STAT243 Problem set3

Tianyi Zhang

September 29, 2015

1 Problem1

The article I chose to read was "Best Practices for Scientific Computing" by Greg Wilson. I have a question about automating repetitive tasks in scientific computing. Generally, writing functions in R will automate repetitive tasks. However, in some cases, some tasks are similar in parts instead of purely repeating. In this scenario, do we make all efforts to spend more time writing a function as general (have quiet more inputs) as possible so that it would be suitable for all similar tasks? Another choice is to write similar functions to similar tasks which will decrease the reproducibility, but spend little time on single project. Moreover, the latter approach also corresponds to the advice that optimizing codes after it works correctly.
My second problem is related to the version control software Git. I found that it was a disaster to use Git and Dropbox (or other sync tools) simutaneously, if I had two computers working on the same project. The dropbox will sync first and then I use git pull there would be a conflict message.

2 Problem2

2.1 2A

I n Problem 2a, I was using regular expressions and XML tools to exract first Debates URLs and Years. In particular, I used a function called toString.XMLNode to transform my data type in nodes to string so that I can do regular expressions.
Besides, I obeserved that all debates happened in Sep or Ocb, which made it easier for me to grep the date. Finally, I wrote a function called selecturl that took year as an input,
and returned the URL for the first debate of that year.

```r
new_html<-htmlParse("http://www.debates.org/index.php?page=debate-transcripts")
##First observe that the text part of the website starts from <p>
listofnodes<-getNodeSet(new_html,"//p//a")
##toString.XMLNode transforms the list element to string so that
## it could be manipulated using regular expressions
stringnode<-unlist(lapply(listofnodes,toString.XMLNode))
selectyear<-stringnode[grep("1996|2000|2004|2008|2012",stringnode)]
first_html<-selectyear[grep("First",selectyear)]
first_html<-str_replace_all(first_html,".*http","http")
first_html<-str_replace_all(first_html,". title.*","")
Dateinfo<-selectyear[grep("First",selectyear)]
Dateinfo<-as.data.frame.Date(str_extract(
  Dateinfo,"(September|October) \\d+, \\d{4}"))
Speechdataframe<-cbind(as.data.frame(first_html),Dateinfo)
Speechdataframe[,2]=str_replace_all(
  Speechdataframe[,2],"(September|October) \\d+, ","")
colnames(Speechdataframe)<-c("first_URL","Year")
```

```
###Write a function about how to extract URL of a year given year as an input
select_url<-function(year){
  return(Speechdataframe[Speechdataframe[,2]==year,1])
}

Speechdataframe

##                                                                    first_URL
## 1    http://www.debates.org/index.php?page=october-3-2012-debate-transcript
## 2              http://www.debates.org/index.php?page=2008-debate-transcript
## 3 http://www.debates.org/index.php?page=september-30-2004-debate-transcript
## 4           http://www.debates.org/index.php?page=october-3-2000-transcript
## 5    http://www.debates.org/index.php?page=october-6-1996-debate-transcript
##   Year
## 1 2012
## 2 2008
## 3 2004
## 4 2000
## 5 1996
```

2.2 2B and 2C

In this section, I took the URLink as an input and returned a dataframe for future use.
This dataframe contained speakernames in the first column: like "OBAMA" "ROMNEY" "OBAMA", with no neighborhood the same (means no "OBAMA" "OBAMA"). In the second column, it's the raw text with laugther and applause tags. In the third column, I name it spoken text because it does not contain non-spoken texts.
Notice that I did eliminate the speaker names at first of some paragraphs, and to combine neighbor chunks by the same person to one chunk, I used a for loop. (I know groupby option in dplyr is a good option, but I am running a ubuntu with R 3.0, which did not support dplyr) By doing this, I can easily take subset of each candidate by data frame operations.

```
textbody<-function(year){
  speech_data<-htmlParse(select_url(year))
  ## By inspecting the Xpath Code of the element in Chrome.
  ## //p/text() will extract the body of the article
  text_data<-xpathSApply(speech_data,"//p/text()",xmlValue)
  ##Good Look
  # cat(paste(text_data,collapse="\n\n"))
  #This step concatenate all text together, and I extract all speaker names
  ## Then I split the original text by "Speakernames:", and throw out the first elemment of the list
  ## After that I created a data frame with names on the left and text on the right
  text_data<-paste(text_data,collapse=" ")
  snames<-as.list(str_replace(unlist(str_extract_all(text_data,"[A-Z]+:")),":",replacement=""))
  text_data<-str_split(text_data,pattern = "[A-Z]+: ")
  text_data<-unlist(text_data)[-1]

  finalframe<-data.frame(cbind(unlist(snames),text_data),stringsAsFactors = FALSE)
  index=1
  index_vec<-c(1)
  for(i in 2:nrow(finalframe)){
    if(finalframe[i,1]!=finalframe[i-1,1]){
      index=i
      index_vec<-c(index_vec,index)
    }
    if(finalframe[i,1]==finalframe[i-1,1]){
      finalframe[index,2]=paste(finalframe[index,2],finalframe[i,2],collapse="\n")
    }
  }
  finalframe<-finalframe[index_vec,]
  rownames(finalframe)<-NULL
  ##This line of code eliminates the non-spoken text. Such as Laughter...
  ## I am, However, willing to retain those information. Thus I place the new text in a new column
  ## By saying that, I will compute the # of tags for each candidate in future steps.
  finalframe[,3]<-str_replace_all(finalframe[,2],
                                  "\\(LAUGHTER\\)|\\(APPLAUSE\\)|\\(Applause\\)|\\(Laughter\\)|\\(CROSS
  colnames(finalframe)<-c("speakernames","raw text","spoken text")
  return(finalframe)
}
```

2.3 2D

In this section, I created a function called splitword to split the text into words and add it as the fourth column in my dataframe, the sentence split is in the fifth column. For simplicity, I will just show the first few words and first few sentences Obama said in 2012 to show that my split is useful.

```
split_word<-function(finalframe){
  withoutpunc<-str_replace_all(finalframe[,3],pattern="\\.|\\,|\\.\\.\\.|\\?|\\!|\\ --|\\ (?![A-Za-z0-9
  # wordsplit<-lapply(withoutpunc,function(x){return(str_split(x,pattern="\\ "))})
  wordsplit<-str_split(withoutpunc,pattern = "\\ ")
  finalframe[,3]<-str_replace_all(finalframe[,3],"Mr\\.","Mr")
  finalframe[,3]<-str_replace_all(finalframe[,3],"Dr\\.","Dr")
  sentencesplit<-str_split(finalframe[,3],pattern = "\\. |\\! |\\? |\\.\\.\\. ")
  newframe<-cbind(as.list(finalframe[,1]),as.list(finalframe[,2]),
                as.list(finalframe[,3]),wordsplit,sentencesplit)
  finalframe<-newframe
  colnames(finalframe)<-c("speakernames","raw text","spoken text","wordsplit","sentencesplit")
  finalframe<-data.frame(finalframe,stringsAsFactors = FALSE)
  return(finalframe)
}
example<-textbody(2012)
example<-split_word(example)
##First few words or sentences OBAMA said in 2012 to
## show that I am able to correspond words to candidates
head(unlist(example[example[,1]=="OBAMA",4]))

## [1] "Well"  "thank" "you"   "very"  "much"  "Jim"

head(unlist(example[example[,1]=="OBAMA",5]))

## [1] "Well, thank you very much, Jim, for this opportunity"
## [2] "I want to thank Governor Romney and the University of Denver for your hospitality"
## [3] "There are a lot of points I want to make tonight, but the most important one is that 20 years ag
## [4] "And so I just want to wish, Sweetie, you happy anniversary and let you know that a year from no
## [5] " You know, four years ago we went through the worst financial crisis since the Great Depression
## [6] "Millions of jobs were lost, the auto industry was on the brink of collapse"
```

2.4 2E and 2F

In this section, I made a function that would take finalframe from last step, and count the words of each candidate and other basic statistics like number of laughters and applauses. To achieve this I start with an empty data frame with all row names and columnames set, then I insert the result to these dataframe by counting the number of occurence using regular expressions. Notice that it's still complex for me to use lapply here because I use regular expression over different columns of my dataframe.

```r
##Part E and F, and Also count the number of tags
###Write a function that will return the data required for a speech.
Candidate_stat<-function(finalframe){
  ##Store speaker names to a vector
  speaker_unique<-unlist(unique(finalframe[finalframe[,1]!="SPEAKERS",1]))
  ##Create an empty data frame to store number of words, average length, etc.
  candidate_data<-data.frame(matrix(numeric(0),ncol=17,nrow=3),stringsAsFactors=FALSE)
  colnames(candidate_data)<-c("wordcount","charachtercount","averagelength",
                              "I","we","American","democracy","republic",
                              "Democrat","Republican","freedom",
                              "war","Jesus","God","GodBless","Laughter","Applause")
  rownames(candidate_data)<-speaker_unique
  ##Now all splitting in word is in the third column of the finalframe
  ## for loop looping from 1 to 3, namely moderator and each candidate
  ## The regexvector contains the basic regular expressions for use, some special ones
  ## will be dealt with seperately.
  regexvector<-c("I[^a-z]","[W|w]e[^a-z]","American?","democracy\\b|democratic\\b",
                 "[R|r]epublic\\b","Democrats?[ic]?","Republicans?",
                 "[F|f]ree[dom]?","[W|w]ars?","Jesus|Christs\\b|Christians?")
  for (i in 1:length(speaker_unique)){
    name=speaker_unique[[i]]
    word_candidate=unlist(finalframe[finalframe[,1]==name,4])
    text_candidate=unlist(finalframe[finalframe[,1]==name,3])
##In order to count Laughters and Applause tags
    raw_candidate=unlist(finalframe[finalframe[,1]==name,2])
##First 3 columns
    candidate_data$wordcount[i]<-length(word_candidate)
    candidate_data$charachtercount[i]<-sum(nchar(word_candidate))
    candidate_data$averagelength[i]=candidate_data$charachtercount[i]/candidate_data$wordcount[i]

    for (k in 1:length(regexvector)){
      candidate_data[i,k+3]<-sum(str_count(word_candidate,pattern=regexvector[k]))
    }

    #### Since God bless has two words, we need to use main text to count.
    candidate_data$God[i]<-sum(str_count(text_candidate,"[G|g]od (?!bless)"))
    candidate_data$GodBless[i]<-sum(str_count(text_candidate,"[G|g]od bless"))
    ###This is one of part c in the problem.
    candidate_data$Laughter[i]<-sum(str_count(raw_candidate,"\\(LAUGHTER\\)|\\(Laughter\\)"))
    candidate_data$Applause[i]<-sum(str_count(raw_candidate,"\\(APPLAUSE\\)|\\(Applause\\)"))
  }
  return(candidate_data)
}

###Combine all functions together, the stat table is the table of statistics
```

```r
main<-function(year){
  finalframe<-textbody(year)
  aftersplit<-split_word(finalframe)
  stat_table<-Candidate_stat(aftersplit)
  rownames(stat_table)<-paste(rownames(stat_table),year)
  return(stat_table)
}
result<-lapply(c(2012,2008,2004,2000,1996),main)
result
```

```
## [[1]]
##             wordcount charachtercount averagelength  I we American
## LEHRER 2012      1524            6834      4.484252 11 17        1
## OBAMA 2012       7239           32608      4.504490 26 65       24
## ROMNEY 2012      7729           33928      4.389701 69 34       41
##             democracy republic Democrat Republican freedom war Jesus God
## LEHRER 2012         0        0        1          1       0   0     0   0
## OBAMA 2012          0        0        8          9       3  12     0   0
## ROMNEY 2012         1        0        7          9       7   3     0   1
##             GodBless Laughter Applause
## LEHRER 2012        0        0        1
## OBAMA 2012         0        3        0
## ROMNEY 2012        0        1        0
##
## [[2]]
##             wordcount charachtercount averagelength  I  we American
## LEHRER 2008      2740           12082      4.409489 30  20        0
## OBAMA 2008      15156           66906      4.414489 54 112       32
## MCCAIN 2008     14178           63344      4.467767 92  60       48
##             democracy republic Democrat Republican freedom war Jesus God
## LEHRER 2008         0        0        2          2       2   0     0   0
## OBAMA 2008          2        0        0          6      10  42     0   0
## MCCAIN 2008         2        0        6         14       8  36     0   0
##             GodBless Laughter Applause
## LEHRER 2008        0        2        4
## OBAMA 2008         0        0        0
## MCCAIN 2008        0        2        0
##
## [[3]]
##             wordcount charachtercount averagelength  I we American
## LEHRER 2004      1365            6598      4.833700  8  2        3
## KERRY 2004       7084           30708      4.334839 51 25       46
## BUSH 2004        6298           27499      4.366307 28 57       24
##             democracy republic Democrat Republican freedom war Jesus God
## LEHRER 2004         1        0        1          1       0   4     0   0
## KERRY 2004          2        0        0          1       4  46     0   0
## BUSH 2004           4        0        0          0      38  27     0   1
##             GodBless Laughter Applause
## LEHRER 2004        0        0        2
## KERRY 2004         1        2        0
## BUSH 2004          0        1        0
##
## [[4]]
##                wordcount charachtercount averagelength  I we American
## MODERATOR 2000      1685            7843      4.654599  9 10        0
## GORE 2000           7170           31520      4.396095 36 16       16
## BUSH 2000           7398           32314      4.367937 46 27       26
##                 democracy republic Democrat Republican freedom war Jesus
## MODERATOR 2000         1        0        1          1       0   1     0
## GORE 2000              1        0        2          2       1   9     0
## BUSH 2000              1        0       12          9       4   6     0
##                 God GodBless Laughter Applause
## MODERATOR 2000   0        0        0        2
## GORE 2000        0        0        0        0
## BUSH 2000        0        0        0        0
## 
## [[5]]
##               wordcount charachtercount averagelength  I we American
## LEHRER 1996        1214            5585      4.600494  2  2        1
## CLINTON 1996       7357           32543      4.423406 34 40       36
## DOLE 1996          8083           35173      4.351478 69 50       50
##               democracy republic Democrat Republican freedom war Jesus God
## LEHRER 1996           0        0        1          2       0   2     0   0
## CLINTON 1996          4        0        1         10       8  17     0   0
## DOLE 1996             0        0       12         12       1   9     0   0
##               GodBless Laughter Applause
## LEHRER 1996          0        0        0
## CLINTON 1996         0        0        0
## DOLE 1996            1        0        0
```

From the table, we have obeserved that 2008 is an unusual case that every part of the script has been counted twice, so the statistics do. Namely, all statistics are even numbers, and it's unlucky that html is not structured. Besides, We can observe that "war" was mentioned significantly more times in 2004, and probably because the happening of the Iraq war. Bush also mentioned freedom a lot in 2004, which also related to the Iraq War. Besides in 2012, Obama got more laughters. Another interesting fact is about the average word length, candidates typically had a average word length of 4.5, which is less than the average word length in typical English documents (5.1), which probably because people tend to say easier and shorter words than writing.

3 Problem 3

3.1 3 A and B

Here I created a function called random walk without using for loops. One thing need to mention is that this function handles gracefully with wrong inputs such as nonintegers, negative numbers, etc.

```
set.seed(11)
randomwalk<-function(nstep=10,start=c(0,0),fullpath=TRUE){
  if (is.numeric(nstep) & nstep%%1==0 & nstep>0){
    randomvector=sample(c("Up","Down","Right","Left"),nstep,replace=TRUE)
    Updown=rep(0,nstep)
    Updown[randomvector=="Up"]=1
    Updown[randomvector=="Down"]=-1
    leftright=rep(0,nstep)
    leftright[randomvector=="Right"]=1
    leftright[randomvector=="Left"]=-1
    xcoordinates<-cumsum(leftright)+start[1]
    ycoordinates<-cumsum(Updown)+start[2]
    finalpos<-c(xcoordinates[nstep],ycoordinates[nstep])
    finalpath<-cbind(xcoordinates,ycoordinates)
    finalpath<-rbind(start,finalpath)
    rownames(finalpath)<-NULL
    if(fullpath==FALSE){
      return(finalpos)
    }
    else{
      return(finalpath)
    }
  }
  else{
    if(is.numeric(nstep) & nstep%%1!=0){
      stop("Your input should be an integer")
    }
    if(is.numeric(nstep) & nstep<=0){
      stop("Your input should be positive")
    }
    else{
      stop("Your input should be a positive integer")
    }
  }
}
randomwalk(10,fullpath=TRUE)
```

```
##       xcoordinates ycoordinates
##  [1,]            0            0
##  [2,]            0           -1
##  [3,]            0            0
##  [4,]            1            0
##  [5,]            1            1
##  [6,]            1            2
##  [7,]            0            2
##  [8,]            0            3
##  [9,]            0            2
## [10,]           -1            2
## [11,]           -1            3
```

```
##Illustration for wrong input
a<-randomwalk(20.5)
```

```
## Error in randomwalk(20.5):  Your input should be an integer
```

```
b<-randomwalk(-10)
```

```
## Error in randomwalk(-10):  Your input should be positive
```

3.2 3C

Then I use a class constructor to create an S3 class called rw, with two attributes in the object, path and final position.

```
walk <- function(nstep=10,start=c(0,0)){
  # constructor for 'rw' class
  path<-randomwalk(nstep,fullpath=TRUE)
  finalpos<-path[nrow(path),]
  obj <- list(finalpos=finalpos,path=path)
  class(obj) <- 'rw'
  return(obj)
}
walk1<-walk(50)
attributes(walk1)

## $names
## [1] "finalpos" "path"
##
## $class
## [1] "rw"
```

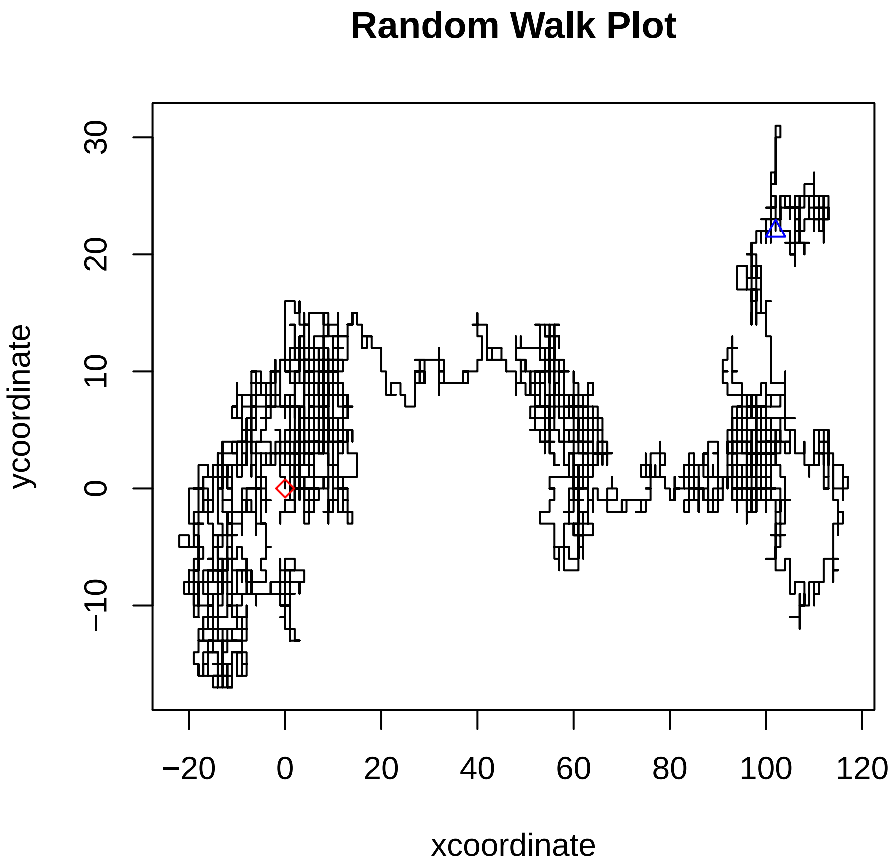
Here I constructed a print and plot method for rw class. In particular, for the plot part, I use the red point (square) to denote the starting point, and use the triangle point to denote the end point. The detail of code explanation is along side with the code. By doing these, I can use plot() and print() directly to rw class objects.

```
print.rw<-function(obj){
  cat("The starting point is:", toString(obj$path[1,]),"\n")
  cat("The end point is: ", toString(obj$path[nrow(obj$path),]))
}
print(walk1)

## The starting point is: 0, 0
## The end point is:  5, -5

plot.rw<-function(obj){
## This step presets an empty plot for future usage.
  plot(0,type="n",xlab="xcoordinate",ylab="ycoordinate",main="Random Walk Plot",
       xlim=range(obj$path[,1]),ylim=range(obj$path[,2]))
  lines(obj$path[,1],obj$path[,2])
  points(cbind(obj$path[1,1],obj$path[1,2]),col="red",pch=23)
  points(cbind(obj$path[nrow(obj$path),1],obj$path[nrow(obj$path),2]),col="blue",pch=24)
}

##more steps will bring more pretty plots
walk1<-walk(5000)
plot(walk1)
```



In this section I created a replacement method start and an operator method to find the ith step. I notice that for the start part, we have to minus the orginal starting point coordinates, so that these operations can be done multiple times.

```
`start<-` <- function(object ,...) UseMethod("start<-");

`start<-.rw` <- function(obj, value){
  obj$path[,1]=obj$path[,1]+value[1]-obj$path[1,1]
  obj$path[,2]=obj$path[,2]+value[2]-obj$path[1,2]
  return(obj)
}
start(walk1)<-c(5,7)
##Print first ten rows of object path for illustration
walk1$path[1:10,]

##       xcoordinates ycoordinates
##  [1,]            5            7
##  [2,]            5            8
##  [3,]            5            7
##  [4,]            4            7
##  [5,]            3            7
##  [6,]            4            7
##  [7,]            5            7
##  [8,]            5            6
##  [9,]            5            5
## [10,]            5            6
```

```
'[.rw'<-function(object,i){
  obj<-object$path
  class(obj)<-"matrix"
  return(obj[i+1,])
}
walk1[3]
```

```
## xcoordinates ycoordinates
##            4            7
```