Problem Set 3

Hsieh Cheng Han

September 28, 2017

1(c)

Comment : The practice of reproducible research

Reproducible research plays an important role in whole academia since it serves as the only thing that an investigator can guarantee about a study. It ensures that the experiment results are correct and the transparency which gives us confidence in understanding exactly what was done.

With maturity and complexity of the technique in data analysis, reproducibility is pretty much the only thing we can hope for. Although time will tell whether we are ultimately right or wrong about any claims, reproducibility is still something that helps to clarify the correctness and reliabilty right now.

The preface of the material mentions the difficulty in reproducible research such as inaccessibility of experiment data, proprietary anaysis software and uninterest from scientific publishing. As a result, a systematic procedure must be developed to make reproducible research, which is mostly time-consuming and laborious, more efficient and smoother.

Then, three stages of basic reproducible workflow are illustrated. The first one is data acquisition from exisiting source, field obersvation, statistical surveys or experimental research. The second stage includes data processing or cleaning, which I think may be the most time-consuming step. The final stage is data analysis, in which formal statistical methods are introduced and executed.

I think it clear to have a simple example to describe how the three stages are going through by R language. The material also shows the importance and efficiency of automation conducting reproducible research. I realize that inside the core of completeing reproducible research and feasibility of automation is modularization, which is also an essential concept in software engineering.

The challenges to reproducibility are called "pain points", including inconsistent computational skills and tool familiarity from researchers, incompatible softwares, disenfranchisement from scientific process. In correspondance with this, a bunch of actionable recommendations like version control, testing framworks and continuous integration are brought up and conducted upon realisitic experiments.

The material gives me an abstract of how reproducible research is progressing and its substantive importance. Last year, a prestigious scholar from National Taiwan University, which is my undergraduate university, was accused of fraud lurk inside his academic papers. This news heavily struck the whole academia in Taiwan and forced everyone to pay more attention to the authenticity of experimental results, which I think was the very evidence to highlight the necessity of reproducible research.

2(a)

```
# First we load the required packages to deal with the following problems.
library(XML)
library(curl)
library(stringr)
library(ggplot2)
```

```r
# Download the whole plain txt file and trun it into a single string.
URL <- "http://www.gutenberg.org/cache/epub/100/pg100.txt"
txt <- readLines(URL)
txt <- toString(txt)

# First, we delete strings "<<THIS...MEMBERSHIP.>>".
txt <- str_replace_all(txt,"<<THIS[^[:lower:]]+MEMBERSHIP\\.>>","")


# We observe the txt and find that every play starts with {years}
# {Name} {"by William Shakespeare"}.
# More importantly, the new line character is ", ".

# Extract the critical sigments appearing on each play.
# Be careful of the "," " " ";" in the name of each play.
startlocation <- str_locate_all(txt, "[[:digit:]]{4}(, )+[[:upper:] ',;]+by William Shakespeare")
endlocation <- str_locate_all(txt, "THE END")

# The two locations are lists with each component including two integers : start and end.
startlocation <- unlist(startlocation)
endlocation <- unlist(endlocation)
playnumber = length(startlocation)/2

# location[1:playnumber] record the starting / ending addresses of each play in the text file.
# We create a list to record each play(total number = 38).
playlist <- list()
for(i in 1 : playnumber){
  playlist[[i]] <- substr(txt,startlocation[i],endlocation[i+playnumber])
}

# Now that playlist has recorded every play, but we need to delete the first and the
# last plays.
playlist <- playlist[ 3 : playnumber - 1]
playnumber <- length(playlist)
```

2(b)

```r
# Create a function to record the number of acts in each play.
numact <- function(play){
  # The number that "ACT I" appears.
  # Notice that "ACT I." and "ACT 1"(only for act 1) are also possible.
  # "Act II" and "Act IV" are also possible.
  # The act number will not exceed 5, so we only count until five.
  numact1 <- length(unlist(str_extract_all(play,"ACT I[\\. ]|ACT 1")))
  numact2 <- length(unlist(str_extract_all(play,"ACT II[\\. ]|Act II[\\.]|ACT Il\\.")))
  numact3 <- length(unlist(str_extract_all(play,"ACT III[\\. ]")))
  numact4 <- length(unlist(str_extract_all(play,"ACT IV[\\. ]|Act IV[\\.]")))
  numact5 <- length(unlist(str_extract_all(play,"ACT V[\\. ]")))
  totalact <- c(numact1,numact2,numact3,numact4,numact5)
  # if numact(i > n) = 0, we judge that there are n acts.
  return(length(totalact[totalact!=0]))
```

```
}

playnumber = length(playlist)
# Create two lists with length equal to the number of total plays and use them to record
# the metadata and the body of each play.
metadata <- vector("list",length = playnumber)
body <- vector("list",length = playnumber)
for(i in 1 : playnumber){
      play <- playlist[[i]]
      # The first four characters are exactly the year of the play.
      year <- substr(play,1,4)
      # Extract the title from the uppercase letters.
      title <- str_extract(play,"([[:upper:]])[[:upper:] ';,]+([[:upper:]])")
      # Calcualte the number of total acts.
      actnumber <- numact(play)
      # Each scene is preceded by "SCENE" or "Scene".
      # There is a "SCENE" on the beginning of each play that isn't associate to
      # any specified scene.
      scenenumber <- length(unlist(str_extract_all(play,"SCENE|Scene"))) - 1
      metadata[[i]] <- list("year" = year,"title" = title, "actnumber" = actnumber, "scenenumber" = scen
      body[[i]] <- str_extract(play,"(SCENE|Scene).+THE END")
} # end of for loop.
# Show a portion above.
head(metadata,3)
```

```
## [[1]]
## [[1]]$year
## [1] "1603"
##
## [[1]]$title
## [1] "ALLS WELL THAT ENDS WELL"
##
## [[1]]$actnumber
## [1] 5
##
## [[1]]$scenenumber
## [1] 23
##
##
## [[2]]
## [[2]]$year
## [1] "1607"
##
## [[2]]$title
## [1] "THE TRAGEDY OF ANTONY AND CLEOPATRA"
##
## [[2]]$actnumber
## [1] 5
##
## [[2]]$scenenumber
## [1] 42
##
##
## [[3]]
```

```
## [[3]]$year
## [1] "1601"
##
## [[3]]$title
## [1] "AS YOU LIKE IT"
##
## [[3]]$actnumber
## [1] 5
##
## [[3]]$scenenumber
## [1] 22

head(body,1)

## [[1]]
## [1] "SCENE:, Rousillon; Paris; Florence; Marseilles, , , ACT I. SCENE 1., Rousillon. The COUNT'S pala
```

2(c)

```
# First we want to replace the Roman letters with corresponding numbers.
# numbervector is correspondance of the romanvector from V to I.
numbervector <- c(5:1)
romanvector <- as.roman(numbervector)
# Loop for the 36 plays and replace the roman letters with numbers.
for(i in 1 : playnumber){
   for(j in 1 : 5){
     body[[i]] <- str_replace_all(body[[i]], paste("ACT ",romanvector[j],sep = ""), paste("ACT ",number
   }
}

# This function is used to extract only speaking portions without other information.
# A dialogue ends with a lowercase letter(or I) and a punctuation like period, exclamation or
# question mark.
cleanchunk <- function(chunk){
  # Locate where the dialogue ends.
  chunkaddress <- unlist(str_locate_all(chunk,"[[:lower:]I]+[\\.?!]"))
  # If rare case happens, we don't deal with it.
  if(length(chunkaddress) != 0)
    return(substr(chunk,3,chunkaddress[length(chunkaddress)]))
  else
    return(chunk)
}

chunk <- vector("list",length = playnumber)

# Loop for 36 plays.
for(i in 1 : playnumber){
  # Locate where the dialogue begins(Speaker must be uppercase letters and are followed by a period).
  # Sometimes the speaker has a blankspace, such as "FIRST LORD".
  # In some play, speaker will appear like "Ber."
  address <- unlist(str_locate_all(body[[i]],"[^[:upper:]] [[:upper:]]{2,}( [[:upper:]]{2,})?\\."))
```

```
  chunknumber <- length(address) / 2
  chunk[[i]] <- vector("list",length = chunknumber)
  # Loop for all chunks and eliminate unnecessary information.
  for(j in 1 : (chunknumber - 1)){
    chunk[[i]][[j]] <- cleanchunk(substr(body[[i]],address[j],address[j+1]-1))
  }
  # When j = chunknumber, address[j+1] is meaningless(It will be the end location of j=1).
  # So we let the substring end at the end of the original body.
  chunk[[i]][[chunknumber]] <- cleanchunk(substr(body[[i]],address[chunknumber],1e7))
}
head(chunk[[1]],3)

## [[1]]
## [1] "COUNTESS. In delivering my son from me, I bury a second husband."
##
## [[2]]
## [1] "BERTRAM. And I in going, madam, weep o'er my father's death anew;,     but I must attend his Ma
##
## [[3]]
## [1] "LAFEU. You shall find of the King a husband, madam; you, sir, a,     father. He that so generall
```

2(d)

```
chunknumber <- vector("list",length = playnumber)
speaker <- vector("list",length = playnumber)
speakernumber <- vector("list",length = playnumber)
sentencenumber <- vector("list",length = playnumber)
word <- vector("list",length = playnumber)
wordnumber <- vector("list",length = playnumber)
averageword <- vector("list",length = playnumber)
uniqueword <- vector("list",length = playnumber)

# Define a function to count the number of the sentences in each chunk.
# We need to minus 1 since the speaker will also followed by a period.
countsentence <- function(chunk){
  return(length(unlist(str_extract_all(chunk,"[\\.?!]"))) - 1)
}

# Define a function to record the words in each chunk.
# A word contains alphabets and sometimes an apostrophe or a dash.
recordword <- function(chunk){
  return(unlist(str_extract_all(chunk,"[[:alpha:]-']+")))
}

# Loop for all 36 plays.
for(i in 1 : playnumber){
  # Record the chunks and its number within this specified play.
  tempchunk <- chunk[[i]]
  tempchunknumber <- length(tempchunk)
  # Initialize variables.
  speakeri <- vector("list",length = tempchunknumber)
```

```
  sentencenumber[[i]] <- 0
  word[[i]] <- vector("list",length = tempchunknumber)
  wordnumber[[i]] <- 0
  # Loop for each chunk.
  for(j in 1 : tempchunknumber){
    speakeri[[j]] <- str_extract(tempchunk[[j]],"[[:upper:]][[:alpha:]]+\\. [[:upper:]]")
    speakeri[[j]] <- str_replace(speakeri[[j]],"([[:upper:]][[:alpha:]]+)\\. [[:upper:]]","\\1")
    sentencenumber[[i]] = sentencenumber[[i]] + countsentence(tempchunk[[j]])
    word[[i]][[j]] <- recordword(tempchunk[[j]])
    # We need to minus 1 otherwise it will count speaker as a word.
    wordnumber[[i]] <- wordnumber[[i]] + length(word[[i]][[j]]) - 1
  }
  # Eliminate duplicate speakers.
  speakeri <- unique(speakeri)
  # Eliminate NULL & NA situations.
  speaker[[i]] <- speakeri[ !is.null(speakeri) & !is.na(speakeri) ]
  speakernumber[[i]] <- length(speaker[[i]])
  chunknumber[[i]] <- tempchunknumber
  averageword[[i]] <- wordnumber[[i]]/tempchunknumber
  uniqueword[[i]] <- length(unique(unlist(word[[i]])))
}
# 4(i)
# The number of unique speakers.
unlist(speakernumber)

##  [1] 20 52 22 16 41 31  6 10 46 49 51 59 42 47 26 47 10 23 41 24 22 23 32
## [24]  8 25 36 54  8 34 19 43 24 28 18 14 29

# 4(ii)
# The number of spoken chunks.
unlist(chunknumber)

##  [1]  940 1208  827  270 1129  877   16   13  918  763  652  809  821  714
## [15]  550  806   18  949  664  909  622 1037  508   10 1190  552 1073   15
## [29]  903  643  783  576 1144  937  871  754

# 4(iii)
# The number of sentences.
unlist(sentencenumber)

##  [1] 1722 2328 1529 1402 2105 2170 3734 2851 1961 1473 1463 1644 1675 1604
## [15] 1230 1823 3625 1877 1648 1672 1388 2073 1169 2628 2668 1333 2130 3006
## [29] 1649 1266 1633 1316 2268 1646 1373 1715

# The number of words spoken.
unlist(wordnumber)

##  [1] 22816 24253 21523 15570 27093 27114 30468 24706 26009 25909 21060
## [12] 24936 24023 24309 20654 19841 24834 21457 17357 21410 21175 21446
## [23] 16126 19731 26457 22043 28997 23516 20616 16230 18177 20241 25271
## [34] 19745 17008 24745

# average number of words per chunk.
unlist(averageword)
```

```
##  [1]   24.27234   20.07699   26.02539   57.66667   23.99734   30.91676
##  [7] 1904.25000 1900.46154   28.33224   33.95675   32.30061   30.82324
## [13]   29.26066   34.04622   37.55273   24.61663 1379.66667   22.61012
## [19]   26.14006   23.55336   34.04341   20.68081   31.74409 1973.10000
## [25]   22.23277   39.93297   27.02423 1567.73333   22.83056   25.24106
## [31]   23.21456   35.14062   22.09003   21.07257   19.52698   32.81830

# 4(iv)
# The number of unique words.
unlist(uniqueword)

##  [1] 4042 4675 3734 2891 4733 4984 5336 4325 4725 5208 4424 4696 4119 4387
## [15] 4111 3381 4623 4349 3900 3824 3761 3723 3430 3245 4361 4269 4819 4146
## [29] 3776 3687 3868 3905 4849 3598 3136 4582
```

2(e)(f)

By carefully checking every body of each play, there are five plays that have inconsistent formatting. Play 7,8,17,24,28 have speakers with lowercase letters such as "Bers." That is the reason causing the particularly small number of speakers. We try to customize an algorithm to deal with them.

```
# Slightly adjust the cleanchunk function.
cleanchunk2 <- function(chunk){
  # Locate where the dialogue ends.
  chunkaddress <- unlist(str_locate_all(chunk,"[[:lower:]I]+[\\.?!]"))
  # If rare case happens, we don't deal with it.
  if(length(chunkaddress) != 0)
    return(substr(chunk,5,chunkaddress[length(chunkaddress)]))
  else
    return(chunk)
}
# Loop for the five plays.
for(i in c(7,8,17,24,28)){
  # First we extract the chunks from the body.
  tempaddress <- unlist(str_locate_all(body[[i]],",   [A-Z][a-z]{2,4}\\. [A-Z]"))
  chunknumber[[i]] <- length(tempaddress) / 2
  chunk[[i]] <- vector("list",length = chunknumber[[i]])
  for(j in 1 : (chunknumber[[i]] - 1)){
    chunk[[i]][[j]] <- cleanchunk2(substr(body[[i]],tempaddress[[j]],tempaddress[[j+1]]-1))
  }
  chunk[[i]][[chunknumber[[i]]]] <- cleanchunk2(substr(body[[i]],tempaddress[[chunknumber[[i]]]],1e7))
  # Then extract the speakers.
  speakeri <-  vector("list",length = chunknumber[[i]])
  for(j in 1 : chunknumber[[i]]){
    speakeri[[j]] <- str_extract(chunk[[i]][[j]],"[A-Z][a-z]{2,4}")
  }
  speakeri <- unique(speakeri)
  speaker[[i]] <- speakeri[ !is.null(speakeri) & !is.na(speakeri) ]
  speakernumber[[i]] <- length(speaker[[i]])
  averageword[[i]] <- wordnumber[[i]]/chunknumber[[i]]
}
# Again show the statistics
unlist(speakernumber)
```

```
##  [1] 20 52 22 16 41 31 26 27 46 49 51 59 42 47 26 47 19 23 41 24 22 23 32
## [24] 22 25 36 54 26 34 19 43 24 28 18 14 29
```

```
unlist(chunknumber)
```

```
##  [1]  940 1208  827  270 1129  877 1048  552  918  763  652  809  821  714
## [15]  550  806 1001  949  664  909  622 1037  508  932 1190  552 1073  755
## [29]  903  643  783  576 1144  937  871  754
```

```
unlist(sentencenumber)
```

```
##  [1] 1722 2328 1529 1402 2105 2170 3734 2851 1961 1473 1463 1644 1675 1604
## [15] 1230 1823 3625 1877 1648 1672 1388 2073 1169 2628 2668 1333 2130 3006
## [29] 1649 1266 1633 1316 2268 1646 1373 1715
```

```
unlist(wordnumber)
```

```
##  [1] 22816 24253 21523 15570 27093 27114 30468 24706 26009 25909 21060
## [12] 24936 24023 24309 20654 19841 24834 21457 17357 21410 21175 21446
## [23] 16126 19731 26457 22043 28997 23516 20616 16230 18177 20241 25271
## [34] 19745 17008 24745
```

```
unlist(averageword)
```

```
##  [1] 24.27234 20.07699 26.02539 57.66667 23.99734 30.91676 29.07252
##  [8] 44.75725 28.33224 33.95675 32.30061 30.82324 29.26066 34.04622
## [15] 37.55273 24.61663 24.80919 22.61012 26.14006 23.55336 34.04341
## [22] 20.68081 31.74409 21.17060 22.23277 39.93297 27.02423 31.14702
## [29] 22.83056 25.24106 23.21456 35.14062 22.09003 21.07257 19.52698
## [36] 32.81830
```

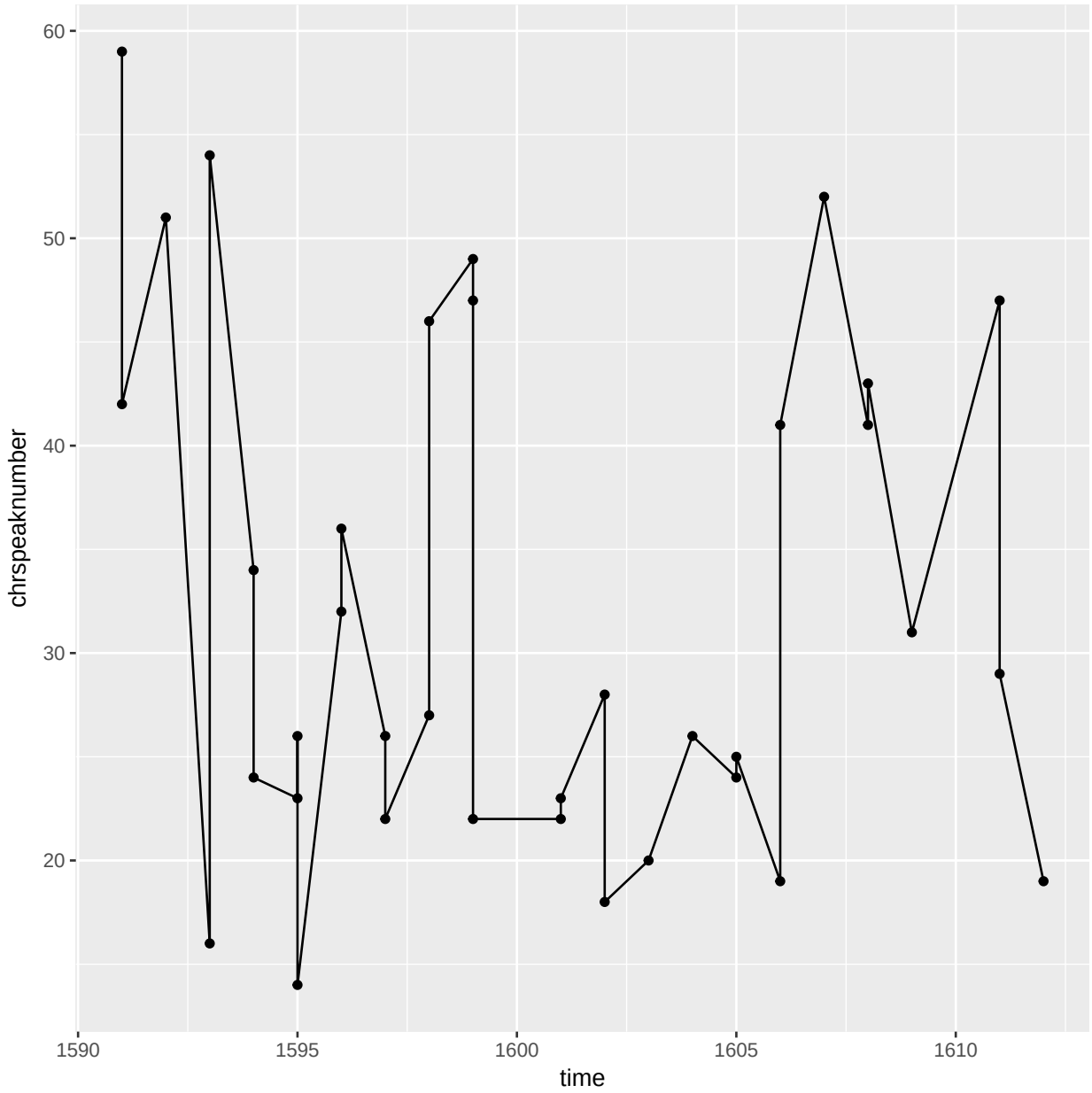
```
unlist(uniqueword)
```

```
##  [1] 4042 4675 3734 2891 4733 4984 5336 4325 4725 5208 4424 4696 4119 4387
## [15] 4111 3381 4623 4349 3900 3824 3761 3723 3430 3245 4361 4269 4819 4146
## [29] 3776 3687 3868 3905 4849 3598 3136 4582
```

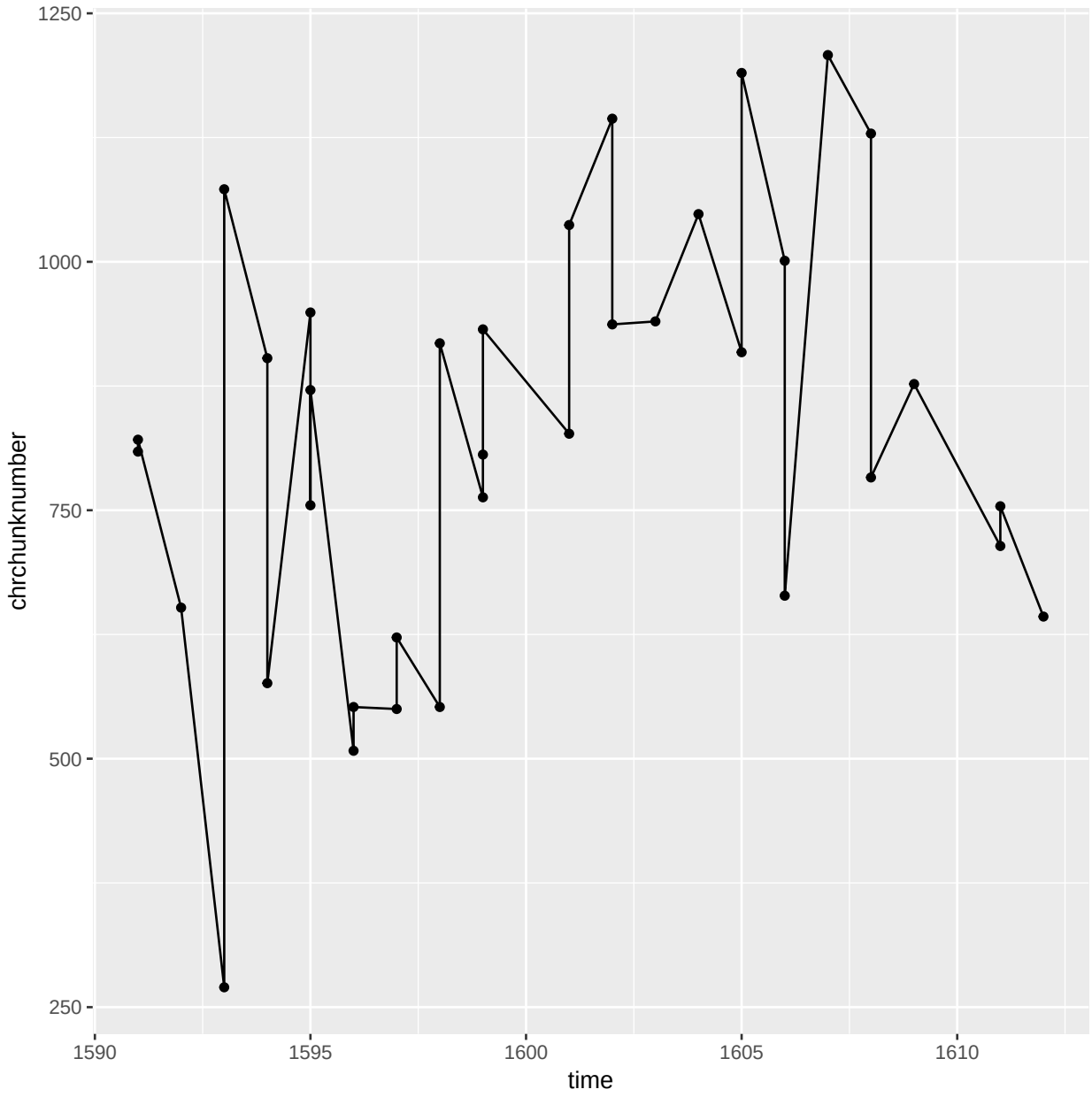
It looks more reasonable compared to the result without adjustment. Then we plot the diagrams to illustrate these statistics.

```
# Extract the year information of each play.
year <- vector("list",length = playnumber)
for(i in 1 : playnumber){
  year[[i]] <- as.integer(metadata[[i]][[1]])
}
# Sort the year in chronological order.
# return = TRUE gives an index vector which shows the order.
year <- unlist(year)
time <- sort.int(year,index.return = TRUE)$x
index <- sort.int(year,index.return = TRUE)$ix

# Create correspoding statistics above in chronological orders.
chrspeaknumber <- vector("numeric",length = playnumber)
chrchunknumber <- vector("numeric",length = playnumber)
chrsentencenumber <- vector("numeric",length = playnumber)
chrwordnumber <- vector("numeric",length = playnumber)
```

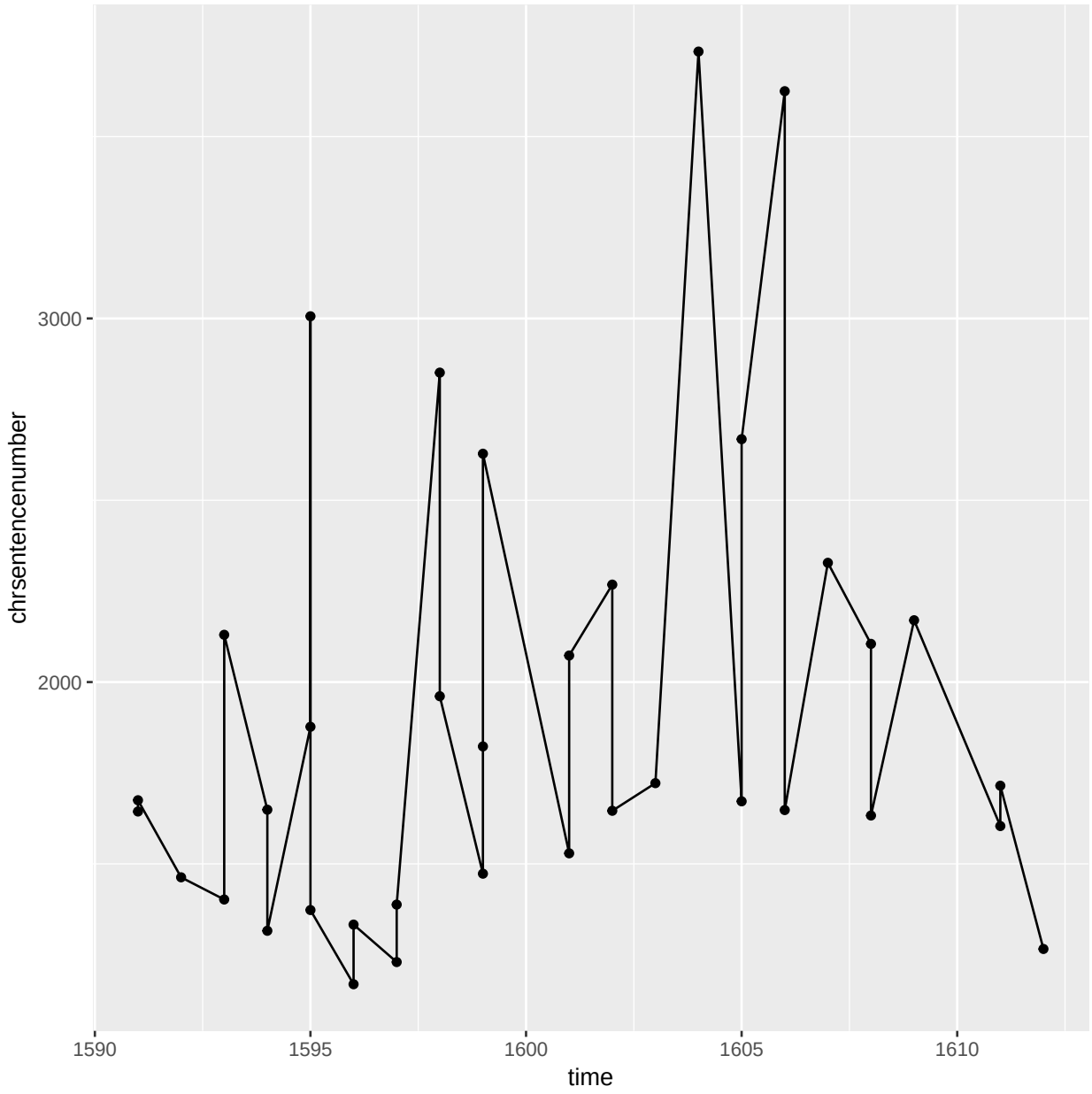
```
chraverageword <- vector("numeric",length = playnumber)
chruniqueword <- vector("numeric",length = playnumber)
chractnumber <- vector("numeric",length = playnumber)
chrscenenumber <- vector("numeric",length = playnumber)

# Using index to help us rearrage the order chronologically.
for(i in 1 : playnumber){
  chrspeaknumber[i]<- as.integer(speakernumber[[index[i]]])
  chrchunknumber[i] <- as.integer(chunknumber[[index[i]]])
  chrsentencenumber[i] <- as.integer(sentencenumber[[index[i]]])
  chrwordnumber[i] <- as.integer(wordnumber[[index[i]]])
  chraverageword[i] <- averageword[[index[i]]]
  chruniqueword[i] <- uniqueword[[index[i]]]
  chractnumber[i] <- metadata[[i]][[3]]
  chrscenenumber[i] <- metadata[[i]][[4]]
}
# Enclosure all statistics into a dataframe
data <- data.frame(time,chrspeaknumber,chrchunknumber,chrsentencenumber,chrwordnumber,chraverageword,ch
# Plot above statistics in function of times.
ggplot(data,aes(time,chrspeaknumber)) + geom_line() + geom_point()
```



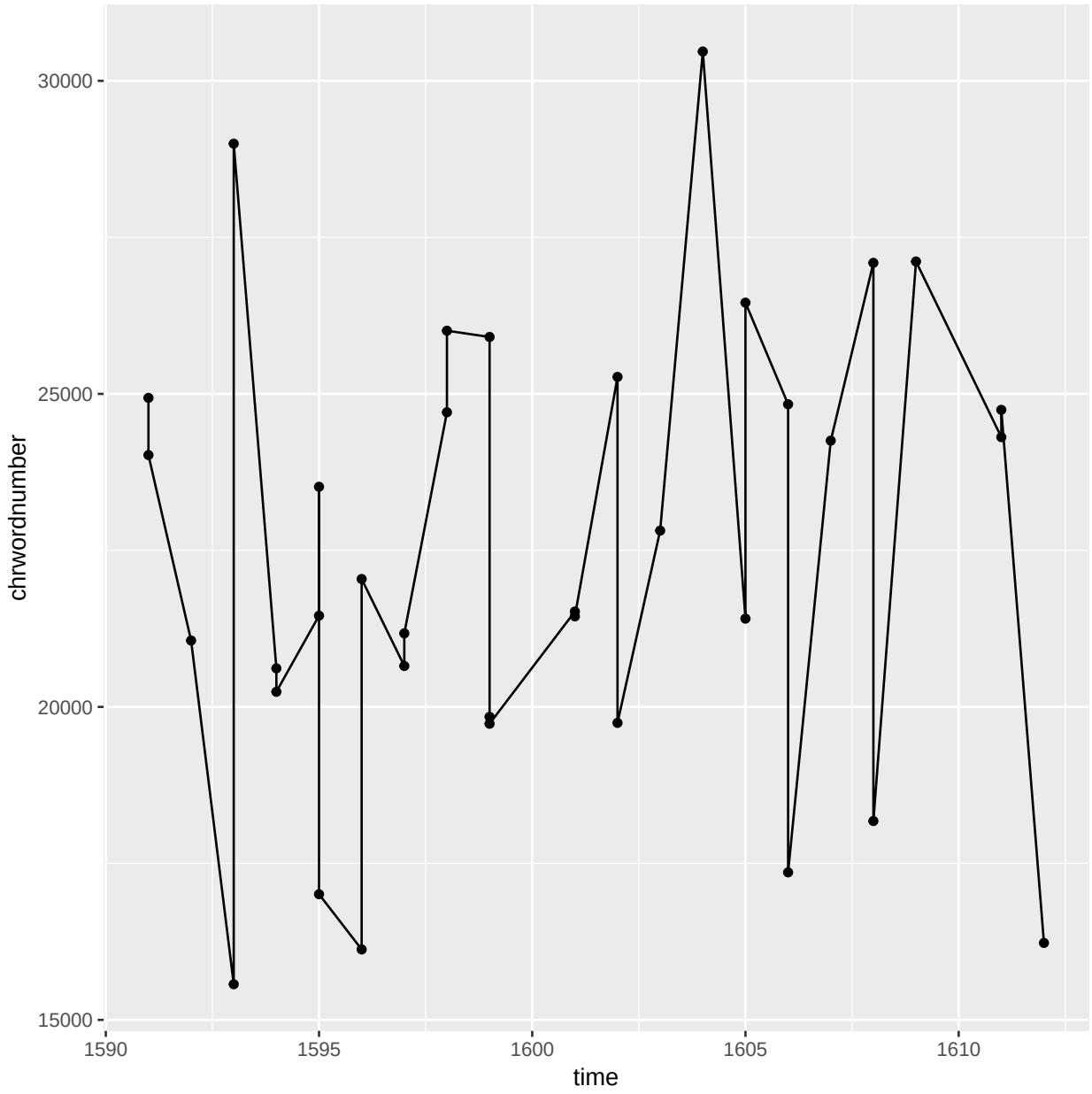

```
ggplot(data,aes(time,chrchunknumber)) + geom_line() + geom_point()
```



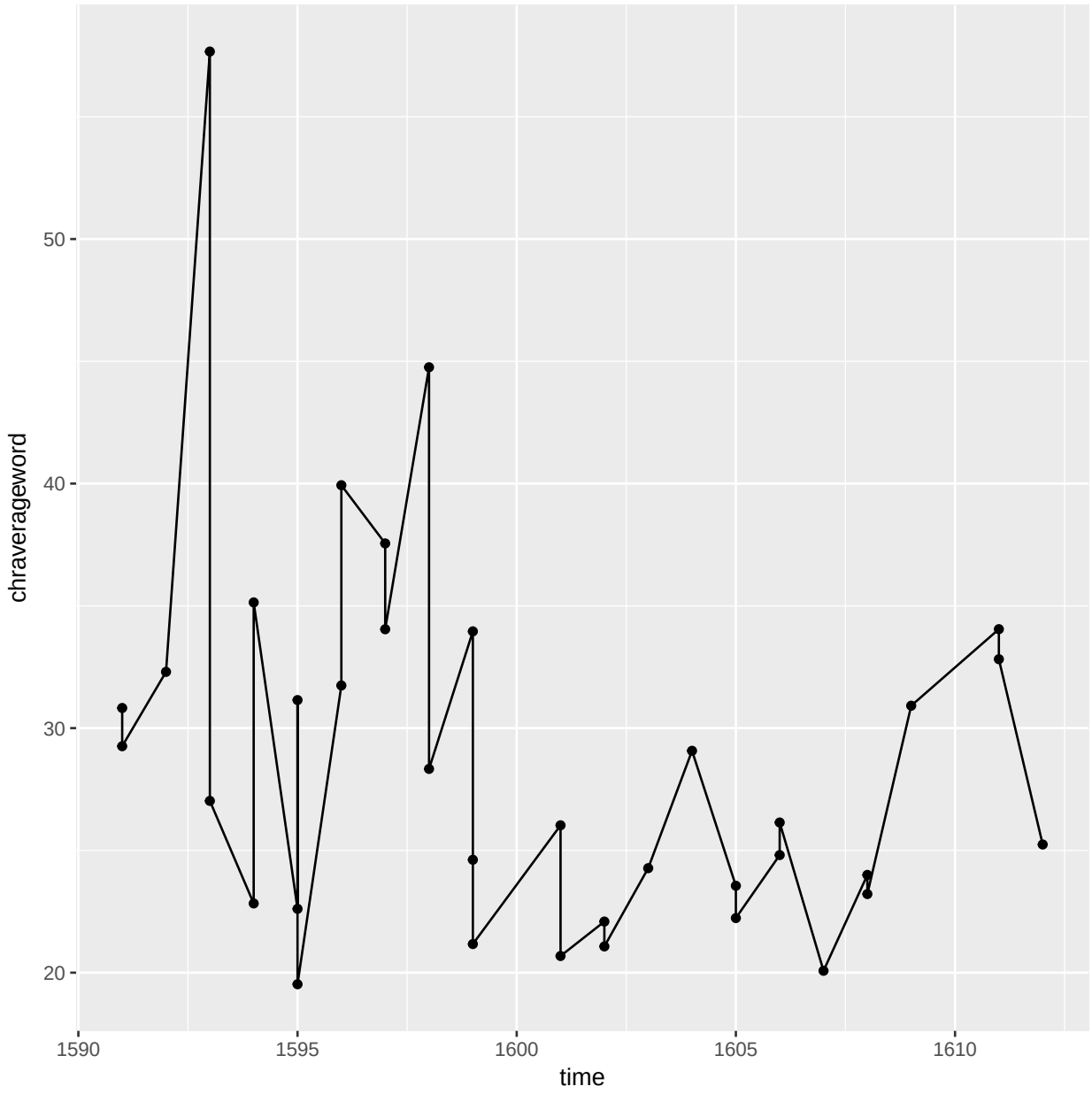

```
ggplot(data,aes(time,chrsentencenumber)) + geom_line() + geom_point()
```



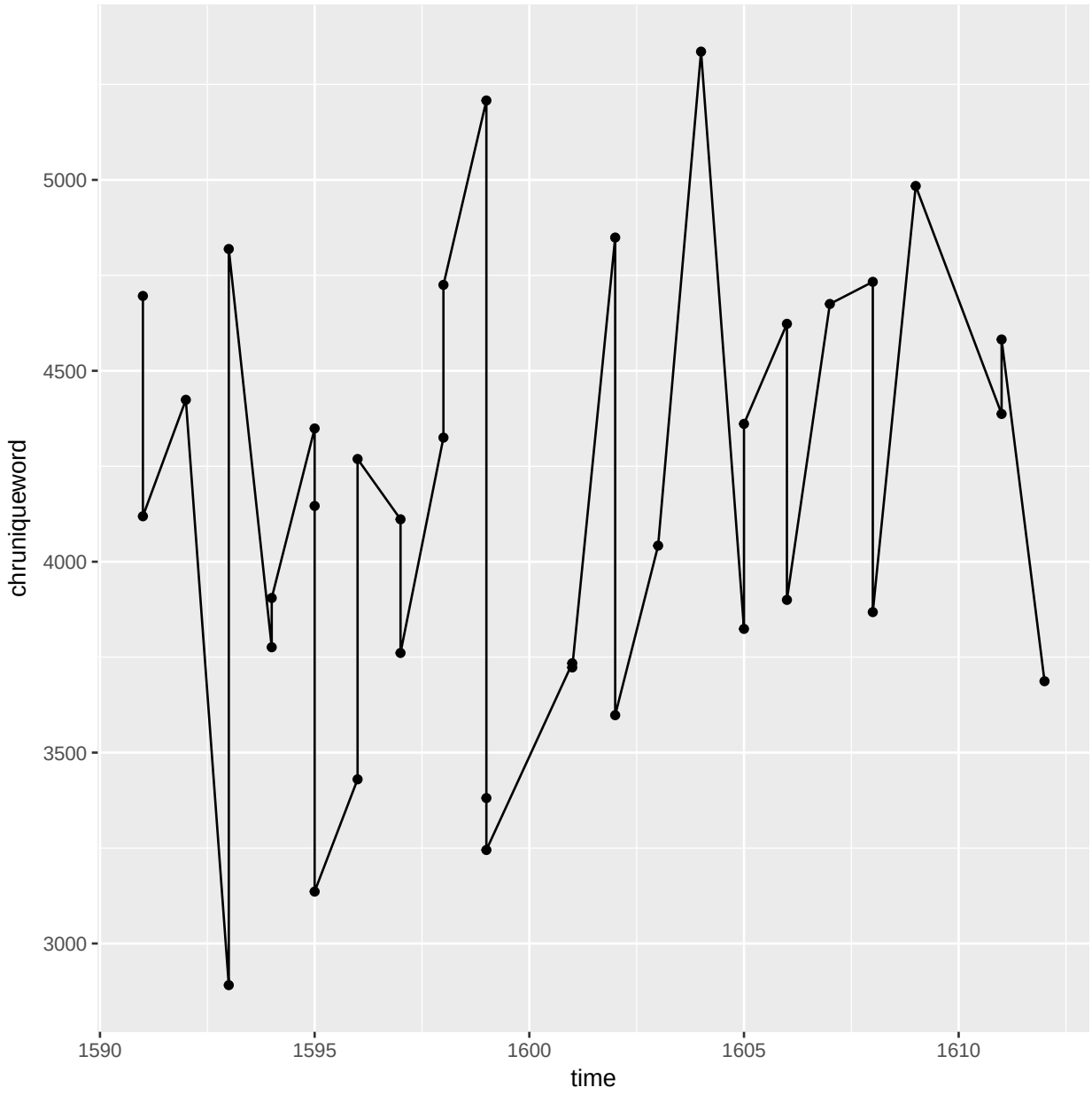

```
ggplot(data,aes(time,chrwordnumber)) + geom_line() + geom_point()
```



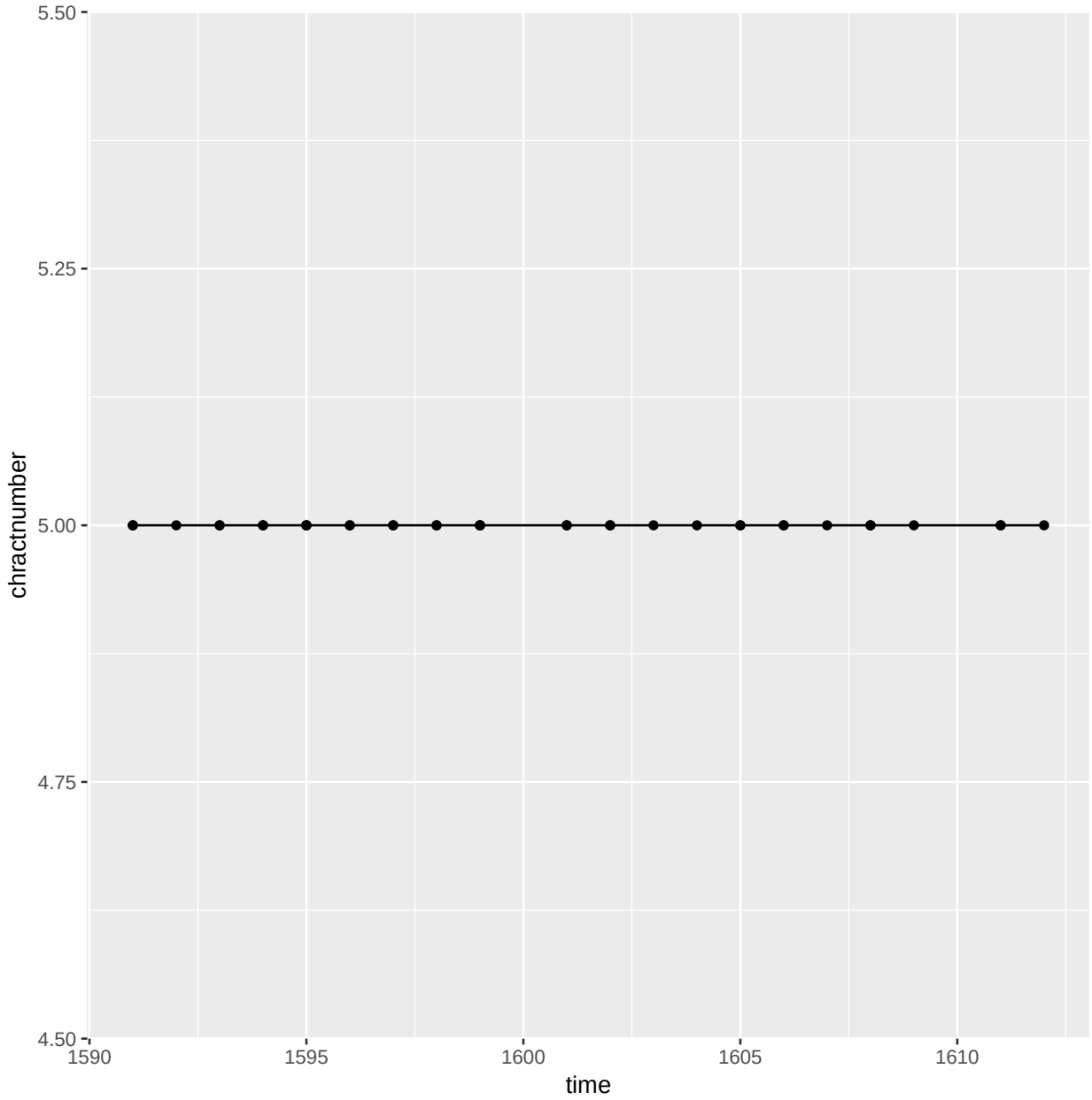

```
ggplot(data,aes(time,chraverageword)) + geom_line() + geom_point()
```



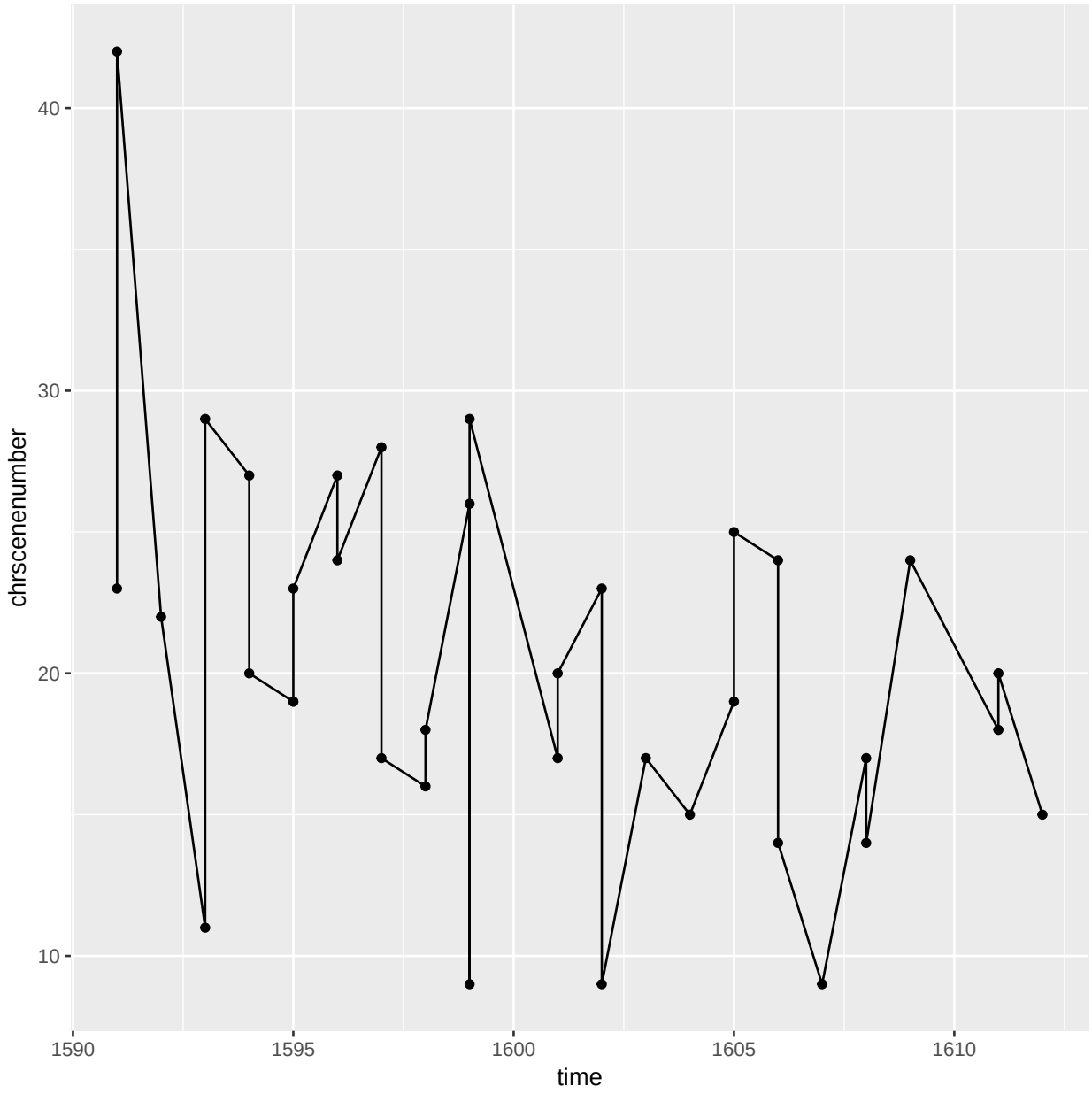

```
ggplot(data,aes(time,chruniqueword)) + geom_line() + geom_point()
```




```
ggplot(data,aes(time,chractnumber)) + geom_line() + geom_point()
```




```
ggplot(data,aes(time,chrscenenumber)) + geom_line() + geom_point()
```



3(a)

We can define a class called "Play" to create a Shakespeare-style play. The slots(fields) are title, year and scene. Scene field is a list indicating how many scenes are in each ACT. For example, we want a play with act 1 having 2 scenes and act 2 having 3 scenes. Then, scene field = list(2,3) More complicatedly, we can set chunks and speakers in the play.

```
library(methods)
setClass("Play",representation(title = "character", year = "numeric", scene = "list", body = "list")
  ,prototype(title = "I LOVE SHAKESPEARE", year = 2017, scene = list(1L), body = list("TRUMP : Make Ame
)
setValidity("Play",
            function(object){
```

```
              if(!is.character(object@title))
                   return("Error : Title must be a string.")
              if(!(object@year %in% 1:2017 ))
                   return("Error : Year must be between 1 and 2017.")
              if(!is.list(object@scene))
                   return("Scene must be a list indicating how many scenes are in each ACT.")
              if(!is.list(object@body)||length(object@body) != length(object@scene))
                   return("Body must be a list with length equal to scene.")
              if(!is.integer(unlist(object@scene)))
                   return("Scene should be an integer list.")
              if(!is.character(unlist(object@body)))
                   return("Body should be a string list.")
              return(TRUE)
            }
)

## Class "Play" [in ".GlobalEnv"]
##
## Slots:
##
## Name:      title      year     scene      body
## Class: character   numeric      list      list

# Try to create a object with wrong arguments.
myplay <- new("Play",title = "WELCOME MY PLAY", year = 2000, scene = list(1L,1L,1L), body = list("HE : I

## Error in validObject(.Object):  invalid class "Play" object:  Body must be a list with length
equal to scene.

# Try to create a object with correct arguments.
myplay <- new("Play",title = "WELCOME MY PLAY", year = 2000, scene = list(1L,1L), body = list("HE : Hey
```

3(b)

Metadata method returns a list including year, title, number of acts and scenes just like 2.(b). Body can be acquired directly by object@body. If we would like to acquire some statistics such as number of unique words/speakers, we could write a method that processes body of the play(object@body) as argument by the algorithm above.

```
setGeneric("metadata",function(object){
  standardGeneric("metadata")
})

## [1] "metadata"

metadata.Play <- function(object){
  metadata <- vector("list",length = 4)
  metadata[[1]] <- object@title
  metadata[[2]] <- object@year
  metadata[[3]] <- length(object@scene)
  # Sum the number of scene in each act.
  metadata[[4]] <- Reduce("+",object@scene)
  return(metadata)
```

```
}
setMethod(metadata, signature = c("Play"), definition = metadata.Play)

## [1] "metadata"
## attr(,"package")
## [1] ".GlobalEnv"

metadata(myplay)

## [[1]]
## [1] "WELCOME MY PLAY"
##
## [[2]]
## [1] 2000
##
## [[3]]
## [1] 2
##
## [[4]]
## [1] 2
```