Text Mining Project

Nick McCulloch

2022-11-02

Contents

Introduction

The process involves use of the twitter APIv2. There are many request formats that can be used including searches by tweet, author, list or other feature. Currently the full archive tweet search is used allowing tweets to be pulled from the full twitter archive and filtered on the basis of various parameters. The code to do this is organized as follows.

1.) Defining Full-Archive Search Function 2.) Setting up the parameters for the search 3.) The function call itself 4.) Data clean up and processing 4.) Saving the data to .csv 5.) Code to control and shape the function's for loop 6.) Full, iterative version of the function to be run 7.) Data quality checks and fixes 8.) Pre-processing 9.) Analysis and Data Visualization

Set Up

install packages

```
install.packages("rmarkdown")
install.packages("knitr")
install.packages("httr")
install.packages("jsonlite")
install.packages("magrittr")
install.packages("dplyr")
install.packages("ggplot2")
install.packages("tinytex")
install.packages("readr")
#install.packages("RTwitterV2")
#install.packages("ggpubr")###
install.packages("tidyverse")
install.packages("Matrix")
install.packages("slam")
install.packages("bench")
install.packages("rJava")
install.packages("qdap")
install.packages("tm")
# Important note: TM will mask "content(")" from httr which is used elsewhere
in this markdown.
install.packages("tidyr")
install.packages("quanteda")
install.packages("quanteda.textstats")
install.packages("quanteda.textplots")
install.packages("quanteda.textmodels")
install.packages("ggthemes")
install.packages("scales")
install.packages("tidytext")
install.packages("lubridate")
install.packages("stringi")
install.packages("stringr")

library(tinytex)
install_tinytex()
```

load packages

```
library(rmarkdown)
library(knitr)
library(httr)
library(jsonlite)
library(magrittr)
library(dplyr)
library(ggplot2)
library(tinytex)
library(readr)
#library(RTwitterV2) #optional
```

```r
#library(ggpubr)  #optional
library(tidyverse)
library(Matrix)
library(slam)
library(bench)
library(rJava)
library(qdap)
library(tm)
# Important note: TM will mask "content()" from httr which is used elsewhere
in this markdown.
library(tidyr)
library(quanteda)
library(quanteda.textstats)
library(quanteda.textplots)
library(quanteda.textmodels)
library(ggthemes)
library(scales)
library(tidytext)
library(lubridate)
library(stringi)
library(stringr)
```

Data Collection

pull tweet function-full archive

Function to pull tweets from the full archive. Organized around tweets but includes expansion fields with author and place info. Rate limit of 1 request per second, 100 tweets per request, and 300 total requests per quarter hour. The function generates an HTML request to the Twitter developer API and the API returns info in a JSON format.

```
#this version of the pull function excludes the place field which is preferab
le when pulling tweets from elected officials.  The iterative function below
is set up to work without place field.
PullTweetsAll<- function(bearer_token,tweet_fields,author_fields,acct_list,st
artdt,stopdt,max) {
  headers <- c(`Authorization` = sprintf('Bearer %s', bearer_token))
  fromseries<- rep(list("OR"),length(acct_list)-1)
  cleanquery <- paste(acct_list,fromseries)
  cleanquery[length(cleanquery)]=acct_list[length(acct_list)]
  cleanquery <- paste0("from:",cleanquery,collapse = " ")
  #stop <- as.Date(start) + 1
  startT <- startdt
  stopT <- stopdt
  params = list(
    `tweet.fields` = tweet_fields,
    `expansions` = 'author_id',
    `user.fields` = author_fields,
    `query` = cleanquery,
    `start_time` = startT,
    `end_time` = stopT,
    `max_results` = max)
  resp <- httr::GET(url = 'https://api.twitter.com/2/tweets/search/all', httr
::add_headers(.headers=headers), query = params)
  return(resp)
}
```

formula/query parameters

These are settings and choices to be used in the function, saved as objects here for convenience.

```
#categories of information that will be returned about the tweet ex: date cre
ated
tweet_field_list <- list('author_id,created_at,id,in_reply_to_user_id,possibl
y_sensitive,public_metrics,referenced_tweets,text')

author_field_list <- list('created_at,description,id,name,public_metrics,url,
username')

#place_field_list <- list('contained_within,country,country_code,full_name,ge
o,id,name,place_type')
```

set up - creating acct matrix

```
#list of 'authors' that will be checked, format = twitter handles, the below
example uses Major party presidential candidates but these can be substituted
for others as is shown later.

#acct_list<-list("POTUS","SenJohnMcCain","BarackObama","POTUS44","MittRomney"
,"JohnMcCain","realDonaldTrump","POTUS45")

#list of consolidated twitter accts from joint govt watch dog group github
#online at: https://github.com/unitedstates/congress-legislators
#main page at:  https://theunitedstates.io/
#data was downloaded, parsed, and reduced to twitter accts only.
#the list doesn't appear exhaustive, only includes current politicians, and o
nly includes official twitter accts (not campaign accts)

#twitter_accts_leg<-read.csv("legislator_social.csv")
twitter_accts_leg<-read.csv("twitter_congress_list.csv")
twitter_accts_leg<-twitter_accts_leg$username
acct_list<-twitter_accts_leg

#API has limits on length of requests so only a subset of accounts can be inc
luded in each request.
#accts are converted here to a matrix which can be submitted sequentially or
by random sample

#method creating matrix to move through sequentially.
#with the other query parameters 45-50 is about the maximum number of handles
that can be included while remaining under the API request character limit.
size_acct_list<-length(acct_list)
print(size_acct_list)
acct_list<-sample(acct_list,size=size_acct_list,replace = FALSE)

acct_list <-matrix(data = acct_list,ncol = 40)

#the number of rows in the new matrix is the number of iterations necessary t
o use all accts.
nrow(acct_list)

# get users by Twitter List such as @TwitterGov "US House of Representatives"
or "US Senate"
#curl "https://api.twitter.com/2/lists/63915645/members?user.fields=id,name,u
sername" -H "Authorization: Bearer $BEARER_TOKEN"

#measuring length of request
# the API will only accept requests of a certain length (a little over 2000 c
haracters).  The function below prints the length of the last request run
## Can only be run after one request has been made.
#print(length(charToRaw(results[["request"]][["url"]])))
```

creating randomized dates

Twitter's API doesn't have a setting to randomly sample tweets and the size of each requested is limited to 100 tweets each. To get a representative sample from the date range in question the follwoing generates a list of dates from the start date to the current date.

```
#set.seed(64)

### It's important to change the size in the date var and hoursvar code to ac
count for the number of iterations you want to make.  Bear in mind that the f
unction (as initially designed) is a nested for loop that iterates through al
l the dates AND the acct list.

#randomizes the year, month, and day of the target range.
ndate <- Sys.Date() - as.Date("2007-01-01")

#ndate <- as.Date("2009-01-01") - as.Date("2007-01-01")
datevar <- as.Date("2007-01-01") + (sample.int(n = ndate, size = 30))

#randomizes the time of day by hour.
nhours <- c('01','02','03','04','05','06','07','08','09','10','11','12','13',
'14','15','16','17','18','19','20','21','22','23','24')
hoursvar <-sample(nhours,30, replace = TRUE)

#pastes randomized dates and times into acceptable format
dates<-paste(datevar,"T",hoursvar,":00:00.000Z",sep= "")
```

function call

Actually calling the function. Can make changes here or in the referenced objects to make different requests.

```
results<-PullTweetsAll(bearer_token = my_bearer_token,
                       tweet_fields = tweet_field_list,
                       author_fields = author_field_list,
                       acct_list = acct_list[1,],
                       startdt = "2007-01-01T00:00:00.000Z",
                       stopdt = "2008-01-01T00:00:00.000Z",
                       max = '100')
```

Clean Up

view response info

This shows the response generated by the API. This is not a good place to view the tweet information itself but rather includes details about the request, status, rate limit, etc. Some pertinent information is extracted here.

```
#View(results)

req_url  <- results$request$url
req_status <- results$status_code
req_date <- results$headers$date
req_length <- results$headers$`content-length`
req_id <- results$headers$`x-transaction-id`
rate_limit <- results$headers$`x-rate-limit-limit`
limit_reset <- results$headers$`x-rate-limit-reset`
limit_remaining <- results$headers$`x-rate-limit-remaining`

request_info<- data.frame(req_url, req_status,req_date,req_length,req_id,rate
_limit,limit_reset,limit_remaining)
```

parsing response

Converts response from JSON to something more interpretable. Should return a list of lists: 1) data-where the tweets and the 'tweet field' data is stored. 2) includes- which contains the user and location data, as well as any other expansions. 3) meta-which includes meta data about the request such as tweet count, oldest/newest tweet-id etc.

```
parsed_df <-
  httr::content(
    results,
    as = 'parsed',
    type = 'application/json',
    simplifyDataFrame = TRUE
  )

#Sanity Check
#should contain three lists; data, includes, meta; and ''includes' should con
tain both users and places, if the place field expansion was included.
#View(parsed_df)
```

extracting and cleaning tweet content

```
#extracting the tweet content from the parsed json data.
tweet_df <- parsed_df$data

#renaming the tweet data and place data expansion.
names(tweet_df)[names(tweet_df) == "id"] <- "tweet_id"
#names(tweet_df)[names(tweet_df) == "geo"] <- "place_id"
```

```r
#creating object with unneeded column names to drop.
vars2drop <- names(tweet_df) %in% c('context_annotations','edit_controls','at
tachments','edit_history_tweet_ids', 'entities')

tweet_df <- tweet_df[!vars2drop]

#flattening the remaining columns into a flat list.
tweet_df <- flatten(tweet_df, recursive = TRUE)

#renaming flattened columns with friendly and short names.
names(tweet_df)[names(tweet_df) == "public_metrics.reply_count"] <- "replies"
names(tweet_df)[names(tweet_df) == "public_metrics.like_count"] <- "likes"
names(tweet_df)[names(tweet_df) == "public_metrics.retweet_count"] <- "retwee
ts"
names(tweet_df)[names(tweet_df) == "public_metrics.quote_count"] <- "quote_co
unt"
names(tweet_df)[names(tweet_df) == "created_at"] <- "date_of_tweet"

#sanity check
#should produce a dataframe where you can read the individual tweets.
#names(tweet_df)
#View(tweet_df)
```

extracting author expansion field content

```r
### creating a data frame for the user data expansion
user_df <- parsed_df$includes$users

names(user_df)[names(user_df) == "id"] <- "author_id"

user_df <- flatten(user_df, recursive = TRUE)

vars2drop <- names(user_df) %in% c("entities.description.mentions","entities.
url.urls","public_metrics.listed_count","location","url")

user_df <- user_df[!vars2drop]

names(user_df)[names(user_df) == "public_metrics.followers_count"] <- "follow
ers"
names(user_df)[names(user_df) == "public_metrics.following_count"] <- "follow
ing"
names(user_df)[names(user_df) == "public_metrics.tweet_count"] <- "num_tweets
"
names(user_df)[names(user_df) == "created_at"] <- "account_created"
names(user_df)[names(user_df) == "description"] <- "account_description"

#sanity check
#View(user_df)
```

extracting place expansion field content

The place field is the most likely field to return a null value. Any query that includes it must build in try/except statements and checks for null values.

```
### creating a data frame for the place data expansion
parsed_dftemp <- parsed_df

geo<-c(parsed_dftemp$includes$places$geo$bbox,recursive = TRUE)

bbox_names <-c('bbox.longitude1','bbox.latitude1','bbox.longitude2','bbox.lat
itude2')

if(is.null(geo)) {
} else {names(geo <- bbox_names)}

parsed_dftemp$includes$places[['geo']] = NULL

places_df <- as.data.frame(c(parsed_dftemp$includes$places,geo))

names(places_df)[names(places_df) == "id"] <- "place_id"
names(places_df)[names(places_df) == "full_name"] <- "place_name"

vars2drop <- names(places_df) %in% c("name","country")

places_df <- places_df[!vars2drop]

#View(places_df)
```

Merging Tweet, place, and user data frames

Run this code if desired. Consider dropping unneeded columns first for better interpretability.

```
merged_df <- merge(x = tweet_df,y = user_df, by = "author_id", all.x = TRUE)

#if(is.null(geo)) {
#} else {merged_df <- merge(x = merged_df,y = places_df, by = "place_id", all
.x = TRUE)}

### and now adding request info
merged_df <- cbind(merged_df,request_info)

#names(merged_df)

complete_df <- merged_df

#rm(geo)
```

saving results to file

```
if(is.null(vox_pop)){
  vox_pop <- complete_df
} else {
  vox_pop <- dplyr::bind_rows(vox_pop,complete_df)
}

#removing duplicate tweets
#opt 1
vox_pop[!duplicated(vox_pop$tweet_id),]

#temp<-matrix(c(1,1,1,1,2,2,2,2,1,1,1,1,1,2,3,4,1,2,3,4),ncol=4,byrow = TRUE)
#View(temp)
#temp[!duplicated(temp),]
#temp2<-temp[!duplicated(temp),]
#temp2
#temp

vox_pop<-vox_pop[!duplicated(vox_pop$tweet_id),]


### strongly consider changing to a data table approach.  Also defining colum
n/variable types and naming them.


# Please Note---append = TRUE is needed to merge new results to existing data
frame but it will not retain column headings if no previous iteration has occ
urred.
readr::write_csv(x = vox_pop, "vox_pop.csv", append = TRUE, progress = show_p
rogress(),eol = "\n",
)

print(req_status)
print(results$headers$`x-rate-limit-remaining`)
print(results$headers$`x-response-time`)
```

rate limit function

gets and returns rate limit, number of requests remaining, and reset point. Only provides info, must be incorporated elsewhere to actually limit functions. Note that the standard rate limit for the full archive search is 300/app, 180/user, and 1/sec. More info online at: https://developer.twitter.com/en/docs/twitter-api/rate-limits.

```
complete_df$limit_remaining[1]
complete_df$limit_reset[1]

### function for getting reset information and rate limit remaining.
rate_limit <- function() {
  remaining <- results$headers$`x-rate-limit-remaining`[1]
```

```
  reset <- results$limit_reset[1]
  rate_result <- c(remaining,reset)
  return(rate_result)
}

rate_limit()
```

Automated Loop

Although the code chunks above can be run to make individual requests, the following chunks are used to make requests iterative.

```
### Set Up

#setwd('C:/Users/nickb/Documents/Course Documents/CSV files/text_mining_proje
ct/')

#library(knitr)
#library(httr)
#library(jsonlite)
#library(magrittr)
#library(dplyr)
#library(readr)

PullTweetsAll<- function(bearer_token,tweet_fields,author_fields,acct_list,st
artdt,stopdt,max) {
  headers <- c(`Authorization` = sprintf('Bearer %s', bearer_token))
  fromseries<- rep(list("OR"),length(acct_list)-1)
  cleanquery <- paste(acct_list,fromseries)
  cleanquery[length(cleanquery)]=acct_list[length(acct_list)]
  cleanquery <- paste0("from:",cleanquery,collapse = " ")
  #stop <- as.Date(start) + 1
  startT <- startdt
  stopT <- stopdt
  params = list(
    `tweet.fields` = tweet_fields,
    `expansions` = 'author_id',
    `user.fields` = author_fields,
    `query` = cleanquery,
    `start_time` = startT,
    `end_time` = stopT,
    `max_results` = max)
  resp <- httr::GET(url = 'https://api.twitter.com/2/tweets/search/all', httr
::add_headers(.headers=headers), query = params)
  return(resp)
}

tweet_field_list <- list('author_id,created_at,id,in_reply_to_user_id,possibl
y_sensitive,public_metrics,referenced_tweets,text')

author_field_list <- list('created_at,description,id,name,public_metrics,url,
username')

###

startlength<-nrow(vox_pop)
```

```r
### it begins...

for (q in 1:100) {
  #setting up accts
  # must read in or create initial list of usernames.

  TweetsPerUser<-as.data.frame(table(vox_pop$username))

  TweetsPerUser<-TweetsPerUser[TweetsPerUser$Freq < 1000,]

  TweetsPerUser<- TweetsPerUser[order(TweetsPerUser$Freq),]

  rownames(TweetsPerUser)<-1:nrow(TweetsPerUser)

  acct_list<-as.vector(TweetsPerUser$Var1)

  #View(TweetsPerUser)
  #TweetsPerUser<-TweetsPerUser[1:500]

  #acct_list<-acct_list[!duplicated(acct_list)]
  #acct_list<-sample(acct_list_vect,size=1195,replace = FALSE)

  #acct_list<-acct_list[!(acct_list %in% c(TweetsPerUser))]

  acct_list <-matrix(data = acct_list,ncol = 20,byrow=TRUE)

  #dates
  ndate <- Sys.Date() - as.Date("2007-01-01")

  #ndate <- as.Date("2009-01-01") - as.Date("2007-01-01")

  #change back to 2007
  datevar <- as.Date("2007-01-01") + (sample.int(n = ndate, size = 100))

  #randomizes the time of day by hour.
  nhours <- c('01','02','03','04','05','06','07','08','09','10','11','12','13
','14','15','16','17','18','19','20','21','22','23','24')
  hoursvar <-sample(nhours,100, replace = TRUE)

  #pastes randomized dates and times into acceptable format
  dates<-paste(datevar,"T",hoursvar,":00:00.000Z",sep= "")


  ###I recommend copying this to a new r script and running it from there.  I
have heard anecdotally that Rmd is slower, though I still need to confirm thi
s.
```

```r
  # measuring how long full loop takes
  # place outside both loops
  startloop <- Sys.time()

  for(m in 1:50){
    for(i in 1:5){
      #timer for the loop
      date_time<-Sys.time()

      #pull tweet function
      results<-PullTweetsAll(bearer_token = my_bearer_token,
                             tweet_fields = tweet_field_list,
                             author_fields = author_field_list,
                             acct_list = acct_list[m,],
                             startdt = "2007-01-01T00:00:00.000Z",
                             stopdt = paste(dates[i]),
                             max = '100')

      #Sys.sleep added because of suspicions that the twitter API was being o
verwhelmed by the number of requests being made, and also to allow user to mo
nitor progress more closely, given the volume of data in question.
      Sys.sleep(time = 1)

      #break added if the status code generated by the request is anything ot
her than "200" which indicates a successful request.
      #if(results$status_code != 200) {
      #  break
      #}

      if(results$status_code != 200) {
        print(results$status_code)
        flush.console()
        #Sys.sleep(1)
        if(results$status_code == 400) {
          bad_requests<-c(bad_requests,results[["request"]][["url"]])
          print(results$status_code)
          next
        } else if (results$status_code == 429) {
          Sys.sleep(1)
          Sys.sleep(1)
          Sys.sleep(1)
        } else {
          print(results$status_code)
          print("broke at line 70")
          break
        }
      }
```

```
      #break added to stop when the rate limit has been reached.
      if(as.numeric(results$headers$`x-rate-limit-remaining`) <5) {
        date_time2<-as.numeric(Sys.time())
        print("waiting")
        repeat {wait<-as.numeric(Sys.time())- date_time2
        if(wait>700) {break}}}

      while((as.numeric(Sys.time()) - as.numeric(date_time))<1){} #dummy whil
e loop

      #start cleaning
      req_url  <- results$request$url
      req_status <- results$status_code
      req_date <- results$headers$date
      req_length <- results$headers$`content-length`
      req_id <- results$headers$`x-transaction-id`
      rate_limit <- results$headers$`x-rate-limit-limit`
      limit_reset <- results$headers$`x-rate-limit-reset`
      limit_remaining <- results$headers$`x-rate-limit-remaining`

      request_info<- data.frame(req_url, req_status,req_date,req_length,req_i
d,rate_limit,limit_reset,limit_remaining)

      #more cleaning
      parsed_df <-
        httr::content(
          results,
          as = 'parsed',
          type = 'application/json',
          simplifyDataFrame = TRUE
        )

      #checking for null
      if(is.null(parsed_df$data)) {
        next
      }

      #extracting the tweet content from the parsed json data.
      tweet_df <- parsed_df$data
      #renaming the tweet data and place data expansion.
      names(tweet_df)[names(tweet_df) == "id"] <- "tweet_id"

      #creating object with unneeded column names to drop.
      vars2drop <- names(tweet_df) %in% c('context_annotations','edit_control
s','attachments','edit_history_tweet_ids', 'entities')

      tweet_df <- tweet_df[!vars2drop]

      #flattening the remaining columns into a flat list.
```

```
      tweet_df <- jsonlite::flatten(tweet_df, recursive = TRUE)

      #renaming flattened columns with friendly and short names.
      names(tweet_df)[names(tweet_df) == "public_metrics.reply_count"] <- "re
plies"
      names(tweet_df)[names(tweet_df) == "public_metrics.like_count"] <- "lik
es"
      names(tweet_df)[names(tweet_df) == "public_metrics.retweet_count"] <- "
retweets"
      names(tweet_df)[names(tweet_df) == "public_metrics.quote_count"] <- "qu
ote_count"
      names(tweet_df)[names(tweet_df) == "created_at"] <- "date_of_tweet"

      ##extracting expansion fields

      #creating a data frame for the user data expansion
      user_df <- parsed_df$includes$users

      names(user_df)[names(user_df) == "id"] <- "author_id"

      user_df <- flatten(user_df, recursive = TRUE)

      names(user_df)[names(user_df) == "public_metrics.followers_count"] <- "
followers"
      names(user_df)[names(user_df) == "public_metrics.following_count"] <- "
following"
      names(user_df)[names(user_df) == "public_metrics.tweet_count"] <- "num_
tweets"
      names(user_df)[names(user_df) == "created_at"] <- "account_created"
      names(user_df)[names(user_df) == "description"] <- "account_description
"

      vars2drop <- names(user_df) %in% c("entities.description.mentions","ent
ities.url.urls","public_metrics.listed_count","location","url","protected","a
ccount_descpription","profile_image_url","pinned_tweet_id")

      user_df <- user_df[!vars2drop]

      merged_df <- tweet_df

      #creating a data frame for the place data expansion
      #parsed_dftemp <- parsed_df

      ## merging data frames

      merged_df <- merge(x = tweet_df,y = user_df, by = "author_id", all.x =
TRUE)

      complete_df<-merged_df
```

```
      vars2drop<- names(complete_df) %in% c("referenced_tweets","in_reply_to_
user_id")

      complete_df<-complete_df[!vars2drop]

      #and now adding request info
      #merged_df <- cbind(merged_df,request_info)

      #names(merged_df3)


      #if(is.null(author_df)){
      #  author_df <- user_df
      #} else {
      #  vox_pop <- dplyr::bind_rows(author_df,user_df)
      #}

      #add code to confirm the colnames match before proceeding.
      wanted_cols_complete_df<-c("author_id","possibly_sensitive","date_of_tw
eet","text","tweet_id","retweets",
                                 "replies","likes","quote_count","username",
                                 "name","account_created","account_descriptio
n","followers","following","num_tweets")

      new_order = sort(colnames(complete_df),decreasing=TRUE)
      complete_df <- complete_df[, new_order]
      colnames(complete_df)

      #View(head(complete_df))
      #wanted_cols_complete_df
      #wanted_cols_complete_df<-matrix(wanted_cols_complete_df,ncol = 16,)

      #colnames(wanted_cols_complete_df)

      #missing referenced tweets
      #in_reply_to_user_id

      wanted_cols_complete_df <- sort(wanted_cols_complete_df,decreasing=TRUE
)
      #new_order
      wanted_cols_complete_df

      coltemp<-c(colnames(complete_df))
      coltemp1<-c(wanted_cols_complete_df)

      if(all(coltemp == coltemp1)) {
      } else {
        print("complte_df doesn't match cols in wanted_cols_complete_df")
```

```
        print(paste("coltemp",coltemp))
        print(paste("coltemp1",coltemp1))

        flush.console()
        break
      }


      test1<-colnames(complete_df)[14] == "author_id"
      test2<-colnames(complete_df)[3] == "text"
      test3<-colnames(complete_df)[2] == "tweet_id"
      test4<-colnames(complete_df)[1] == "username"

      col_order_tests<-c(test1,test2,test3,test4)

      if(all(col_order_tests)) {
      } else {
        print("col order test failed")
        flush.console()
        break
      }

      if(length(coltemp==coltemp1)) {
      } else {
        print("col number test failed")
        flush.console()
        break
      }

      if(!exists("vox_pop")) {
        print("got to line 214, 'exists(vox_pop) returned 'FALSE'")
      }

      if(exists("vox_pop")) {
        vox_pop <- dplyr::bind_rows(vox_pop,complete_df)
      } else {vox_pop<-complete_df}

      #removing duplicate tweets
      #opt 1
      vox_pop<-vox_pop[!duplicated(vox_pop$tweet_id),]

      #author_df[!duplicated(author_df$author_id,)]

      print(paste("Beep Boop... current progress report:"))
      print(paste("iteration  ",q,m,i,sep = "."))
      print(paste("req status ",req_status))
      print(paste("rate limit remaining ",results$headers$`x-rate-limit-remai
ning`))
      print(paste("response time ",results$headers$`x-response-time`))
```

```
      print(paste("tweets collected = ",length(vox_pop$tweet_id)-startlength)
)
      #print(paste("unique users = ",length(unique(vox_pop$username))))

      flush.console()

      ## optional break if code is taking too long to run.
      #if(results$headers$`x-response-time`> 60) {
      #  break
      #}

      #Sys.sleep(1)

      while((as.numeric(Sys.time()) - as.numeric(date_time))<1){} #dummy whil
e loop

      if(as.numeric(Sys.time()) - as.numeric(date_time) < 1) {
        repeat {wait2<-as.numeric(Sys.time())-date_time
        if(wait2>1) {break}
        }}
    }

    #break added if the status code generated by the request is anything othe
r than "200" which indicates a successful request.
    if(results$status_code != 200) {
      next
    } else if(results$status_code == 429) {
      break} else {next}

    #break added to stop when the rate limit has been reached.
    #if(results$headers$`x-rate-limit-remaining`<5) {
    #  break
    #}
  }

  #break added if the status code generated by the request is anything other
than "200" which indicates a successful request.
  if(results$status_code != 200) {
    next
  } else if(results$status_code == 429) {
    break
  } else {next}


  #break added to stop when the rate limit has been reached.
  #if(results$headers$`x-rate-limit-remaining`<5) {
  #  break
  #}
```

```
  targetfile<-filenames[q]

  ### strongly consider changing to a data table approach.  Also defining col
umn/variable types and naming them.
  cur_length <- nrow(vox_pop)

  if(cur_length-startlength>10000){
    if(file.exists("C:/Users/nickb/Documents/Course Documents/CSV files/text_
mining_project/vox_pop.csv")){
      readr::write_csv(x = vox_pop, file = paste(targetfile), append = TRUE,e
ol = "\n")
    } else {readr::write_csv(x = vox_pop, file = paste(targetfile), append =
FALSE, eol = "\n", col_names = TRUE)
    }
  }
  #measuring end of code
  stoploop <-Sys.time()

  runtime <- stoploop - startloop
  print(paste("runtime was ",runtime))
}
```

measuring code performance

```
### to be placed at beginning

# measuring how long full loop takes
# place outside both loops
startloop <- sys.time()

#place inside m loop

### to be placed at the end

#place outside of both loops, measures full function time.
stoploop <-sys.time()

#progress monitor - to be placed inside loop
print(req_status)
print(results$headers$`x-rate-limit-remaining`)
print(results$headers$`x-response-time`)

#measuring length of request
print(length(charToRaw(results[["request"]][["url"]])))
```

Analysis and Visualization

reading in data

```
#read in vox pop

#the code below was used to write it out in the first place
#colnames(vox_pop)->vox_names
#readr::write_csv(vox_pop,"vox_pop.csv",col_names=TRUE,quote="needed",escape
= "double")
#write.csv(vox_pop,"vox_pop_write.csv.csv")
#readr::write_csv2(vox_pop,"vox_pop_write_csv2")

readr::read_csv("C:/Users/nickb/Documents/Course Documents/CSV files/text_min
ing_project/Final Docs/vox_pop.csv",col_names = TRUE,show_col_types = FALSE)-
>vox_pop

#checking column names to ensure integrity
##names(temp)

#checking full document
##identical(vox_pop,temp)

##all.equal(vox_pop,temp)

#vox_pop <- readr::read_csv("vox_pop.csv")

names(vox_pop)
```

```
##  [1] "author_id"           "username"            "tweet_id"
##  [4] "text"                "retweets"            "replies"
##  [7] "quote_count"         "possibly_sensitive"  "num_tweets"
## [10] "name"                "likes"               "following"
## [13] "followers"           "date_of_tweet"       "account_description"
## [16] "account_created"     "year"                "month"
## [19] "day"                 "party"
```

```
#head(temp)
length(vox_pop$tweet_id)
```

```
## [1] 343978
```

```
#length(temp$tweet_id)
```

data checks

Performing various tests on the data to check for issues.

```
#Does the data have column names

names(vox_pop)
```

```
# number of unique accounts collected,
length(unique(vox_pop$author_id))

#the code below counts the number of tweets per user.  It reveals an interest
ing issue, namely, some users are much more active on twitter than others, by
an order of magnitude.  Now, that reveals an interesting fact about twitter u
sage but also means the corpus of data will be more informative about the twi
tter usage of those highly active accounts.
TweetsPerUser <- as.data.frame(table(vox_pop$username))

## extracting tweets by date (year) for inspection
# converting the ISO 4601 format of teh date to somehthing usable.
TweetsByDate<- strptime(vox_pop$date_of_tweet,"%Y-%m-%dT%H:%M:%S")

# parsing out the year
TweetsByDate<- as.numeric(format(TweetsByDate,"%Y"))

# tabulating year frequencies
TweetsByDate<- table(TweetsByDate)
```

parsing date column

If your data has been saved with only the original date column the code below can parse the dates out into individual sections. It should not be run otherwise.

```
vox_pop<-vox_pop %>% dplyr::mutate(year = lubridate::year(date_of_tweet),
                month = lubridate::month(date_of_tweet),
                day = lubridate::day(date_of_tweet))
```

basic analysis

```
text.df <- as_tibble(vox_pop)

# summary of vox_pop data set
summary(text.df)
```

```
##    author_id           username           tweet_id             text
##  Min.   :5.558e+06   Length:343978      Min.   :4.452e+08   Length:343978
##  1st Qu.:3.792e+07   Class :character   1st Qu.:4.293e+17   Class :charact
er
##  Median :2.408e+08   Mode  :character   Median :8.436e+17   Mode  :charact
er
##  Mean   :1.558e+17                      Mean   :8.215e+17
##  3rd Qu.:1.340e+09                      3rd Qu.:1.240e+18
##  Max.   :1.469e+18                      Max.   :1.596e+18
##     retweets            replies          quote_count        possibly_sensi
tive
##  Min.   :      0.0   Min.   :    0.00   Min.   :    0.000   Mode :logical
##  1st Qu.:      1.0   1st Qu.:    0.00   1st Qu.:    0.000   FALSE:343896
##  Median :      3.0   Median :    0.00   Median :    0.000   TRUE :82
```

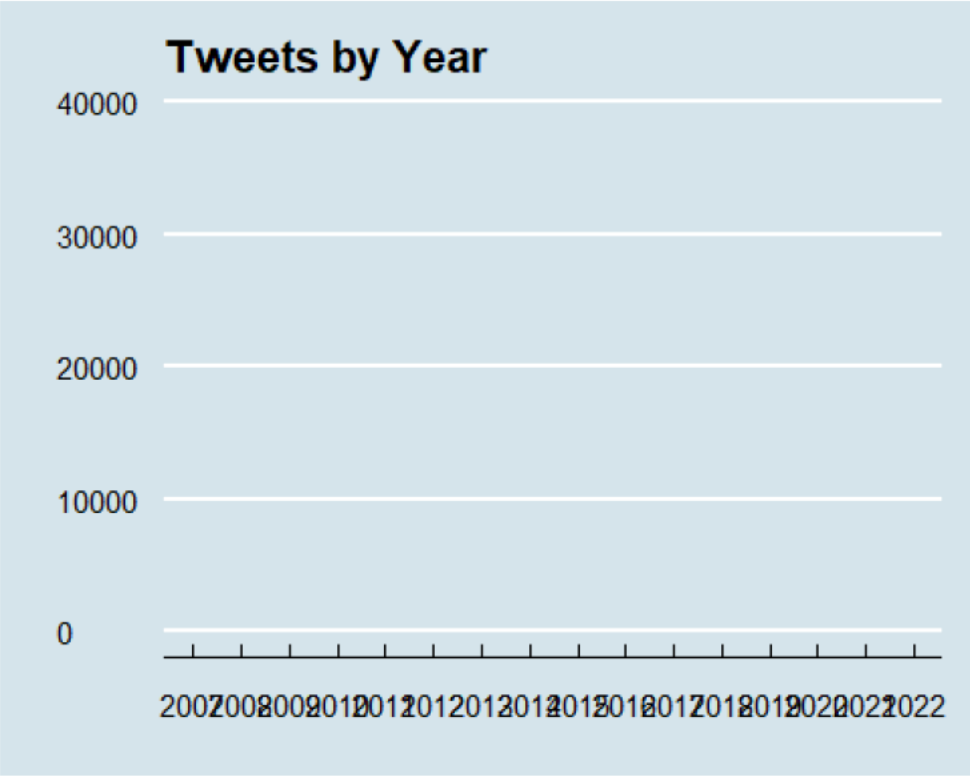
```
##  Mean   :    226.8   Mean   :   27.93   Mean   :    5.303
##  3rd Qu.:     11.0   3rd Qu.:    3.00   3rd Qu.:    0.000
##  Max.   :3046203.0   Max.   :44056.00   Max.   :11687.000
##    num_tweets         name               likes            following
##  Min.   :     4   Length:343978      Min.   :     0.0   Min.   :     0
##  1st Qu.:  3208   Class :character   1st Qu.:     0.0   1st Qu.:   589
##  Median :  6721   Mode  :character   Median :     2.0   Median :  1080
##  Mean   :  8848                      Mean   :   226.3   Mean   :  2318
##  3rd Qu.: 11298                      3rd Qu.:    14.0   3rd Qu.:  2223
##  Max.   :106757                      Max.   :335235.0   Max.   :126990
##    followers        date_of_tweet                     account_description
##  Min.   :       4   Min.   :2007-11-26 15:17:47.00   Length:343978
##  1st Qu.:   17919   1st Qu.:2014-01-31 14:52:39.75   Class :character
##  Median :   37224   Median :2017-03-19 22:34:38.00   Mode  :character
##  Mean   :  346203   Mean   :2016-12-26 09:17:18.99
##  3rd Qu.:  121015   3rd Qu.:2020-03-16 20:34:52.75
##  Max.   :20035626   Max.   :2022-11-24 22:26:20.00
##  account_created                    year          month
##  Min.   :2007-04-27 16:05:52.00   Min.   :2007   Min.   : 1.000
##  1st Qu.:2009-05-05 13:18:55.00   1st Qu.:2014   1st Qu.: 4.000
##  Median :2011-01-20 20:14:21.00   Median :2017   Median : 7.000
##  Mean   :2012-02-10 14:53:37.02   Mean   :2016   Mean   : 6.688
##  3rd Qu.:2013-04-09 19:02:01.00   3rd Qu.:2020   3rd Qu.:10.000
##  Max.   :2021-12-08 20:21:30.00   Max.   :2022   Max.   :12.000
##       day          party
##  Min.   : 1.00   Mode:logical
##  1st Qu.: 8.00   NA's:343978
##  Median :17.00
##  Mean   :15.84
##  3rd Qu.:23.00
##  Max.   :31.00
```

```
# Interestingly the summary revealed 82 tweets marked as potentially sensitiv
e.  Upon closer inspection, the concern seems overblown, most reference disas
ters, or war (especially World War) which may be the cause.  A surprising num
ber of the tweets were about California public health.  None were controversi
al, but the coverage of disasters might have involved graphic images.
sensitive_tweets <- vox_pop$text[vox_pop$possibly_sensitive == TRUE]
sensitive_tweets <- vox_pop[vox_pop$possibly_sensitive == TRUE,]
#View(sensitive_tweets)

rm(text.df)
```

Date Range

Below tweets are broken out and graphed by year. This demonstrates a notable increase in tweets since 2007. This increase can be explained by the growth in Twitter's popularity, and, it reflects the dataset expanding as current legislators win office and enter the dataset studied here. One other trend of note is the

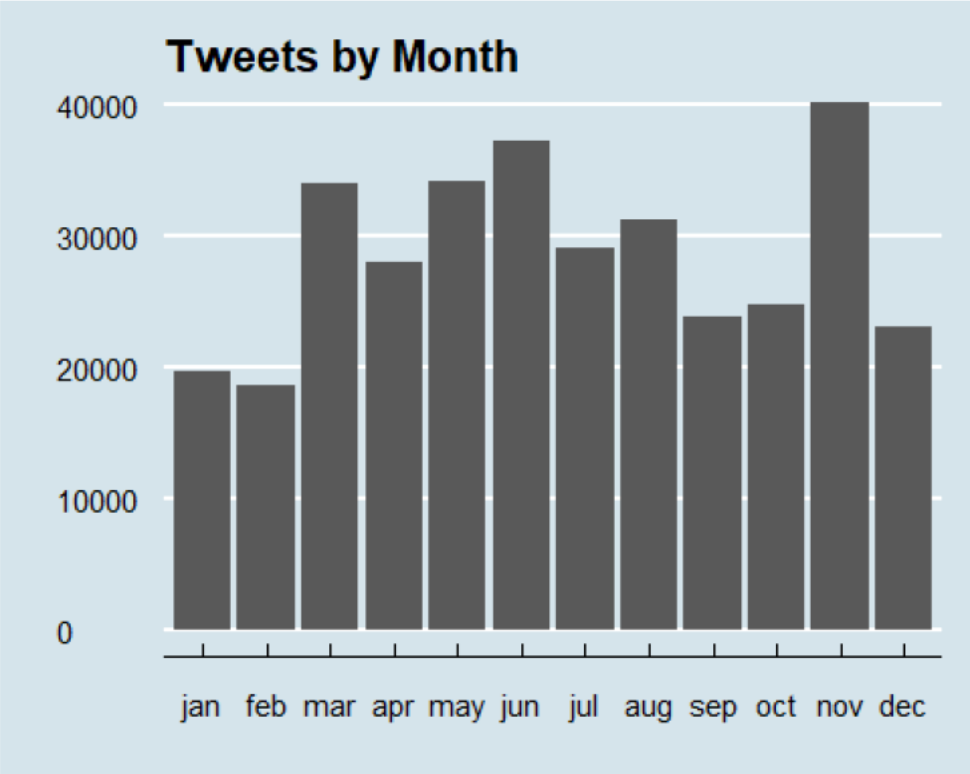
```
TweetsByDate<-as.data.frame(table(vox_pop$year))


colnames(TweetsByDate)[1]<-"Year"


ggplot(data = TweetsByDate) + aes(Year, Freq) +
  geom_line(stat ="identity") + theme_economist() + labs(title = "Tweets by Y
ear") + xlab("") + ylab("")

## `geom_line()`: Each group consists of only one observation.
## ℹ Do you need to adjust the group aesthetic?
```



Tweets by Month

```
#TweetsByMonth<-table(vox_pop$month)
TweetsByMonth<-vox_pop %>% select(year,month)

TweetsByMonth<-as.data.frame.table(table(vox_pop$month),responseName = "Freq"
)

MonthsAbb<-c("jan", "feb", "mar", "apr", "may", "jun","jul", "aug", "sep", "o
ct", "nov", "dec")

colnames(TweetsByMonth)[1]<-"Month"
```

```
TweetsByMonth$MonthAbb<-MonthsAbb[TweetsByMonth$Month]


ggplot(data = TweetsByMonth) + aes(Month,Freq) +
  geom_bar(stat = "identity") + theme_economist()+labs(title = "Tweets by Mon
th")+xlab("")+ylab("")+scale_x_discrete(labels = MonthsAbb)
```



adding party id

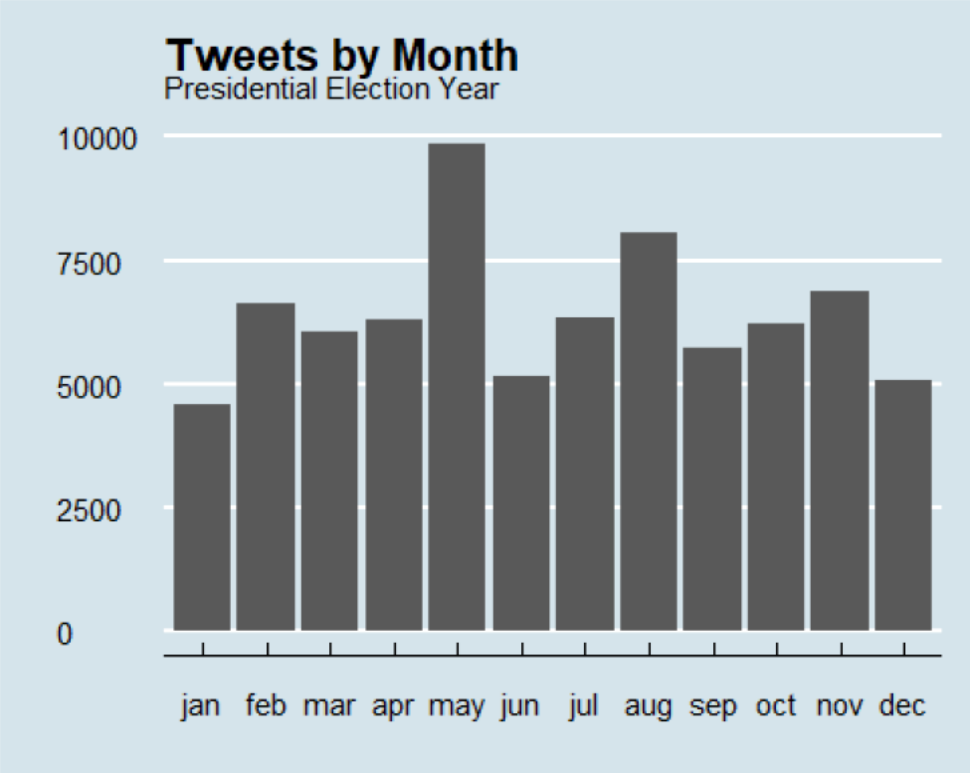
Code to add party identification to authors if available.

```
party_id<- read.csv("legislators-current.csv")
colnames(party_id)
party_id<-party_id[,c("twitter","party")]
names(party_id)[1]<-"author_id"

#vox_pop_temp<-vox_pop

vox_pop<-merge(x=vox_pop,y=party_id,by= "author_id", all.x = TRUE)

length(is.na(vox_pop$party))
```

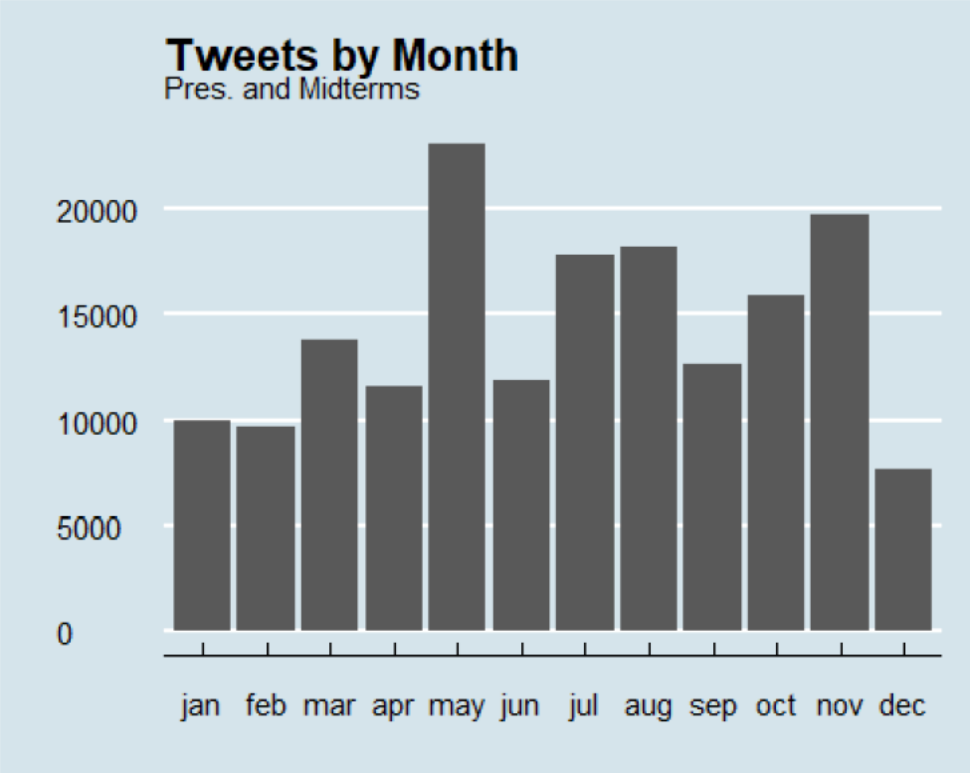
election year effect-plots

```
require(magrittr)
TweetsByElectionYear<-vox_pop %>% select(year,month,date_of_tweet)
```

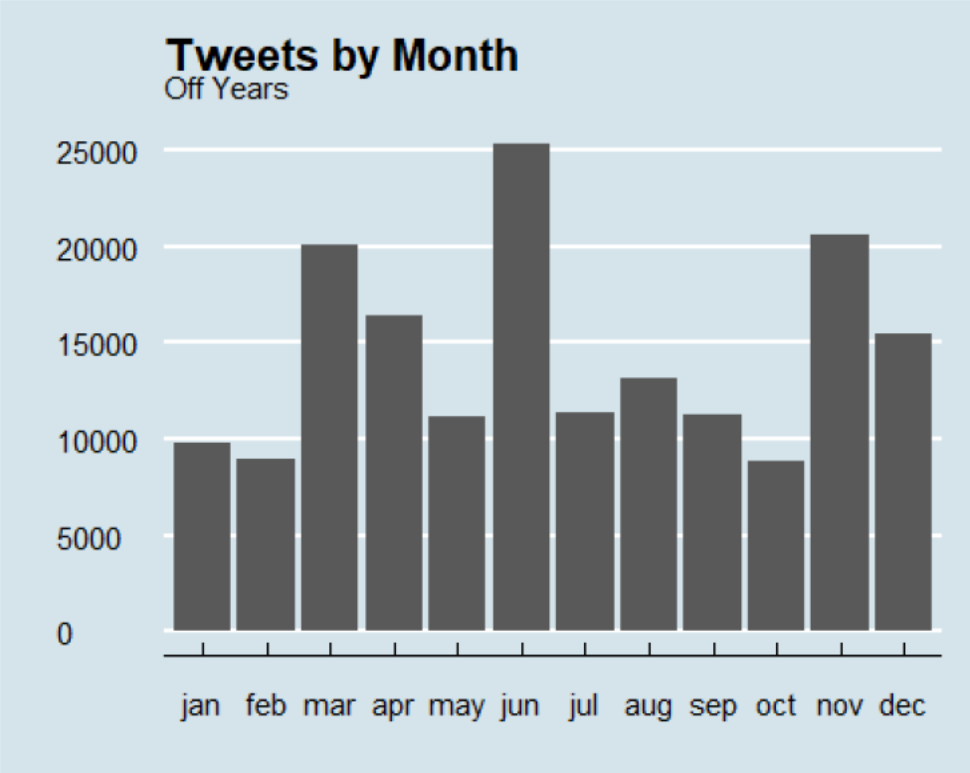
```r
PresElectionYears<-c(2008,2012,2016,2020)
PresAndMidterms<-c(seq(from=2008,to=2022,by=2))
OffYear<-c(2008:2022)
OffYear<-OffYear[!OffYear %in% PresAndMidterms]

PresElectionYearDF<-TweetsByElectionYear[TweetsByElectionYear$year %in% PresElectionYears,]
PresAndMidtermsDF<-TweetsByElectionYear[TweetsByElectionYear$year %in% PresAndMidterms,]
OffYearDF<-TweetsByElectionYear[TweetsByElectionYear$year %in% OffYear,]

PresElectionYearDF<-as.data.frame.table(table(PresElectionYearDF['month']),responseName = 'DFreq')

PresAndMidtermsDF<-as.data.frame.table(table(PresAndMidtermsDF['month']),responseName = "DFreq")

OffYearDF<-as.data.frame.table(table(OffYearDF['month']),responseName = "DFreq")

names(PresElectionYearDF)[1]<-"month"
PresElectionYearDF$MonthsAbb<-MonthsAbb[PresElectionYearDF$month]

names(PresAndMidtermsDF)[1]<-"month"
PresAndMidtermsDF$MonthsAbb<-MonthsAbb[PresAndMidtermsDF$month]

names(OffYearDF)[1]<-"month"
OffYearDF$MonthsAbb<-MonthsAbb[OffYearDF$month]

par(mfrow=c(1,3))

ggplot(data = PresElectionYearDF) + aes(month,DFreq) +
  geom_bar(stat = "identity") + theme_economist()+labs(title = "Tweets by Month",subtitle = "Presidential Election Year")+xlab("")+ylab("")+scale_x_discrete(labels = MonthsAbb)
```




```
ggplot(data = PresAndMidtermsDF) + aes(month,DFreq) +
  geom_bar(stat = "identity") + theme_economist()+labs(title = "Tweets by Mon
th",subtitle = "Pres. and Midterms")+xlab("")+ylab("")+scale_x_discrete(label
s = MonthsAbb)
```

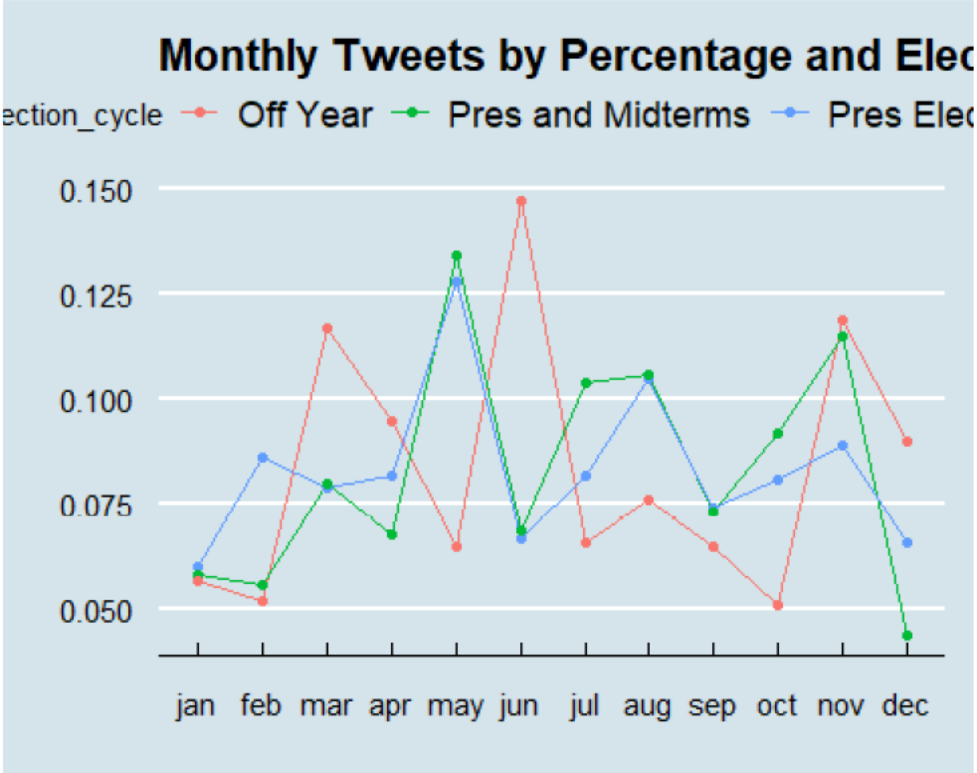



```
ggplot(data = OffYearDF) + aes(month,DFreq) +
  geom_bar(stat = "identity") + theme_economist()+labs(title = "Tweets by Mon
th",subtitle = "Off Years")+xlab("")+ylab("")+scale_x_discrete(labels = Month
sAbb)
```



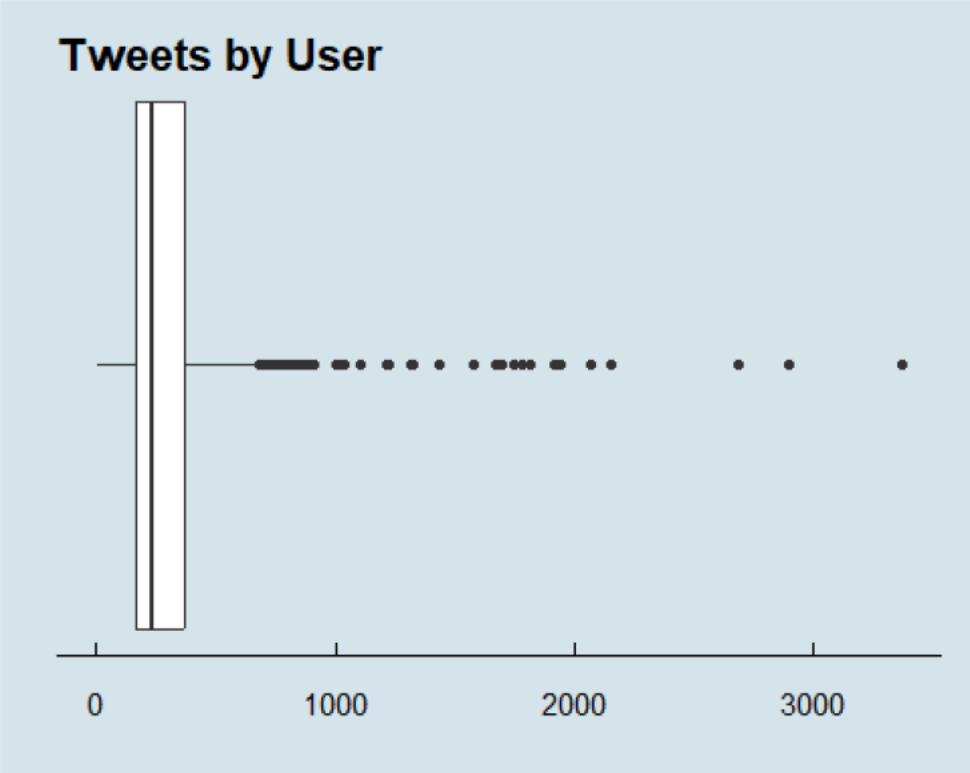
comparing tweets by election cycles

```
df1<-PresElectionYearDF
df2<-PresAndMidtermsDF
df3<-OffYearDF

df1<-cbind(df1,"Pres Election Year")
df1$DFreq<-round(df1$DFreq/sum(df1$DFreq),3)
names(df1)[4]<-"election_cycle"

df2<-cbind(df2,"Pres and Midterms")
df2$DFreq<-round(df2$DFreq/sum(df2$DFreq),3)
names(df2)[4]<-"election_cycle"

df3<-cbind(df3,"Off Year")
df3$DFreq<-round(df3$DFreq/sum(df3$DFreq),3)
names(df3)[4]<-"election_cycle"

linebymonthdf<-rbind(df1,df2,df3)

ggplot(linebymonthdf, aes(x=month, y=DFreq, group=election_cycle)) +
  geom_line(aes(color=election_cycle))+
  geom_point(aes(color=election_cycle))+
  theme_economist()+xlab("")+ylab("")+scale_x_discrete(labels = MonthsAbb)+la
bs(title="Monthly Tweets by Percentage and Election Cycle")
```



Tweets by User

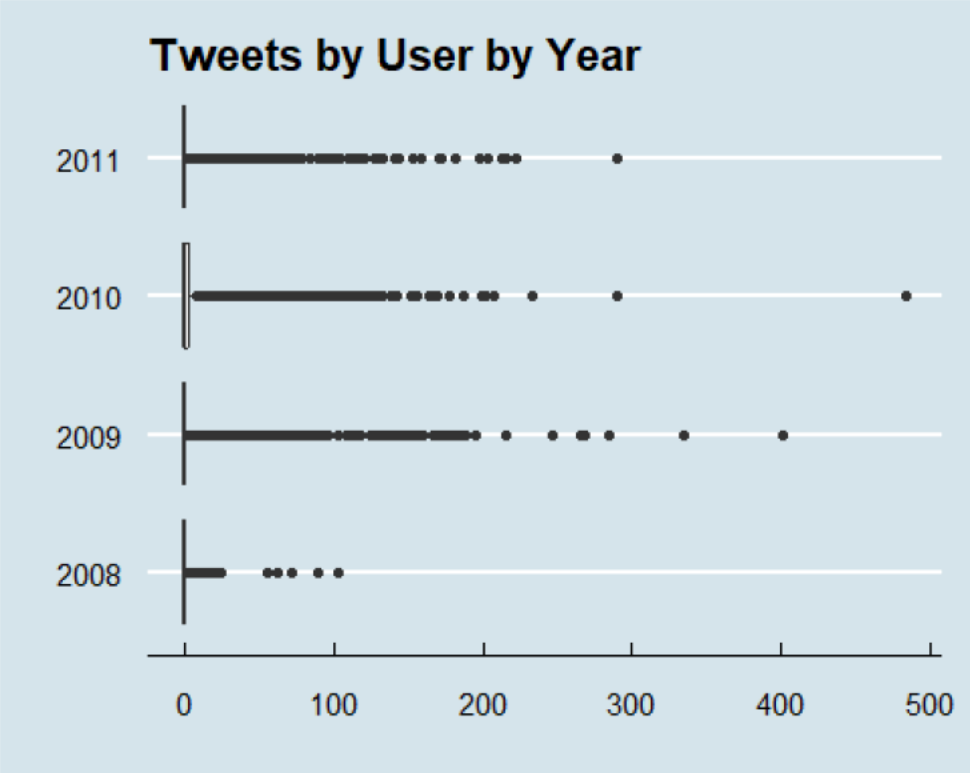
```
#the code below counts the number of tweets per user.  It reveals an interest
ing issue, namely, some users are much more active on twitter than others, by
an order of magnitude.  Now, that reveals an interesting fact about twitter u
sage but also means the corpus of data will be more informative about the twi
tter usage of those highly active accounts.  This persisted even when the dat
a collection methods were altered to prioritize userns with few tweets.

TweetsPerUser <- as.data.frame(table(vox_pop$username))
names(TweetsPerUser)[1]<-"username"
#View(TweetsPerUser)

par(mfrow=c(2,1))

ggplot(data = TweetsPerUser,aes(x = Freq)) +
  geom_boxplot()+theme_economist()+labs(title = "Tweets by User")+xlab("")+yl
ab("")+scale_y_continuous(breaks = NULL)
```



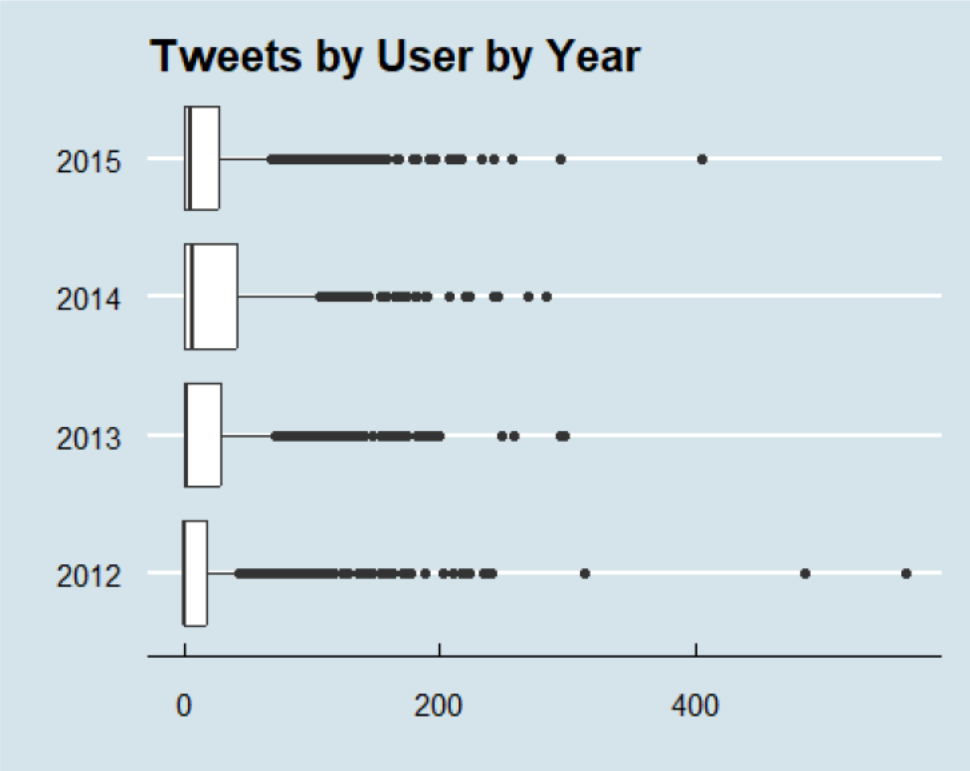

```
summary(TweetsPerUser$Freq)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     2.0   161.0   236.0   311.3   367.0  3375.0
```

Tweets By User/Year

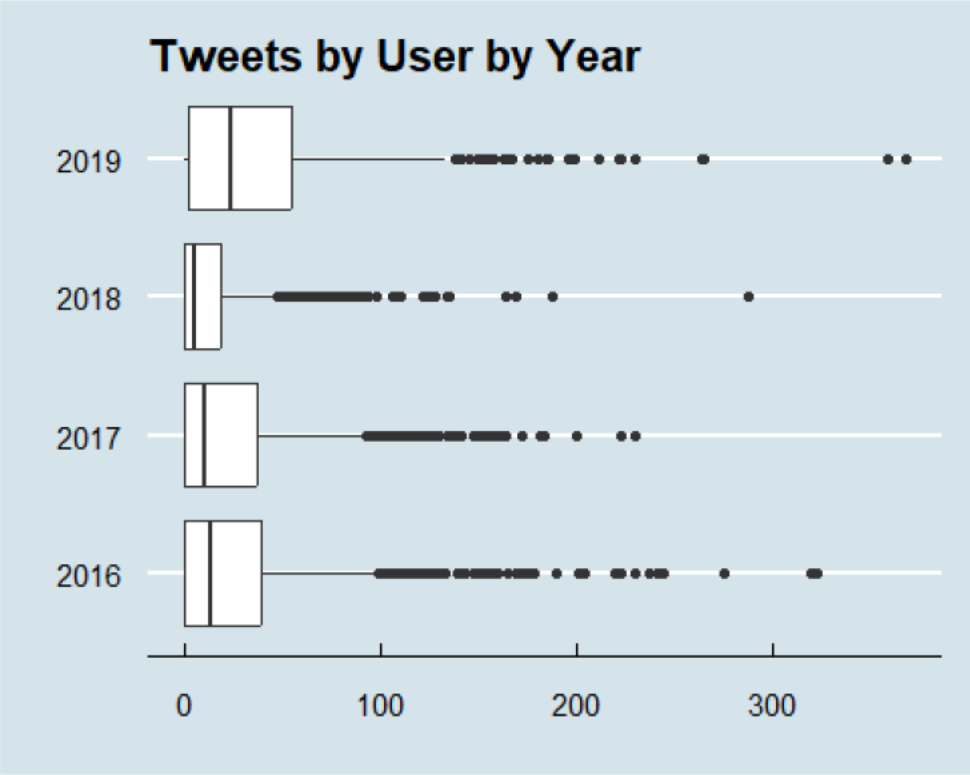
```
TweetsPerUser <- as.data.frame(table(vox_pop[,c('username','year')]))

df1<-c(2008:2011)
df2<-c(2012:2015)
df3<-c(2016:2019)
df4<-c(2020:2022)

df1<-TweetsPerUser[TweetsPerUser$year %in% df1,]
df2<-TweetsPerUser[TweetsPerUser$year %in% df2,]
df3<-TweetsPerUser[TweetsPerUser$year %in% df3,]
df4<-TweetsPerUser[TweetsPerUser$year %in% df4,]

par(mfrow = c(2, 2))

ggplot(data = df1,aes(x = Freq, y = year)) +
  geom_boxplot()+theme_economist()+labs(title = "Tweets by User by Year")+xla
b("")+ylab("")
```



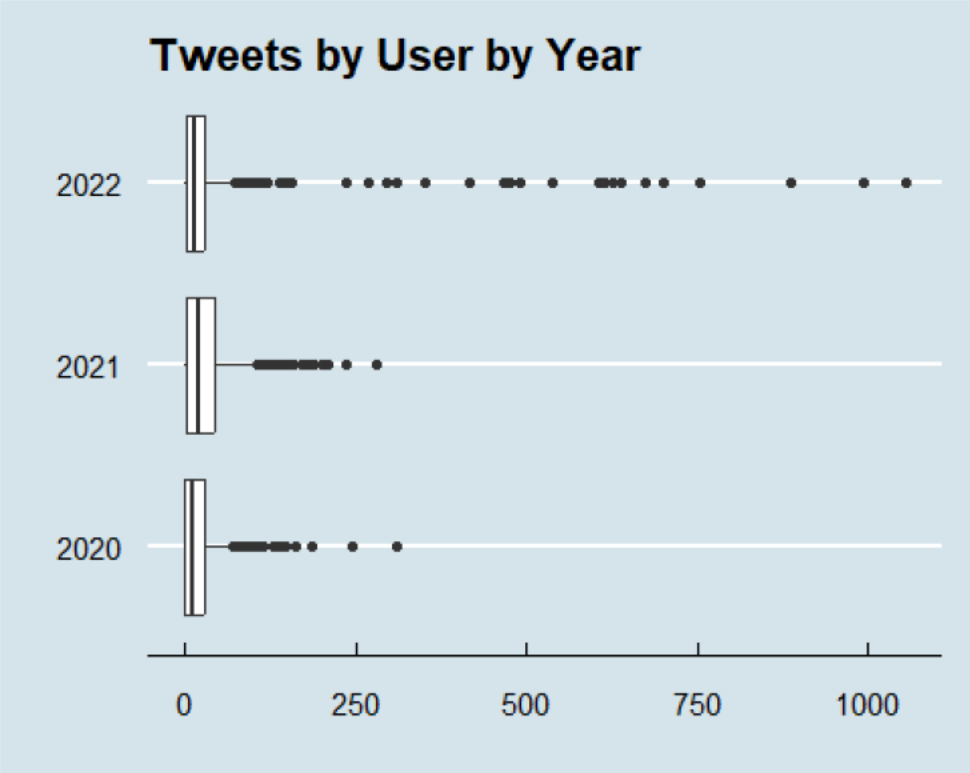

```
ggplot(data = df2,aes(x = Freq, y = year)) +
  geom_boxplot()+theme_economist()+labs(title = "Tweets by User by Year")+xla
b("")+ylab("")
```



```
ggplot(data = df3,aes(x = Freq, y = year)) +
  geom_boxplot()+theme_economist()+labs(title = "Tweets by User by Year")+xla
b("")+ylab("")
```




```
ggplot(data = df4,aes(x = Freq, y = year)) +
  geom_boxplot()+theme_economist()+labs(title = "Tweets by User by Year")+xla
b("")+ylab("")
```



Text Analysis and Visualization

preliminary text analysis

Performign preliminary text analysis which we can expect to demonstrate the process but return mainly stopwords.

```
corp<-corpus(x=vox_pop, docid_field = "tweet_id", text_field = "text",unique_
docnames = FALSE, meta = colnames(vox_pop[,c('year',"month","date_of_tweet","
day","author_id","username")]))

#corp<-corpus(vox_pop$text)

toks<- tokens(corp,remove_punct = TRUE, what = "fastestword", remove_symbols
= TRUE, remove_numbers = TRUE,remove_url = TRUE,remove_separators = TRUE)

#dfm_trim(min_termfreq = 200,min_docfreq = 200)%>%

prelim_dfm<-dfm(toks)

topfeatures(prelim_dfm)

##    the     to    and     of     in    for      a     on     is     rt
## 302028 280612 145996 133229 124644 115457 109437  78272  68471  67767
```

creating corpus

```
corp <- corpus(vox_pop, text_field = "text")

summary(corp, 1)

## Corpus consisting of 343978 documents, showing 1 document:
##
##   Text Types Tokens Sentences    author_id username     tweet_id retweets
##  text1    37     46         2 1.002631e+18  RepCori 1.408194e+18       21
##  replies quote_count possibly_sensitive num_tweets                   name
##       13           8              FALSE       1684 Congresswoman Cori Bush
##  likes following followers       date_of_tweet
##    108       295    114733 2021-06-24 22:41:33
##
account_description
##  Doing the most for MO-01 (STL)—starting with those with the greatest need
. Politician + Activist = Politivist. Member, @HouseJudiciary, @OversightDems
. She/her.
##      account_created year month day party
##  2018-06-01 19:20:32 2021     6  24    NA

colnames(docvars(corp))

##  [1] "author_id"           "username"            "tweet_id"
##  [4] "retweets"            "replies"             "quote_count"
##  [7] "possibly_sensitive"  "num_tweets"          "name"
## [10] "likes"               "following"           "followers"
## [13] "date_of_tweet"       "account_description" "account_created"
## [16] "year"                "month"               "day"
## [19] "party"
```

pre-processing and tokenization

```
custom.stopwords <- c(stopwords("english"), "the","amp","&amp;","rt")

toks <- tokens(corp, remove_punct = TRUE, remove_url = TRUE, remove_separator
s = TRUE, include_docvars = TRUE, what = "word") %>%
  tokens_remove(pattern = c("#*","@*"))

toks <-tokens_select(toks, pattern = custom.stopwords, min_nchar = 2, selecti
on = "remove", padding = FALSE)

toks<-tokens_remove(x= toks, "\\p{Z}", valuetype = "regex")
```

creating reduced dfm

```
dfm_slim<-dfm(toks)%>%
dfm_trim(max_termfreq = 2872403)
```

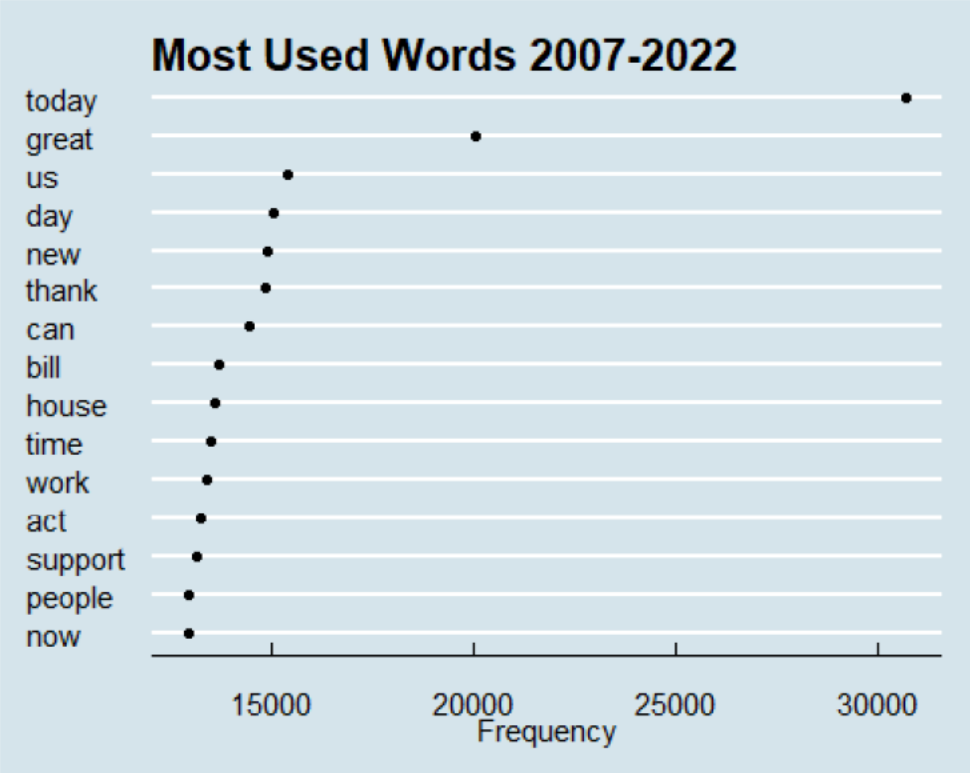
dfm analysis - Top Words

```
topwords<-textstat_frequency(dfm_slim, n = 100)

topwords<-topwords[1:100,c(1,3)]

array(topwords$feature)
```

```
##   [1] "today"       "great"       "us"          "day"         "new"
##   [6] "thank"       "can"         "bill"        "house"       "time"
##  [11] "work"        "act"         "support"     "people"      "now"
##  [16] "need"        "help"        "health"      "congress"    "must"
##  [21] "thanks"      "get"         "american"    "just"        "vote"
##  [26] "proud"       "families"    "president"   "make"        "care"
##  [31] "americans"   "one"         "join"        "happy"       "see"
##  [36] "last"        "state"       "senate"      "country"     "working"
##  [41] "week"        "community"   "year"        "like"        "good"
##  [46] "women"       "first"       "jobs"        "back"        "family"
##  [51] "right"       "years"       "every"       "national"    "keep"
##  [56] "many"        "tax"         "morning"     "u.s"         "protect"
##  [61] "continue"    "watch"       "law"         "take"        "live"
##  [66] "fight"       "federal"     "america"     "know"        "read"
##  [71] "county"      "service"     "office"      "trump"       "veterans"
##  [76] "honor"       "across"      "forward"     "bipartisan"  "stop"
##  [81] "economy"     "important"   "communities" "tonight"     "news"
##  [86] "go"          "democrats"   "students"    "meeting"     "legislation
"
##  [91] "public"      "discuss"     "energy"      "way"         "want"
##  [96] "security"    "plan"        "hearing"     "everyone"    "nation"
```

Top Words - Plot

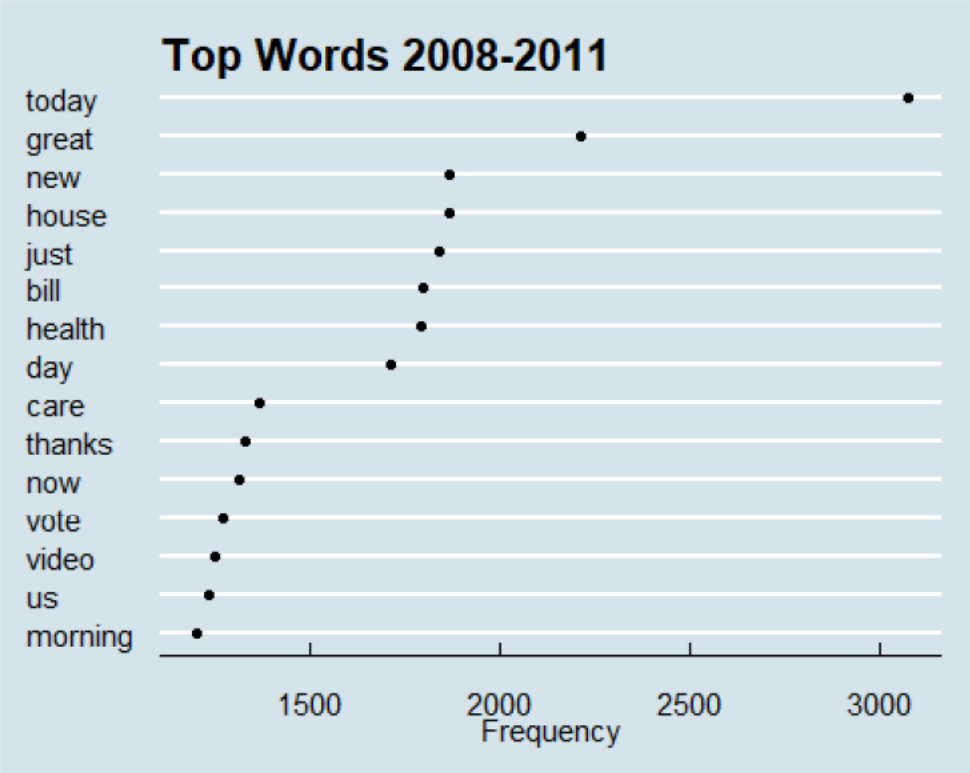
```
dfm_slim %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(title = "Most Used Words 2007-2022",x = NULL, y = "Frequency") +
  theme_economist()
```



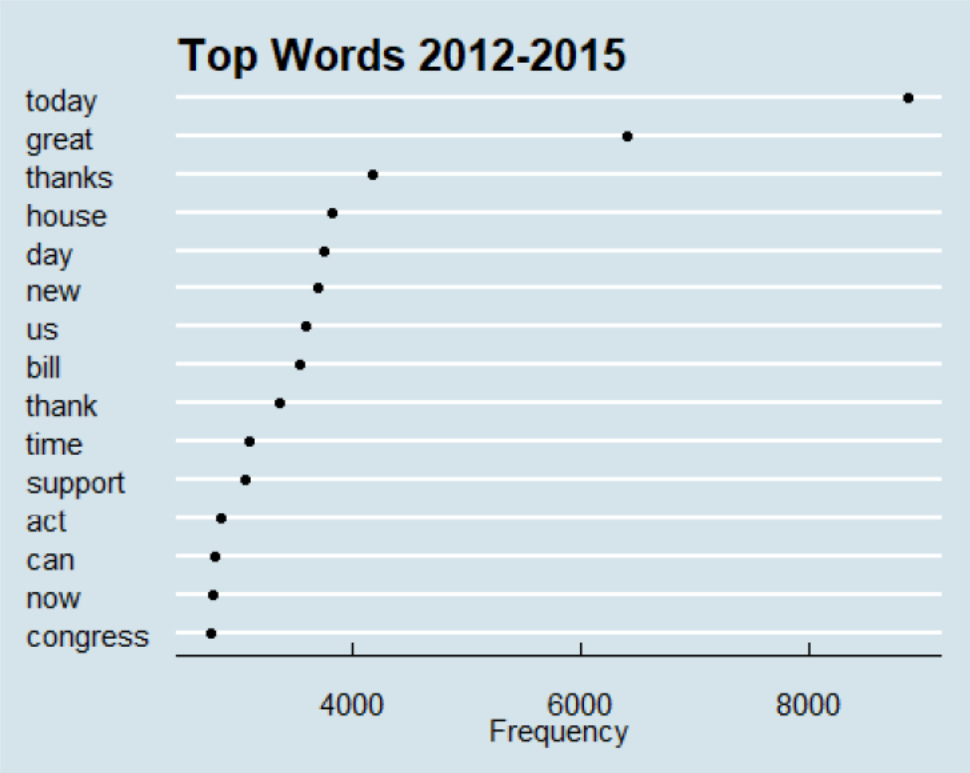
Top Words by Year

Key Takeaways 2008-11 was the only time a hashtag made it to the top words list. 2020-22 was the only few years where "American" it.

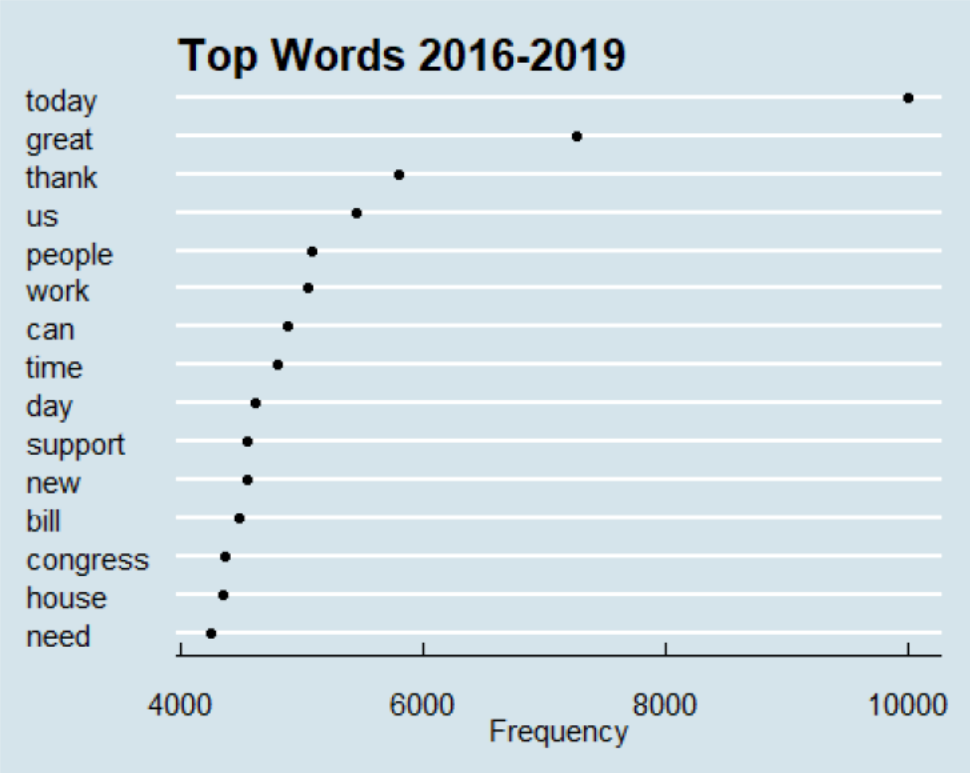
```
y08_11<-c(2008:2011)
y12_15<-c(2012:2015)
y16_19<-c(2016:2019)
y20_22<-c(2020:2022)

df1<-dfm_subset(dfm_slim, year %in% y08_11)
df2<-dfm_subset(dfm_slim, year %in% y12_15)
df3<-dfm_subset(dfm_slim, year %in% y16_19)
df4<-dfm_subset(dfm_slim, year %in% y20_22)

par(mfrow = c(2, 2))

df1 %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency", title = "Top Words 2008-2011") +
  theme_economist()
```



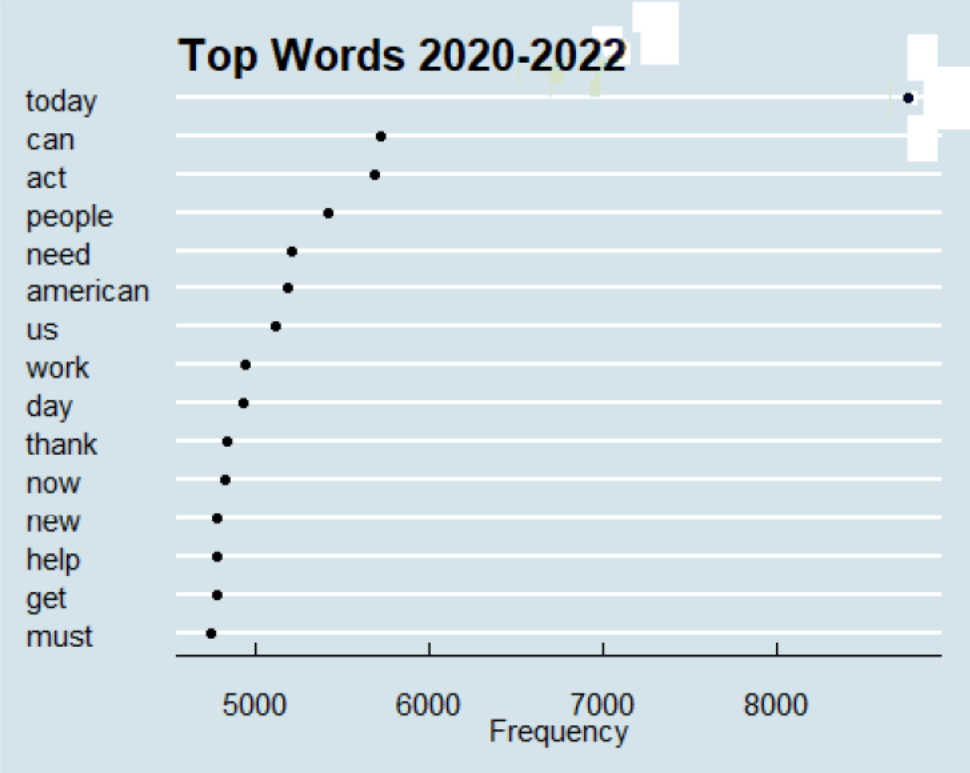

```
df2 %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency", title = "Top Words 2012-2015") +
  theme_economist()
```




```
df3 %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency", title = "Top Words 2016-2019") +
  theme_economist()
```



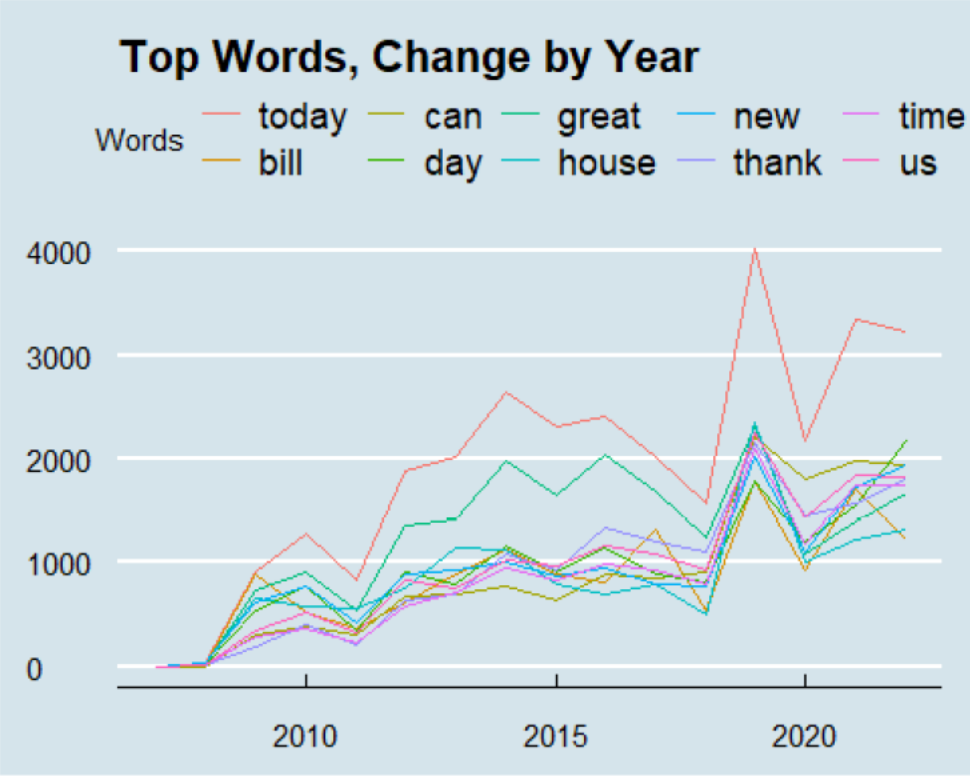

```
df4 %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency", title = "Top Words 2020-2022") +
  theme_economist()
```



Top Words by Year-Line Plot

```
Top10_Words <-textstat_frequency(dfm_slim,n = 10)

Top10_Words<-Top10_Words$feature

tokstop10<-tokens_select(toks, pattern = Top10_Words, selection = "keep")

dfm_top10<-dfm(tokstop10,remove_padding = TRUE)

dfm_top10<-dfm_trim(x=dfm_top10, min_termfreq = 2500, min_docfreq = 2500)

# This converts the more complex DFM object into a data frame that can be mor
e easily loaded into ggplot.
# however, it won't work if the sparsity of the DFM is too high.  To reduce s
parsity the function dfm_trim(), aggregate, and selection can be used to redu
ce the empty rows to an acceptable size.

dfm_top10 <- data.frame(convert(dfm_top10, to = "data.frame"), Year = dfm_top
10$year)

# dropping tweet_id from df
dfm_top10<-dfm_top10[,2:length(colnames(dfm_top10))]

#aggregating the count of features(the hashtags in this case) by year.
dfm_top10<-aggregate(dfm_top10,by = list(dfm_top10$Year),FUN = sum)
```

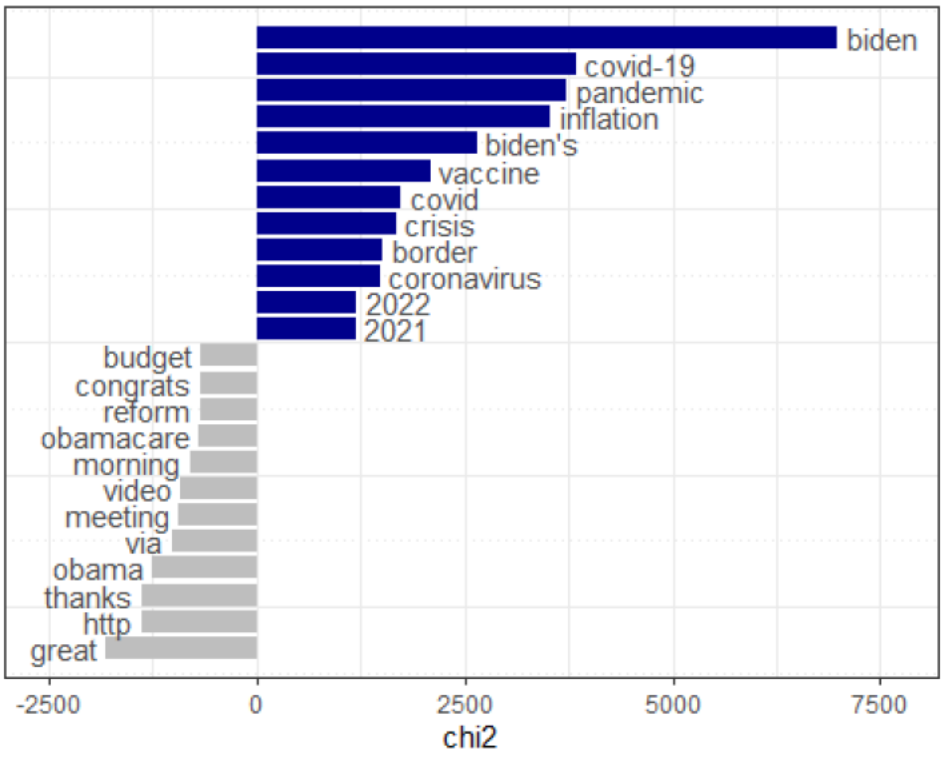
```
tempcol2drop<-length(colnames(dfm_top10))-1

#dropping extra column
dfm_top10<-dfm_top10[,1:tempcol2drop]

#sanity check
#colnames(dfm_top10)

#renaming grouping variable to year
dfm_top10<-dfm_top10 %>% rename(Year=Group.1)

#converting df from a wide to a long format
dfm_top10<- dfm_top10 %>%
  pivot_longer(names_to = "feature",cols = !Year,names_prefix = "X.",values_t
o = "frequency")

#creating plot
ggplot(dfm_top10, aes(x = Year, y = frequency, color = reorder(feature,!frequ
ency)))+
  geom_line()+theme_economist()+labs(x = NULL, y = NULL, title = "Top Words,
Change by Year", color = "Words")
```



Top Word Keyness - Before and After the Pandemic

```
tstat_key_words <- textstat_keyness(dfm_slim,
                                    target = dfm_slim$date_of_tweet >= '2020-03-01
```

```
00:00:00')
textplot_keyness(tstat_key_words, n = 12, show_legend = FALSE)

## Warning: The `guide` argument in `scale_*()` cannot be `FALSE`. This was d
eprecated in
## ggplot2 3.3.4.
## ℹ Please use "none" instead.
## ℹ The deprecated feature was likely used in the quanteda.textplots package
.
##   Please report the issue at
##   <]8;;https://github.com/quanteda/quanteda.textplots/issueshttps://github
.com/quanteda/quanteda.textplots/issues]8;;>.
```



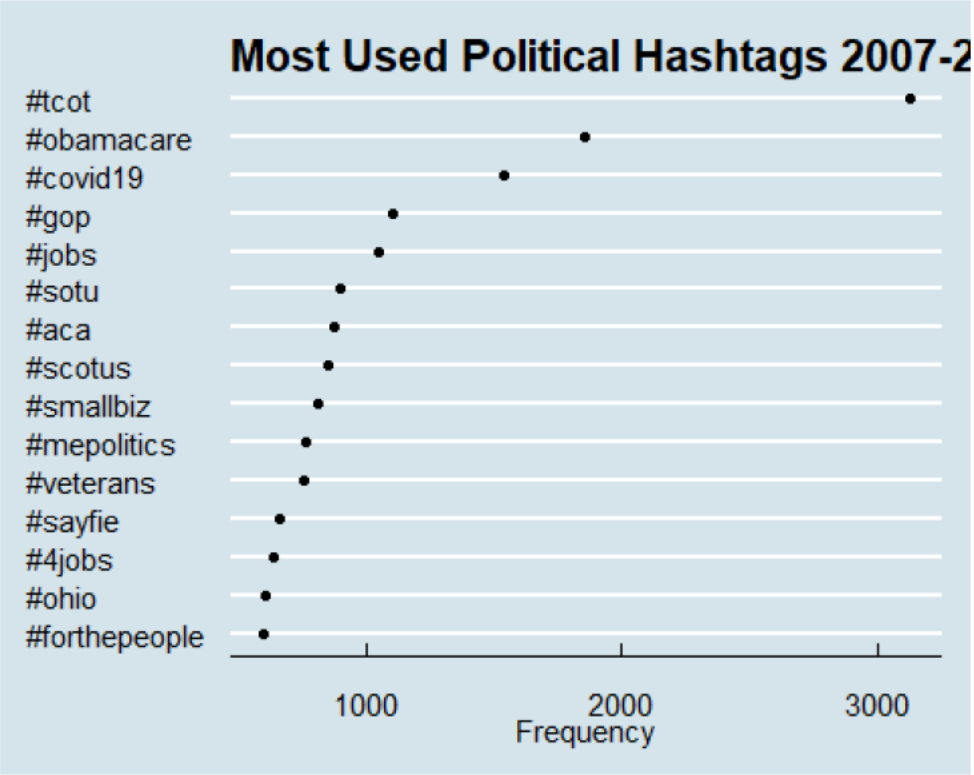
creating hashtag dfm

```
toks_tweets<- tokens(corp, remove_punct = TRUE, remove_url = TRUE, remove_sep
arators = TRUE, include_docvars = TRUE, what = "word") %>% tokens_keep(patter
n = "#*")

dfmat_tweets<-dfm(toks_tweets)
```

Hashtag plots

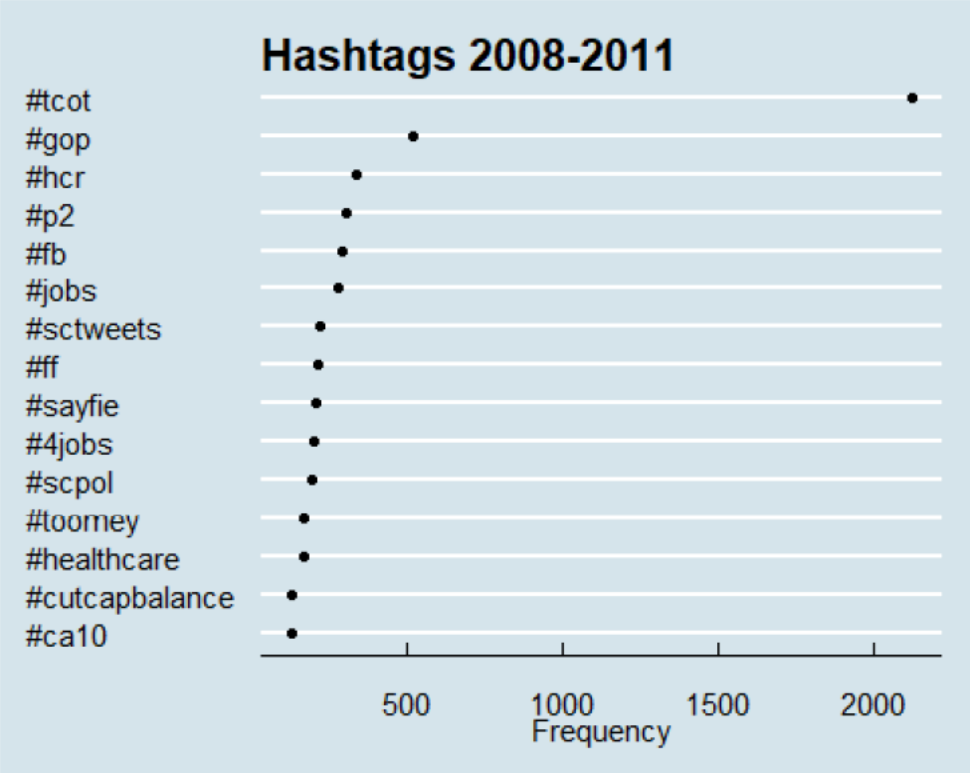
```
dfmat_tweets %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
```

```
  labs(title = "Most Used Political Hashtags 2007-2022",x = NULL, y = "Freque
ncy") +
  theme_economist()
```

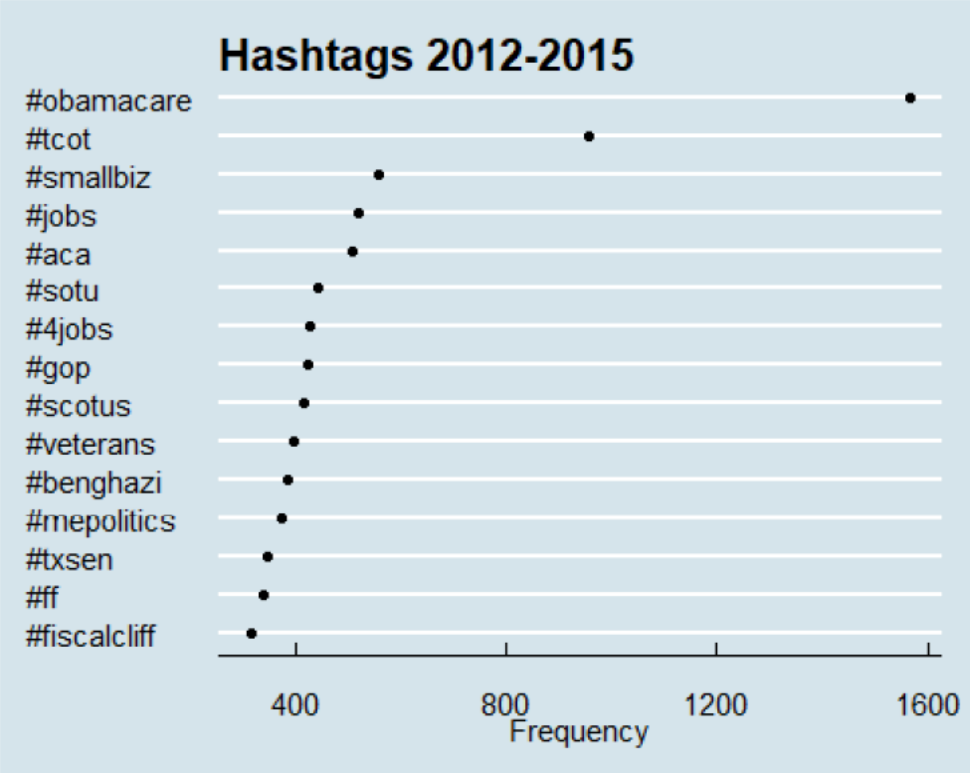


plots by year

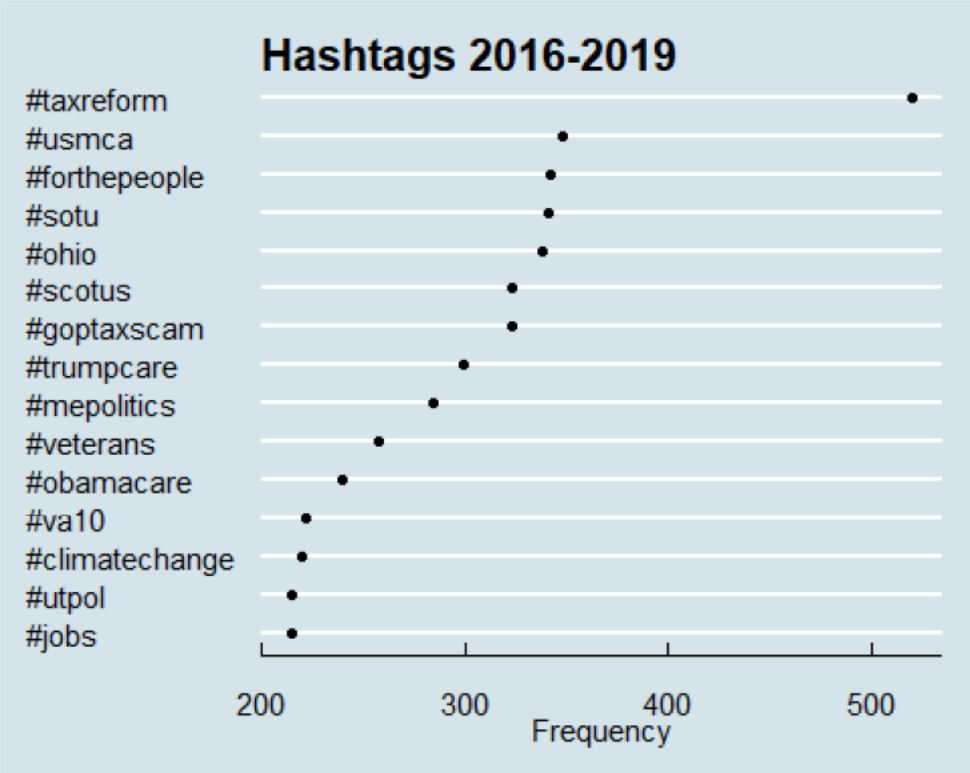
```
y08_11<-c(2008:2011)
y12_15<-c(2012:2015)
y16_19<-c(2016:2019)
y20_22<-c(2020:2022)

#dfmat_tweets<-dfm_subset(dfmat_tweets, year %in% PresElectionYears)

df1<-dfm_subset(dfmat_tweets, year %in% y08_11)
df2<-dfm_subset(dfmat_tweets, year %in% y12_15)
df3<-dfm_subset(dfmat_tweets, year %in% y16_19)
df4<-dfm_subset(dfmat_tweets, year %in% y20_22)

par(mfrow = c(2, 2))

df1 %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency", title = "Hashtags 2008-2011") +
  theme_economist()
```



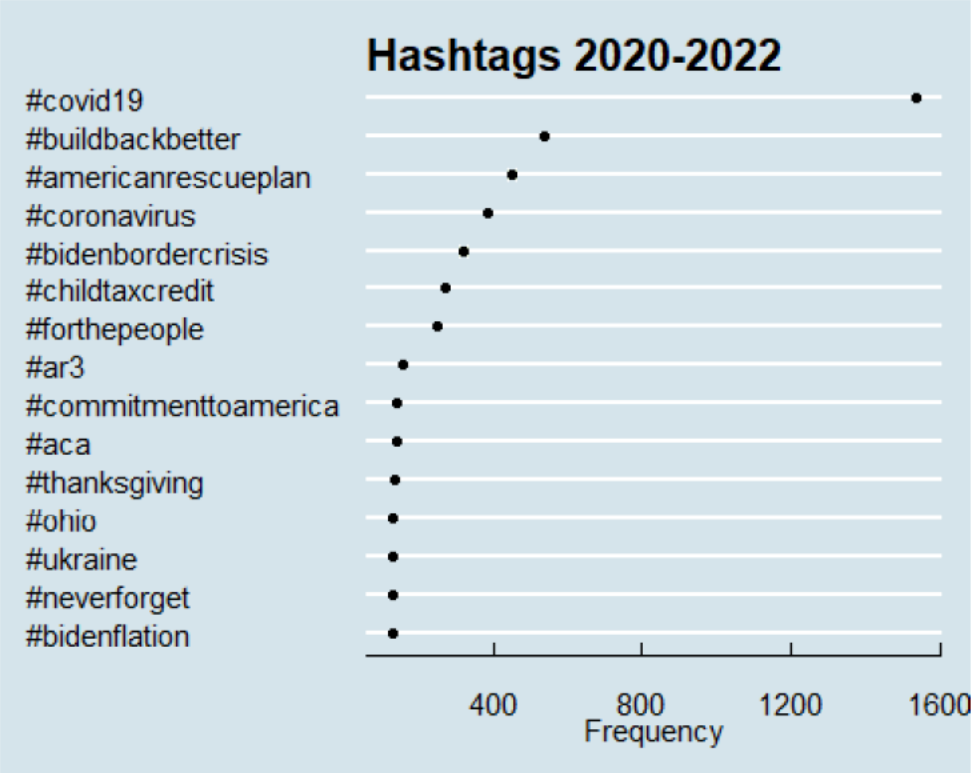

```
df2 %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency", title = "Hashtags 2012-2015") +
  theme_economist()
```




```
df3 %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency", title = "Hashtags 2016-2019") +
  theme_economist()
```




```
df4 %>%
  textstat_frequency(n = 15) %>%
  ggplot(aes(x = reorder(feature, frequency), y = frequency)) +
  geom_point() +
  coord_flip() +
  labs(x = NULL, y = "Frequency", title = "Hashtags 2020-2022") +
  theme_economist()
```



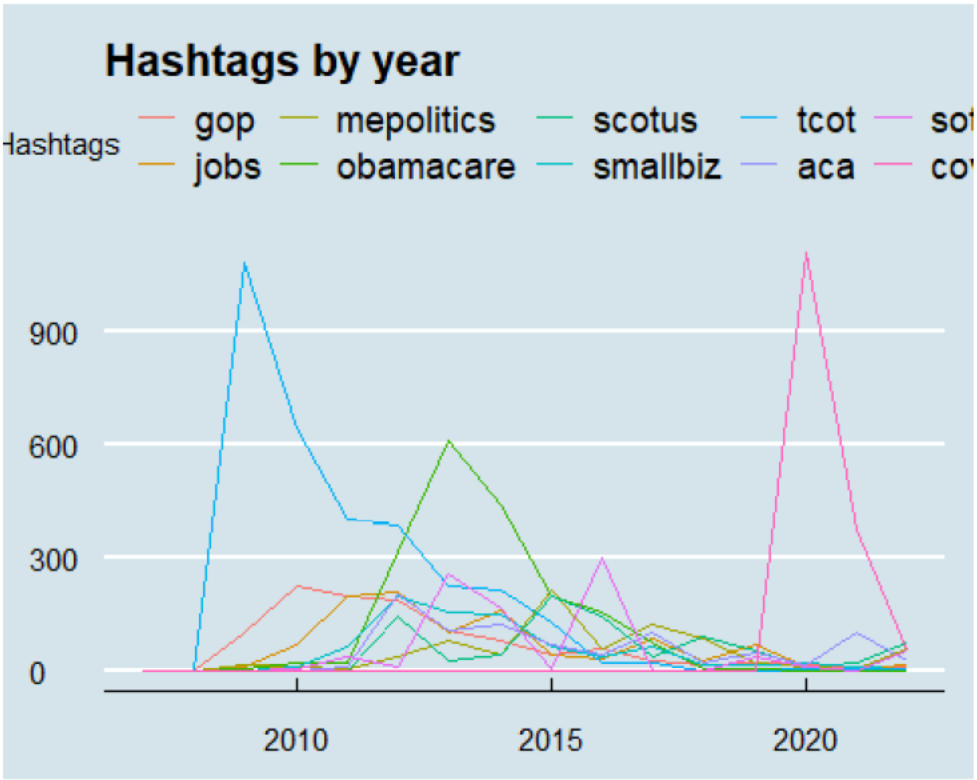
Creating DFM of Top Hashtags

```
toks_tweets<- tokens(corp, remove_punct = TRUE, remove_url = TRUE, remove_separators = TRUE, include_docvars = TRUE, what = "word") %>% tokens_keep(pattern = "#*",padding = FALSE)

hashtag_Top10<-textstat_frequency(dfmat_tweets,n = 10)

hashtag_10_list<-hashtag_Top10$feature

hash_toks<-tokens_select(toks_tweets, pattern = hashtag_10_list, selection = "keep")

hashtag_dfm<-dfm(hash_toks,remove_padding = TRUE)
```

converting DFM to usable data frame

```
# This converts the more complex DFM object into a data frame that can be more easily loaded into ggplot.
# however, it won't work if the sparsity of the DFM is too high.  To reduce sparsity the function dfm_trim(), aggregate, and selection can be used to reduce the empty rows to an acceptable size.

df_hashtag <- data.frame(convert(hashtag_dfm, to = "data.frame"), Year = hashtag_dfm$year)

# dropping tweet_id from df
df_hashtag<-df_hashtag[,2:12]
```

```r
#aggregating the count of features(the hashtags in this case) by year.
df_hashtag<-aggregate(df_hashtag,by = list(df_hashtag$Year),FUN = sum)

#dropping extra column
df_hashtag<-df_hashtag[,1:11]

#sanity check
#colnames(df_hashtag)[2:11]

#renaming grouping variable to year
df_hashtag<-df_hashtag %>% rename(Year=Group.1)


#converting df from a wide to a long format
df_hashtag<- df_hashtag %>%
  pivot_longer(names_to = "feature",cols = !Year,names_prefix = "X.",values_t
o = "frequency")

#creating plot
ggplot(df_hashtag, aes(x = Year, y = frequency, color = reorder(feature,!freq
uency)))+
  geom_line()+theme_economist()+labs(x = NULL, y = NULL, title = "Hashtags by
year", color = "Hashtags")
```



Hashtag Keyness - Before and After the Pandemic

```
tstat_key <- textstat_keyness(dfmat_tweets,
                              target = dfmat_tweets$date_of_tweet >= '2020-03
-01 00:00:00')
textplot_keyness(tstat_key,n=12,margin = 0.1, show_legend = FALSE)
```