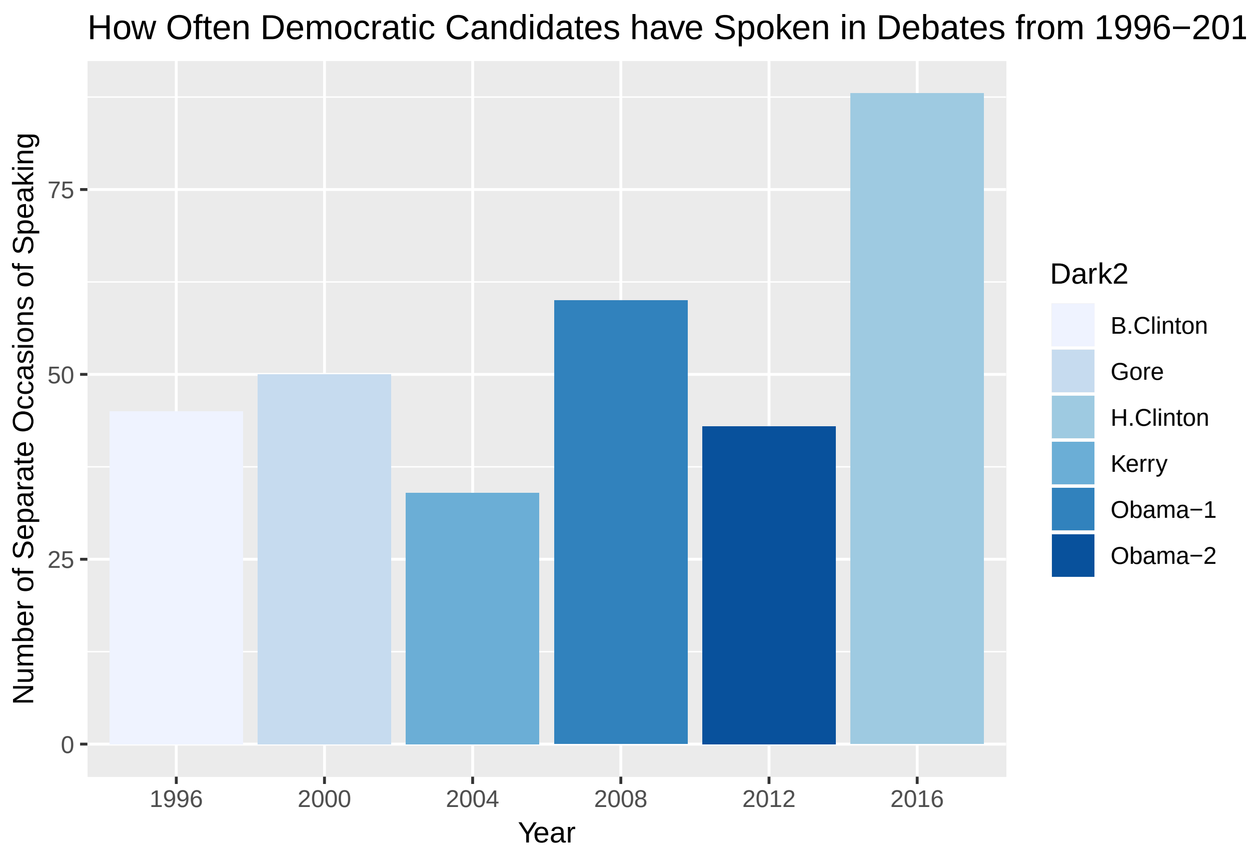


d) (continued) To make some more plots, I created a dataframe for the number of chunks per year per party and graphed those to compare how often the Democratic and Republican candidates spoke in debates througout the years.

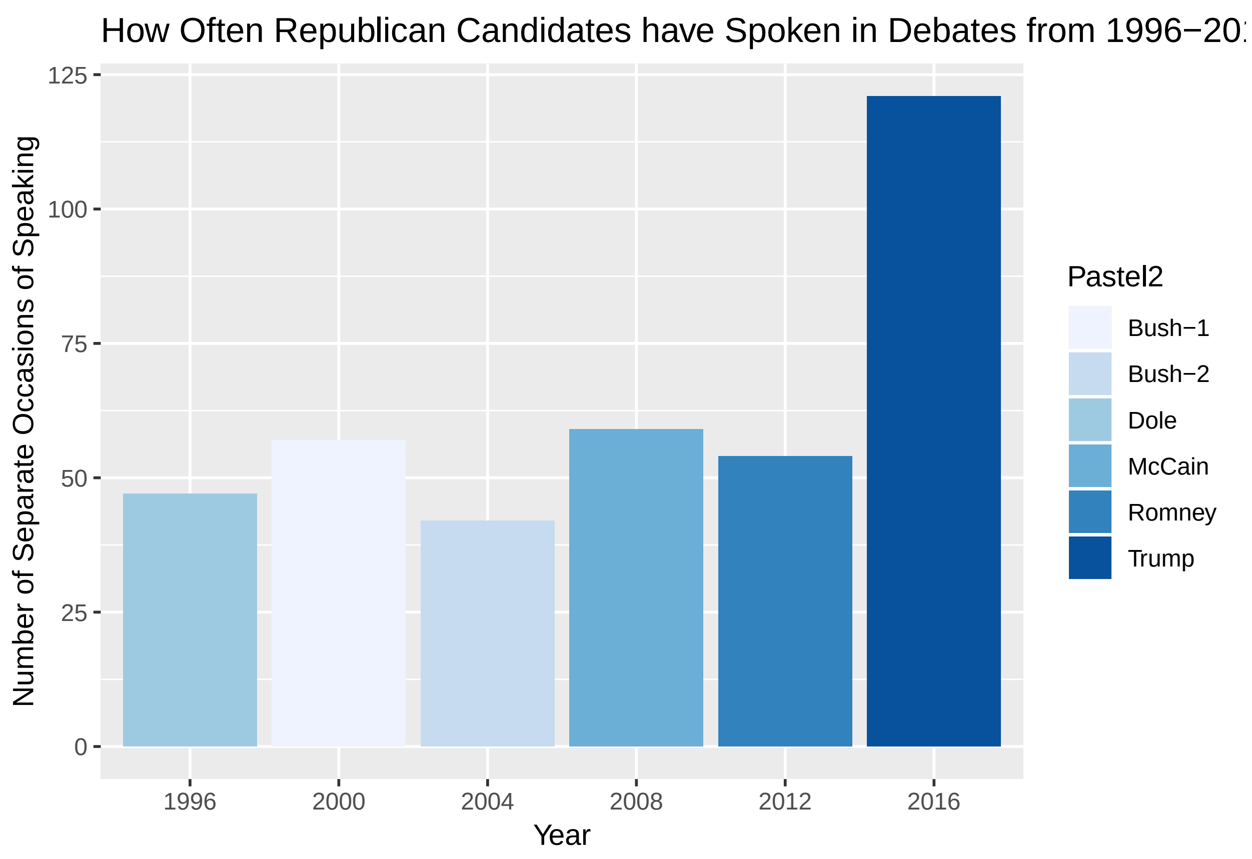
```
years <- c("1996", "2000", "2004", "2008", "2012", "2016")
dem_cands <- c("B.Clinton", "Gore", "Kerry", "Obama-1", "Obama-2",
    "H.Clinton")
rep_cands <- c("Dole", "Bush-1", "Bush-2", "McCain", "Romney",
    "Trump")
dem_chunks_length <- c(45, 50, 34, 60, 43, 88)
rep_chunks_length <- c(47, 57, 42, 59, 54, 121)
chunks_by_year <- data.frame(dem_chunks_length, rep_chunks_length,
    dem_cands, rep_cands, row.names = years)
chunks_by_year
```

```
##      dem_chunks_length rep_chunks_length dem_cands rep_cands
## 1996                45                47 B.Clinton      Dole
## 2000                50                57      Gore    Bush-1
## 2004                34                42     Kerry    Bush-2
## 2008                60                59   Obama-1    McCain
## 2012                43                54   Obama-2    Romney
## 2016                88               121 H.Clinton     Trump
```

```
dem_chunksbyyear_plot <- ggplot(data = chunks_by_year, mapping = aes(x = years,
    y = dem_chunks_length, fill = dem_cands)) + geom_bar(stat = "identity") +
    ggtitle("How Often Democratic Candidates have Spoken in Debates from 1996-2016") +
    xlab("Year") + ylab("Number of Separate Occasions of Speaking")
dem_chunksbyyear_plot + scale_fill_brewer("Dark2")
```



```
rep_chunksbyyear_plot <- ggplot(data = chunks_by_year, mapping = aes(x = years,
    y = rep_chunks_length, fill = rep_cands)) + geom_bar(stat = "identity") +
    ggtitle("How Often Republican Candidates have Spoken in Debates from 1996-2016") +
    xlab("Year") + ylab("Number of Separate Occasions of Speaking")
rep_chunksbyyear_plot + scale_fill_brewer("Pastel2")
```



When looking at these graphs and also the data from the debates, we notice that the 2016 debate was significantly longer than past years' debates. From 1996-2012, the length of the debates are fairly close together. There does not seem to be a clear trend of the candidate that speaks more or less tends to win the election. However, in this most recent election Trump spoke around 120 times while Clinton only spoke around 80-90 times during the debate, which is big disparity. Also, from talking with classmates and looking at the results, I noticed the religious terms aren't said too often during debates and "I" and "we" are the most used words on that list, which makes sense. Overall, this assignment helped me see some trends and patterns in the debates and gain a better understanding of what types of words they use.

e) Please include unit tests for your various functions.

Through out my functions, I used expect_that in the test_that package to make sure that lapplying the function would indeed give me back a list of length 6 to make sure the function and lapply preserved each debate/ candidate slot. I also used expect_that within functions to make sure that certain parts of my function were returning the right type of object that I was expecting, like a vector or integer, as the entire code for this problem set involves lists, vectors with characters, vectors with integers, etc. so I just wanted to make sure that I was getting the correct type of output before moving onto the next part.

3. I got help from some classmates (R.J. and Andrea) on how to complete this project using an object oriented programming language, such as Python or C++. They explained that the point was to define classes in a meaningful way and to write methods as analagous to the functions we wrote for problem 2. Based on the reading and discussion in class, I would want each class to inherit characteristics from the one above it as each class is built off of the one before it.

Class 1: Raw_Debate This class would be the debates without any processing and/ or formatting. The method for this class would be called `webscrape_debates` analagous to the webscraping code that Chris provided us with.

Class 2: Separated_Debates This class would be similar to the results of the question 2a; the debates would be cleaned up but would not be separated into different lists/ vectors for each candidate yet. The first method, `clean_debates` would be similar to my first step in 2a, where I removed the extra slashes and newline marks in the text. The next method would be `format_by_name` so that each debate is formatted to look like a script with each chunk of text is clearly with the person who said it. For the third method, I would create `paste_together`, where each time someone speaks twice, it pastes those two lines/ chunks together. My fourth method would be `count_laughter_and_applause`, where this method would be similar to the get_laugh_applause_count function by looking at how often a specific candidate got laughter and applause from the audience through the non-verbal tags. Lastly, after counting all theh laughs and applause, the fifth method is `no_tags`, which removes the non-verbal tags from each debate.

Class 3: Candidate_Debates This class would be the debates separated by candidate and by year, so each candidate has their own list/ vector of their spoken text stored under their name. It is also sorted by year so even though Obama is in both the 2008 and 2012 debates, he would have separate lists for each year. The methods of this class would be similar to the functions in 2b, 2c, and 2d as those functions required the debates to be sorted by candidate. The first method would be `candidate_sentences`, which would split each candidate's chunks into separate sentences. Then, using the results from this method, the next method would be `candidate_words`, which would split the sentences into separate words and numbers, getting rid of the special characters. Next, my third method would be `candidate_word_length`, which would use the output of `candidate_words` to calculate a candidate's average word length. Lastly, using either the output of `candidate_sentences` or `candidate_wod=rds`, the fourth method would look at how often candidates use certain words, like "I", "we", "democratic", etc.