final Project

Christoffer M. Kramer

26/11/2020

# Introduction

In this paper, I will examine how democratic and republican presidential and vice-presidential candidates have changed their rhetoric during election debates since 1960. This is done by text-mining debate transcripts from the *The Commission on Presidential Debates’* [website](https://www.debates.org/). By combining simple text-mining techniques, such as sentiment analysis, stop-word filtering, and term frequency - inverse document fequency, with visualizations, I’m gonna do an exploratory analysis of the debates. This is done with the aim of investigating, how the discourse during presidential debates have changed. This paper concludes by discussing how data-science and distant reading can be used as a starting point for a traditional anthropological discourse analysis.

**Additional information** \* *Libraries:* I’m gonna use the following libraries:

library(tidyverse) #Cleaning data  
library(tidytext) #Text-mining  
library(ggplot2) #Plotting  
library(ggwordcloud) #Plotting word clouds  
library(rvest) #Web scraping  
library(lubridate) #Cleaning Dates  
library(zoo) #Dealing with na's

\**License*: I contacted The Commission on Presidential Debates and asked for permission to use their transcripts. They informed me, that the transcripts are in the public domain.

* *Repository and metadata:* My scripts and data are stored on [this github repository](https://github.com/Digital-Methods-HASS/au590388_Christoffer_Kramer/tree/master/final_project). Relevant metadata are provided in the readme file.
* *figures:* For higher readability all figures and plots will be provided in an appendix.

# Web scraping

I need to get web-scrape each transcript since 1960. This is done in the code chunk below. For an explanation of this code see pp. X-X, FIXME, or [my github repository for that portfolio](https://github.com/Digital-Methods-HASS/au590388_Christoffer_Kramer/tree/master/learning_journal_and_assignments/week_44_webscrape).

# Scrape Debate function --------------------------------------------------  
scrape\_debates <- function(website) {  
 p\_html <- read\_html(website) %>%  
 html\_nodes("p") %>%  
 html\_text()  
  
 vect\_p\_html <- c(p\_html) #Save the output in a vector.  
}  
  
# Get and store links to debates ---------------------------------------------------  
link\_html <- read\_html("https://www.debates.org/voter-education/debate-transcripts/") %>%  
 html\_nodes("blockquote") %>%  
 html\_children() %>%  
 html\_nodes("a") %>%  
 html\_attr("href")  
  
vect\_link <- c(link\_html) #save the output in a vector

When I wrote the portfolio on web scraping, the transcripts for the 2020 debates had not yet been uploaded. Now that they have, I need to do some additional cleaning. I will, therefore, remove the hostname ("<https://www.debates.org/>) from the strings and replace them with the path to each transcript.

#Clean messy links  
vect\_link[1] <- "/voter-education/debate-transcripts/september-29-2020-debate-transcript/"  
vect\_link[2] <- "/voter-education/debate-transcripts/vice-presidential-debate-at-the-university-of-utah-in-salt-lake-city-utah/"  
vect\_link[3] <- "/voter-education/debate-transcripts/october-22-2020-debate-transcript/"

I have made two changes, since I wrote my last portfolio. Rather than creating a new object for each debate transcripts, I start by creating an empty tibble called *all\_debates\_raw* and row bind each debate transcript. I also removed the vector *debate\_names*, since the row-binding makes it redundant.

# Loop through the links and store the content ----------------------------  
all\_debates\_raw <- tibble()  
  
for (link in vect\_link[!is.na(vect\_link)]) {  
 date <- str\_extract(link, "[A-Za-z]+-\\d+-\\d+")  
  
 if(is.na(date)) {  
  
 date <- str\_extract(link, ".+")  
  
 } #end if  
  
 website <- paste0("https://www.debates.org/", link)  
 debate <- scrape\_debates(website) #create an object storing the transcript  
 debate <- tibble(line = 1:length(debate), text = debate, date = date)  
 all\_debates\_raw <- bind\_rows(all\_debates\_raw, debate) #row bind the transcript to all\_debates   
} #end loop

In order to make my data reproducible, I will save my results from the web-scraping in a csv-file. By doing this, I can make sure, that my data stays intact even if website debates.org ceases to exist. This also makes my results more reproducible, since researchers can reproduce my results without web-scraping.

write.csv(all\_debates\_raw, "../data/all\_debates\_raw.csv", row.names=FALSE) #Save as csv

# Data Cleaning and Data Wrangling

I have created a data set called “candidates\_since 1960.csv”, which contains a list of presidential and vice presidential candidates since 1960. This data set will be used for data wrangling. I will save it in an object called *candidates*. After that I transform *candidates* to a *tibble*. Lastly, I mutate all last names to uppercase. By writing all names in uppercase it will be easier to match their last name in the transcripts using a regex, since those are written in uppercase.

candidates <- read.csv(file = "../data/candidates\_since\_1960.csv", sep = ";") #Load csv  
candidates <- tibble(candidates) %>% #make data frame a tibble.  
 mutate(last\_name = toupper(last\_name)) #Make all names uppercase

I then create the tibble *all\_debates* which contains the csv file *all\_debates\_raw.csv*.

# Save transcripts as csv and create a tibble --------------------------------------------------------------  
all\_debates <- read.csv("../data/all\_debates\_raw.csv") #load csv  
all\_debates <- tibble(all\_debates) #make data frame a tibble

This data set still needs a lot of cleaning, in order to follow Hadley Wickhams principles for tidy data (Wickham, 2014). The data does not tell, who is speaking or what their party affiliation is. Moreover, the dates do not follow the datacarpetry’s recommendation of seperating year, month and day into seperate column (datacarpentry.org, n.d.).

## Cleaning dates

I will start the data wrangling by cleaning the dates. this is done by creating a function. It uses *str\_replace\_all* to replace a pattern with a new string.

#Functions that cleans names by replacing an existing string with a new string  
clean\_dates <- function(dataset, old\_pattern, new\_replacement) {  
 mutate\_if(dataset,  
 is.character,  
 str\_replace\_all, pattern = old\_pattern, replacement = new\_replacement)  
}

Now I need to find all dates, which are wrong. I can match a regex with the function *filter* from the *tidyverse* package (Wickham et al., 2019) and *grepl*. This regex is the same, as the one used in the previous loop. However with the *!* operator, it inverts the regex, thereby returning strings that do not follow the structure of “month-day-year” (e.g. “october-20-2020”). I then store the output in an object called *wrong names*.

#Find the wrong dates  
wrong\_dates <- all\_debates %>%   
 filter(!grepl("[A-Za-z]+-\\d+-\\d+", date)) %>%  
 select(date) %>%   
 unique()

There 4 wrong dates 3 of them don’t include a date, and the last is a link to the translations of the debate\_transcripts. Firstly I remove the translation page from *all\_debates*:

#Remove translations page from list of dates and the dataset  
all\_debates <- all\_debates[!(all\_debates$date == "/voter-education/debate-transcripts/2000-debate-transcripts-translations/"),]

Then I replace the wrong dates with the correct dates using my function *clean\_dates*.

#Replace wrong dates with correct dates  
all\_debates <- clean\_dates(all\_debates, old\_pattern = "voter-education/debate-transcripts/vice-presidential-debate-at-the-university-of-utah-in-salt-lake-city-utah/",  
 new\_replacement = "october-7-2020")  
  
all\_debates <- clean\_dates(all\_debates, old\_pattern ="/voter-education/debate-transcripts/2008-debate-transcript/",  
 new\_replacement = "september-26-2008")  
  
all\_debates <- clean\_dates(all\_debates, old\_pattern ="/voter-education/debate-transcripts/2008-debate-transcript-2/",  
 new\_replacement = "october-2-2008")

It appears to be working. However, I still need to separate day, month, and year, into different columns. This is done with the lubridate package (Grolemund & Wickham, 2011):

# Make "date" a proper date with different columns for day, month and year ------------------------------------------------  
all\_debates <- all\_debates %>%   
 mutate(date = mdy(date)) %>%   
 mutate(day = day(date),  
 month = month(date),  
 year = year(date)  
 )

Now I’m ready to differentiate between speakers.

## Who is speakning?

In the transcript Different speakers are marked at the start of a paragraph with their last name in uppercase followed by a semicolon. In older debates speakers are referred to as MR. or MS. , and some speakers have lowercase letters in their names (e.g. the moderator “McGee” ). In order to extract all speakers, I use a series str\_extract and a regexes to remove semicolon and MR. or MS., I then save the result as a tibble in an object called *last\_name*.

# Find all names and save as a tibble  
last\_name <- str\_extract(all\_debates$text, "^[A-Za-z]+:|^M[RS]\\..+:") %>% #Find name  
 str\_extract("[A-Za-z]+:") %>% #remove MR. and MS.  
 str\_extract("[A-Za-z]+") #Remove semicolon  
last\_name <- tibble(last\_name) #Make last\_name a tibble

Then I combine the columns of *all\_debates* and *last\_name*.

# bind column to all debates  
all\_debates <- cbind(all\_debates, last\_name)

Every time a speaking is not mentioned, it is the last named speaker who is speaking. Therefore by replacing all NA’s with the previous known value, it is possible to assign a speaker to each line. This is done with the function *na.locf* from the *zoo* package (Zeileis & Grothendieck, 2005).

all\_debates <- na.locf(all\_debates) #fill out every cell with the last known value.

Lastly, I transform all Last\_names to Uppercase so I can do a left\_join.

all\_debates <- all\_debates %>%   
 mutate(last\_name = toupper(last\_name)) # Make all last\_names uppercase

Then I left\_join *all\_debates* with *candidates* by column *last\_name* and *year*. Then I replace all NA’s with the string “not\_a\_candidate”, everywhere. This is done in order to follow Broman and Woo’s principles of data organization in spreadsheets, which, among other principles, states, that NA’s should not be included in spreadsheets (Broman & Woo: 4, 2018).

#Left\_join candidates and all\_debates by last\_name and year  
all\_debates <- left\_join(all\_debates, candidates, by = c("last\_name" = "last\_name", "year" = "year"))  
all\_debates[is.na(all\_debates)] <- "not\_a\_candidate" #fill NA's

Since each speaker is declared in the column *last\_name*. I’m going to use the same regex as previously for finding names, and then remove them from the transripts:

# Remove the names from the text  
 all\_debates <- all\_debates %>%  
 mutate(text = str\_remove(text, "^[A-Za-z]+:|^M[RS]\\..+:"))

Lastly, I will reorder the columns so *first\_name* and *last\_name* are placed in front of the text. This is purely an aestethic choice.

# Reorder columns  
all\_debates <- all\_debates %>%   
 relocate(last\_name, .before = text)  
#Reorder columns  
all\_debates <- all\_debates %>%   
 relocate(first\_name, .before = last\_name)

Now my data set follows a nice and tidy structure.

# Text-mining

I will do three types text-mining: *tf-idf*, *sentiment analysis*, and *stop word filtering*. I’m mainly inspired by Julia Silge and David Robinsons approach to text-mining in their book: “Text Mining with R: A Tidy Approach” (2020). Therefore, I’m gonna use the the tidytext package (Silge & Robinsong, 2016).

Since I’m gonna be making a lot of plots, I will make a function for saving my plots. This will make writing the code easier, and it will make it easier to combine all pdf files into one big appendix file, since they all have the same size.

save\_plot\_pdf <- function(name, path\_destination = "../plots/", width\_value = 8.26, height\_value = 11.69){  
 dev.copy(pdf, paste(path\_destination, name, ".pdf", sep = ""), width= width\_value, height= height\_value) #Units are inches  
 dev.off()  
}

I’m also gonna make a lot of word clouds. Therefore, I will make a custom function, which does this for me. Most parameters have a default value, which will make it a lot easier to plot, but still give me the opportunity to customize the plot.

plot\_wordclouds <- function(data\_set, label\_value = word, size\_value = n, max\_size = 10, color\_value = n, shape\_value ="diamond") {  
 ggplot(data\_set, aes(label = {{label\_value}}, size = {{size\_value}})) +  
 geom\_text\_wordcloud\_area(aes(color = {{color\_value}}), shape = shape\_value) +  
 scale\_size\_area(max\_size) +  
 theme\_minimal()   
 }

## Tokenizing

I will start out by creating a tibble called *my\_stop\_words*, which contains stopwords that are not included in the stopword lexicon. Then I tokenize the text by word. This is done with the **unnest\_tokens** function in the tidytext package (Silge & Robinson, 2016). This ensures that my data set still follows the tidy format, but now each row represents a word (Silge & Robinson, 2017). After that I’m using a regex to filter out digits, and then, using anti\_join, I’m filtering out rows that matches *my\_stop\_words*. The output is saved in the object *debate\_words*.

my\_stop\_words <- tibble(word = c("uh", "uhh")) #custom stop words  
  
debate\_words <- all\_debates %>%   
 unnest\_tokens(word, text) %>%   
 filter(!grepl("[[:digit:]]", word)) %>% #Remove digits  
 anti\_join(my\_stop\_words)

## Stop word filtering

Most of the debates uses a single right quotation mark, rather than an apostrophe. This causes problems, when using the anti\_join. Therefore, Luckily, I found an answer to this problem in [this stack overflow question](https://stackoverflow.com/questions/47209828/r-tidytext-stop-words-are-not-filtering-consistently-from-gutenbergr-downloads). The second line mutate single right quotation marks to apostrophes. Lastly I do an anti\_join in order to remove stop words.

stop\_words\_removed <- debate\_words %>%   
 mutate(word = gsub("\u2019", "'", word)) %>% #Make read it at unix  
 anti\_join(stop\_words)

Now that each stop word has been filtered out, I’m ready for my exploratory analysis.

### Plotting Stop word filtering

Let’s try filtering out the democrats and counting each word in each year, and then pipe it into my function *plot\_wordclouds*, which will be faceted by year,

stop\_words\_removed %>%   
 filter(grepl("[DR]", party)) %>%   
 count(year, party, word, sort = TRUE) %>%   
 group\_by(year, party) %>%   
 slice\_max(order\_by = n, n = 20) %>%   
 plot\_wordclouds() +  
 scale\_color\_gradientn(colors = c("darkgreen","blue","red")) +  
 facet\_wrap(~year + party) +  
 labs(title = "Fig. 1: Stop Word Filtering")

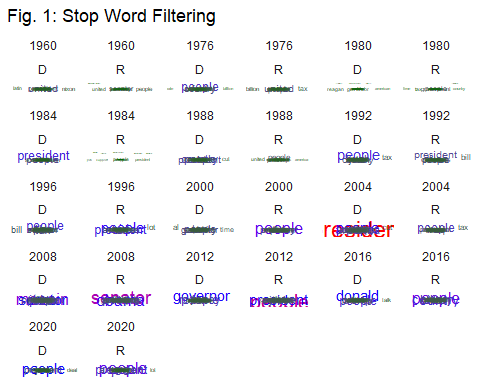


Fig. 1: Stop Word Filtering

save\_plot\_pdf("fig\_1")

Hm.. interestingly enough the parties’s primary talking points since 2008 appears to have been the opposite candidates. It is quite astonishing that the opposite candidate’s name appear so often, that otherwise common themes, such as national debt, jobs or budgets, are pushed aside. However, it might be a result of changes to the debate format, such as longer debates, which naturally would increase how often a candidate is mentioned. Let’s try using *term frequency - inverse document frequency*, which looks at the word frequency across all documents in order investigate whether this is a general phenomenon or the result of changes to the debate format.

## Term Frequency - Inverse Document Frequency

I make the tf-idf analysis by counting how often a word appears in each year and save the output in an object called *word\_count\_year*.

word\_count\_year <- debate\_words %>%   
 count(year, word, sort = TRUE)

Then I calculate the total amount of words each year, and save the result in an object called *total\_words\_year*.

total\_words\_year <- word\_count\_year %>%  
 group\_by(year) %>%   
 summarise(total = sum(n))

I then left join *total\_words\_year* and *word\_count\_year* and save the result in *word\_count\_year*.

word\_count\_year <- left\_join(word\_count\_year, total\_words\_year)

Lastly, I calculate tf-idf for each word in each year, by using the function *bind\_tf\_idf* from the tidytext package and save the output in an object called *word\_tf\_idf\_year*:

word\_tf\_idf\_year <- word\_count\_year %>%   
 bind\_tf\_idf(word, year, n)

## TF\_IDF plotting

Let’s plot it:

word\_tf\_idf\_year %>%  
 group\_by(year) %>%  
 slice\_max(order\_by = tf\_idf, n = 20) %>%  
 plot\_wordclouds(size\_value = tf\_idf, color\_value = tf\_idf) +  
 scale\_color\_gradientn(colors = c("darkgreen","blue","red")) +  
 facet\_wrap(~year, ncol = 3, scales = "free") +  
 labs(title = "Fig. 2: Tf-idf each year - Both Parties")

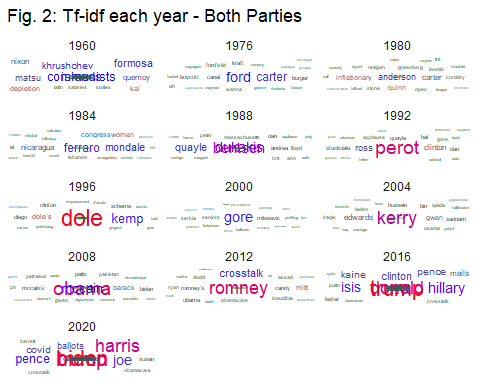


Fig. 2: TF-IDF each year - Both Parties

save\_plot\_pdf("fig\_2")

Hmm interesting, It appears that the candidates are some of the most common talking points across all years, and that this trend genereally started in 1996 with Bob Dole, and really took off in 2016 and 2020, which are really dominated by names.

## Sentiment analysis

Generally it appears that each party have been spending more time discussing the opposite candidate each year. This certainly points towards an increasing polarization, since I highly doubt that they are praising their opponent. But what about their sentiment, has it changed over the last couple of years, and does it differ between the parties? Let’s do a sentiment analysis on the debates since 1992.

I will start by removing word, which distorts the result. This is “trump” and “vice”. Trump is, in this case, a name, but is defined as a word with a positive sentiment in most sentiment dictionaries. Vice refers to vice-presidents, but it has a negative sentiment in most dictionaries. I will save these words in a tibble.

distortion\_words <- tibble(word = c("trump", "vice"))

Then I filter out all debates before 1992, then I anti-join the distortion words, then I inner-join the sentiment dictionary “afinn” and the dictionary “bing”:

sentiment\_debate\_words <- debate\_words %>%   
 filter (year >= 1992) %>%  
 anti\_join(distortion\_words) %>%   
 inner\_join(get\_sentiments("afinn")) %>%   
 inner\_join(get\_sentiments("bing"))

### Plot sentiment

I want to make a pie-chart to visualize the bing values. Many author reject pie-charts, but, as Claus Wike points out, they work well in showing simple fractions such as one-half, one-third, one-quarter and so on (Wilke, XXXX - FIXME), and since I only have two values, the pie-chart is a useful tool for visualizing how large a portion each sentiment makes up.

I make a function called *plot\_pie\_chart*, which takes a data set and creates a new column called *total* which is the sum of a column called *n*. Then it creates a column called *share*, which is *n* divided by *total*, and it show what share of the total amount of words each count makes up. Then I ggplot a *geom\_bar*, where y represents the share, and it is filled dependening on sentiment and then I use *coord\_polar* to make the bar chart round rather than rectangular. Lastly I use a blank theme, where the x-axis does not contain any text:

#Plot Pie chart function  
plot\_pie\_chart <- function(data\_set, fill\_value = sentiment){  
 data\_set %>%   
 mutate(total = sum(n)) %>%   
 mutate(share = n/total) %>%   
 ggplot() +   
 geom\_bar(aes(x = "", y = share, fill = {{fill\_value}}), stat = "identity") +  
 scale\_fill\_brewer(palette = "Set3") +  
 coord\_polar("y", start = 0) +  
 theme(axis.text.x = element\_blank())   
 }

Let’s start with the republican, because I need to summarize the result I will use the function *tally* instead of *count*:

#Republicans  
sentiment\_debate\_words %>%   
 filter(party == "R") %>%   
 group\_by(year, sentiment) %>%  
 tally() %>%  
 plot\_pie\_chart() +  
 facet\_wrap(~year, ncol = 7) +  
 labs(title = "Fig. 3: Republicans' sentiment by year - BING",  
 x = "",  
 y = "",  
 fill = "sentiments")

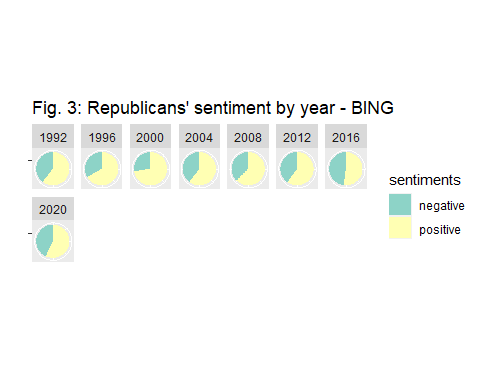


Fig. 3: Republicans’ sentiment for each year - BING

save\_plot\_pdf("fig\_3")

Interestingly enough 1996 and 2000 was some of the most positive sentiment for the republicans, and then it has become more negative, culminating in 2016, where almost half of all counted words had a negative sentiment. Let’s look at the democrats:

#Democrats  
sentiment\_debate\_words %>%  
 filter(party == "D") %>%   
 filter(year >= 1992) %>%   
 group\_by(year, sentiment) %>%   
 tally() %>%   
 plot\_pie\_chart() +  
 facet\_wrap(~year, ncol = 7) +  
 labs(title = "Fig. 4: Democrats sentiment by year - BING",  
 x = "",  
 y = "",  
 fill = "sentiments")

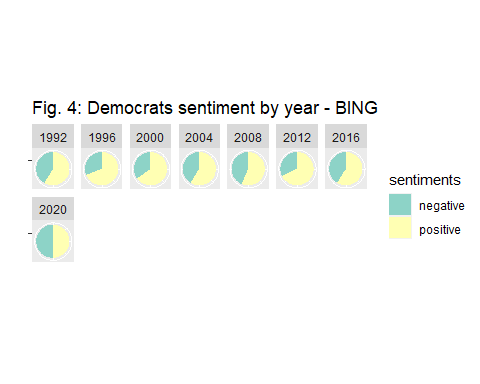


Fig. 4: Democrats’ sentiment for each year - BING

save\_plot\_pdf("fig\_4")

Apperantly democrats almost follow the same pattern, with 1996 and 2000 being postive, and then becoming more negative in 2004 and 2008. However, back in 2012, they appeared to be a lot more postive than republicans. This certainly indicates, that the 2012 debate should be studied in a qualitative analysis in order to determine, why this is the case. Moreover, democrats, had a very negative sentiment in 2020, where about half of the counted words was negative. Because of co-vid this is certainly not surprising, however, what is surprising, is that republicans was more negative in 2016 than in 2020.

## Combining Sentiment and wordclouds

Lastly, let’s get an overview of which words they use, how often they are used, and their sentiment. This can be achieved with a word cloud, where the size of each word is determined by the count, the color is determined by the sentiment (red = negative, green = postive) according to the “afinn” lexicon. Let’s start with the democrats:

sentiment\_debate\_words %>%  
 filter(grepl("[DR]", party)) %>%   
 count(year, word, party, value, sort = TRUE) %>%  
 group\_by(year, party) %>%   
 slice\_max(order\_by = n, n= 50) %>%   
 plot\_wordclouds(color\_value = value) +  
 scale\_color\_gradient(low = "red", high = "green") +  
 facet\_wrap(~year + party) +  
 labs(title = "Fig. 5: Democrats' sentiment and word count year")

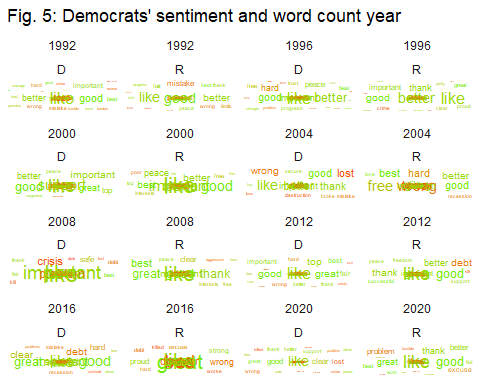


Fig. 5: Afinn Word cloud. The size of each word is the count, the color is the sentiment

save\_plot\_pdf("fig\_5")

Interestingly the same words tends to be used across all debates. Words such as “Like”, “important”, “safe” and “better” are often used as positive words. The word “like” is especially problematic here, since it can be an adjective, a verb, a preposition and an adverb, which changes its sentiment, this suggest that the debates are even more negative than our previous analysis showed. Moreover we see that words like “lost”, “debt”, “crisis”, “problem”, and “wrong” are often used as negative words. Moreover, we can see that especially 2020 contain a lot of red words which supports my previous analysis, which found that 2020 was an especially negative debate. Republicans tends to use the same positive words as democrats. However their negative words differ. They use words like “terror” “threat”, and “bad” a lot more. Moreover, even though they do use the word “crisis” it is much less used by the republicans compared to the democrats. Moreover, the republicans appeared to us a lot more negative words in 2016, compared to 2020. What is interesting, is how much overlap there actually is between both parties. They tend to use the same words a lot, and the only real difference in their use of loaded words is when we look at less common sentiments such as “terror”, which is used by republicans" and “crisis”, which is more commonly used by democrats.

# Conclusion

# Literature

(Wickham, 2014) (datacarpentry.org, n.d.): <https://datacarpentry.org/spreadsheets-socialsci/03-dates-as-data/index.html>

“Text Mining with R: A Tidy Approach” (2020).

Wilke, C. O. (XXXX - FIXME). “Chapter 10: Visualizing proportions” in Wilke, C. O. *Fundamentals of Data Visualization*, O’Reilly Media, Inc, URL: <https://clauswilke.com/dataviz/visualizing-proportions.html>

**Packages** Zeileis A, Grothendieck G (2005). “zoo: S3 Infrastructure for Regular and Irregular Time Series.” Journal of Statistical Software, 14(6), 1–27. doi: 10.18637/jss.v014.i06

Garrett Grolemund, Hadley Wickham (2011). Dates and Times Made Easy with lubridate. Journal of Statistical Software, 40(3), 1-25. URL <http://www.jstatsoft.org/v40/i03/>

Karl W. Broman & Kara H. Woo (2018) Data Organization in Spreadsheets, The American Statistician, 72:1, 2-10, DOI: 10.1080/00031305.2017.1375989

Silge J, Robinson D (2016). “tidytext: Text Mining and Analysis Using Tidy Data Principles in R.” JOSS, 1(3). doi: 10.21105/joss.00037, <http://dx.doi.org/10.21105/joss.00037>

Wickham H, et al. (2019). “Welcome to the tidyverse.” Journal of Open Source Software, 4(43), 1686. doi: 10.21105/joss.01686

Wickham H (2007). “Reshaping Data with the reshape Package.” Journal of Statistical Software, 21(12), 1–20. <http://www.jstatsoft.org/v21/i12/>

Fellows, Ian (n.d.). “wordcloud: Word Clouds”, located at CRAN here: <https://cran.r-project.org/web/packages/wordcloud/index.html>

# Ekstra