ps3

Kehsin Su Esther 3033114294

September 28, 2017

```
knitr::opts_chunk$set(tidy = TRUE, cache = TRUE)
```

1 Q1

1.1 (c)

I have read Best Practice for Scientific Computing, and I have raised the question that why the author does not recommend research groups to implement all the 10 recommendations at once since it mentioned that these approaches can gain productivity in short time?
Also, how did the author measure that the time cost to implement these recommendations can be offset "immediately" by productivity? (Honestly, I think the time cost to modify the coding habits and familiar with a new coding environment is very high)

1.2 (d)(i)

I think the 7th method is very useful, and test-driven development is very interesting!Even though the author mentioned there is not significant impact on productivity, I guess it can help me to know what I should code down more clearly.
I think the 8th suggestion should not use such determined word "only" since sometimes if people are working on very huge programs, the efficiency is also important, otherwise, they need to wait for a long time to make sure their results are correct. Although accuracy is generally more important than efficiency, programmers should still consider how long their code will consume the testers time and how long they can get the result and make sure it is correct.

1.3 (d)(ii)

I am currently using the second method. As the author mentioned, I think to keep typing repetitive code is not efficient at all and very easy to make mistakes. Moreover, it seems a little hard for me to remember all the structures of each function, and with that method, I can just modify some part of my codes to complete most of the tasks. It's a more easy way for me to manage my codes.

1.4 (d)(iii)

For the first method, I guess that, sometimes, researcher just do some easy trails, so they do not want to waste their time to type long words to name meaningful functions or set good formats.Instead, for the convenience, they will only name their functions or variables x, y and results.
I believe, normally, they will use method 2, 3, 5 and 6 to track their experiment results and reduce mistakes. Among these recommendations, I think 2,3, 6 are easiest to implement since they only need to install editors or change their working environment.On the other hand, 4, 9 and 10 are the most difficult part to implement as it will take longer time to ask a programmer to change their coding habits.

1.5 (d)(iv)

I think 1, 2, 3, 6 and 9 are the general rules for both data analysis and software engineering.
1, 3 and 9 are especially important for data analysis since the key concept for data analysis is to make good use of data to obtain useful information. The concept why to choose such method is quite important, so use recommendation 1 and 9 to write down meaningful function names and add some comment on the approach is essential. Also, people will obtain lots of variables while analysis data, so they need use recommendation 3 to record the history.
For software engineering 4, 5, 7, 8, and 10 seem more for it. As software programmers usually deal with huge programs, they need to collaborate together and work on it for longer times. At such situation, how to track where is modified, when such modification made, how to debug and how to collaborate with other programmers are necessary. Also, programmers usually care about the efficiency since they have learned more about algorithms, so recommendation 8 is the rule they should keep in mind.

2 Q2

2.1 Check the working directory

```
getwd()

## [1] "C:/Users/Esther/Desktop/stat243"

setwd("C:/Users/Esther/Desktop/stat243-fall-2017-master/ps")
getwd()

## [1] "C:/Users/Esther/Desktop/stat243-fall-2017-master/ps"
```

```
#use tm package to do text mining.
#Set the file path to where it contains the document I want to work on.
library(tm)
#vignette("tm")
cname <- file.path("C:/Users/Esther/Desktop/stat243-fall-2017-master", "ps")
dir(cname)
docs <- VCorpus(DirSource(cname))
summary(docs)
inspect(docs[8])
#tdm = as.matrix(TermDocumentMatrix(plays[[2]], control = list(wordLengths = c(1, Inf))))
#uniqueWords = function(d) {
  return(paste(unique(strsplit(d, " ")[[1]]), collapse = ' '))
}
##corpus = tm_map(plays[[2]], content_transformer(uniqueWords))
#tdm = as.matrix(TermDocumentMatrix(corpus, control = list(wordLengths = c(1, Inf))))
#rowSums(tdm)
#writeLines(as.character(docs[8]))
#findFreqTerms(mydata, 30)
```

2.2 Input the data and named it as mystring

```
library(readr)

## Warning: package 'readr' was built under R version 3.3.3
##
## Attaching package: 'readr'
## The following object is masked from 'package:curl':
##
##     parse_date

library(stringr)
mystring <- read_file("C:/Users/Esther/Desktop/stat243-fall-2017-master/ps/pg100.txt")
```

2.3 Clean the data

2.3.1 Transfer ACT and SCENE into the standatd type like ACT 3. SCENE 1.

```
# Make a copy to start data cleaning
mystring_new <- mystring
# Note: Must use descending method! Because if we use acesending order, when
```

```r
# the string equal to 'ACT II', it will first catch 'ACT I' and transfer
# into ACT 1
num_type <- c(15:1)
rom_type <- as.roman(num_type)
mystring_new <- gsub("Act ", "ACT ", mystring_new)
mystring_new <- gsub("ACT_", "ACT ", mystring_new)
mystring_new <- gsub("ACT_", "ACT ", mystring_new)
for (i in c(1:length(num_type))) {
    mystring_new <- gsub(paste("ACT ", rom_type[i], sep = ""), paste("ACT ",
        num_type[i], ".", sep = ""), mystring_new)
    mystring_new <- gsub(paste("SCENE ", rom_type[i], sep = ""), paste("SCENE ",
        num_type[i], sep = ""), mystring_new)
}
# check
if ((gregexpr("Act ", mystring_new, FALSE)[[1]][1] == -1) * (gregexpr("Act_",
    mystring_new, FALSE)[[1]][1] == -1)) {
    print("Sucessfully Transfer all Act and SCENE types")
}

## [1] "Sucessfully Transfer all Act and SCENE types"
```

2.4 Create functions

```r
# obtain strings with a criteria, but ignore some strings at the beginning
# at the ends of the criteria when getting the result
textobtain <- function(mydata, rule, ignore_st1 = "", ignore_st2 = "", case_sen = TRUE) {
    match_st <- gregexpr(rule, mydata, case_sen)
    len = length(match_st[[1]])
    obtains <- vector("list", len)
    for (i in c(1:len)) {
        obtains[i] = substr(mydata, match_st[[1]][i] + nchar(ignore_st1), match_st[[1]][i] +
            attr(match_st[[1]], "match.length")[i] - nchar(ignore_st2) - 1)
    }
    return(obtains)
}
# obtain stirngs between two rules Between two mactched strings, so checking
# whether the two criterias obtains same lengthes of results is necessary
# endsep is another option if I the two criteria I want to obtain are the
# same, the functions still will obtain the words between such criteria, but
# change the end point to grep by the endsep criteria.
textobtain2 <- function(mydata, rule1, ignore_st1 = "", case_sen1 = TRUE, rule2,
    ignore_st2 = "", case_sen2 = TRUE, endsep = "\r\n", case_sen3 = TRUE) {
    match_st1 <- gregexpr(rule1, mydata, case_sen1)
    match_st2 <- gregexpr(rule2, mydata, case_sen2)
    len1 = length(match_st1[[1]])
    len2 = length(match_st2[[1]])
    obtains <- vector("list", len1)
    if (rule1 == rule2) {
        for (i in c(1:(len1 - 1))) {
            obtains[i] = substr(mydata, match_st1[[1]][i] + nchar(ignore_st1),
                match_st1[[1]][i + 1] - 1)
        }
        obtains[len1] = substr(mydata, match_st1[[1]][len1] + nchar(ignore_st1),
            gregexpr(endsep, mydata, case_sen3)[[1]][1] + nchar(endsep) - 1)
    } else if (len1 == len2) {
```

```
        for (i in c(1:len1)) {
            obtains[i] = substr(mydata, match_st1[[1]][i] + nchar(ignore_st1),
                match_st2[[1]][i] + attr(match_st2[[1]], "match.length")[i] -
                  nchar(ignore_st2) - 1)
        }
    } else {
        print("Did not enter correct rules")
    }
    return(obtains)
}
# obtain string between two criteria, but start with part of the criteria
textobtain3 <- function(mydata, rule, case_sen = TRUE, begin = "", endsep = "\r\n",
    case_sen3 = TRUE) {
    match_st <- gregexpr(rule, mydata, case_sen)
    len1 = length(match_st[[1]])
    obtains <- vector("list", len1)
    for (i in c(1:(len1 - 1))) {
        obtains[i] = substr(mydata, match_st[[1]][i] + attr(match_st[[1]], "match.length")[i] -
            nchar(begin), match_st[[1]][i + 1] - 1)
    }
    obtains[len1] = substr(mydata, match_st[[1]][len1] + attr(match_st[[1]],
        "match.length")[i] - nchar(begin), gregexpr(endsep, mydata, case_sen3)[[1]][1] +
        nchar(endsep) - 1)
    return(obtains)
}


# identified how many number of acts in each play
group_by_ind <- function(mydata, ind_pos1, ind_pos2) {
    act_num <- vector("list", length(mydata))

    for (k in c(1:length(mydata))) {
        act_num[k] <- as.numeric(substr(mydata[k], ind_pos1, ind_pos2))
    }
    total_act <- act_num[[length(mydata)]]
    ind <- list(total_act, act_num)
    return(ind)
}
# combined same act
comb_act <- function(mydata, ind_pos1, ind_pos2) {
    total_act <- group_by_ind(mydata, ind_pos1, ind_pos2)[[1]]
    act_num <- group_by_ind(mydata, ind_pos1, ind_pos2)[[2]]
    act <- vector("list", total_act)
    for (x in c(1:total_act)) {
        for (y in c(1:length(mydata))) {
            if (act_num[[y]] == x) {
                act[[x]] <- paste(act[x], mydata[[y]])
            }
        }
    }
    return(act)
}
```

2.5 Begin to seperate data

2.5.1 Seperate plays

```
# (a) grep the plays from the mystrings each play are begin with whitespace
# and four digital numbers(year) and ending with 'THE END'
plays <- textobtain2(mydata = mystring_new, rule1 = "[\r\n]+[[:digit:]]{4}[\r\n\r\n]",
    rule2 = "THE END", case_sen2 = FALSE)
# remove first and last ones
plays <- plays[-1]
plays <- plays[-37]
nplays <- length(plays)
cat("Total are ", nplays, " plays")

## Total are  36  plays
```

2.5.2 Allocate the list

```
# Allocate in advance
year <- vector("list", nplays)
title <- vector("list", nplays)
body <- vector("list", nplays)
act_sep <- vector("list", nplays)
num_act <- vector("list", nplays)
act_per <- vector("list", nplays)
sce_per <- vector("list", nplays)
peo <- vector("list", nplays)
pure_peo <- vector("list", nplays)
peodi <- vector("list", nplays)
act <- vector("list", nplays)
sum_sc <- vector("list", nplays)
num_chunk_peo <- vector("list", nplays)
num_chunk_dia <- vector("list", nplays)
num_pure_peo <- vector("list", nplays)
sum_sent <- vector("list", nplays)
sum_word <- vector("list", nplays)
mean_sent <- vector("list", nplays)
mean_word <- vector("list", nplays)
cl_st <- vector("list", nplays)
unique_words <- vector("list", nplays)
```

2.5.3 Seperate elements inside each play

```
# (b)
for (i in c(1:nplays)) {
    year[i] <- textobtain(mydata = plays[i], rule = "\r\n\r\n[[:digit:]]{4}\r\n\r\n",
        ignore_st1 = "\r\n\r\n", ignore_st2 = "\r\n\r\n")
    title[i] <- textobtain2(mydata = plays[i], rule1 = "[[:digit:]]{4}\r\n\r\n",
        ignore_st1 = "1234\r\n\r\n", rule2 = "\r\n\r\nby ", ignore_st2 = "\r\n\r\nby ")
    title[i] <- gsub("\r", "", title[i])
    title[i] <- gsub("\n", "", title[i])
    # body are some words begin with 'SCENE.' or 'SCENE:' and end with 'THE END'
    body[i] <- textobtain2(mydata = plays[i], rule1 = "SCENE[:.]", case_sen1 = FALSE,
```

```
        rule2 = "THE END", case_sen2 = FALSE)
    act_sep[i] <- list(textobtain2(mydata = plays[i], rule1 = "ACT [[:digit:]]",
        case_sen1 = FALSE, rule2 = "ACT [[:digit:]]", ignore_st2 = "ACT 1",
        case_sen2 = FALSE, endsep = "THE END", case_sen3 = FALSE))
    num_act[[i]] <- group_by_ind(act_sep[[i]], 5, 5)[[1]]
    act[i] <- list(comb_act(act_sep[[i]], 5, 5))
    sce_per[i] <- length(gregexpr("SCENE|Scene", plays[i], FALSE)[[1]]) - 1
    sum_sc[i] <- 0
    for (j in c(1:num_act[[i]])) {
        c_sc <- vector("list", num_act[[i]])
        c_sc[[j]] <- length(gregexpr("SCENE|Scene", act[[i]][j], FALSE)[[1]])
        sum_sc[[i]] <- sum_sc[[i]] + c_sc[[j]]
    }
    # speaker's name must begin with the combination of two white space and a
    # capital letter and end with the combination of comma and white space and a
    # capital letter
    peo[i] <- list(textobtain(mydata = plays[i], rule = "\r\n  [A-Z][A-Za-z ]+[.] [A-Z]",
        ignore_st1 = "\r\n  ", ignore_st2 = ". A", case_sen = FALSE))  #obtain speakers' names
    pure_peo[i] <- length(unique(peo[[i]], TRUE))  #obtain unique speaker's name
    # at the beginning of sentence must be a speaker's name, so set it as a
    # criteria
    peodi[i] <- list(textobtain3(mydata = plays[i], rule = "\r\n  [A-Z]+[A-Za-z ]+[.] [A-Z]",
        case_sen = FALSE, begin = "A", endsep = "THE END", case_sen3 = FALSE))  #obtain sentence within
}

# print out the rsult
cat("Total act each play:\n", unlist(num_act))

## Total act each play:
##  5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5

cat("Total scene each play:\n")

## Total scene each play:

unlist(sce_per)

##  [1] 23 42 22 11 29 27 20 19 19 23 27 24 28 17 16 18 26  9 29 17 20 23  9
## [24] 17 15 19 25 24 14  9 17 14 24 18 20 15

k <- rep("NA", 36)
for (i in 1:36) {
    k[i] <- length(gregexpr("SCENE[.:]", plays[[i]], FALSE)[[1]])
}
```

2.6 Do some statistics

At the beginning, check whether the number of speakers same as the number is chunk.
USe a for loop to calculate how many sentence and words in each chunk.
Usually, sentence are seperate by "." and words are seperate by whitespace, so we can use these two criteria to match the number of words and sentence.
Then, use termFreq function to identify different words(but we need to remove all the punctuations and ° before calculate!)
Finally, print out the result by paste them togehter.

```
# (d) begin to grep the chunk within each play
for (i in 1:nplays) {
    if (length(peo[[i]]) != length(peodi[[i]])) {
        print("Do not get chunk correctly!")
    } else {
        num_chunk_peo[[i]] <- length(peo[[i]])
        num_chunk_dia[[i]] <- length(peodi[[i]])
    }
    sent_per <- 0
    word_per <- 0
    for (j in 1:length(peodi[[i]])) {

        sent_per <- sent_per + length(gregexpr("[.]", peodi[[i]][[j]], FALSE)[[1]])
        word_per <- word_per + length(gregexpr("\\W+", peodi[[i]][[j]], TRUE)[[1]])
    }
    sum_sent[[i]] <- sent_per
    sum_word[[i]] <- word_per
    mean_sent[[i]] <- round(sent_per/length(peodi[[i]]), 4)
    mean_word[[i]] <- round(word_per/length(peodi[[i]]), 4)
    cl_st[[i]] <- removePunctuation(plays[[i]])
    cl_st[[i]] <- gsub("\r\n", "", cl_st[[i]])
    unique_words[[i]] <- length(termFreq(cl_st[[i]]))
}
```

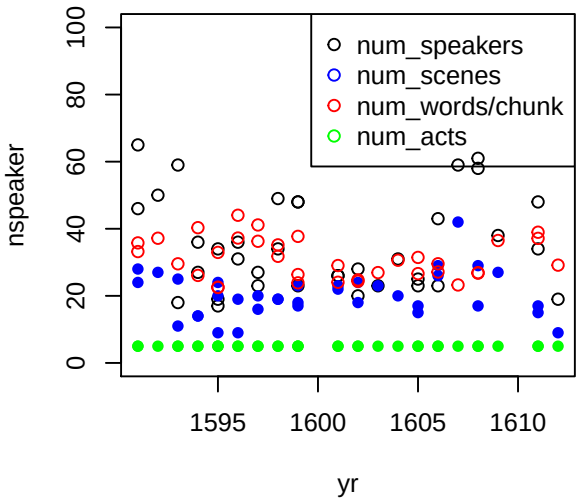
2.6.1 Print out the result

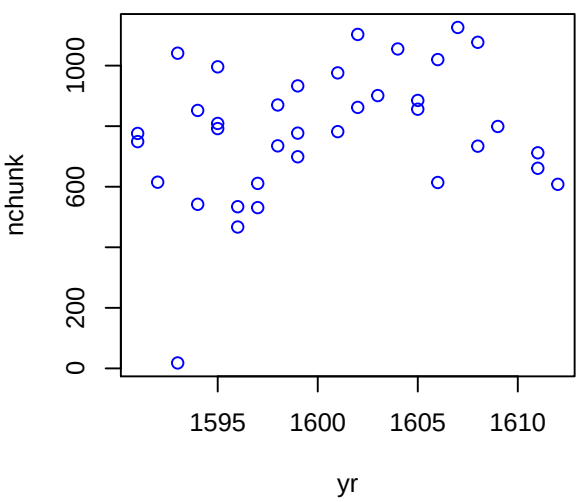
```
yr <- unlist(year)
nact <- unlist(num_act)
nsce <- unlist(sce_per)
nchunk <- unlist(num_chunk_dia)
nspeaker <- unlist(pure_peo)
n_sum_sent <- unlist(sum_sent)
n_mean_sent <- unlist(mean_sent)
n_sum_words <- unlist(sum_word)
n_mean_words <- unlist(mean_word)
n_uni_words <- unlist(unique_words)
cat("Total unique speakers each play:\n")

## Total unique speakers each play:

unlist(pure_peo)

##  [1] 23 59 26 18 61 38 31 34 49 48 50 65 46 48 27 48 23 19 43 23 23 26 31
## [24] 23 25 36 59 34 36 19 58 27 28 20 17 34

cat("Total chunck each play:\n")

## Total chunck each play:

nchunk

##  [1]  901 1126  782   18 1077  799 1055  735  870  699  615  749  776  661
## [15]  531  777 1020  996  614  856  611  976  467  933  885  534 1041  792
## [29]  852  608  734  542 1103  862  809  712

cat("Total sentences each play:\n")
```

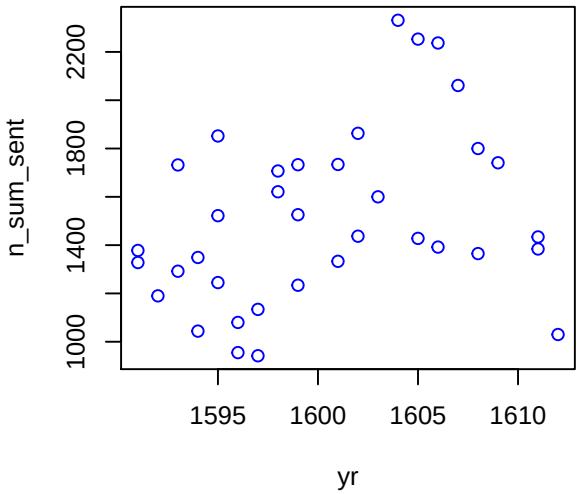
```
## Total sentences each play:

n_sum_sent

##  [1] 1600 2061 1333 1292 1800 1741 2331 1707 1621 1234 1190 1328 1378 1384
## [15]  942 1526 2237 1522 1392 1428 1134 1734  955 1733 2253 1080 1732 1852
## [29] 1349 1030 1365 1044 1863 1437 1245 1434

cat("Average sentences per chunk each play:\n")

## Average sentences per chunk each play:

n_mean_sent

##  [1]  1.7758  1.8304  1.7046 71.7778  1.6713  2.1790  2.2095  2.3224
##  [9]  1.8632  1.7654  1.9350  1.7730  1.7758  2.0938  1.7740  1.9640
## [17]  2.1931  1.5281  2.2671  1.6682  1.8560  1.7766  2.0450  1.8574
## [25]  2.5458  2.0225  1.6638  2.3384  1.5833  1.6941  1.8597  1.9262
## [33]  1.6890  1.6671  1.5389  2.0140

cat("Total words each play:\n")

## Total words each play:

n_sum_words

##  [1] 24268 26180 22711 15313 28999 29160 32300 25871 27656 26374 22864
## [12] 26802 25765 25757 21842 20479 27658 22492 18153 22792 22163 23471
## [23] 17390 22278 27840 23510 30762 26097 22227 17727 19642 21854 27316
## [34] 20986 18195 26440

cat("Average words per chunk each play:\n")

## Average words per chunk each play:

n_mean_words

##  [1]  26.9345  23.2504  29.0422 850.7222  26.9257  36.4956  30.6161
##  [8]  35.1986  31.7885  37.7310  37.1772  35.7837  33.2023  38.9667
## [15]  41.1337  26.3565  27.1157  22.5823  29.5651  26.6262  36.2733
## [22]  24.0482  37.2377  23.8778  31.4576  44.0262  29.5504  32.9508
## [29]  26.0880  29.1562  26.7602  40.3210  24.7652  24.3457  22.4907
## [36]  37.1348

cat("Total unique words each play:\n")

## Total unique words each play:

n_uni_words

##  [1] 3588 4028 3284 3010 4104 4327 4801 3936 4208 4612 3915 4138 3666 3799
## [15] 3648 2927 4221 3774 3405 3367 3324 3332 3014 3071 3808 3764 4201 3778
## [29] 3296 3271 3344 3454 4333 3131 2757 3935
```

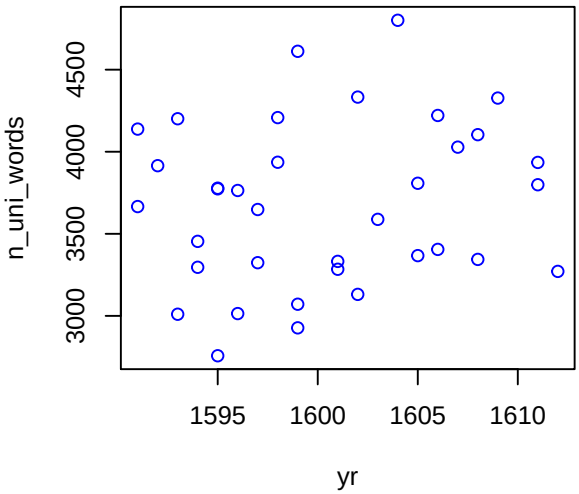
2.7 Graph to see to results

```r
# (e)
df <- data.frame(yr, nact, nsce, nchunk, nspeaker, n_sum_sent, n_mean_sent,
    n_sum_words, n_mean_words, n_uni_words)
# view the data
View(df)
# plot
par(mfrow = c(2, 2))
plot(yr, ylim = c(0, 100), nspeaker, col = "black")
points(x = yr, y = nsce, pch = 16, col = "blue")
points(x = yr, y = n_mean_words, col = "red")
points(x = yr, y = nact, pch = 16, col = "green")
legend("topright", pch = 1, c("num_speakers", "num_scenes", "num_words/chunk",
    "num_acts"), col = c("black", "blue", "red", "green"))
plot(yr, nchunk, col = "blue")
plot(x = yr, y = n_sum_sent, col = "blue")
plot(x = yr, y = n_uni_words, col = "blue")
```


num_speakers
num_scenes
num_words/chunk
num_acts
nspeaker
0
20
40
60
80
100
1595
1600
1605
1610
yr


nchunk
0
200
600
1000
1595
1600
1605
1610
yr


n_sum_sent
1000
1400
1800
2200
1595
1600
1605
1610
yr


n_uni_words
3000
3500
4000
4500
1595
1600
1605
1610
yr

3 Q3

3.1 (a)

My design for above problem is as following:

class	slot name	typeof	method	input	output
data	play	char	textobtain2	char, char	list of char
data	year	char	textobtain2	char, char	list of char
pdata	title	char	textobtain2	char, char	list of char
data	body	char	textobtain2	char, char	list of char
stat	num chunk	integer	length	list of char	list of integer
stat	num speakers	integer	length	list of char	list of integer
stat	num sentence	integer	length, gregexpr	list of char	list of integer
stat	num words	integer	length, gregexpr	list of char	list of integer
stat	unique speakers	integer	length, unique	list of char	list of integer
stat	unique words	integer	length, unique	list of char	list of integer
graphes	yearly trend for num act	graph	plot	year, num act	graph
graphes	yearly trend for num scene	graph	plot	year, num scene	graph
graphes	yearly trend for num words	graph	plot	year, num words	graph
graphes	yearly trend for unique words	graph	plot	year, unique words (vectors)	graph
graphes	yearly trend for num speakers	graph	plot	year, unique words (vectors)	graph
graphes	yearly trend for unique speakers	graph	plot	year, unique people (vectors)	graph
graphes	yearly trend for num sentence	graph	plot	year, num sentence (vectors)	graph

3.2 (b)

The corrspoinding methods are mentioned as above.
The following are some describes of each methods:
textobtain1/textobtain2: a method to grep strings with 1 or 2 criteria inputs, which are strings in regular expression
gregexpr: a method that find the position of match strings within the data, the inputs are data and strings
unique: get the unique items from data input, which is a vector
length: count the length of data input, which is a vector
plot: come out a graph with two input data, which are two vecotrs
A lots of slots are create by the method of textobtain2 methods(e.g. year, title), number of speakers, words, are calculate by the method of length, and unique method create the slot of unique words.