PS3

ShubeiWang

9/18/2018

1

(a)

I read the preface and workflow template chapter of The Practice of Reproducible Research. It illustrates the necessity and significance of reproducible research. Hard as it is, we should try to work as transparently and reproducibly as possible so that our claims or results can be replicated by others. To achieve that, it's important to establish the basic workflow template. The basic reproducible research workflow contains three main stages: data acquisition, data processing and data analysis. In this chapter the author demonstrates this workflow by a simple project. I find this workflow very practical and useful as I walk through the whole process. It introduces a common convention of how the structure should be and how the files are named, etc. Also it emphasizes making the code readable and automating the process. I reckon that it would be more convenient and possible for me to use or assess others' work under this workflow.

But I do have some questions after reading it. Would this workflow applicable to all projects? What if their are some confidential data that others' don't have access to? Is there still a way to assess the work and automate the process? Also, sometimes the research uses some simulations to generate raw data and it's likely to produce different data every time. In that case, how would it be possible to verify the claim/results?

(d)

Strengths:

1. Their code has a good style in terms of spacing and naming, which makes it easier for others to read and understand it.
2. They use different files for each process and combine them in a main file. Also they provide intermediate data so that others don't need to download all the raw data(which can take hours).
3. They give a lot of comments and a thorough readme file to explain how their work was done.

Weaknesses:

1. They use absolute path in their code. It may not work on others' computers.
2. Their work is not modular. It's not convenient if others only need the outputs for one specific part.
3. There is no testing in their codes.

2

(a)

For this question I create a list called "debate_content" to store the chunks as a character vector for everyone. The structure of the list is like {2016:{candidate1: chunks, candidate2: chunks, moderator: chunks}, . . . , 1996:{. . . }}.

```r
source("ps3prob2.R")
```

```r
library(stringr)
library(assertthat)
library(testthat)

# deal with the repetition of transcript of 2018
debates_body[3] <- str_extract(debates_body[3],"(.*)and good night\\.")
# add "END" at the end of every transcript to make it easier to extract the ending chunk
fun <- function(x) return(paste0(x,"END"))
debates_body <- lapply(debates_body, fun)

## clean chunks
clean <- function(chunk,names){

  # remove all the non-spoken text
  chunk <- str_remove_all(chunk,"[(\\[].*[)\\]]")

  # remove all the "\n"
  chunk <- str_remove_all(chunk, "\\n")

  # remove all the \"
  chunk <- str_remove_all(chunk, '\\\"')

  # remove candidates' and moderators' names and "END"
  for(n in names){
    ex <- paste0(n,"(: )*")
    chunk <- str_remove(chunk,ex)
    chunk <- str_remove(chunk,"END")
  }

  # check if the chunk is empty
  assert_that(length(chunk)!=0, msg = "Check why this chunk is empty.")
  return(chunk)
}

## process the text of a certain year.
## returns a list: the first element is a list of the chunks of each person, and the second element is
## a list of times of laughter/applause

get_chunks <- function(candidate, moderator, text){

  # create an empty list to store the content for this year
  mylist1 <- vector("list",0)
  mylist2 <- vector("list",0)

  # split the debate content into a character vector for future use
  mytext <- str_split(text,"")[[1]]

  dem <- candidate[1]
  rep <- candidate[2]
  mod <- moderator
  names <- c(dem, rep, mod)
```

```
  pattern1 <- regex(paste0(dem,":(.*?)","(",rep,"|",mod,"|END)"), dotall = TRUE)
  pattern2 <- regex(paste0(rep,":(.*?)","(",dem,"|",mod,"|END)"), dotall = TRUE)
  pattern3 <- regex(paste0(mod,":(.*?)","(",dem,"|",rep,"|END)"), dotall = TRUE)
  pattern <- list(pattern1, pattern2, pattern3)

  for(j in 1:3){

    # specify whose response we're searching for
    name <- names[[j]]

    # create an empty character vector to store the chunks
    chunks <- c()
    laughter <- 0
    applause <- 0

    # find the positions of matching characters and store them in a matrix
    location <- str_locate_all(text, pattern[[j]])[[1]]
    location <- matrix(as.numeric(location),ncol=2)

    # find all his/her chunks
    loop <- nrow(location)
    for(k in 1:loop){

      start <- location[k,1]
      end <- location[k,2]
      indices <- start:end
      strs <- mytext[indices]
      chunk <- paste(strs, collapse = '')
      laughter <- laughter+str_count(chunk,regex("[(\\[]laughter[)\\]]",ignore_case = TRUE))
      applause <- applause+str_count(chunk,regex("[(\\[]applause[)\\]]",ignore_case = TRUE))
      chunks[k] <- clean(chunk,names)
    }

    mylist1[[j]] <- chunks
    mylist2[[j]] <- list("laughter"= laughter, "applause"= applause)
  }

  # name the list
  names(mylist1) <- names
  names(mylist2) <- names
  mylist <- list(mylist1, mylist2)
  return(mylist)
}


# create empty lists to store the content, times of laughter and applause for each person
debate_content <- list()
reply <- list()

# extract chunks for each year with a loop
for(i in 1:6){
  yr <- as.character(2020-4*i)
  candidate_yr <-candidates[[i]]
```

```
  moderator_yr <- moderators[i]
  text_yr <- debates_body[i]

  debate_content[[i]] <- get_chunks(candidate_yr,moderator_yr,text_yr)[[1]]
  reply[[i]] <- get_chunks(candidate_yr,moderator_yr,text_yr)[[2]]
}

names(debate_content) <- as.character(seq(2016,1996,-4))
names(reply) <- as.character(seq(2016,1996,-4))

# extract the information for candidates only
candidate_content <- debate_content
for(i in 1:6) candidate_content[[i]][[3]] <- NULL


# print out the number of chunks for each candidate
for(i in 1:6){
  yr <- names(candidate_content)[i]
  for(j in 1:2){
    name <- names(candidate_content[[i]])[j]
    print(paste0(yr," ",name," ", length(candidate_content[[i]][[j]])))
  }
}
```

```
## [1] "2016 CLINTON 87"
## [1] "2016 TRUMP 123"
## [1] "2012 OBAMA 42"
## [1] "2012 ROMNEY 54"
## [1] "2008 OBAMA 60"
## [1] "2008 MCCAIN 58"
## [1] "2004 KERRY 33"
## [1] "2004 BUSH 41"
## [1] "2000 GORE 49"
## [1] "2000 BUSH 56"
## [1] "1996 CLINTON 45"
## [1] "1996 DOLE 46"
```

(b)

Create a list "sentence_word" of the same structure as debate_content to store the sentences and words in character vectors.

```
## extract the sentences and words in each chunk

extract <- function(element){
  sentences <- unlist(str_extract_all(element,"[A-Z](.+?)[.?!](?= )"))
  words <- unlist(str_extract_all(element,"[a-zA-Z]+('[a-z]+)?"))
  ls <- list("sentences"=sentences, "words"=words)
  return(ls)
}

# test if function "extract" works well
test_that("extract function works well", expect_equal(extract("This is a test. Does it work? "),
```

```
list("sentences"= c("This is a test.", "Does it work?"),
"words"=c("This", "is", "a", "test", "Does", "it", "work"))))

# store sentences and words in a list which has the same structure as debate_content
sentence_word <- debate_content
for(i in 1:6)
    sentence_word[[i]] <- lapply(debate_content[[i]], extract)

## sanity check
sentence_word[[1]][[1]][[1]][1:3]
```

```
## [1] "How are you, Donald?"
## [2] "Well, thank you, Lester, and thanks to Hofstra for hosting us.The central question in this elec
## [3] "Today is my granddaughter's second birthday, so I think about this a lot."
```

```
sentence_word[[1]][[1]][[2]][1:30]
```

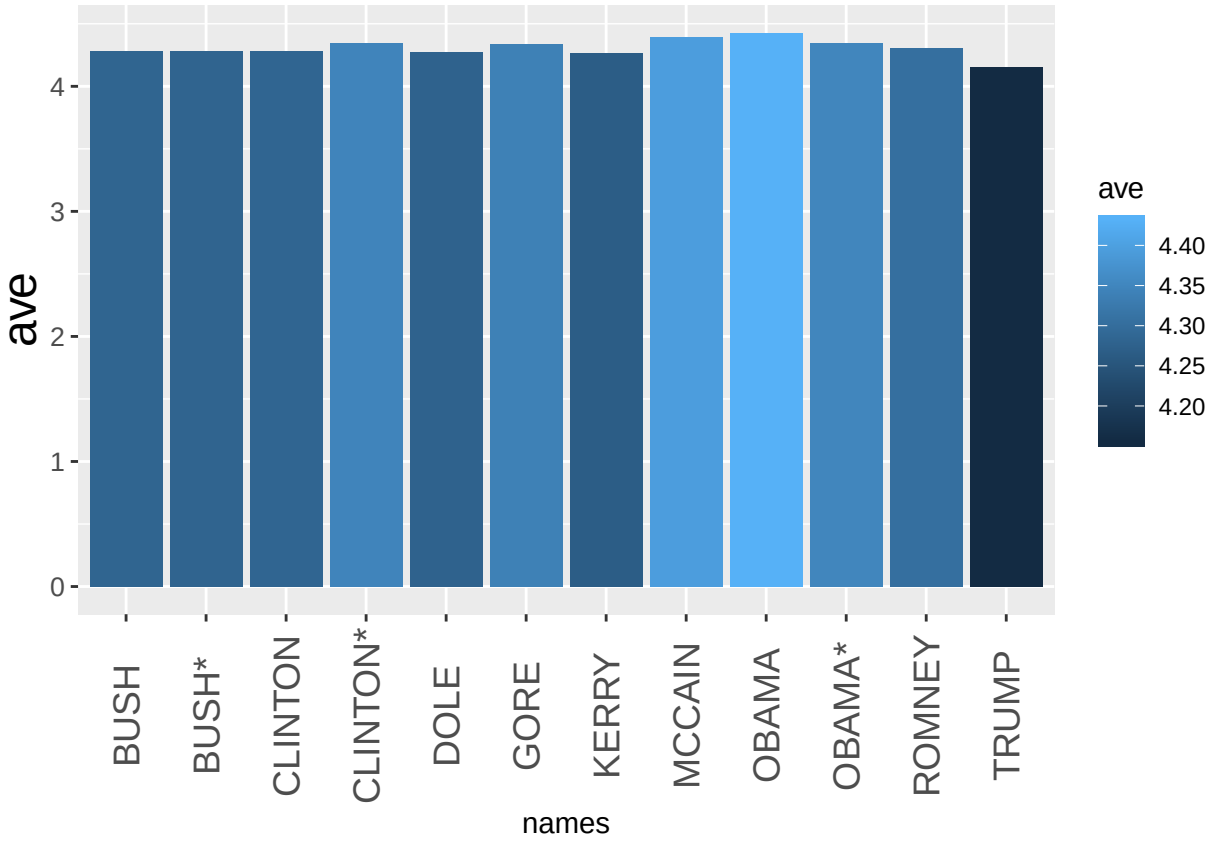
```
##  [1] "How"      "are"      "you"      "Donald"   "Well"     "thank"
##  [7] "you"      "Lester"   "and"      "thanks"   "to"       "Hofstra"
## [13] "for"      "hosting"  "us"       "The"      "central"  "question"
## [19] "in"       "this"     "election" "is"       "really"   "what"
## [25] "kind"     "of"       "country"  "we"       "want"     "to"
```

(c)

For each candidate, for each debate, I count the number of words and characters and compute the average word length and store all the information in the list called "count_list". Also I use a vector which consists of all average word length to see how it varies. The result shows that the mean of average word length is about 4.3 and the standard deviation is small. It seems the average word length of everyone is very close.

```
count_list <- candidate_content

## count the number of characters in each chunk
char_count <- function(element){
  count <- str_count(element,"[A-Za-z]")
  return(count)
}

# test function char_count works well
expect_equal(char_count("123Rmd"),3)
expect_equal(char_count("!?> null"),4)

## count the number of words and characters and compute the average word length for each chunk
num_count <- function(chunks){
  num_words <- length(chunks)
  num_char <- sum(unlist(lapply(chunks, char_count)))
  average_len <- num_char/num_words
  ls <- list("num_words"=num_words,"num_char"=num_char,"average_len"=average_len)
  return(ls)
}

# store all the average length in a vector called "average_length"
average_length <- c()
for(i in 1:6)
```

```
  for(j in 1:2){
  chunks <- lapply(sentence_word[[i]], '[[',2)[[j]]
  count_list[[i]][[j]] <- num_count(chunks)
  average_length[2*(i-1)+j] <- as.numeric(num_count(chunks)["average_len"])
  }


library(ggplot2)


names <- unlist(candidates)
names[5] <- "OBAMA*"
names[10] <- "BUSH*"
names[11] <-"CLINTON*"
names(average_length)<-names
mydata <- data.frame(names = names(average_length), ave = average_length)
ggplot(data = mydata, mapping = aes(x = names , y = ave, fill=ave)) + geom_bar(stat = 'identity')+theme
```




```
mean(average_length)
```

```
## [1] 4.308754
```

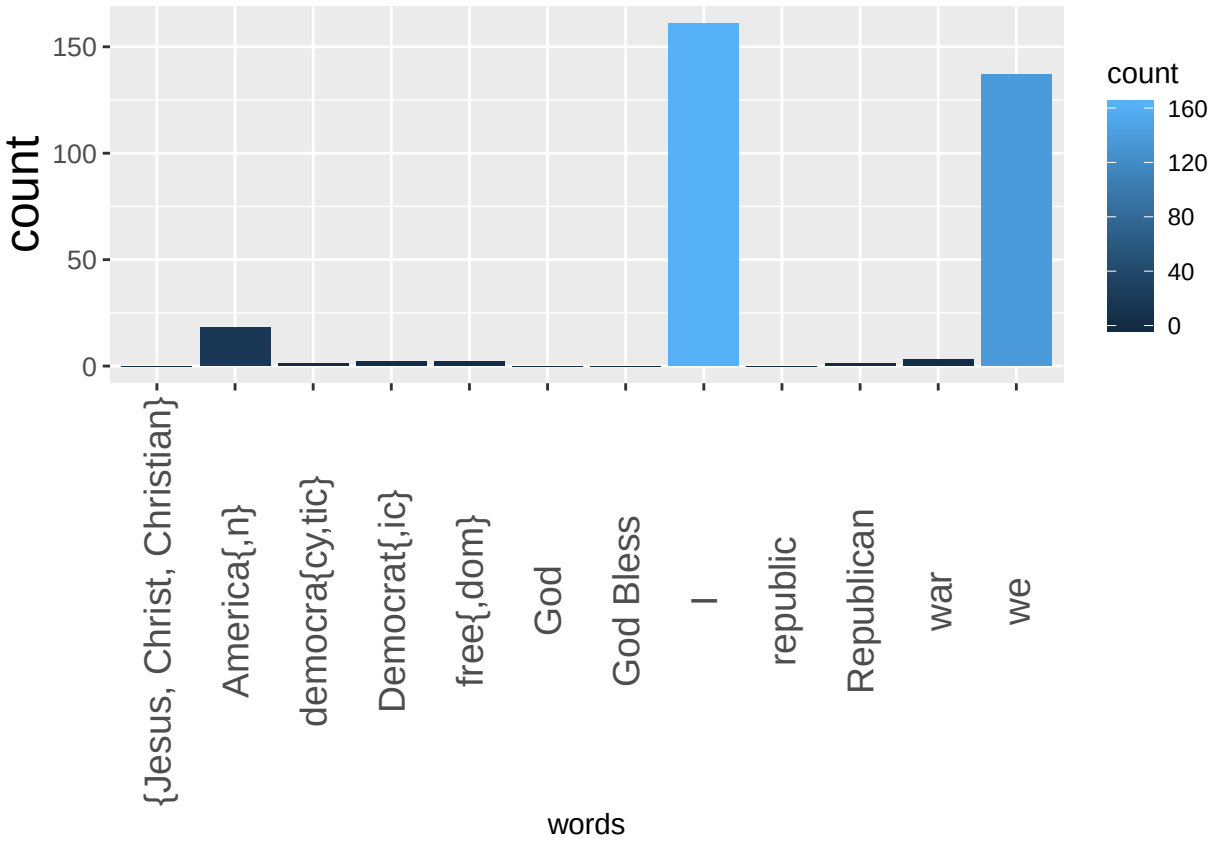
```
sd(average_length)
```

```
## [1] 0.06970935
```

(d)

For each candidate, I count the special words or word stems and store the information in a list called sp_words_count.

```
sp_words <- c("I", "we", "America{,n}", "democra{cy,tic}", "republic", "Democrat{,ic}", "Republican", "
sp_pattern <- c("(?<![[:alpha:]])I(?![[:alpha:]])",
                "(?<![[:alpha:]])we(?![[:alpha:]])",
                "(America|American)(?![[:alpha:]])",
                "(?<![[:alpha:]])(democracy|democratic)(?![[:alpha:]])",
                "(?<![[:alpha:]])republic(?![[:alpha:]])",
                "(Democrat|Democratic)(?![[:alpha:]])",
                "Republican(?![[:alpha:]])",
                "(?<![[:alpha:]])(free|freedom)(?![[:alpha:]])",
                "(?<![[:alpha:]])war(?![[:alpha:]])",
                "God(?![[:blank:]]bless)(?![[:alpha:]])",
                "God Bless(?![[:alpha:]])", "(Jesus|Christ|Christian)(?![[:alpha:]])")
## count the number of each word for one person
sp_count <- function(element){
  # test if the argument is a character string
  assert_that(is.character(element), msg = "wrong argument")
  ls <- list()
  for(word in sp_words){
    index <- which(sp_words == word)
    pattern <- paste0(sp_pattern[index])
    ls[[index]] <- sum(str_count(element,pattern))
  }
  names(ls) <- sp_words
  return(ls)
}
# make the list have the same structure as candidate_content
sp_words_count <- candidate_content
for(i in 1:6) sp_words_count[[i]] <- lapply(candidate_content[[i]], sp_count)
# make a plot for Clinton in 2016
clinton <- as.numeric(sp_words_count$`2016`$CLINTON)
mydata2 <- data.frame(count=clinton, words=sp_words)
ggplot(data = mydata2, mapping = aes(x = words , y = count, fill=count)) + geom_bar(stat = 'identity')+
```



3

class“debate”:

methods:

- get_chunk: uses “candidates”, “moderator” and input text and returns a list of the chunks for everyone
- get_spoken_chunks: clean the “chunks” and returns a list of the spoken chunks for the candidates
- reply: use “chunks” and returns a list of the counts of “laughter” and “applause” for candidates
- count: use “spoken chunks” and count the number of some special words

fields:

- candidates: a list of the names of candidates
- moderator: name the moderator
- chunks: store the chunks for everyone
- spoken_chunks: store the spoken chunks for the candidates
- sentences: store the sentences for candidates
- words: store the words for candidates
- word_count: store the number of words for candidates
- char_count: store the number of characters for candidates
- ave_length: store the average word length
- sp_count: store the number of some special words for candidates