PS03

Yingyue Luan

September 23 2017

Problem 1

Gentzkow and Shapiro (http://www.brown.edu/Research/Shapiro/pdfs/CodeAndData.pdf)
Overall the handbook provides suggestion and instructions to social scientists who do not have degrees in programming or software engineering, who are also reluctant to change their work styles. I benefit a lot from chapter two talking about automation. Writing key script in a system shell offers readers a clear understanding of the process from beginning to end and allows replication. We should also automate as much as we could. Instead of moving file from one folder to another, we should write script to do so. I think the author makes a good point in chapter three introducing vision control. It allows us to easily refer back to previous edits and make comparison. This is extremely helpful for collaboration when two people have different ideas and they all want to try out. Also, creating function for similar steps reduces redundancy and creates readability. But it might be slightly difficult to test the function and make sure it works for all scenarios. Moreover, the author gives good recommendations on documentation. Besides only commenting information from other sources, we should name our variables descriptively and easy to read using complete words and underscore. As the author mentions in the appendix, we should fully take advantage of the debugging tools and informative error handling provided by most of the programming languages. If those are not enough, try writing unit tests instead of manually checking such as using different numbers to check a quadratic root formula.

Questions 1. In chapter three, the author states that vision control helps maintain one single, authoritative vision of the directory and it will help resolve conflicts when two people make changes simultaneously. How can it help? 2. In chapter three, if one person delete one input file and re-run the whole directory. Will the deleted file be called back if we go back to the previous log and where did the file go? 3. In chapter five, I am not very clear of what panel data is? 4. What are some more advanced task management systems? Evernote and OneNote only provide basic options like checkbox.

Problem 2

a

```r
library(stringr)
text = readLines("http://www.gutenberg.org/cache/epub/100/pg100.txt")
```

```r
# remove all the empty lines
text = text[sapply(text, nchar) > 0]
# remove all warning after each play
pat = "(<<THIS ELECTRONIC VERSION OF THE COMPLETE WORKS OF WILLIAM|SHAKESPEARE IS COPYRIGHT 1990-199
text = text[!(str_detect(text, pattern = pat))]
# seperate plays by their years and remove information and the first by
# starting from the second play
start = grep("^(15|16)[0-9]{2}$", text)
text = text[start[2]:length(text)]
year = grep("^(15|16)[0-9]{2}$", text)
# put all plays except the last one in a list
play = list()
for (i in 1:(length(year) - 1)) {
    start = year[i]
    end = year[i + 1] - 1
    onePlay = list(text[start:end])
    play[i] <- onePlay
}
```

```
pat = "(<<THIS ELECTRONIC VERSION OF THE COMPLETE WORKS OF WILLIAM|SHAKESPEARE IS
     COPYRIGHT 1990-1993 BY WORLD LIBRARY, INC., AND IS|PROVIDED BY PROJECT
    GUTENBERG ETEXT OF ILLINOIS BENEDICTINE COLLEGE|WITH PERMISSION.  ELECTRONIC
    AND MACHINE READABLE COPIES MAY BE|DISTRIBUTED SO LONG AS SUCH COPIES|PERSONAL
     USE ONLY|COMMERCIALLY.  PROHIBITED COMMERCIAL DISTRIBUTION INCLUDES BY ANY|
    SERVICE THAT CHARGES FOR DOWNLOAD TIME OR FOR MEMBERSHIP.>>)"
```

b

```r
# create a function extracting the body of the play, starting from line with
# 'SCENE' and ending with 'THE END'
getBody <- function(x) {
    bodyStart = grep("(SCENE|Scene)[[:space:]][[:upper:][:digit:]]+", x)[1]
    bodyEnd = grep("THE END", x)
    body = x[bodyStart:bodyEnd]
}
# apply the function to all the plays and get the body of each play
body = sapply(play, getBody)

# create a function extracting the year of the play, the title, the number
# of acts, and the number of scenes
getList <- function(x) {
    year = x[1]
    title = x[2]
    act = sum(str_count(x, pattern = "(ACT|Act)\\s\\w+\\.\\s((SCENE |Scene )(1|I))"))
    scene = sum(str_count(x, pattern = "(SCENE|Scene)[[:space:]][[:upper:][:digit:]]+"))
    cbind(year, title, act, scene)
}
# apply the function to all the plays and get meta data of each play
mD = sapply(play, getList)
metaData = data.frame(mD)
```

c

```
# we first create two empty lists which will contain the speakers and spoken
# texts of all the plays
speakersAll = list()
chunksAll = list()
# dealing with one play at a time
for (j in 1:length(play)) {
    x = play[[j]]
    # create a data frame containing the starting and ending position of speaker
    # name. The row number of the data frame is the line number where the
    # speaker name appears
    findSpeaker = data.frame(str_locate(x, (ifelse(j == 4, "^[[:upper:]]{1}[A-Za-z]+( [[:upper:]]{1}
        "^[[:space:]]{2}[[:upper:]]{1}[A-Za-z]+( [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s"))))
    findSpeaker = na.omit(findSpeaker)
    lineSpokenText = as.numeric(rownames(findSpeaker))
    # create two lists for speakers and spoken texts in this play
    speakers = list()
    spokenTexts = list()
    # for each line number, extracting the text in between and removing the
    # speaker name in the begining, we get the spoken text. We get the speaker
    # name by subsetting the sentence by its starting and ending positions
    n = length(lineSpokenText) - 1
    for (i in 1:n) {
        start = lineSpokenText[i]
        end = lineSpokenText[i + 1] - 1
        spokenText = x[start:end]
        speaker = str_sub(spokenText[1], start = 3, end = findSpeaker[i, 2] -
            2)
        spokenText = str_replace(spokenText, (ifelse(j == 4, "^[[:upper:]]{1}[A-Za-z]+( [[:upper:]]{
            "^[[:space:]]{2}[[:upper:]]{1}[A-Za-z]+( [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s")),
            "")
        speakers = c(speakers, speaker)
        spokenTexts = c(spokenTexts, list(spokenText))
    }
    # add back the speaker and spoken chunk information of the last chunk
    lastStart = lineSpokenText[n]
    lastEnd = lineSpokenText[length(lineSpokenText)]
    lastSpokenText = x[lastStart:lastEnd]
    lastSpeaker = str_sub(lastSpokenText[1], start = 3, end = findSpeaker[n,
        2] - 2)
    lastSpokenText = str_replace(lastSpokenText, (ifelse(j == 4, "^[[:upper:]]{1}[A-Za-z]+( [[:upper
        "^[[:space:]]{2}[[:upper:]]{1}[A-Za-z]+( [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s")),
        "")
    speakers = c(speakers, lastSpeaker)
    spokenTexts = c(spokenTexts, list(lastSpokenText))
    # put result of each play into list. Each list will contain 35 elements
    speakersAll = c(speakersAll, list(speakers))
    chunksAll = c(chunksAll, list(spokenTexts))
}
# combine list of speakers and list of chunks
combinedAll = cbind(speakersAll, chunksAll)
```

```
findSpeaker = data.frame(str_locate(x, (ifelse(j == 4, "^[[:upper:]]{1}[A-Za-z]+(
     [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s", "^[[:space:]]{2}[[:upper:]]{1}[A-Za-z]+(
     [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s"))))
spokenText = str_replace(spokenText, (ifelse(j == 4, "^[[:upper:]]{1}[A-Za-z]+(
    [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s", "^[[:space:]]{2}[[:upper:]]{1}[A-Za-z]+(
    [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s")), "")
lastSpokenText = str_replace(lastSpokenText, (ifelse(j == 4, "^[[:upper:]]{1}[A-
    Za-z]+( [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s", "^[[:space:]]{2}[[:upper:]]{1}[A-
    Za-z]+( [[:upper:]]{1}[A-Za-z]+){0,}\\.\\s")), "")
```

d

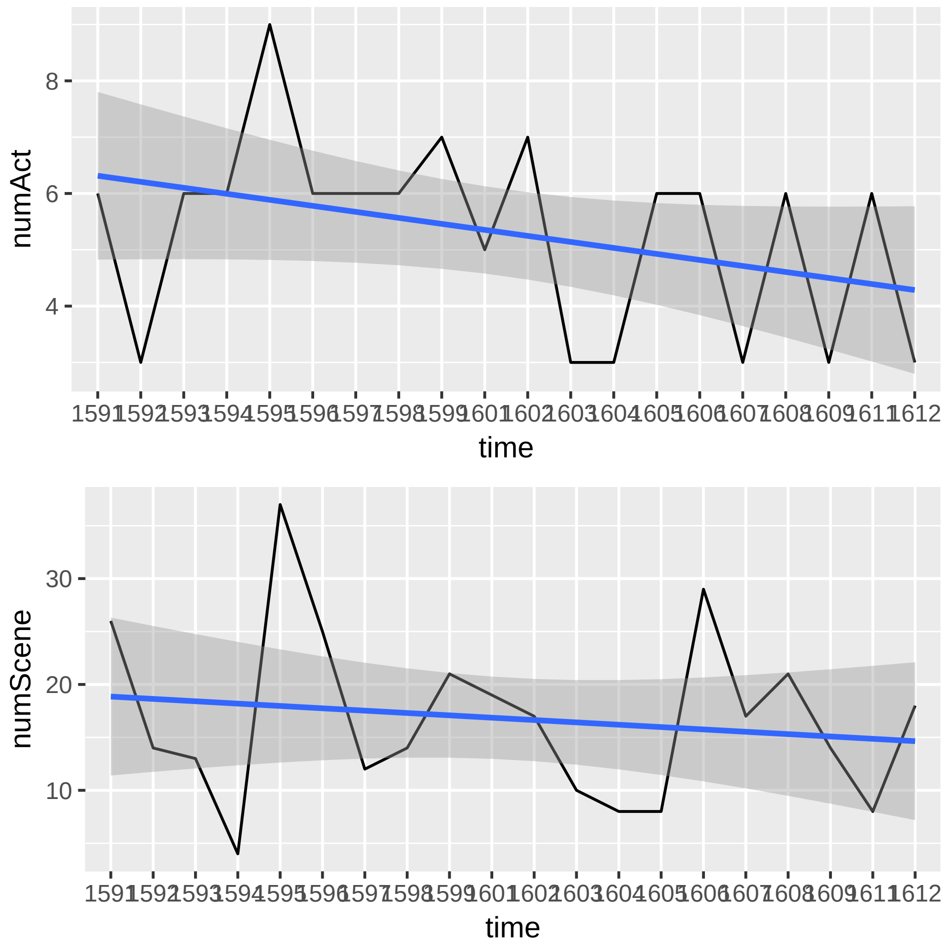
```
# The number of unique speakers of each play
numUniqueSpeaker = lapply(speakersAll, function(x) {
    length(unique(x))
})

# The number of spoken chunks of each play
numSpokenChunk = lapply(chunksAll, length)

# the number of sentences of every spoken chunk
numSentence = lapply(chunksAll, function(y) lapply(y, function(x) {
    length(unlist(gregexpr("[[:alnum:] ][.!?]", x))[unlist(gregexpr("[[:alnum:] ][.!?]",
        x)) > 0])
}))
# the number of sentences of each play
numSentence = lapply(numSentence, function(x) Reduce("+", x))
```

```r
# the number of words of every spoken chunk
numWord = lapply(chunksAll, function(y) lapply(y, function(x) {
    sum(str_count(x, pattern = "\\S+"))
}))
# the number of words of each play
numWord = lapply(numWord, function(x) Reduce("+", x))

# the average number of words per chunk of each play
avgWord = round(unlist(numWord)/unlist(numSpokenChunk))

# the number of unique words of every spoken chunk
numUniqueWord = lapply(chunksAll, function(y) lapply(y, function(x) {
    length(unlist(unique(str_extract(x, pattern = "\\S+"))))
}))
# the number of unique words of each play
numUniqueWord = lapply(numUniqueWord, function(x) Reduce("+", x))

# create summary statistics
summaryStats = data.frame(cbind(numUniqueSpeaker, numSpokenChunk, numSentence,
    numWord, avgWord, numUniqueWord))
summaryStats
```

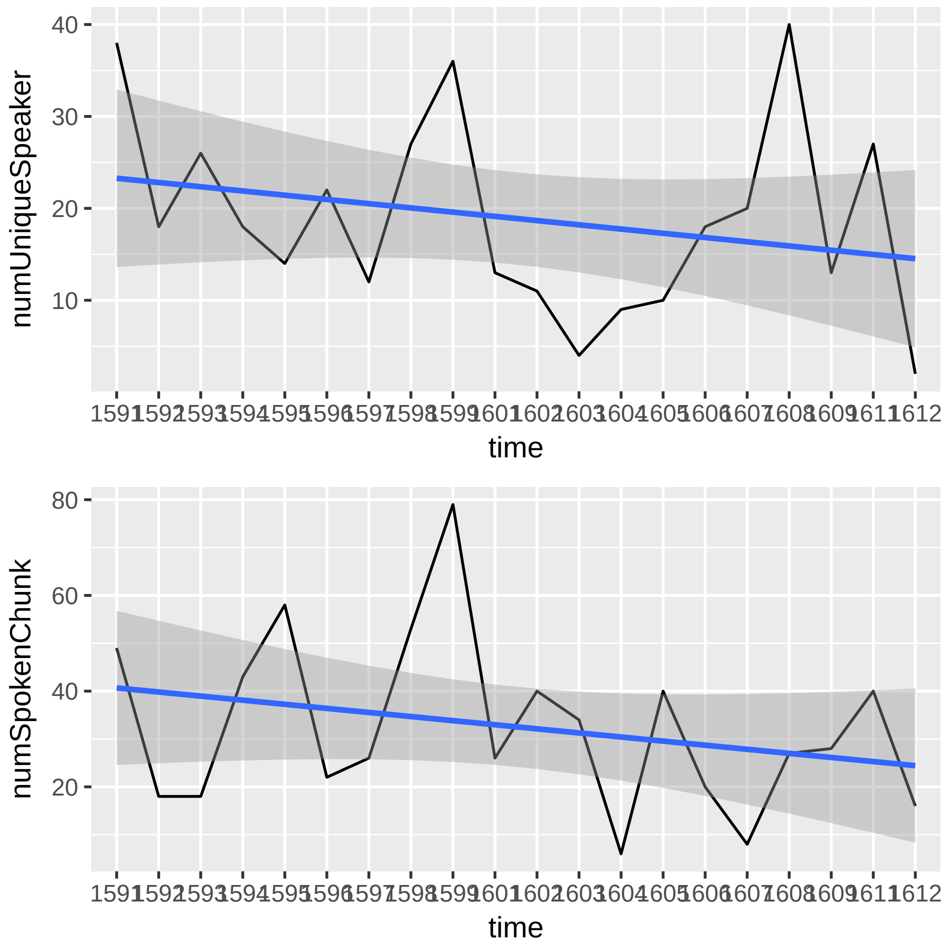
```
##    numUniqueSpeaker numSpokenChunk numSentence numWord avgWord
## 1                 23            933        1738   23195      25
## 2                 59           1172        2361   24978      21
## 3                 26            806        1544   21604      27
## 4                 23            610        1065   14756      24
## 5                 62           1104        2121   27711      25
## 6                 39            854        2184   27720      32
## 7                 32           1118        2784   30673      27
## 8                 33            743        2146   24754      33
## 9                 50            902        1955   26063      29
## 10                48            717        1458   25215      35
## 11                53            647        1478   21763      34
## 12                66            790        1663   25646      32
## 13                47            815        1679   24472      30
## 14                48            704        1600   24336      35
## 15                27            548        1230   20785      38
## 16                48            793        1832   19817      25
## 17                23           1061        2817   26315      25
## 18                19           1044        1788   21737      21
## 19                43            643        1663   17138      27
## 20                23            895        1678   21738      24
## 21                25            631        1374   21405      34
## 22                27           1015        2116   22236      22
## 23                33            504        1171   16610      33
## 24               23             954        1982   21385      22
## 25               26            1180        2672   26488      22
## 26               36             552        1337   22355      40
## 27               63            1073        2125   29196      27
## 28               34             817        2479   24778      30
## 29               36             889        1663   21145      24
## 30               19             641        1265   16496      26
## 31               58             759        1651   18447      24
## 32               26             562        1329   20901      37
## 33               29            1138        2278   25894      23
## 34               21             920        1655   19744      21
## 35               17             856        1383   17264      20
## 36               34             743        1717   24933      34
##    numUniqueWord
## 1           2818
## 2           3614
## 3           2484
## 4           1743
## 5           3660
## 6           3524
## 7           3789
## 8           2768
## 9           2947
## 10          2816
## 11          2819
## 12          2968
## 13          2809
## 14          3136
## 15          2427
## 16          2484
## 17          3416
## 18          2618
## 19          2375
## 20          2701
## 21          2457
## 22          2693
## 23          1975
## 24          2443
## 25          3368
## 26          2616
## 27          3722
## 28          2979
## 29          2549
## 30          2244
## 31          2379
## 32          2429
## 33          3304
## 34          2383
## 35          2141
## 36          3097
```

e

```
# create dataframe reporting the number of acts and scenes, number of unique
# speakers, and number of chunks for each play
time = as.numeric(mD[1, ])
numAct = unlist(mD[3, ])
numScene = unlist(mD[4, ])
numUniqueSpeaker = unlist(numUniqueSpeaker)
numSpokenChunk = unlist(numSpokenChunk)
report = cbind(time, numAct, numScene, numUniqueSpeaker, numSpokenChunk)
report = data.frame(report)
# group by year and sum by year
groupByYear = aggregate(. ~ time, FUN = sum, data = report)
# plot variable in report against time
library(ggplot2)
library(gridExtra)
plot1 = ggplot(groupByYear, aes(time, numAct, group = 1), color = variable) +
    geom_line() + stat_smooth(method = lm)
plot2 = ggplot(groupByYear, aes(time, numScene, group = 1), color = variable) +
    geom_line() + stat_smooth(method = lm)
plot3 = ggplot(groupByYear, aes(time, numUniqueSpeaker, group = 1), color = variable) +
    geom_line() + stat_smooth(method = lm)
plot4 = ggplot(groupByYear, aes(time, numSpokenChunk, group = 1), color = variable) +
    geom_line() + stat_smooth(method = lm)
grid.arrange(plot1, plot2)
```



```
grid.arrange(plot3, plot4)
```



f

I added a ifelse statement to include play 4 which has the act name starting without indentation like other plays.

Problem 3

a

A object of class "subPlay"
Slots:
Name: body, metaData, speakersAll, chunksAll, summaryStats, report
Class: list, data.frame, list, list, data.frame, data.frame
A object of class "body"
Slots:

Name: unnamed list variable
Class: character
A object of class ”metaData”
Slots:
Name: unnamed field
Class: factor
A object of class ”speakersAll”
Slots:
Name: unnamed field
Class: list (containing fields of character)
A object of class ”chunksAll”
Slots:
Name: unnamed field
Class: list (containing fields of character)
A object of class ”summaryStats”
Slots:
Name: unnamed field
Class: list (containing fields of integer)
A object of class ”report”
Slots:
Name: unnamed field
Class: factor

b

```
# The methods relating to processing the text of the plays to produce the
# fields could be the functions like str_detect() and str_locate() from the
# stringr library as well as grep(). Inputs to these methods are the string
# text and the methods perform some actions like extract and remove. Fields
# like speakersAll and chunksAll might be modified and generate list of
# names and list of spoken chunks.

# Other methods relating to providing information to a user who wants to
# know something about a play or see the text of the play can be the
# getList() and getBody() created before or some functions like str_count()
# embedded in the sapply() or lapply(). These methods count or grasp the
# information users want. Fields like summartStats might be created to show
# users the information they want.
```