ps3

Weijie Yuan

9/18/2018

Answers for Problem 1

(a)

Question

1) What does 'Version Control' exactly mean? I search for it on 'git', however, it is equivocal to understand clearly. And what kind of software should I install for my personal computer to conduct version control.
2) How to demarcate whether a code chunk need a detailed document? Because in a coding-extensive project, it is time-consuming to document all the code chunk in detail. And I think writing program for people with different level of foundation of statistics knowledge and code programming require different kind of documents.

Comment

Previously, I never find it important to make one's workflow replicable until when I was assigned to work on a machine learning project in another lesson IEOR290. In that project, we are expected to put forward a new method to improve the NLP model based on original work from students last year. Fortunately, students last year make all their work replicable. They upload the data and code to Github and separate script, code, instruction, document and result into different subdirectory. Furthermore, they write a README. file to illustrate the original thought and exact order of their workflow. Those behaviors make it super convenient for us to continue their work or develop new branches of this project. Although there are millions of reason for a student not to attach importance to make his work replicable because of time, effort, data restrictions, a lack of reward, we should enhance our determination to change this situation. Python is famous and popular because it is a completely open source language. This era of data is bound to be more open and we all should contribute our strength to be more academic, which means we are expected to conduct more transparent and replicable programming or analysis.

(d)

Strength: * File descriptions * Workflow diagram * clear construction * detailed documents

Weakness: * Absolute paths * Use of non-generalizable codes * Change all of directories * Assertion/chunk for existence of URLs * Non-modular design * Package versions specified * Hard to distinguish the grobal variables, local variables and function arguments * No test function, which leads to trouble for others to read the code

Answers for Problem 2

In my work on problem set 3, I process the debate text in a vectorized way and do not use any loop.

Download the html and pre-process the string data.

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

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
```

When we preprocess the text data, we find that debate transcript in 2018 repeats twice in html text. So firstly, we try to strip out the repeated text. Use 'str_locate_all' to find the index of the end of first transript and use substring to extract the unique debate transcript.

```r
str_locate_all(debates_body[3],'END')
```

```
## [[1]]
##       start    end
## [1,]  91182  91184
## [2,] 182375 182377
```

```r
debates_body[3] <- substring(debates_body[3], 1, 91181)
```

(a)

If one do not need the information about time order of each spoken chunks. Then we can just use the regular expression to extract the spoken word of each speaker. And another requirement is that two chunks in a row spoken by a candidate, it's supposed to combine them into a single chunk. Put Democratic candidates' names, Republican candidates' names and moderators' names into respective regular expression to aviod looping. Besides, to put two chunks in a row spoken by a candidate into a single one. I use to match the name of one candidate and stop matching when it finds the names of another candidates or moderators.

```r
# construct the regular expressions corresponding to each party and moderators.
dem = sapply(candidates,'[[', 1)
rep = sapply(candidates,'[[', 2)
dem_colon = paste0(dem,":")
rep_colon = paste0(rep,":")
dem_or = paste0(dem_colon, collapse = '|')
rep_or = paste0(rep_colon, collapse = '|')
mod_colon = paste0(moderators,":")
mod_or = paste0(mod_colon, collapse = '|')

end_pattern = c('(Good night, everyone.)|(Thank you, and good night.)|
                (Thank you and good night.)|(Thank you. God bless America.)')

# duplicate the debates_body to aviod changing the original data
debates_speak <- debates_body[1:6]

# strip out formatting and non-spoken text
# including '()', '[]', '\n', \" so on and so forth
debates_speak <- sapply(debates_speak, str_replace_all,
                       pattern='\\(.*?\\)|\\[.*?\\]|(\\n)|\\\"',replacement='')

# self-define function to put each candidate's response into a respective character vector
# the function return a list which is seperated into three parts for each year
extract_chunk <- function(vector){
  # match the speaking content from Democratic candidate and put them into a character vector
  dem_reg <- regex(paste0("(",dem_or,")","(.*?)","(",rep_or,"|",mod_or,"|",end_pattern,")")
                  ,dotall = TRUE)
  dem_speech <- str_extract_all(vector, pattern = dem_reg)
  dem_name <- sapply(dem_speech, str_extract_all, pattern = dem_or)[[1]][1]
  dem_vector <- as.vector(sapply(dem_speech, str_replace_all,
                                 pattern = paste0(mod_or,'|',dem_or,'|',rep_or),
                                 replacement =''))

  # match the speaking content from Republican candidate and put them into a character vector
  mod_reg <- regex(paste0("(",mod_or,")","(.*?)","(",rep_or,"|",dem_or,"|",end_pattern,")")
                   ,dotall = TRUE)
  mod_speech <- str_extract_all(vector, pattern = mod_reg)
  mod_name <- sapply(mod_speech, str_extract_all, pattern = mod_or)[[1]][1]
  mod_vector <- as.vector(sapply(mod_speech, str_replace_all,
                                 pattern = paste0(mod_or,'|',dem_or,'|',rep_or),
                                 replacement =''))

  # match the speaking content from moderators and put them into a character vector
  rep_reg <- regex(paste0("(",rep_or,")","(.*?)","(",dem_or,"|",mod_or,"|",end_pattern,")")
                   ,dotall = TRUE)
  rep_speech <- str_extract_all(vector, pattern = rep_reg)
  rep_name <- sapply(rep_speech, str_extract_all, pattern = rep_or)[[1]][1]
  rep_vector <- as.vector(sapply(rep_speech, str_replace_all,
                                 pattern = paste0(mod_or,'|',dem_or,'|',rep_or),
                                 replacement =''))

  # put three character vectors generated above into a list
  result_list <- list(dem_vector,
                      rep_vector,mod_vector)
  names(result_list) <- c(dem_name , rep_name , mod_name)
  return(result_list)
}


test_that('test year 2016 works',
          expect_true(is.list(extract_chunk(debates_speak[[1]]))))

# apply the 'extract_chunk()' to 'debates_speak' in a vectorized way using 'lapply'
speak_chunk <- lapply(debates_speak, extract_chunk)
# assign the name of the result list
names(speak_chunk)<-c(2016,2012,2008,2004,2000,1996)
```

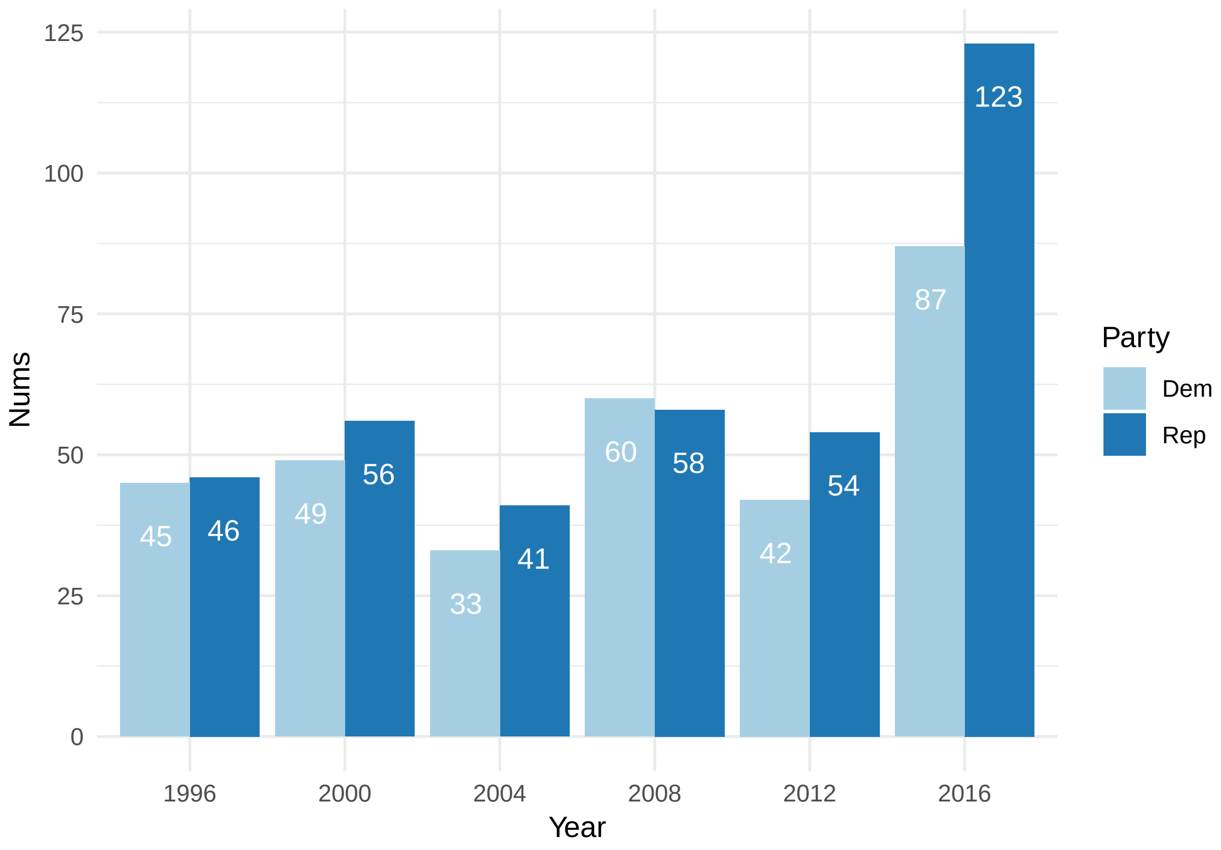
Print out and plot the number of chunks for the candidates.

```r
# self-define a function to count the number of chunks
count_num<-function(vector){
  count_list <- sapply(vector,length)

  # exclude the names of moderators from the result
  count_list <- count_list[names(count_list)!='LEHRER:'&
                             names(count_list)!='MODERATOR:'&
                             names(count_list)!='HOLT:']
  return(count_list)
}

test_that("test year 2016 not include moderator's name",
          expect_true(all(names(count_num(speak_chunk[[1]]))!='HOLT:')))
```

```r
# apply it to 'debates_speak' in a vectorized way
num_of_chunk <- lapply(speak_chunk , count_num)
names(num_of_chunk) <- c(2016,2012,2008,2004,2000,1996)
# show the result
print(num_of_chunk)
```

```
## $`2016`
## CLINTON:   TRUMP:
##       87      123
##
## $`2012`
##  OBAMA: ROMNEY:
##      42      54
##
## $`2008`
##  OBAMA: MCCAIN:
##      60      58
##
## $`2004`
## KERRY:  BUSH:
##     33     41
##
## $`2000`
## GORE: BUSH:
##    49    56
##
## $`1996`
## CLINTON:    DOLE:
##       45       46
```

```r
# try to plot the number of chunks for the candidates.
library(ggplot2)
df_count <- data.frame(Party = rep(c('Dem','Rep'),each=6),
                       Nums = c(sapply(num_of_chunk,'[[',1),sapply(num_of_chunk,'[[',2)),
                       Year = rep(c(2016,2012,2008,2004,2000,1996),2))
ggplot(data=df_count, aes(x=Year, y=Nums, fill=Party)) +
geom_bar(stat="identity", position=position_dodge()) +
  geom_text(aes(label=Nums), vjust=3, color="white",
            position = position_dodge(3.5), size=4)+
  scale_fill_brewer(palette="Paired")+
  scale_x_discrete(limits=c(2016,2012,2008,2004,2000,1996))+
  theme_minimal()
```



There should be some sort of metadata or attributes so that you can easily extract only the chunks for one candidate.

```r
# take OBAMA in 2008 as an example
speak_chunk[["2008"]]['OBAMA:'][[1]][1:3]
```

```
## [1] " Well, thank you very much, Jim, and thanks to the commission and the University of Mississippi
## [2] " We haven't seen the language yet. And I do think that there's constructive work being done out
## [3] " Well, I think Senator McCain's absolutely right that we need more responsibility, but we need 
```

In this case, one can extract all chunks spoken by a certain speaker in a certain year. Take Trump in 2016 as an example.

```r
num_of_chunk[['2016']]['TRUMP:']
```

```
## TRUMP:
##    123
```

which is obviouly very convenient.

In the other hand, if one need the information about time order, we can use the code below simply manipulate the original list rather than store the information extracted from the original one to a new one.

```r
# self-define function to seperate and clean the string
string_clean <- function(vector){
  transcript = strsplit(vector,
                      split=regex(paste0(mod_or,'|',dem_or,'|',rep_or),
                                  dotall = TRUE))[[1]]
  # delete the first element which is the title not dialogue
  dialogue = tail(transcript,-1)
  # find the speaker of each
  speaker <- str_extract_all(vector, paste0(mod_or,'|',dem_or,'|',rep_or))[[1]]
  names(dialogue) <- speaker
  return(dialogue)
}
debates_dialogue<-lapply(debates_speak,string_clean)

test_that('test year 2016 works',
          expect_true(is.vector(string_clean(debates_speak[[1]]))))

# if one only want to check the chunks for one candidate
# take OBAMA in 2008 as an example
debates_2018 <- debates_dialogue[[3]]
debates_2018[names(debates_2018)=='OBAMA:'][1:3]
```

```
##
## " Well, thank you very much, Jim, and thanks to the commission and the University of Mississippi, Ol
##
##
##
##
```

However, using this method is somewhat difficult to combine two chunks in a row spoken by a candidate into a single chunk. Just for reference. And we can check the total number of chunks of each candidate, which is correspondingly equal to or larger than the numbers counted above. That makes sense because combining successive chunks only decreases the number of chunks.

```
# print out or plot the number of chunks for the candidates.
name_list = sapply(debates_dialogue,names)
# exclude the chunks for the moderators when call table() function
count_candidates = lapply(name_list,table,exclude = mod_colon)
# rename the list to illustrate the years
names(count_candidates)<-c(2016,2012,2008,2004,2000,1996)
# print out the number of chunks for the candidates every year.
count_candidates
```

```
## $`2016`
##
## CLINTON:   TRUMP:
##       87      124
##
## $`2012`
##
##  OBAMA: ROMNEY:
##      56      71
##
## $`2008`
##
## MCCAIN:  OBAMA:
##      63      63
##
## $`2004`
##
##  BUSH: KERRY:
##     41     33
```

```
##
## $`2000`
##
## BUSH: GORE:
##    56    49
##
## $`1996`
##
## CLINTON:    DOLE:
##       45       46
```

Althouth the processing method showed above is undesirable considering the requirement of this question, it can simplify the process of counting the number of times of special words involved in later questions because it decreases the number of layers of the chunk list.

For the Laughter and Applause tags, calculate the number of times it occurred in the debate for each candidate mainly using 'table()' function in a vectorized way.

```
# extract information about the number of times laughter and applause occurred in the debate
# for each candidate.
debates_nontext<-lapply(debates_body, string_clean)

# self-define a function to calculate the numbers of times laughter and applause occurred
# in the debate for each candidate every year
string_count_al <- function(vector){
  # use regular expression to match laughter and applause
  list_al <- sapply(vector, str_extract_all,
                    pattern=regex('(\\(|\\[)(applause|laughter)(\\)|\\])', ignore_case = T))
  applause_and_laughter <- unlist(list_al)

  # use table() function to calculate the numbers of times laughter and applause for
  # each candidate
  table_al <- table(names(applause_and_laughter),applause_and_laughter,
                    exclude = mod_colon)
  return(table_al)
}

test_that('test year 2016 has 2*2 structure',
          expect_true(all(dim(string_count_al(debates_nontext[[1]]))==2)))

# use sapply() to apply the self-defined function in a vectorized way
table_applause_and_laghter <- sapply(debates_nontext,string_count_al)

# assign proper title to illustrate the year
names(table_applause_and_laghter)<-c(2016,2012,2008,2004,2000,1996)
table_applause_and_laghter
```

```
## $`2016`
##           applause_and_laughter
##             [applause] [laughter]
##   CLINTON:          4          4
##   TRUMP:            5          1
##
## $`2012`
##          applause_and_laughter
##            (APPLAUSE) (LAUGHTER)
##   OBAMA:           0          3
##   ROMNEY:          0          1
##
## $`2008`
##          applause_and_laughter
##           (APPLAUSE) (LAUGHTER)
##   MCCAIN:          0          1
##
## $`2004`
##          applause_and_laughter
##           (APPLAUSE) (LAUGHTER)
##   BUSH:          0          1
##   KERRY:         0          2
##
## $`2000`
## < table of extent 0 x 2 >
##
## $`1996`
## < table of extent 0 x 0 >
```

(b)

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

```r
# '(\\.\\.\\.)?[[:space:]]([a-zA-Z]+)(.*?)(\\.|\\?)'
# .*?(?<!Mr|Mrs)(\.|\?)(?= [A-Z]|$)

#self-define function to split sentences in speaking chunks
extract_sentence <- function(vector){
  vector_sentence <- sapply(vector, str_extract_all,
    pattern = '(.*?)(\\s)?(.*?)(?<!Mr|Mrs|St|J|U|S|A)(\\.|\\?)(\\s)?(?=[[:space:]]?[A-Z]|$)')
  return(vector_sentence)
}

test_that("test year 2016 Clinton's first chunk",
          expect_true(length(extract_sentence(debates_nontext[[1]][[1]]))==1))

# apply the self-define function to speaking chunks in a vectorized way
extracted_sentence = lapply(speak_chunk, extract_sentence)
extracted_sentence[[1]][['CLINTON:']][1:3]
```

```
## [[1]]
## [1] " How are you, Donald? "
##
## [[2]]
##  [1] " Well, thank you, Lester, and thanks to Hofstra for hosting us."
##  [2] "The central question in this election is really what kind of country we want to be and what ki
##  [3] "Today is my granddaughter's second birthday, so I think about this a lot. "
##  [4] "First, we have to build an economy that works for everyone, not just those at the top. "
##  [5] "That means we need new jobs, good jobs, with rising incomes."
##  [6] "I want us to invest in you. "
##  [7] "I want us to invest in your future. "
##  [8] "That means jobs in infrastructure, in advanced manufacturing, innovation and technology, clean
##  [9] "We also have to make the economy fairer. "
## [10] "That starts with raising the national minimum wage and also guarantee, finally, equal pay for 
## [11] "I also want to see more companies do profit-sharing. "
## [12] "If you help create the profits, you should be able to share in them, not just the executives a
## [13] "And I want us to do more to support people who are struggling to balance family and work. "
## [14] "I've heard from so many of you about the difficult choices you face and the stresses that you'
## [15] "So let's have paid family leave, earned sick days. "
## [16] "Let's be sure we have affordable child care and debt-free college."
## [17] "How are we going to do it? "
## [18] "We're going to do it by having the wealthy pay their fair share and close the corporate loophol
## [19] "Finally, we tonight are on the stage together, Donald Trump and I. "
## [20] "Donald, it's good to be with you. "
## [21] "We're going to have a debate where we are talking about the important issues facing our countr
## [22] "You have to judge us, who can shoulder the immense, awesome responsibilities of the presidency
## [23] "I hope that I will be able to earn your vote on November 8th."
##
## [[3]]
##  [1] " Well, I think that trade is an important issue. "
##  [2] "Of course, we are 5 percent of the world's population; we have to trade with the other 95 perc
##  [3] "And we need to have smart, fair trade deals."
##  [4] "We also, though, need to have a tax system that rewards work and not just financial transactio
##  [5] "And the kind of plan that Donald has put forth would be trickle-down economics all over again.
##  [6] "In fact, it would be the most extreme version, the biggest tax cuts for the top percent of the
##  [7] "I call it trumped-up trickle-down, because that's exactly what it would be. "
##  [8] "That is not how we grow the economy."
##  [9] "We just have a different view about what's best for growing the economy, how we make investmen
## [10] "I think we come at it from somewhat different perspectives. "
## [11] "I understand that. "
## [12] "You know, Donald was very fortunate in his life, and that's all to his benefit. "
## [13] "He started his business with $14 million, borrowed from his father, and he really believes tha
## [14] "I don't buy that. "
## [15] "I have a different experience. "
## [16] "My father was a small-businessman. "
## [17] "He worked really hard. "
## [18] "He printed drapery fabrics on long tables, where he pulled out those fabrics and he went down 
## [19] "And so what I believe is the more we can do for the middle class, the more we can invest in yo
## [20] "That's the kind of economy I want us to see again."
```

The whole result is quite verbose. I'll show some special situation to evaluate the high performance of my regular expression.

- The regular expression ignore words like Mr./Mrs./Dr./St. considering '.' is one of terminators of a sentence.

```r
print(extracted_sentence[[1]][['MODERATOR:']][[1]][13])
```

```
## [1] "I will invite you to applaud, however, at this moment, as we welcome the candidates: Democratic
```

```r
print(extracted_sentence[[1]][['MODERATOR:']][[3]][2])
```

```
## [1] "Mr. Trump, the same question to you. "
```

- The regular expression takes sentences begin with '. . . ' or end with '. . . ' into consideration.

```r
print(extracted_sentence[[1]][['MODERATOR:']][[67]][1])
```

```
## [1] " ... why is your-why is your judgment..."
```

```
print(extracted_sentence[[1]][['MODERATOR:']][[80]][1])
```

```
## [1] " Your two minutes is..."
```

```
print(extracted_sentence[[1]][['MODERATOR:']][[81]][1])
```

```
## [1] " ... is expired."
```

- The regular expression takes sentences begin with a space or not into consideration.

```
print(extracted_sentence[[1]][['MODERATOR:']][[95]])
```

```
## [1] " Mr. Trump, very quickly, same question. "
## [2] "Will you accept the outcome as the will of the voters?"
```

(c)

For each candidate, for each debate, count the number of words and characters and compute the average word length for each candidate.

Firstly, count the number of words for each candicate for each debate.

```
count_word <- function(vector){
  # extract all the words splitted by whitespace
  vector_word <- sapply(vector, str_extract_all,
                        pattern = '[[:space:]][a-zA-z0-9]+')
  # count the number
  num_word <- lapply(vector_word,length)
  vec_num <- unlist(num_word)
  table_num <- table(names(vec_num),vec_num)
  df_num <- data.frame(table_num)

  # count the total number by multiplying the number and frequency
  df_num['word_num'] <- as.numeric(as.character(df_num$vec_num))*
    as.numeric(as.character(df_num$Freq))
  # using groupby to acquire the sum of data processed above
  result_num <- df_num %>% group_by(Var1) %>% summarize(sum_wordnum = sum(word_num))
  names(result_num)[1]<-'speaker'

  # exclude the moderators' data
  result_num <- result_num[result_num['speaker']!='LEHRER:'&
                           result_num['speaker']!='MODERATOR:'&
                           result_num['speaker']!='HOLT:',]
  return(result_num)
}

test_that('test year 2016 works',
          expect_true(is.list(count_word(debates_dialogue[[1]]))))

counted_word = lapply(debates_dialogue, FUN = count_word)
names(counted_word)<-c(2016,2012,2008,2004,2000,1996)
counted_word
```

```
## $`2016`
## # A tibble: 2 x 2
##   speaker  sum_wordnum
##   <fct>          <dbl>
## 1 CLINTON:        6233
## 2 TRUMP:          8320
##
## $`2012`
## # A tibble: 2 x 2
##   speaker sum_wordnum
##   <fct>         <dbl>
## 1 OBAMA:         7143
## 2 ROMNEY:        7613
##
## $`2008`
## # A tibble: 2 x 2
##   speaker sum_wordnum
##   <fct>         <dbl>
## 1 MCCAIN:        6966
## 2 OBAMA:         7438
##
## $`2004`
## # A tibble: 2 x 2
##   speaker sum_wordnum
##   <fct>         <dbl>
## 1 BUSH:          6142
## 2 KERRY:         6921
##
## $`2000`
## # A tibble: 2 x 2
##   speaker sum_wordnum
##   <fct>         <dbl>
## 1 BUSH:          7403
## 2 GORE:          7170
##
## $`1996`
## # A tibble: 2 x 2
##   speaker  sum_wordnum
##   <fct>          <dbl>
## 1 CLINTON:        7593
## 2 DOLE:           8029
```

Then, count the number of characters for each candicate for each debate.

```
# Using the similar way to count the total number of character including letters and digits
count_character <- function(vector){
  vector_character <- sapply(vector, str_extract_all,
                             pattern = '[a-zA-z0-9]')
  num_character <- lapply(vector_character,length)
  vec_num <- unlist(num_character)
  table_num <- table(names(vec_num),vec_num)

  df_num <- data.frame(table_num)
  df_num['character_num'] <- as.numeric(as.character(df_num$vec_num))*
    as.numeric(as.character(df_num$Freq))
  result_num <- df_num %>% group_by(Var1) %>% summarize(sum_characternum = sum(character_num))
  names(result_num)[1]<-'speaker'
  result_num <- result_num[result_num['speaker']!='LEHRER:'&
                           result_num['speaker']!='MODERATOR:'&
                           result_num['speaker']!='HOLT:',]
  return(result_num)
}

test_that('test the number of years',
          expect_true(length(lapply(debates_dialogue, FUN = count_character))==6))

counted_character = lapply(debates_dialogue, FUN = count_character)
names(counted_character)<-c(2016,2012,2008,2004,2000,1996)
counted_character
```

```
## $`2016`
## # A tibble: 2 x 2
##   speaker  sum_characternum
##   <fct>               <dbl>
## 1 CLINTON:            27291
## 2 TRUMP:              35632
##
## $`2012`
## # A tibble: 2 x 2
##   speaker sum_characternum
##   <fct>              <dbl>
## 1 OBAMA:             32212
## 2 ROMNEY:            33435
##
## $`2008`
## # A tibble: 2 x 2
##   speaker sum_characternum
##   <fct>              <dbl>
## 1 MCCAIN:            31347
## 2 OBAMA:             33116
##
## $`2004`
## # A tibble: 2 x 2
##   speaker sum_characternum
##   <fct>              <dbl>
## 1 BUSH:              27160
## 2 KERRY:             30359
##
## $`2000`
## # A tibble: 2 x 2
##   speaker sum_characternum
##   <fct>              <dbl>
## 1 BUSH:              31928
## 2 GORE:              31253
##
## $`1996`
## # A tibble: 2 x 2
##   speaker  sum_characternum
##   <fct>               <dbl>
## 1 CLINTON:            33371
## 2 DOLE:               34624
```

Compute the average word length for each candidate using the number of words and the number of characters derived from above. Call the two self function constructed above and count the average word length by deviding.

```r
average_word_len <- function(vector){
  df_word <- data.frame(count_word(vector))
  df_character <- data.frame(count_character(vector))
  result_df <- data.frame(speaker = df_word['speaker'],
                        average_word_len=
                          df_character['sum_characternum']/df_word['sum_wordnum'])
  return(result_df)
}
# vectorized way

test_that('test the number of dataframe',
          expect_true(any(names(average_word_len(debates_dialogue[[1]]))=='speaker')))

average_len = lapply(debates_dialogue, FUN = average_word_len)
names(average_len)<-c(2016,2012,2008,2004,2000,1996)
average_len
```

```
## $`2016`
##    speaker sum_characternum
## 1 CLINTON:         4.378469
## 2   TRUMP:         4.282692
##
## $`2012`
##   speaker sum_characternum
## 1  OBAMA:          4.50959
## 2 ROMNEY:          4.39183
##
## $`2008`
##   speaker sum_characternum
## 1 MCCAIN:         4.500000
## 2  OBAMA:         4.452272
##
## $`2004`
##   speaker sum_characternum
## 1   BUSH:         4.422012
## 2  KERRY:         4.386505
##
## $`2000`
##   speaker sum_characternum
## 1   BUSH:         4.312846
## 2   GORE:         4.358856
##
## $`1996`
##    speaker sum_characternum
## 1 CLINTON:         4.394969
## 2    DOLE:         4.312368
```

*Comments All the average word length of around 4.5. Democratic candidates' average length is a little more than that of Republican candidates in four of these six years and the other two years the contrary is the case. The difference is pretty small and always smaller than 0.1 or a little more than 0.1. So in my opinion, speech diction strategy of one side depends on the other candidate's choice of words on the spot.

(d)

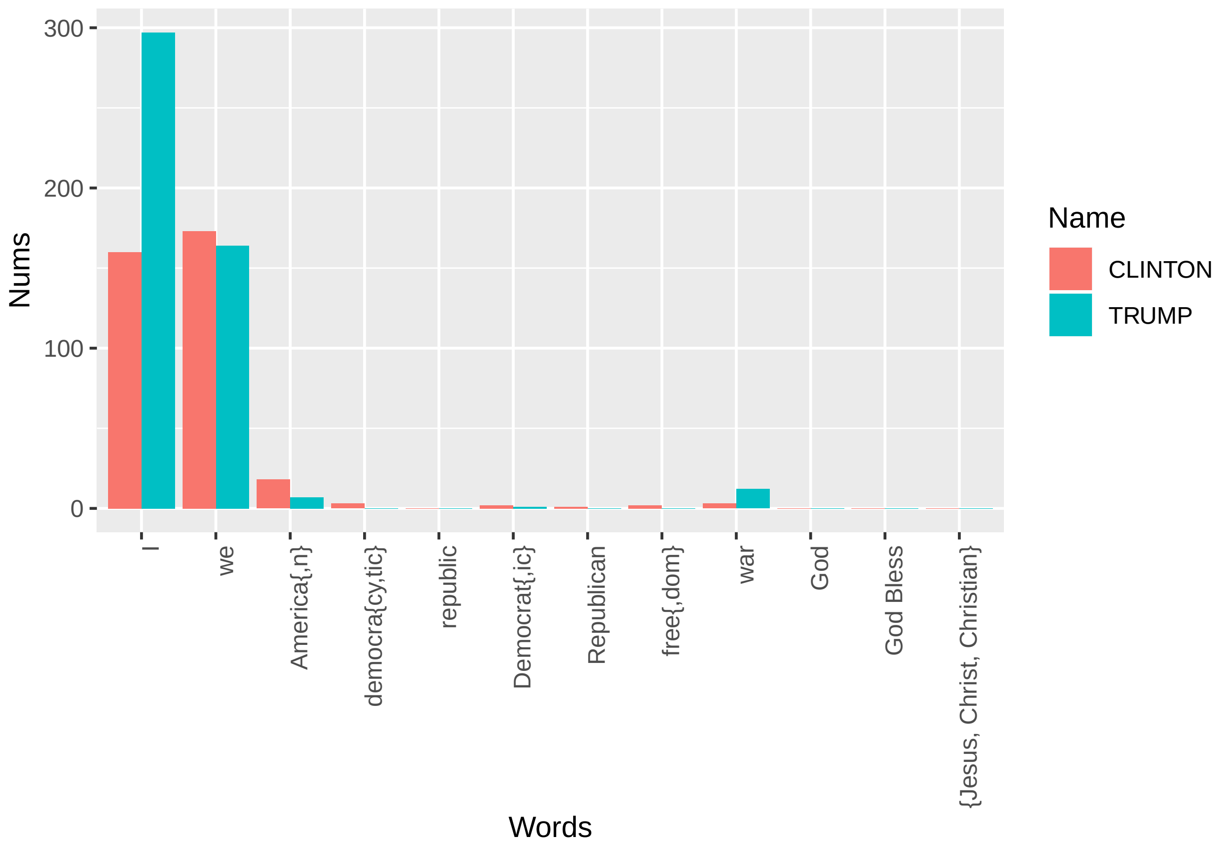
For each candidate, count the following words or word stems and store in a multi-layer list: I, we, America{,n}, democra{cy,tic}, republic, Democrat{,ic}, Republican, free{,dom}, war, God [not including God bless], God Bless, {Jesus, Christ, Christian}. Because the structure of target list is multi-layer, so I self-define multi-layer functions respectively handle with the single word in inner function and the single year in external function to avoid looping.

```r
# use regular expression to extract the special word
count_special_word <- function(vector){
  # in order to extract the special words followed by punctuations
  extract_I <- str_extract_all(vector , pattern = "(I)(\\s|\\'|\\.|\\?)")
  # use length() function in a vectorized way
  # use sum() function to acquire the total number of occurances of each specified word
  count_I <- sum(sapply(extract_I , length))

  extract_we <- str_extract_all(vector , pattern = regex("(we)(\\s|\\'|\\.|\\?|\\,)" ,
                                                      ignore_case = TRUE))
  count_we <- sum(sapply(extract_we , length))

  extract_america <- str_extract_all(vector ,
                                     pattern = regex("(america)(n)?(\\s|\\'|\\.|\\?|\\,)" ,
                                                      ignore_case = TRUE))
  count_america <- sum(sapply(extract_america , length))

  extract_democra <- str_extract_all(vector ,
                              pattern = regex("(democra)((cy)|(tic))?(\\s|\\'|\\.|\\?|\\,)" ,
                                                      ignore_case = TRUE))
  count_democra <- sum(sapply(extract_democra , length))

  extract_republic <- str_extract_all(vector ,
                                  pattern = regex("(republic)(\\s|\\'|\\.|\\?|\\,)" ,
                                                      ignore_case = TRUE))
  count_republic <- sum(sapply(extract_republic , length))

  extract_Democrat <- str_extract_all(vector ,
                                  pattern = regex("(Democrat)(ic)?(\\s|\\'|\\.|\\?|\\,)" ,
                                                      ignore_case = TRUE))
  count_Democrat <- sum(sapply(extract_Democrat , length))

  extract_Republican <- str_extract_all(vector ,
                                  pattern = regex("(Republican)(\\s|\\'|\\.|\\?|\\,)" ,
                                                      ignore_case = TRUE))
  count_Republican <- sum(sapply(extract_Republican , length))

  extract_free <- str_extract_all(vector ,
                                  pattern = regex("(free)(dom)?(\\s|\\'|\\.|\\?|\\,)" ,
                                                      ignore_case = TRUE))
  count_free <- sum(sapply(extract_free , length))

  extract_war <- str_extract_all(vector ,
                                  pattern = regex("(war)(\\s|\\'|\\.|\\?|\\,)" ,
                                                       ignore_case = TRUE))
  count_war <- sum(sapply(extract_war , length))

  extract_god <- str_extract_all(vector ,
                                  pattern = regex("(God)(\\!|\\s|\\'|\\.|\\?|\\,)" ,
                                                       ignore_case = TRUE))
  count_god <- sum(sapply(extract_god , length))

  extract_bless <- str_extract_all(vector ,
                              pattern = regex("((God)\\s(Bless))(\\!|\\s|\\'|\\.|\\?|\\,)" ,
                                                       ignore_case = TRUE))
  count_bless <- sum(sapply(extract_bless , length))

  extract_other <- str_extract_all(vector ,
                      pattern = regex("(Jesus|Christ|Christian)(\\!|\\s|\\'|\\.|\\?|\\,)" ,
                                                       ignore_case = TRUE))
  count_other <- sum(sapply(extract_other , length))

  # put all the data into a single list
  count_list<-list(count_I,
                   count_we,
                   count_america,
                   count_democra,
                   count_republic,
                   count_Democrat,
                   count_Republican,
                   count_free,
                   count_war,
                   count_god,
                   count_bless,
                   count_other)
  # assign the names of this list
  names(count_list)<-c('I','we','America{,n}','democra{cy,tic}',
                       'republic','Democrat{,ic}','Republican',
                       'free{,dom}', 'war', 'God' , 'God Bless',
                       '{Jesus, Christ, Christian}')
  return(count_list)
}

# self-define function to handle with the data each year and retuen the whole list
count_each_candidate <- function(year){
  year_count <- sapply(year, count_special_word)
  return(year_count)
}

test_that('test the number of dataframe',
          expect_true(length(count_special_word(debates_dialogue[[1]]))==12))

# select the list excluding the names of moderators
speak_candidate <- lapply(speak_chunk,'[',c(1,2))
lapply(speak_candidate, count_each_candidate)
```

```
## $`2016`
##                             CLINTON: TRUMP:
## I                           160      297
## we                          173      164
## America{,n}                 18       7
## democra{cy,tic}             3        0
## republic                    0        0
## Democrat{,ic}               2        1
## Republican                  1        0
## free{,dom}                  2        0
## war                         3        12
## God                         0        0
## God Bless                   0        0
## {Jesus, Christ, Christian}  0        0
##
## $`2012`
##                             OBAMA: ROMNEY:
## I                           118    217
## we                          182    124
## America{,n}                 18     33
## democra{cy,tic}             3      2
## republic                    0      0
## Democrat{,ic}               4      4
## Republican                  5      5
## free{,dom}                  3      7
## war                         3      0
## God                         0      1
## God Bless                   0      0
## {Jesus, Christ, Christian}  0      0
##
## $`2008`
##                             OBAMA: MCCAIN:
## I                           144    212
## we                          267    168
## America{,n}                 13     18
## democra{cy,tic}             1      1
## republic                    0      0
## Democrat{,ic}               1      1
## Republican                  2      2
## free{,dom}                  2      3
## war                         13     9
## God                         0      0
## God Bless                   0      0
## {Jesus, Christ, Christian}  0      0
##
## $`2004`
##                             KERRY: BUSH:
## I                           196    177
## we                          150    170
## America{,n}                 42     24
## democra{cy,tic}             2      4
## republic                    0      0
## Democrat{,ic}               0      0
## Republican                  1      0
## free{,dom}                  4      37
## war                         37     24
## God                         1      1
## God Bless                   1      0
## {Jesus, Christ, Christian} 0      0
##
## $`2000`
##                             GORE: BUSH:
## I                           229   213
## we                          96    109
## America{,n}                 13    19
## democra{cy,tic}             2     1
## republic                    0     0
## Democrat{,ic}               1     2
## Republican                  1     1
## free{,dom}                  1     3
## war                         7     4
## God                         0     0
## God Bless                   0     0
## {Jesus, Christ, Christian} 0     0
##
## $`1996`
##                             CLINTON: DOLE:
## I                           242      274
## we                          159      166
## America{,n}                 34       42
## democra{cy,tic}             6        5
## republic                    0        0
## Democrat{,ic}               2        7
## Republican                  7        11
## free{,dom}                  8        1
## war                         7        7
## God                         0        1
## God Bless                   0        1
## {Jesus, Christ, Christian} 0        0
```

Make a plot and comment briefly on the above results. I just pick a typical year to analyse the plot because it is kind of difficult to acquire a plot including completely three-dimension data obtained above.

```r
df_word <- data.frame(Name = rep(c('CLINTON','TRUMP'),each=12),
                      Nums = c(160,173,18,3,0,2,1,2,3,0,0,0,
                               297,164,7,0,0,1,0,0,12,0,0,0),
                      Words=c('I','we','America{,n}','democra{cy,tic}',
                              'republic','Democrat{,ic}','Republican',
                              'free{,dom}','war','God','God Bless',
                              '{Jesus, Christ, Christian}'))
ggplot(data=df_word, aes(x=Words, y=Nums, fill=Name)) +
geom_bar(stat="identity", position=position_dodge()) +
  scale_x_discrete(limits=c('I','we','America{,n}','democra{cy,tic}',
                              'republic','Democrat{,ic}','Republican',
                              'free{,dom}','war','God','God Bless',
                              '{Jesus, Christ, Christian}'))+
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Comment

The most mentioned words are I, we, America and war. On the contrary, the least mentioned words are God, Jesus and similar religious words. From my perspective, candidates try to be patriotic, self-motivated and irreligious, which exhibits features of mainstream and avoids being biased by voters of other religious sects. Besides, war is a main topic of debates. Candidates try to be anti-war and peaceful, which is widely accepted by the public. Another thing interesting is that words stem from republic and democrat are also rare. In my opinion, it may result from the fact that candidates do not want to win the popular support by belittling the other party or discussing the policies of the other party. Candidates try to express their own notion and policies objectively.

e)

Please include unit tests for your various functions. For the sake of time, you can keep this to a test or two for each function.

The test_that functions are included into self-define functions above.

Answers for Problem 3

Class I Called debates_text

I will construct a class called debates_text and regard debates body of each year and year as objects of the designed class. (e.g. one can pass along the debates_body[[1]] as an object into the debates_text class) Fields:

- Year (self$year)

- Content of debates such as self$text.
- names of candidates such as self$dem, self$rep and self$mod. These fields can be set as NULL and get their values from methods and functions.
- Individual chunks of text spoken by each speaker such as self$dem_chunk, self$rep_chunk and self$mod_chunk. These fields also can be set as NULL and get their values from methods which comes up later.

Functions and Methods:

- Strip out the non-spoken text and first paragraph which is not spoken text as well. (This method is optional because when we want to count the number of laughter and applause, we should not use this method to clean the character string)
- Get the names of candidates and moderators (extract names before colon and other information including candidates and moderators list mentioned in problem 2)
- Obtain the character vectors or individual chunks of text spoken by each speaker (using similar way as mentioned in 2(a))

Class II Called debates_chunk

Additional class which is needed to represent the spoken chunks and metadata on the chunks for the candidates takes individual chunk of text spoken by each speaker and the name of candidate as initialized objects. And these objects can be acquired from Class I.

Fields:

- Name of the specific candidate
- Content of the spoken chunk (in order to represent the spoken chunks, one can define a standard print function in methods)
- Number of laughter, number of applause, number of special words. These fields can be set as NULL and get their values from methods.
- Average length of words, number of sentences in chunk, number of words in chunk. These fields can be set as NULL and get their values from methods.

Functions and methods:

- Count the number of laughter, applause and special words using similar ways mentioned in problem 2 and assign the result to corresponding fields.
- Extact all the words and characters in the chunk. Then count the number and assign the quotient to field of average length of words. The number of words and characters should be assigned to fields of number of sentences in chunk and number of words in chunk.
- Print all the words and characters extracted in last method, although it is not necessary.

In these two classes, all methods and fields can be public because there is little concern about privacy in the problem.