STAT243-PS3

Jinhui Xu

September 2017

1 Other students

I discuss some problems with Xin Shi.

2 Question 1

Material four : Millman and Perez

The article emphasizes the importance of good practice of computational work. It first introduces its vision for scientific software development in everyday research. Then gives specific recommendations for computational work.

I think the article is meaningful. As when I was confused about my project thesis, I would search solutions on the internet. However, the code I found usually did not work because I did not know how to use the data and which tools should I use. Therefore I think that good practice of computational work is necessary.

The article gives recommendations on each step of research work from initial exploration of ideas and data to final results,including Individual exploration, Collaboration, Production-scale execution, Publication and Education.

But I am not familiar with some tools and process the article mentions, so I have two questions:

Q1: the article says that the Python language has a simple, expressive, and accessible syntax that emphasizes code readability. I dont know what code readability means? Whether R or other languages do well in code readability?

Q2: In the section of collaboration, the article mentions the concept of distributed version control systems. Are Mercurial or other DVCS similar with git? Whether git is enough for us?

3 Question 2

3.1 (a)

In this section, I add each play into a list which call playlist. As at the begin of each play, there is year of play, so I grep the str start with four digits and get 38 years. Delete the first one and the last one, I can obtain plays by reading the content between two years.

```
#read the original txt and store it in text
text=readLines("http://www.gutenberg.org/cache/epub/100/pg100.txt")
#get rownumber of year in text and store 36 years in vector year_play,it is for question (b)
year_rownumber=grep('^[[:digit:]]{4}',text)
year_play=text[year_rownumber[2:(length(year_rownumber)-1)]]
```

```
#store each play in playlist,I believe that the content between year of play_i
#and year of play_i+1 is exactly the content of play_i
playlist=list()
for(i in 1:36) playlist[[i]]=text[(year_rownumber[i+1]):(year_rownumber[i+2]-1)]
```

3.2 (b)

I have get year_play in Q(a), and get the title of play by obtaining content in the second row (not NULL) of a play. The way to get the number of acts and the number of scenes is similar so I write a function named count_number(str_target,searchlist).str_target is a regex to be matched in searchlist. And the output is a list contain content_list and count_number.

```
#the second row is exactly the title of play.
title=c();
for(i in 1:36) title[i]=playlist[[i]][which(playlist[[i]]!="")[2]]
#get the number of scenes by locate the str , and the number of acts equels to the number of unique nam
#the function to get the number of acts and scenes
library(stringr)
count_number=function(str_target,searchlist){
  content_list=list()
  content_number=c()
  for(i in 1:length(searchlist)){
    str_locate=c()
    str_locate=grep(str_target,searchlist[[i]])
    content_list[[i]]=searchlist[[i]][str_locate]
    content_number[i]=length(str_locate)
  }
  list=list(content_list,content_number)
  return(list)
}
```

Run the function to get act_number and scene_number

```
scene_list=list();act_list=list();scene_number=c();act_number=c()
scene_str="[S][Cc][Ee][Nn][Ee][[:blank:]]"
act_str="A[Cc][Tt] "
scene_list=count_number(scene_str,playlist)[[1]]
scene_number=count_number(scene_str,playlist)[[2]]
act_list=count_number(act_str,playlist)[[1]]
act_number=c();for(i in 1: 36) act_number[i]=length(unique(str_extract_all(act_list[[i]],
'A[Cc][Tt][[:blank:]]?[[:alpha:]]*')))
```

Then extract the body and store five factors in a list. the information of play_i is stored in result[[i]]

```
body_list=list();
for(i in 1:36) body_list[[i]]=playlist[[i]][grep("^[[:blank:]]*[Ss][Cc][Ee][Nn][Ee][[:punct:]]"
,playlist[[i]]):length(playlist[[i]])]
result=list()
for(i in 1:36) {
  play_result=list()
  play_result[[1]]=year_play[i]
  play_result[[2]]=title[i]
  play_result[[3]]=act_number[i]
```

```
  play_result[[4]]=scene_number[i]
  play_result[[5]]=body_list[[i]]
  names(play_result)=c('year','title','number of acts','number of scenes','body')
  result[[i]]=play_result
}
```

view the result of play_1 except the body of play (too large)

```
result[[1]]$year

## [1] "1603"

result[[1]]$title

## [1] "ALLS WELL THAT ENDS WELL"

result[[1]]$'number of acts'

## [1] 5

result[[1]]$'number of scenes'

## [1] 23
```

3.3 (c)

First,update the body_list,delete the first 5 rows of body. I suppose that part of information are confused.

```
body_update=body_list
library(stringr)
for(i in 1:36){
  n=grep("[S][Cc][Ee][Nn][Ee][[:blank:]]",body_list[[i]])
  for(j in 1:length(n))
  body_update[[i]][n[j]:(n[j]+4)]='null'
}
```

get the unique speaker name.I suppose that every speaker's name begins with a upper,every chunk must have at least one lower.

```
speaker_first_content=list()
name=list()
speaker_locate=list()
speaker_number=c()
for(i in 1:36){

#delete several specific stage direction like Exit,Enter...
  body_update[[i]]=str_replace_all(body_update[[i]],'((Exit|Exeunt|Enter|Re-enter)([[:graph:]
[:blank:]])*)|(\\[([[:graph:][:blank:]])+\\])','')

#find the sentences that begin with a speaker's name plus [.]
  speaker_locate[[i]]=grep('^[[:blank:]]{0,4}[[:upper:]]+[[:lower:]]*([[:blank:]]*[[:upper:]]+
[[:lower:]]*)*[.][[:blank:]]*[[:upper:]][[:blank:]]*[[:punct:]]*([[:blank:]][[:upper:]])?[[:blank:]]*
[[:punct:]]?[[:lower:]]',body_update[[i]])
  speaker_first_content[[i]]=body_update[[i]][speaker_locate[[i]]]
```

```
#group each speaker's names in to list of name.and delete trashy blanks
  name[[i]]=unique(str_replace_all(str_extract(speaker_first_content[[i]],"^[[:blank:]]{0,4}
[[:upper:]]+[[:lower:]]*([[:blank:]]*[[:upper:]]*[[:lower:]]*){0,2}"),"^[[:blank:]]*",''))

#I suppose that  play's names are either all upper cases or all lower cases except the first one.
#So find plays whose names are all upper cases and delete confused 'names' that contains lower cases.
  if((length(grep('[[:lower:]]+',name[[i]]))/length(name[[i]]))<0.3&&(length(grep('[[:lower:]]+',
name[[i]]))/length(name[[i]]))>0)             name[[i]]=name[[i]][-grep('[[:lower:]]+',name[[i]])]

#calculate the number of unique speakers required in Q(d)
  speaker_number[i]=length(name[[i]])
}
```

Store all information of chunks in a list called chunk_name_play. chunk_name_play[[i]] contains all all chunks of each speaker in play_i, and chunk_name_play[[i]][[j]] contains all chunks of speaker_j in play_i.

```
chunk_name_play=list();
for(r in 1:36){
  chunk=list();for(k in 1:length(name[[r]])){
     # in play_r,get the location of speakers_k'names,store them in vector l
     l=c()
     l=grep(paste('^[[:blank:]]*',name[[r]][k],sep=''),body_update[[r]])
     rownumber=c();
     # use for loop to decide the length of each chunk
     for(i in 1:length(l)){
         j=0;rownumber[i]=1;
         # if the line does not begin with a name and it is not a empty line, then I
         #suppose it is the content of chunk.
          while((length(grep('^[[:blank:]]{0,4}[[:upper:]]+[[:lower:]]*([[:blank:]]*
[[:upper:]]+[[:lower:]]*)*[.]',body_update[[r]][(l[i]+j+1)]))==0)&&
(length(grep("[[:lower:]]",body_update[[r]][(l[i]+j+1)]))==1)){
             j=j+1
            rownumber[i]=rownumber[i]+1
          }
      }
      # namei contains the chunks of speaker_k in play_r
     namei=list()
      for(i in 1:length(l)){
         namei[[i]]=body_update[[r]][l[i]:(l[i]+rownumber[i]-1)]
         namei[[i]]=str_replace(namei[[i]],name[[r]][k],'')
         namei[[i]]=str_replace(namei[[i]],'[[:blank:]]{2,}','')
         namei[[i]]=str_replace(namei[[i]],'\\. ','')
      }
      # list of chunk store the information of play_r
      chunk[[k]]=namei
  }
  # all information is stored in chunk_name_play
  chunk_name_play[[r]]=chunk
}
```

3.4 (d)

Create vector that store the targeted result

```
chunk_number=c(rep(0,36))
sentences_number=c(rep(0,36))
words_number=c(rep(0,36))
unique_words=c()
```

Suppose that words are made up with alpha plus ; And sentences are splited by[,!];

```
word_extract=list()
sentences_extract=list()
for(i in 1:36){
  unique_words[i]=length(unique(unlist(str_extract_all(chunk_name_play[[i]],"[[:alpha:]]+
[']?[[:alpha:]]*"))))-1
  word_extract=str_extract_all(chunk_name_play[[i]],'[[:graph:]]+')
  sentences_extract=str_extract_all(chunk_name_play[[i]],'[.!]')
  for (j in 1:speaker_number[i]){
    chunk_number[i]=chunk_number[i]+length(chunk_name_play[[i]][[j]])
    sentences_number[i]=sentences_number[i]+length(sentences_extract[[j]])
    words_number[i]=words_number[i]+length(word_extract[[j]])
  }
}
```

Store result in a matrix called data_final.

```
words_perchunk=words_number/chunk_number
data_final=cbind(act_number,scene_number,speaker_number,chunk_number,sentences_number,words_perchunk,uni
```

3.5 (e)

First show the original data we get from the above question.

```
data_final
```

```
##       act_number scene_number speaker_number chunk_number sentences_number
##  [1,]          5           23             24          940             1075
##  [2,]          5           42             59         1179             1447
##  [3,]          5           22             26          811              943
##  [4,]          5           11             20          610              812
##  [5,]          5           29             62         1128             1311
##  [6,]          5           27             39          859             1149
##  [7,]          5           20             38         1233             1707
##  [8,]          5           19             42          861             1421
##  [9,]          5           19             49          941             1265
## [10,]          5           23             48          886             1155
## [11,]          5           27             54          665             1040
## [12,]          5           24             66          879             1295
## [13,]          5           28             46          821             1226
## [14,]          5           17             48          714              884
## [15,]          5           16             27          553              859
## [16,]          5           18             48          791             1137
## [17,]          5           26             22         1145             1774
## [18,]          5            9             19         1049             1302
## [19,]          5           29             42          648              870
## [20,]          5           17             22          884             1018
```

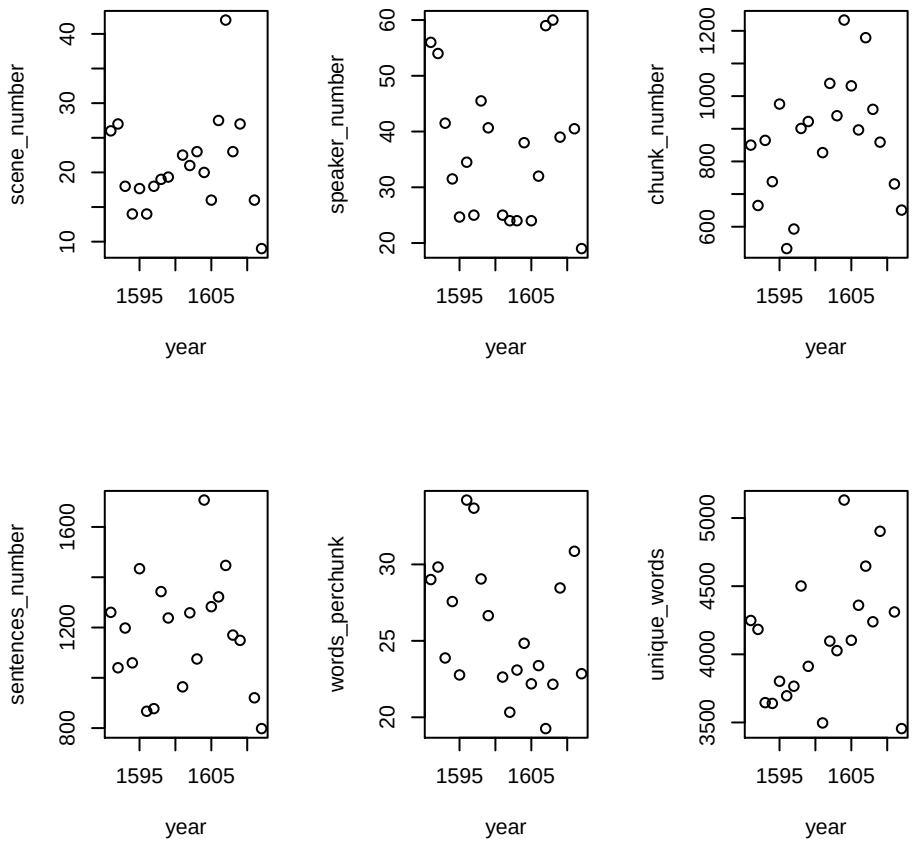
```
## [21,]          5           20             23          633             896
## [22,]          5           23             24          843             985
## [23,]          5            9             33          510             815
## [24,]          5           17             26         1090            1422
## [25,]          5           15             26         1179            1548
## [26,]          5           19             36          556             919
## [27,]          5           25             63         1119            1584
## [28,]          5           24             38         1021            2004
## [29,]          5           14             36          898            1159
## [30,]          5            9             19          651             798
## [31,]          5           17             58          791            1029
## [32,]          5           14             27          578             960
## [33,]          5           24             29         1154            1465
## [34,]          5           18             19          924            1052
## [35,]          5           20             17          857             996
## [36,]          5           15             33          748             957
##       words_perchunk unique_words
##  [1,]       23.09149         4027
##  [2,]       19.25869         4647
##  [3,]       25.17016         3657
##  [4,]       23.30820         2832
##  [5,]       22.92730         4740
##  [6,]       28.46333         4903
##  [7,]       24.84428         5132
##  [8,]       31.51800         4404
##  [9,]       26.57811         4599
## [10,]       34.99661         5017
## [11,]       29.82857         4183
## [12,]       30.37088         4585
## [13,]       27.65286         3912
## [14,]       31.21849         4202
## [15,]       35.45389         3897
## [16,]       23.31479         3306
## [17,]       22.29083         4886
## [18,]       19.39371         4117
## [19,]       24.47377         3834
## [20,]       22.96719         3807
## [21,]       31.89889         3635
## [22,]       20.08185         3336
## [23,]       30.98235         3347
## [24,]       21.65138         3413
## [25,]       21.41306         4399
## [26,]       37.43525         4044
## [27,]       24.45666         4459
## [28,]       29.54750         4261
## [29,]       21.86526         3565
## [30,]       22.84946         3455
## [31,]       21.39191         3738
## [32,]       33.27163         3715
## [33,]       20.92721         4736
## [34,]       19.73593         3458
## [35,]       19.36289         3030
## [36,]       30.50401         4420
```

We need to analyse the data by year, so I sort the year and mean the data in the same year

```
for(i in 1:7) data_final[,i]=as.numeric(data_final[,i])
year_sort=sort(unique(year_play))
n=length(year_sort)
data_sort_byyear=c()
for(i in 1:n){
  temp=data_final[which(year_play==year_sort[i]),]
# When the year is unique, dim equals 0, then there would be a error when use apply.
# so we need to deal with data in two ways.
  if(length(dim(temp))>0){
  year_mean=apply(temp,2,mean)
  data_sort_byyear=rbind(data_sort_byyear,year_mean)
  }
  else
  data_sort_byyear=rbind(data_sort_byyear,temp)
  rownames(data_sort_byyear)[i]=i
}
data_sort_byyear=cbind(as.numeric(year_sort),data_sort_byyear)
colnames(data_sort_byyear)[1]='year'
data_sort_byyear=round(data_sort_byyear,2)
```

First see the result of scatter plot of each element in term of function of time. As all plays have five acts, I do not plot it.

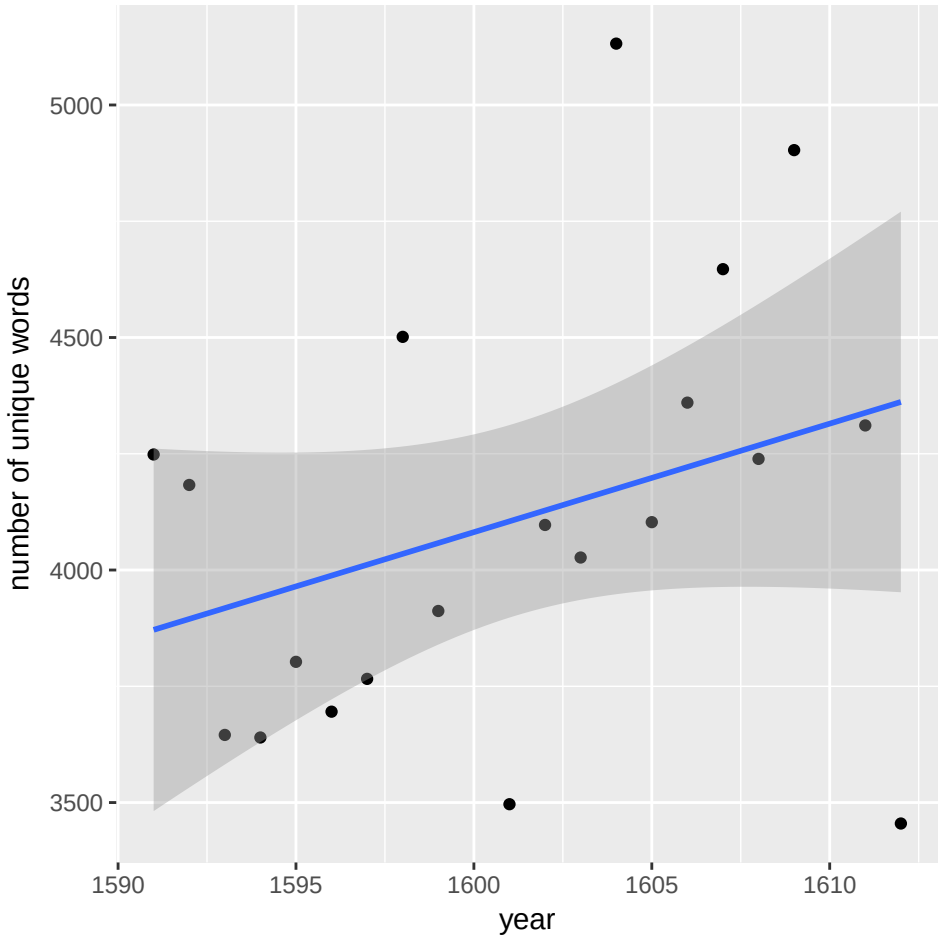
```
par(mfrow=c(2,3))
for(i in 3:8) plot(data_sort_byyear[,c(1,i)])
```



According to the plot, I guess that number of unique words might be increasing by year in term of statistics. Other elements seems not change with year. So I use ggplot2 get a clearer view in the sixth plot.

```
qplot(data_sort_byyear[,1],data_sort_byyear[,8],geom=c('point','smooth'),method='lm',xlab
= 'year',ylab = 'number of unique words')

## Warning:  Ignoring unknown parameters:  method
```



In addition, I find the p-value is about 0.1 by using lm. So I think unique words increased by year in Shakepeare's play, but the trend is not obvious.

4 Question 3

Fields:
year='numeric' year published of the play
title='character' title of the play
text='character' original content of the play
body='character' body of the play
chunk='character' all spoken chunks in the play
acts_number='numeric' number of acts
scenes_number='numeric' number of scenes
speakers_number='numeric' number of unique speakers
chunks_number='numeric' number of chunks
sentences_number='numeric' number of sentences
words_number='numeric' number of total words
unique_words='numeric' number of unique words
words_perchunk='numeric' number of words per chunk
we can also use prototype to set default of fields except text

Methods:
in the following methods, input mean the content of methods really need. For example, in method get_basic_information,

we can input play and use play@text to get text, rather than input text directly.
1.get_basic_information : input text of the play and output: year,title,body,acts_number,scenes_number
2.get_chunk : input body of the play and output chunk
3.get_chunk_information : input chunk and output: speakers_number,chunks_number,sentences_number,words_number, unique_words,words_perchunk