Problem 1

To have a function that lists the objects in the workspace in a nicely formatted manner, serveral auxiliary printing function (*printObj*, *printUnit*) is created first for cleaner coding.

```
## Aux Printing Fun: Define printing control global parameter
wid = 20  # object name priting width
digit = 10  # byte size printing digits
## Print out file size according to auto-match units
printUnit <- function(obj) {
    units = c("", "k", "M", "G")
    i = 1
    while (obj >= 1024) {
        obj = obj/1024
        i = i + 1
    }  # get proper units
    if (i == 1) {
        cat(format(obj, width = digit, justify = "right"), "\n", sep = "")
    } else {
        cat(format(as.integer(obj), width = digit - 1, justify = "right"), units[i],
            "\n", sep = "")
    }
}
## Print out the object list in a nicely formatted way
printObj <- function(objList, objName, pretty) {
    cat(format("object", width = wid), format("bytes", width = digit, justify = "right"),
        "\n", sep = "")
    if (pretty == FALSE) {
        # pretty==TRUE: it will call the printUnit to print out proper units for sizes
        t <- sapply(1:length(objList), function(i) {
            cat(format(objName[i], width = wid), format(objList[i], width = digit,
                justify = "right"), "\n", sep = "")
        })
    } else {
        # default:      it will just print out number of bytes
        t <- sapply(1:length(objList), function(i) {
            cat(format(names(objList)[i], width = wid))
            printUnit(objList[i])
        })
    }
}
```

The main function will list objects ordered by size from largest to smallest; it allows the user to either specify how many objects to list or to specify that objects larger than a certain size to be listed. The argument *pretty*, when TRUE, results in the size of the objects being printed with the units tailored by function *printUnit*. By default, *lsObj<-function(numls=100, sizeB=0, pretty=FALSE)* will give sensible results. Also, the main function shown below is already written to be able to print out only objects in the frame of the function call when called from within another function.

```
lsObj <- function(numls = 100, sizeB = 0, pretty = FALSE) {
    listLs <- ls(envir = parent.frame())  # object list
    sizeLs <- sapply(listLs, function(x) {
        object.size(get(x, envir = parent.frame(n = 4)))
    })
    # get all object sizes from within the called frame;
    # n=4:FUN->sapply->lsObj->call(lsObj)
    tmpEnv <- NULL
    nameEnv <- NULL  # store object sizes and names within a certain user-defined ENV
    tmpFun <- NULL
    nameFun <- NULL  # store object sizes and names within a function closure
    for (item in listLs) {
        # elborate objects to find hidden ones
        objitem <- get(item, envir = parent.frame())
        if (is.environment(objitem)) {
            # get sizes and store names with ENV prefix
            tt <- sapply(ls(envir = objitem), function(x) {
                object.size(get(x, envir = objitem))
            })
            tmpEnv <- c(tmpEnv, tt)
            nameEnv <- c(nameEnv, paste(item, "$", ls(envir = objitem), sep = ""))
        }
        if (is.function(objitem) && identical(environment(objitem), parent.frame()) ==
            FALSE) {
            # get sizes and store names with FUN prefix
            tt <- sapply(ls(envir = environment(objitem)), function(x) {
                object.size(get(x, envir = environment(objitem)))
            })
            tmpFun <- c(tmpFun, tt)
            nameFun <- c(nameFun, paste(item, "$", ls(envir = environment(objitem)),
                sep = ""))
        }
    }
    names(tmpEnv) <- nameEnv
    names(tmpFun) <- nameFun
    if (length(tmpEnv) > 0)
        {
            sizeLs <- c(sizeLs, tmpEnv)
        }  #if there is use-defined ENV
    if (length(tmpFun) > 0)
        {
            sizeLs <- c(sizeLs, tmpFun)
        }  #if there is FUN closure
    sizeLs <- sizeLs[sizeLs[] > sizeB]  # find objects larger than sizeB(bytes)
    if (numls > length(sizeLs))
        {
            numls = length(sizeLs)
        }  #prune incorrect listing number
    sizeLs <- sort(sizeLs, decreasing = TRUE)[1:numls]  # find the numls largest objects
    nameLs <- names(sizeLs)  # get the name of vector
    printObj(sizeLs, nameLs, pretty)  # print using the aux functions
}
```

The above main function has the capability to deal with user-created environments and objects within a closure. It elborates the objects obtained from the *ls()* function to see if there are user-defined ENV or FUN closure. When detected, it then goes one frame down to the enclosing environment to get the hidden objects and assign names to them with ENV/FUN prefix.

To create objects of various sizes for testing, I generated vectors of random normals of different lengths, along with serveral hidden ENV/FUN examples for testing.

```
#### Generate objects test cases
ob0 <- rnorm(1)
ob1 <- rnorm(10)
ob2 <- rnorm(100)
ob3 <- rnorm(1000)
ob4 <- rnorm(10000)
ob5 <- rnorm(1e+05)
ob6 <- rnorm(1e+06)
ob7 <- rnorm(1e+07)
```

```
## **********Test Case 1**********
## lsObj(5,128)

## object                   bytes
## ob7                      8e+07
## ob6                      8e+06
## ob5                      8e+05
## ob4                      80040
## ob3                       8040

## **********Test Case 2**********
## lsObj(sizeB = 1024)

## object                   bytes
## ob7                      8e+07
## ob6                      8e+06
## ob5                      8e+05
## ob4                      80040
## ob3                       8040

## **********Test Case 3**********
## lsObj(numls = 3)

## object                   bytes
## ob7                      8e+07
## ob6                      8e+06
## ob5                      8e+05

## **********Test Case 4**********
## lsObj(pretty = TRUE)

## object                   bytes
## ob7                        76M
## ob6                         7M
## ob5                       781k
## ob4                        78k
```

```
## ob3                          7k
## ob2                        840
## ob1                        168
## ob0                         48
```

```
testFunc <- function () {
             data <- rnorm(10000); data2 <- rnorm(100000)
             myFun <- function(theta) {
                       dist <- rnorm(theta); return(exp(dist / theta)); }
lsObj()}
testEx <- function () {
           x <- rnorm(100000); data <- rnorm(1000)
           e <- new.env(); e$a <- x # an object hidden in an environment
           e2 <- new.env(); e2$a2 <- x; e2$b2 <- 52
           myFun <- function(theta) {
                     dist <- rnorm(theta); return(exp(dist / theta)); }
           myFun1 <- function(theta) {
                      dist <- rnorm(theta); return(exp(dist / theta)); }
           myFun2 <- with(list( data = x ), # an object hidden in a closure
                          function(theta) {
                          dist <- rdist(data); return(exp(dist / theta)); } )
            myFun3 <- with(list( data2 = x ),
                          function(theta2) {
                          dist <- rdist(data); return(exp(dist / theta2)); } )
lsObj()}
```

```
## **********Test Case 5**********
## lsObj() in testFunc()

## object                    bytes
## data2                     8e+05
## data                      80040
## myFun                      6072

## **********Test Case 6**********
## lsObj() in test testEx()

## object                    bytes
## x                         8e+05
## e$a                       8e+05
## e2$a2                     8e+05
## myFun2$data               8e+05
## myFun3$data2              8e+05
## data                       8040
## myFun                      6072
## myFun1                     6072
## myFun2                     6072
## myFun3                     6072
## e                            56
## e2                           56
## e2$b2                        48
```

Problem 2

After reading in the character vector text from the online traffic logs *IPs.RData*, the code below could extract out all the IP numbers. In the meanwhile, it can determine how many IP addresses are in each element of the vector with function *getIPnum*.

```
patIP <- "((\\d{1,3}\\.){3}\\d{1,3})"  #Perl style pattern for IP
getIPnum <- function(t) {
    if (length(grep(patIP, t, perl = TRUE)) == 0) {
        return(0)
    } else {
        return(length(gregexpr(patIP, t, perl = TRUE)[[1]]))
    }
}  # get number of IPs per text element
getIPidx <- function(t) {
    return(gregexpr(patIP, t, perl = TRUE)[[1]])
}  # get index of IPs in text element
ipNum <- sapply(text, getIPnum, USE.NAMES = FALSE)  # apply to the character vector
ipIdx <- lapply(text, getIPidx)  # maintain list structure of index for each IP log
ipStr <- regmatches(text, ipIdx)
# obtain IPs in list; multiple IPs per element stored in list items
```

The results of getting the number of IPs within one element of *text* is one integer vector of the length *length(text)*. All the extracted IPs are stored in the list *ipStr*, with each element of a character vector in length *ipNum[i]* of all the IPs from line *text[i]*. NA or no-IP results are treated as empty string.

```
## The results of # of IPs in each text element (1:100):

##   [1] 1 0 1 0 2 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 0 1 0 1 1 1 0 0 1 1
##  [36] 2 1 1 1 1 1 0 0 0 1 2 0 0 1 0 1 1 1 1 1 1 1 1 1 1 2 0 1 0 0 1 0 0 0 0
##  [71] 0 2 1 1 0 1 1 1 0 0 1 0 1 1 0 1 2 1 1 2 1 0 1 0 1 1 1 1 1 1

## The first 10 lines of results:

## $`from munnari.OZ.AU (localhost [127.0.0.1]) by delta.cs.mu.OZ.AU
## (8.11.6/8.11.6) with ESMTP id g7MBQPW13260; Thu, 22 Aug 2002 18:26:25
## +0700 (ICT)`
## [1] "127.0.0.1"
##
## $`from SpoolDir by EMS-SRV0 (Mercury 1.44); 22 Aug 02 14:50:31 +0000`
## character(0)
##
## $`from ee.ed.ac.uk (sxs@dunblane [129.215.34.86]) by
## postbox.ee.ed.ac.uk (8.11.0/8.11.0) with ESMTP id g7ME1Li02942 for
## <forteana@yahoogroups.com>; Thu, 22 Aug 2002 15:01:21 +0100 (BST)`
## [1] "129.215.34.86"
##
## $`from SpoolDir by EMS-SRV0 (Mercury 1.44); 22 Aug 02 15:01:34 +0000`
## character(0)
##
## $`from [192.168.0.4] (chello062178142216.4.14.vie.surfer.at
## [62.178.142.216]) (authenticated bits=0) by mail.uptime.at (8.12.5/8.12.5)
## with ESMTP id g7MEI7Vp022036 for
## <spamassassin-devel@lists.sourceforge.net>; Thu, 22 Aug 2002 16:18:07
```

```
## +0200`
## [1] "192.168.0.4"    "62.178.142.216"
##
## $`from m206-56.dsl.tsoft.com ([198.144.206.56] helo=perkel.com) by
## darwin.ctyme.com with asmtp (TLSv1:RC4-MD5:128) (Exim 3.35 #1) id
## 17htgP-0004te-00; Thu, 22 Aug 2002 08:15:37 -0700`
## [1] "198.144.206.56"
##
## $`by jlooney.jinny.ie (Postfix, from userid 500) id 4F57189D;
## Thu, 22 Aug 2002 16:25:45 +0100 (IST)`
## character(0)
##
## $`from dcu.ie (136.206.21.115) by hawk.dcu.ie (6.0.040) id
## 3D6203D3000136AD for iiu@taint.org; Thu, 22 Aug 2002 16:59:17 +0100`
## [1] "136.206.21.115"
##
## $`from [66.218.67.174] by n19.grp.scd.yahoo.com with NNFMP;
## 22 Aug 2002 16:11:27 -0000`
## [1] "66.218.67.174"
##
## $`from [66.218.67.189] by n10.grp.scd.yahoo.com with NNFMP;
## 22 Aug 2002 16:17:40 -0000`
## [1] "66.218.67.189"
```

Problem 3

The American Presidency Project at UCSB has the text from all of the State of the Union speeches by US presidents, in which the president speaks to Congress to report on the situation in the country. We will use web scraping, text formatting and pattern matching to grab the data; and then do some statistical analysis on them.

(a)

From the website, I download the *index.html* file and use pattern matching to pull out the individual URLs for each speech in order to download individual HTML files. Files are converted to UNIX line-ending using *fromdos*.

```
#### Download all html files
system("wget -q -O 'index_pres.html'
        'http://www.presidency.ucsb.edu/sou.php#axzz265cEKp1a'")
system("fromdos index_pres.html")
indexPres <- readLines('index_pres.html',warn= FALSE)
# Get speech text source
patUrl1 <-
'\\s{16}<td width=\\"\\d{2}\\" align=\\"center\\" class=\\"doclist\\"><a href=\\"'
indexPres <- indexPres[grep(patUrl1,indexPres,perl= TRUE)]
indexPres <- sapply(indexPres,
                    function(x){gsub(patUrl1,"",x)},USE.NAMES= FALSE)
patUrl2 <- '\\">\\d{4}<\\/a>(\\*|)<\\/td>'
indexPres <- sapply(indexPres,
                    function(x){gsub(patUrl2,"",x)},USE.NAMES= FALSE)
# Get file id from the source url
patUrl3 <-
'http:\\/\\/www\\.presidency\\.ucsb\\.edu\\/ws\\/index\\.php\\?pid='
fileid <- sapply(indexPres,
                 function(x){gsub(patUrl3,"",x)},USE.NAMES= FALSE)
# Download all files and convert to unix
sapply(1:length(fileid),
       function(i){
       system(paste("wget -q -O '",fileid[i],".html' '",indexPres[i],"'",sep=""));
       system(paste("fromdos ",fileid[i],".html",sep=""))})
```

(b)

For each speech, I use pattern matching to extract the body of the speech while retaining the name of the president and the year of the speech. The function is applied with vector operations using *sapply()*.

For the *speechVec*, text pre-processing is done by

1. Replacing HTML line-end with UNIX ones
2. Removing all the HTML format operators
3. Modifying all the HTML special characters to similar UTF-8 ones

```
# Import all *.html lines
ff <- sapply(fileid, function(x) {
    readLines(paste(x, ".html", sep = ""), warn = FALSE)
})
# Get the president name
patName <- "^<title>(.*?)<\\/title>"
namePres <- ff[grep(patName, ff, perl = TRUE)]
namePres <- sapply(namePres, function(x) {
    gsub(patName, "\\1", x)
}, USE.NAMES = FALSE)
namePres <- sapply(namePres, function(x) {
    return(unlist(strsplit(x, ":"))[1])
}, USE.NAMES = FALSE)
# Get the talk date
patDate <- "^.*<span class=\\\"docdate\\\">(.*?)<\\/span>.*$"
dateTalk <- ff[grep(patDate, ff, perl = TRUE)]
dateTalk <- sapply(dateTalk, function(x) {
    gsub(patDate, "\\1", x)
}, USE.NAMES = FALSE)
dateTalk <- sapply(dateTalk, function(x) {
    gsub("^.*\\s", "", x)
}, USE.NAMES = FALSE)
# Get the speech content text and prune for nice-format print
patText <- "^.*<span class=\\\"displaytext\\\">(.*?)<\\/span>.*$"
speechVec <- ff[grep(patText, ff, perl = TRUE)]
speechVec <- sapply(speechVec, function(x) {
    gsub(patText, "\\1<p>", x)
}, USE.NAMES = FALSE)  # grab speech text
speechVec <- sapply(speechVec, function(x) {
    gsub("(<p.*?>|<\\/p>|<br>)", "\n", x)
}, USE.NAMES = FALSE)  # for line ending
speechVec <- sapply(speechVec, function(x) {
    gsub("<.*?>", "", x)
}, USE.NAMES = FALSE)  # remove all html format
speechVec <- sapply(speechVec, function(x) {
    x <- gsub("&mdash;", " -- ", x)
    x <- gsub(" ", "  ", x)
    x <- gsub("&lsquo;", " ' ", x)
    x <- gsub("&#8226;", " \\. ", x)
    x <- gsub("&lt;", " < ", x)
    x <- gsub("&deg;", " degree ", x)
    x <- gsub("&pound;", " pound ", x)
    x <- gsub("&fra.*?;", " 1/2 ", x)
    x <- gsub("&O.*?;", "O", x)
    x <- gsub("&e.*?;", "e", x)
}, USE.NAMES = FALSE)  # html special char
```

(c)

Each speech is stored as a single character vector with all non-text stripped out. The encoding is converted from WINDOWS-1251 to UTF-8. Meanwhile, the information about the tags of "Laughter" and "Applause" and the number of times it was used are kept as a record for each speech. The *speechVec[i]* will be printed out in a nicely-formatted manner.

```
# Remove audience response tags (laughter & applause)
patLau <- "\\[.*?(Laughter|laughter).*?\\]"
patApp <- "\\[.*?(Applause|applause).*?\\]"
getlauNum <- function(x) {
    if (length(gregexpr(patLau, x, perl = TRUE)[[1]]) == 1 && gregexpr(patLau,
        x, perl = TRUE)[[1]] == -1) {
        return(0)
    } else {
        return(length(gregexpr(patLau, x, perl = TRUE)[[1]]))
    }
}  # get the number of [laughter]
getappNum <- function(x) {
    if (length(gregexpr(patApp, x, perl = TRUE)[[1]]) == 1 && gregexpr(patApp,
        x, perl = TRUE)[[1]] == -1) {
        return(0)
    } else {
        return(length(gregexpr(patApp, x, perl = TRUE)[[1]]))
    }
}  # get the number of [applause]
lauNum <- sapply(speechVec, getlauNum, USE.NAMES = FALSE)
appNum <- sapply(speechVec, getappNum, USE.NAMES = FALSE)
speechVec <- sapply(speechVec, function(x) {
    iconv(x, from = "WINDOWS-1251", to = "UTF-8", sub = " ")
})
speechVec <- sapply(speechVec, function(x) {
    x <- gsub(patLau, "", x, perl = TRUE)
    x <- gsub(patApp, "", x, perl = TRUE)
})
# remove all the non-verbal tags from the speech text
names(speechVec) <- NULL  # clean the speechVec name (otherwise, long and messy)
```

(d)

The collection of speeches is stored in a clean fashion of list elements. The ith entries of all the elements in *listSpeech* give information about the ith speech. This is easy later for plotting variables changes over time.

```
listSpeech <- list()
listSpeech$id <- fileid
listSpeech$name <- namePres
listSpeech$date <- as.integer(dateTalk)  # conversion for plotting, sorting
listSpeech$numLaughter <- lauNum
listSpeech$numApplause <- appNum
listSpeech$speech <- speechVec
```

(e) (f)

Words and sentences are extracted from each speech, and are stored as individual elements of a (rather long) character vector. Counts are also done on both words and sentences.

```
# Speech analysis Grab words from the speech by replacing all non-word
# char with space and split
getWords <- function(x) {
    x <- gsub("'", "", x, perl = TRUE)
    x <- gsub("\\W+", " ", x, perl = TRUE)
    xs <- unlist(strsplit(x, "[ ]+", perl = TRUE))
    return(xs[xs != ""])
}
# Grab sentences from the speech by spliting with ending chars like [.!?]
getSents <- function(x) {
    x <- gsub(" (Mr|Ms|Mrs|Dr|St|Sr|Jr)\\.", "\\1", x, perl = TRUE)
    x <- gsub("[\\.!\\?][ \t]+", "\n", x, perl = TRUE)
    xs <- unlist(strsplit(x, "\n", perl = TRUE))
    return(xs[xs != ""])
}
listSpeech$wc <- sapply(speechVec, function(x) {
    return(length(getWords(x)))
}, USE.NAMES = FALSE)  # word count
listSpeech$sc <- sapply(speechVec, function(x) {
    return(length(getSents(x)))
}, USE.NAMES = FALSE)  # sentence count
listSpeech$wMean <- sapply(speechVec, function(x) {
    return(mean(nchar(getWords(x))))
}, USE.NAMES = FALSE)  #avg word length
listSpeech$wSD <- sapply(speechVec, function(x) {
    return(sd(nchar(getWords(x))))
}, USE.NAMES = FALSE)  # word length sd
listSpeech$sMean <- sapply(speechVec, function(x) {
    return(mean(nchar(getSents(x))))
}, USE.NAMES = FALSE)  # avg sentence length
listSpeech$sSD <- sapply(speechVec, function(x) {
    return(sd(nchar(getSents(x))))
}, USE.NAMES = FALSE)  # sentence length sd
```

(g) (h)

We now start to extract some features of interest from the speeches to analyze how the speeches have changed over time. The result of all this is a list with each element containing the information about a speech: the speech as a single string, the vector of sentences, the vector of words and the additional quantification of variables about the speech from (g) as well as the non-verbal variables from (c).

1. Length in words and sentences *wc,sc*
2. Average and SD of word and sentence lengths *wMean,wSD,sMean,sSD*
3. Number of quotations in each speech, mean length (in words), and SD of length (in words) of the quotations in each speech *quoNum,quoMean,quoSD*
4. The most common meaningful words, where non-meaningful words are pre-defined *cmw*
5. Counts of the following words or word stems:

   I, we
   America,n
   democracy,tic

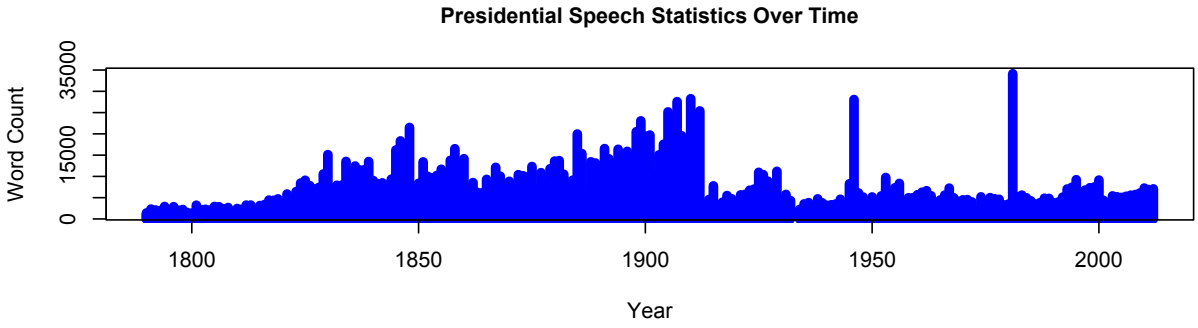
republic
Democrat,ic
Republican
free,dom
war
God – not including God bless
God Bless
Jesus, Christ, Christian
Woman – I think would be interesting

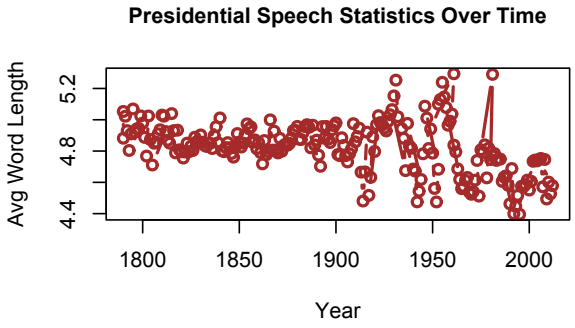
```
tmpList <- matrix(rep(0, 226 * 15), nrow = 226, ncol = 15)
## Speech list with element-wise analysis
speechList <- list()  #empty list
# Get the non-meaningful words file
system("wget -O 'cmw.txt' 'http://www.textfixer.com/resources/common-english-words.txt'")
commonWords <- readLines("cmw.txt", warn = FALSE)
commonWords <- unlist(strsplit(commonWords, ",", perl = TRUE))
# Element-wise analysis on each speech
for (i in 1:length(fileid)) {
    ss <- list()  #empty list element
    # Global attributes: from results before (reproduce for better storage)
    ss$id <- fileid[i]
    ss$name <- namePres[i]
    ss$date <- as.integer(dateTalk[i])
    ss$numLaughter <- lauNum[i]
    ss$numApplause <- appNum[i]
    ss$speech <- speechVec[i]
    # Indiv attributes: get from within each speech
    talkWords <- getWords(speechVec[i])
    talkSents <- getSents(speechVec[i])
    ss$words <- talkWords  # words vector
    ss$sents <- talkSents  # sentence vector
    ss$wc <- length(talkWords)  # word count
    ss$sc <- length(talkSents)  # sentence count
    ss$wMean <- mean(nchar(talkWords))  # avg word length
    ss$wSD <- sd(nchar(talkWords))  # word length sd
    ss$sMean <- mean(nchar(talkSents))  # avg sentence length
    ss$sSD <- sd(nchar(talkSents))  # sentence length sd; ss[14]
    patQuo <- "\"(.*?)\""  # quotation pattern
    quo <- talkSents[grep(patQuo, talkSents, perl = TRUE)]
    if (length(quo) != 0) {
        # get quotation attributes
        quo <- sapply(quo, function(x) {
            gsub(patQuo, "\\1", x)
        }, USE.NAMES = FALSE)
        ss$quoNum <- length(quo)
        ss$quoMean <- mean(nchar(quo))
        ss$quoSD <- sd(nchar(quo))
    } else {
        ss$quoNum <- 0
        ss$quoMean <- 0
        ss$quoSD <- 0
    }  #ss[17]
```

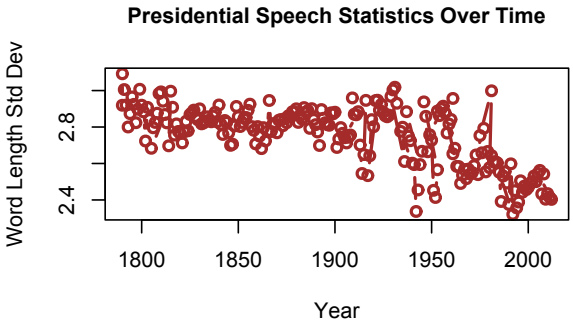
```
    cmw <- sort(table(talkWords), decreasing = TRUE)
    cmw <- cmw[which(!(names(cmw) %in% commonWords))]  # get meaningful words
    ss$cmw <- cmw[cmw >= 10]  #arbitrary cut-off for display
    ss$strIwe <- cmw[grep("^(I|[Ww]e)$", names(cmw))]  #string 'I|We'; ss[19]
    ss$strAme <- cmw[grep("[Aa]merica(|n)", names(cmw))]  #string 'America'
    ss$strDem <- cmw[grep("[Dd]emocra(cy|tic)", names(cmw))]  #string 'democratic'
    ss$strRep <- cmw[grep("[Rr]epublic(|n)", names(cmw))]  #string 'republican'
    ss$strFree <- cmw[grep("^[Ff]ree(|dom)$", names(cmw))]  #string 'free'
    ss$strWar <- cmw[grep("^[Ww]ar(|s)$", names(cmw))]  #string 'war'
    ss$strGod <- cmw[grep("^[Gg]od(|s)$", names(cmw))]  #string 'God'
    ss$strChr <- cmw[grep("(Jesus|Christ|Christian)", names(cmw))]  #string 'Chirst|Jesus'
    ss$strWoman <- cmw[grep("^[Ww]om[ae]n$", names(cmw))]  #Mystring 'Woman'; ss[27]
    ssGodBless <- talkSents[grep("[Gg]od [Bb]less", talkSents, perl = TRUE)]
    if (length(ssGodBless) != 0) {
        #string 'God Bless' from sentences
        ss$strGodBless <- sapply(ssGodBless, function(x) {
            return(length(gregexpr("[Gg]od [Bb]less", x, perl = TRUE)[[1]]))
        }, USE.NAMES = FALSE)
    } else {
        ss$strGodBless <- 0
    }
    # add to speechList and listSpeech
    speechList[[i]] <- ss
    tmpList[i, 1:3] <- unlist(ss[15:17])  #quo attr
    tmpList[i, 4:13] <- sapply(ss[19:28], sum)  #cmw related attr
}
# prepare for plotting, store as listSpeech elements
listSpeech$quoNum <- tmpList[, 1]
listSpeech$quoMean <- tmpList[, 2]
listSpeech$quoSD <- tmpList[, 3]
listSpeech$strIwe <- tmpList[, 4]
listSpeech$strAme <- tmpList[, 5]
listSpeech$strDem <- tmpList[, 6]
listSpeech$strRep <- tmpList[, 7]
listSpeech$strFree <- tmpList[, 8]
listSpeech$strWar <- tmpList[, 9]
listSpeech$strGod <- tmpList[, 10]
listSpeech$strChr <- tmpList[, 11]
listSpeech$strWoman <- tmpList[, 12]
listSpeech$strGodBless <- tmpList[, 13]
```

(i) (j)

Some basic plots that show how the variables have changed over time are given below.

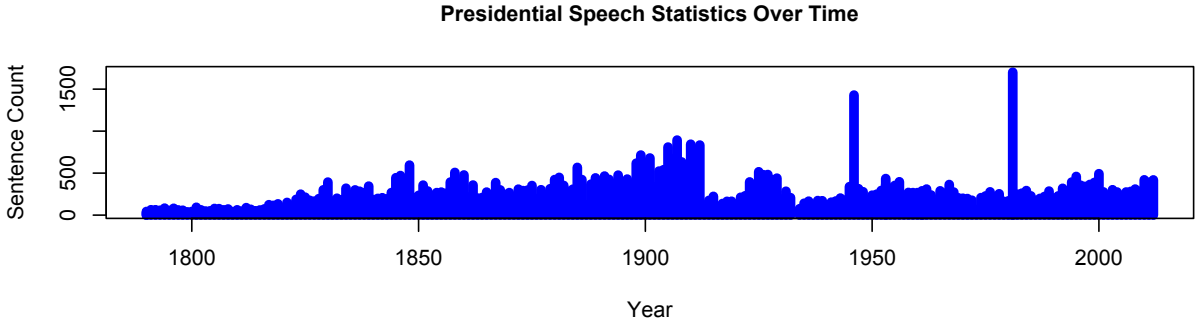


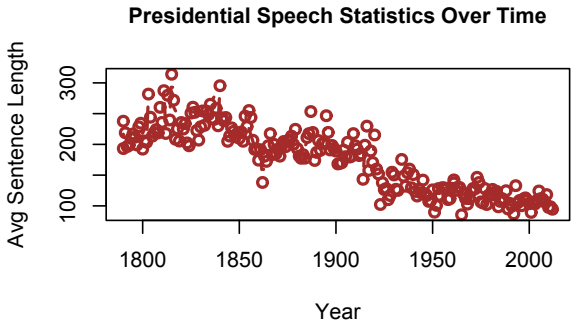


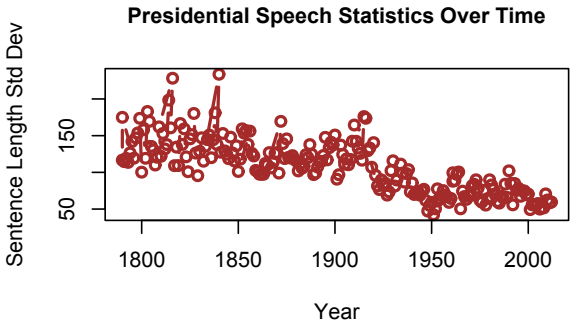


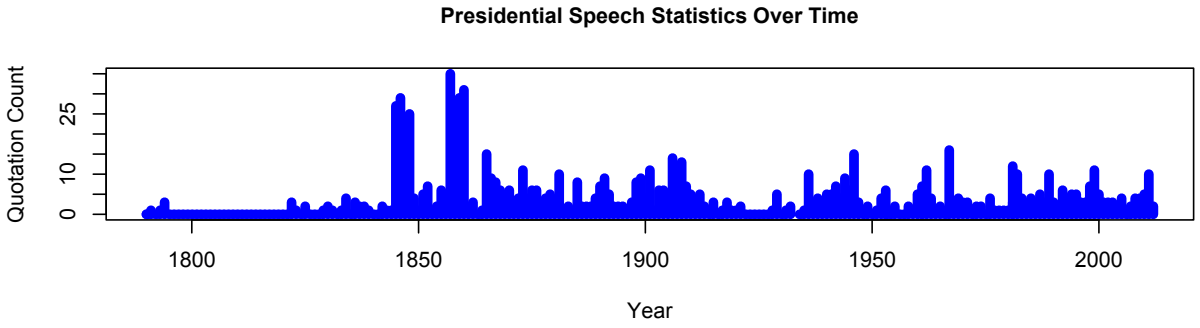
We can see that the word counts of the speeches are higher prior to 1920s than the modern ages. So does the avg word length and word length sd. The presidents nowadays seem to like brevity.

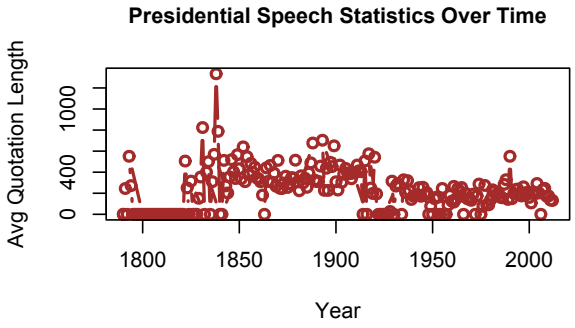
Similar plots for sentence counts and quotation counts are given below. The similar trends stand.

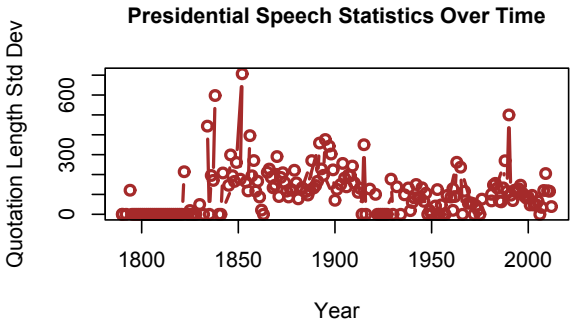




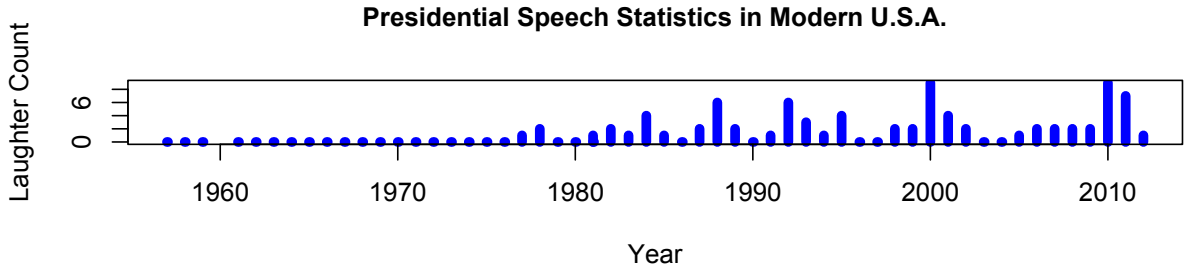


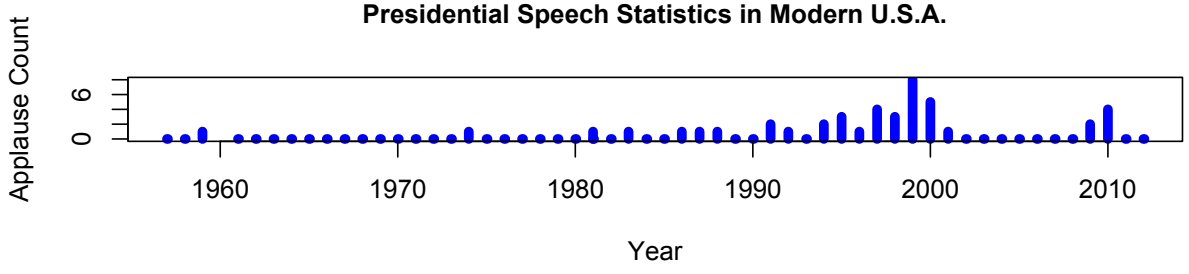




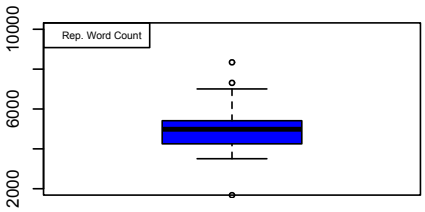


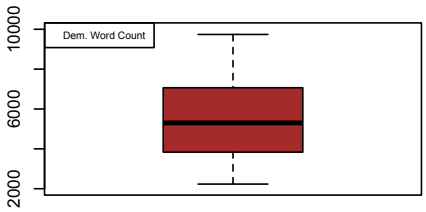
In recent years, with full records of non-verbal attributes, we can see that [Laughter] and [Applause] counts have some positive correlation. The speeches receive more responses from the audience around 2000 than other years.

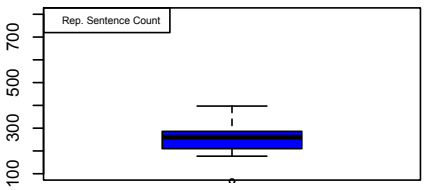


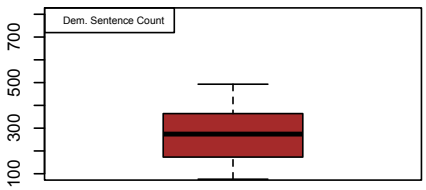


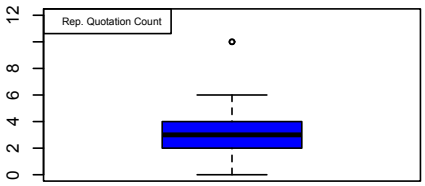
And for presidents since Franklin Roosevelt in 1932, comparison between Republican presidents (Eisenhower, Nixon, Ford, Reagan, G. Bush, G.W. Bush) and Democratic presidents (Roosevelt, Truman, Kennedy, Johnson, Carter, Clinton, Obama) are also given below in the box-plot comparison.

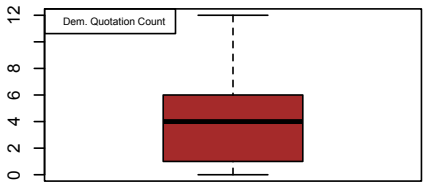

Rep. Word Count
2000
6000
10000

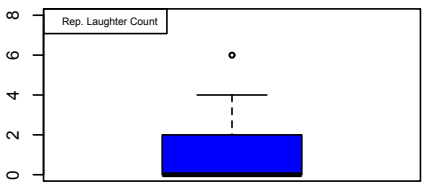

Dem. Word Count
2000
6000
10000

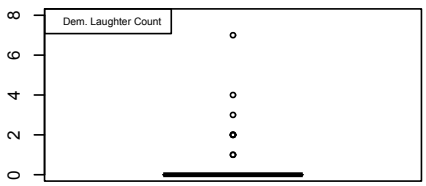

Rep. Sentence Count
100
300
500
700

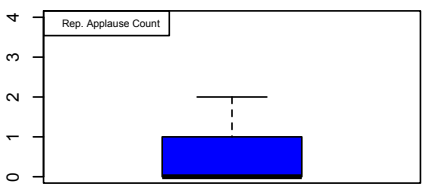

Dem. Sentence Count
100
300
500
700

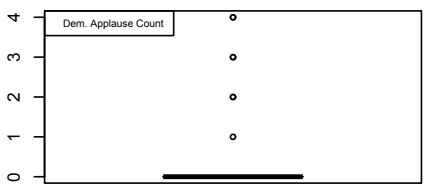

Rep. Quotation Count
0
2
4
6
8
12

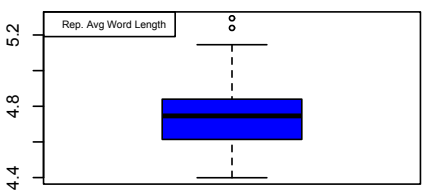

Dem. Quotation Count
0
2
4
6
8
12

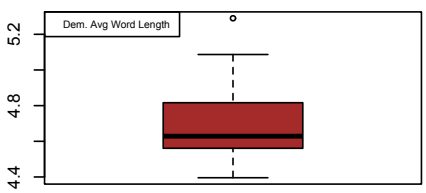

Rep. Laughter Count
0
2
4
6
8

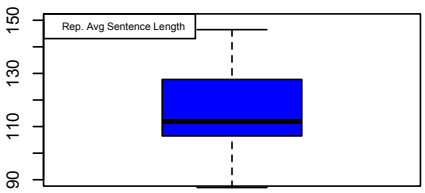

Dem. Laughter Count
0
2
4
6
8

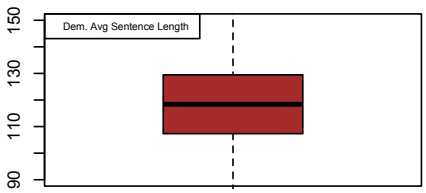

Rep. Applause Count
0
1
2
3
4

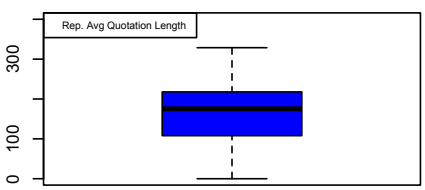

Dem. Applause Count
0
1
2
3
4

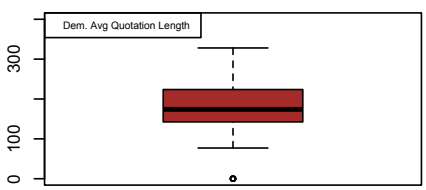

Rep. Avg Word Length
4.4
4.8
5.2


Dem. Avg Word Length
4.4
4.8
5.2

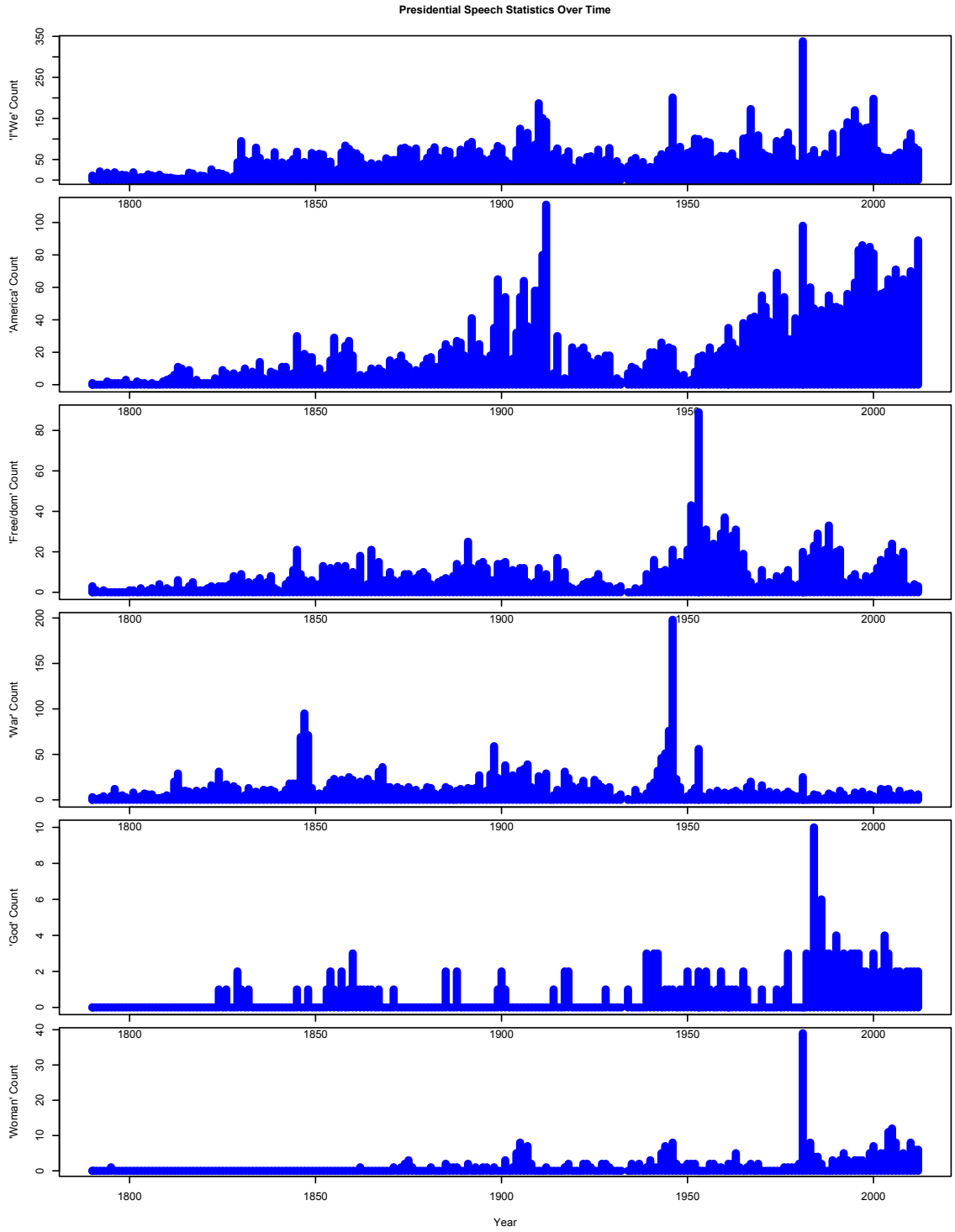

Rep. Avg Sentence Length
90
110
130
150


Dem. Avg Sentence Length
90
110
130
150

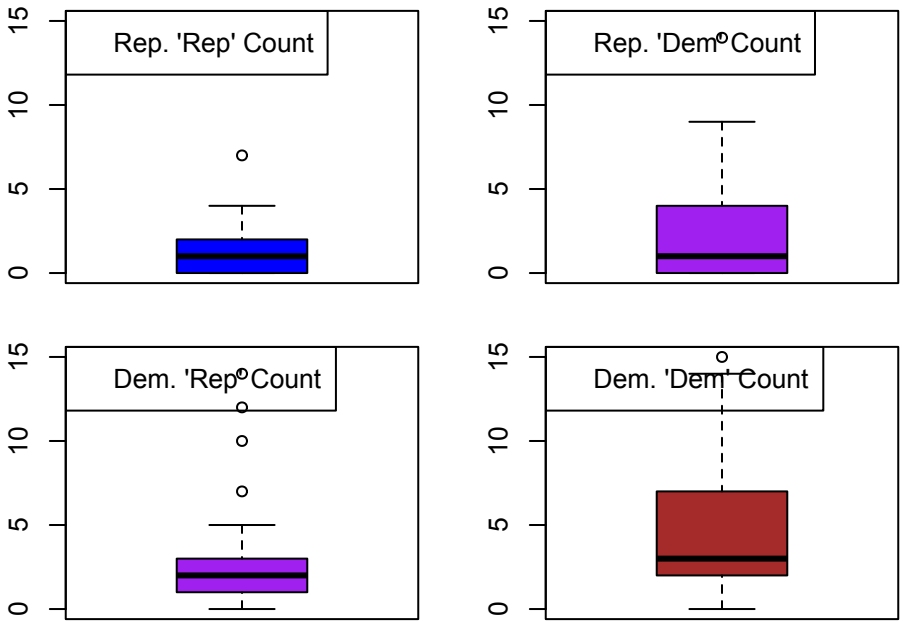

Rep. Avg Quotation Length
0
100
300


Dem. Avg Quotation Length
0
100
300

Some additional research with plotting that illustrates how the speeches have changed over time are shown below. The different keywords have evolved over time, with peak in WAR during wartime and GOD-rich in recent years.



The comparison between Reps and Dems could be more interesting with the strings REPUBLICAN and DEMOCRATIC. It seems that the Dems are more focused on their persuit while Reps are impartial to both words.



Some additional variables that quantify speech in interesting ways are produced. The number of digits shown within the speech is a good indicator whether the president speaks more quantatively, or vice versa, more qualitatively.