Final Exam

*Christoffer Mondrup Kramer*

*08-12-2020*

*Student number: 201704767*

*Cultural Data Science*

*Final exam*

*Aarhus University*

**Contents**

[Week 40: Shell 3](#_Toc58365763)

[Week 41: Start with R 7](#_Toc58365764)

[Week 43: Functions 10](#_Toc58365765)

[Question 1 10](#_Toc58365766)

[Question 2 11](#_Toc58365767)

[Question 3 13](#_Toc58365768)

[Assignment week 44: Web-Scraping 17](#_Toc58365769)

[Lightning Talk - Slides 25](#_Toc58365770)

[Final Project: Text-mining US Election Debates 26](#_Toc58365771)

[Introduction 27](#_Toc58365772)

[Web Scraping 27](#_Toc58365773)

[Data Cleaning 29](#_Toc58365774)

[Cleaning Dates 29](#_Toc58365775)

[Who is Speaking? 30](#_Toc58365776)

[Text-mining: Preparation 32](#_Toc58365777)

[Text-mining: Stop Word Filtering 32](#_Toc58365778)

[Plotting Stop Word Filtering 33](#_Toc58365779)

[Text-mining: Term Frequency-Inverse Document Frequency 34](#_Toc58365780)

[Text-mining: Sentiment Analysis 36](#_Toc58365781)

[Plotting Sentiments 37](#_Toc58365782)

[Conclusion 39](#_Toc58365783)

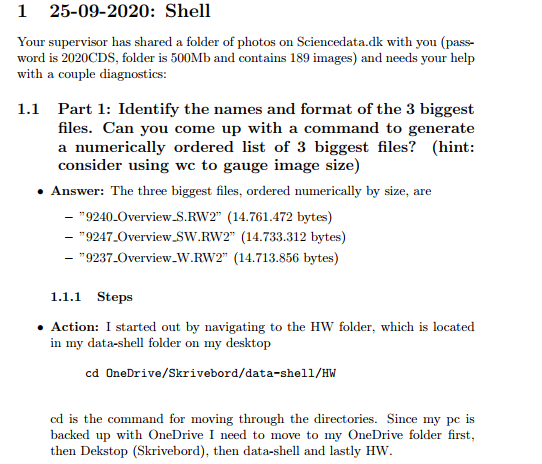
[Literature 40](#_Toc58365784)

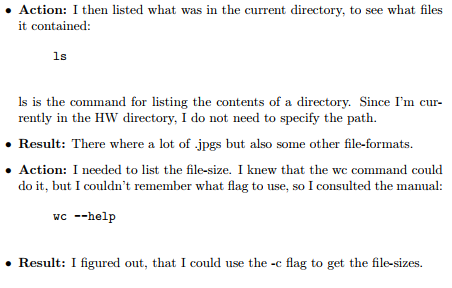
[Packages 40](#_Toc58365785)

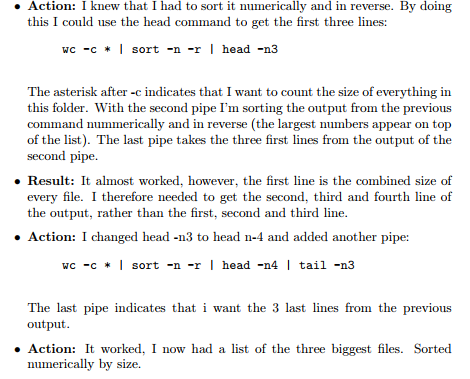
# Week 40: Shell

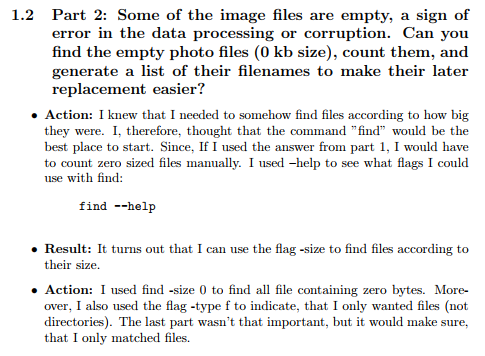
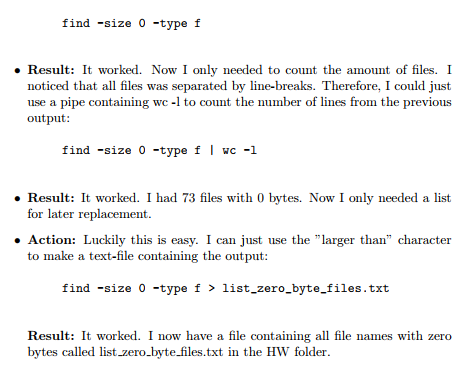
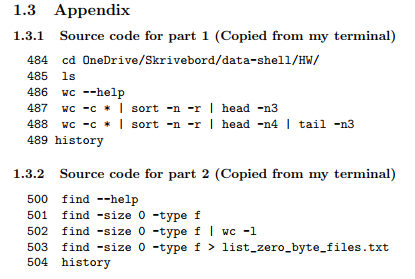
**Note:** Originally made in overleaf. Have taken screenshots of the portfolio, in order to keep the proper formatting.

**GitHub repository with pdf-file:** <https://github.com/Digital-Methods-HASS/au590388_Christoffer_Kramer/tree/master/learning_journal_and_assignments/week_40>





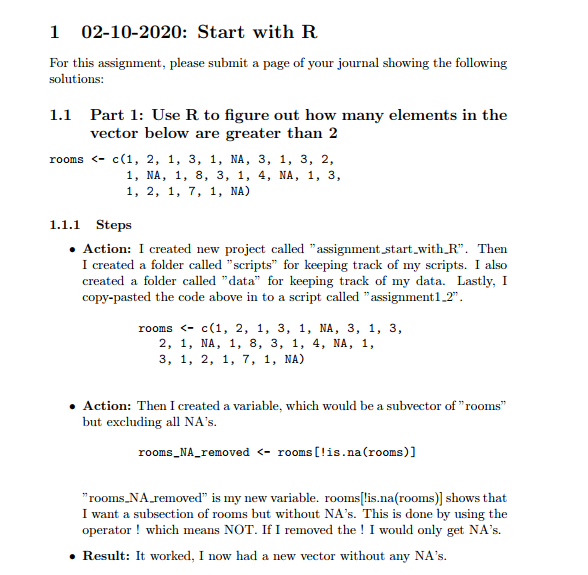


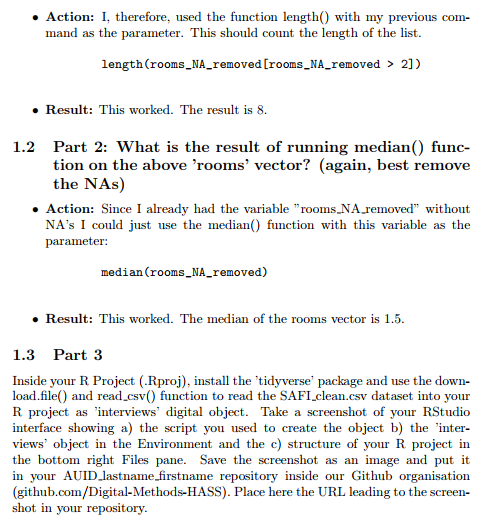
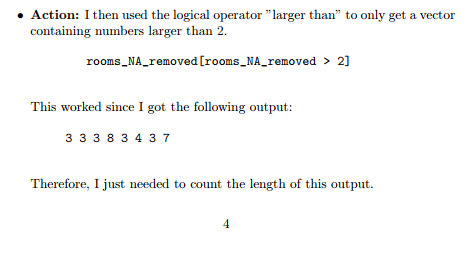
  

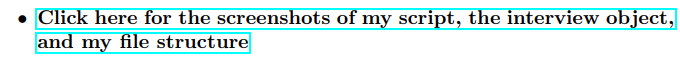
# Week 41: Start with R

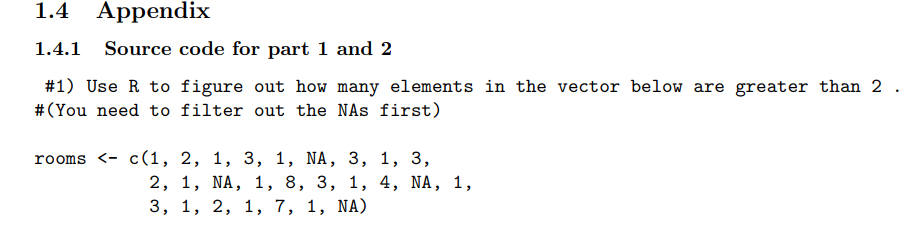
**Note:** Originally made in overleaf. Have taken screenshots of the portfolio, in order to keep the proper formatting.

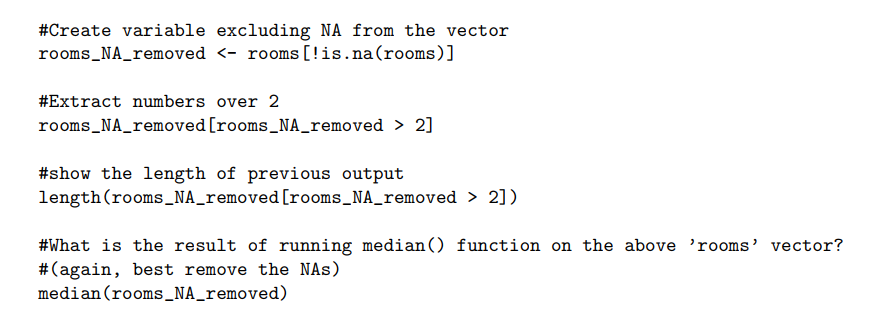
**GitHub repository:** <https://github.com/Digital-Methods-HASS/au590388_Christoffer_Kramer/tree/master/learning_journal_and_assignments/week_41>











# Week 43: Functions

Christoffer M. Kramer

21/10/2020

GitHub repository: <https://github.com/Digital-Methods-HASS/au590388_Christoffer_Kramer/tree/master/learning_journal_and_assignments/week_43_functions>

## Question 1

*Define a defensive function that calculates the Gross Domestic Product of a nation from the data available in the gapminder data set. Using that function, calculate the GDP of Denmark in the following years: 1967, 1977, 1987, 1997, 2007, and 2017.*

First, I load the relevant libraries and disable scientific notation.

library(gapminder)  
library(tidyverse)

## Warning: package 'tidyverse' was built under R version 4.0.3

## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.2 v purrr 0.3.4  
## v tibble 3.0.3 v dplyr 1.0.2  
## v tidyr 1.1.2 v stringr 1.4.0  
## v readr 1.3.1 v forcats 0.5.0

## Warning: package 'ggplot2' was built under R version 4.0.3

## Warning: package 'tidyr' was built under R version 4.0.3

## -- Conflicts ------------------------------------------ tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()

options(scipen = 999) #disable scientific notation

Then I create a function to calculate gdp. This solution is found through [software carpentry’s tutorial](http://swcarpentry.github.io/r-novice-gapminder/10-functions/index.html). However, I have changed the names, since I find longer, but more descriptive names easier to understand. It calculates GDP by sub setting the Gapminder data from the parameters “year” and “country”, then it multiplies “gdpPerCap” with “pop” in that subset. The default parameters are Null, and it is defensive since it starts out by checking if the parameters are NULL. If no parameters are given, it will work on the whole data set. If only one parameter is given it will create a subset based on that parameter. Lastly, it creates a new column called GDP.

calcGDP <- function(dataset, year=NULL, country=NULL) {  
 if(!is.null(year)) { #If year is not null  
 dataset <- dataset[dataset$year %in% year,] #subset the dataset by year  
 }  
 if (!is.null(country)) { #if country is not null  
 dataset <- dataset[dataset$country %in% country,] #subset the dataset by country  
 }  
   
 gdp\_result <- dataset$pop \* dataset$gdpPercap #calculate gdp by multiplying GDP per capita with the total population.  
 #Save the result in the object "gpd\_result"  
   
 new\_column <- cbind(dataset, gdp=gdp\_result) #create a new column called GDP which contains the object "gdp\_result"   
 return(new\_column)  
}

Lastly, I just need to write the relevant values in the parameters. Since I don’t want to repeat a lot of code, I create a vector called “years”, which contains all the years, and use it as a parameter

years <- c(1967, 1977, 1987, 1997, 2007, 2017) #Create a vector containing all the years  
calcGDP(gapminder, year = years, country = "Denmark") #calculate the gdp for each year in the vector

## country continent year lifeExp pop gdpPercap gdp  
## 1 Denmark Europe 1967 72.960 4838800 15937.21 77116977700  
## 2 Denmark Europe 1977 74.690 5088419 20422.90 103920280028  
## 3 Denmark Europe 1987 74.800 5127024 25116.18 128771236166  
## 4 Denmark Europe 1997 76.110 5283663 29804.35 157476118456  
## 5 Denmark Europe 2007 78.332 5468120 35278.42 192906627081

Since there aren’t any data from 2017 the function only returns 5 rows,

## Question 2

*Write a script that loops over each country in the gapminder dataset, tests whether the country starts with a ‘B’ and print out whether the life expectancy is smaller than 50, between 50 and 70, or greater than 70.*

I start by creating an object called “mean\_life\_country”. This object groups by country. It then filters out all rows, where the country does not start with the letter B. Then it summarizes by calculating the average life expectancy for each country and creates the column “mean\_life”.

mean\_life\_country <- gapminder %>% #Create a new object called "mean\_life\_country"  
 group\_by(country) %>% #group by country  
 filter(str\_detect(country, "^B")) %>% #filter out rows where the first letter of the column "country" is not B  
 summarise(mean\_life = mean(lifeExp)) #create a new column called "mean\_life"

## `summarise()` ungrouping output (override with `.groups` argument)

I then create a new csv document which contains the previous object. Then I set the value of the object “mean\_life\_country” to be the csv file.

write.csv(mean\_life\_country, "mean\_life\_country.csv") #create a csv document called "mean\_life\_country.csv  
mean\_life\_country <- read.csv("mean\_life\_country.csv") #let the object "mean\_life\_country" contain the csv file

I then loop through each country in my csv file. I get the value of the column “mean\_life” and store it in a new object called “mean\_life\_i”. Lastly, I use if-else statements to check if the value of “mean\_life\_i” is smaller than 50, between 50 and 70, or above 70. It then prints out the value of the country (i) and the value of “mean\_life\_i”.

for (i in mean\_life\_country$country) { #for every country in "mean\_life\_country"  
   
 mean\_life\_i <- mean\_life\_country[mean\_life\_country$country == i, "mean\_life"] #create an object called "mean\_life\_i, which is a subvector of "mean\_life\_country"   
   
 if(mean\_life\_i < 50) { #if "mean\_life\_i i smaller than 50  
 cat("The life expectancy of", i, "is smaller than 50", "(",mean\_life\_i,")\n") #print this out  
 }  
 else if (mean\_life\_i > 50 && mean\_life\_i < 70) { #if "mean\_life\_i" is larger than 50 and smaller than 70  
 cat("The life expectancy of", i, "is between 50 and 70", "(",mean\_life\_i,")\n") #print this out  
   
 } else { #if "mean\_life\_i" is larger than 70  
 cat("The life expectancy of", i, "is larger than 70", "(",mean\_life\_i,")\n") #print this out  
   
 }  
  
}

## The life expectancy of Bahrain is between 50 and 70 ( 65.60567 )  
## The life expectancy of Bangladesh is smaller than 50 ( 49.83408 )  
## The life expectancy of Belgium is larger than 70 ( 73.64175 )  
## The life expectancy of Benin is smaller than 50 ( 48.77992 )  
## The life expectancy of Bolivia is between 50 and 70 ( 52.50458 )  
## The life expectancy of Bosnia and Herzegovina is between 50 and 70 ( 67.70783 )  
## The life expectancy of Botswana is between 50 and 70 ( 54.5975 )  
## The life expectancy of Brazil is between 50 and 70 ( 62.2395 )  
## The life expectancy of Bulgaria is between 50 and 70 ( 69.74375 )  
## The life expectancy of Burkina Faso is smaller than 50 ( 44.694 )  
## The life expectancy of Burundi is smaller than 50 ( 44.81733 )

## Question 3

*Write a script that loops over each country in the gapminder dataset, tests whether the country starts with a ‘M’ and graphs life expectancy against time (using plot() function) as a line graph if the mean life expectancy is under 50 years.*

I use same approach as above. But now I need to also keep the years. I, therefore, create two csv files.

The first csv file (country\_m) filters out all countries, where the first letter is “M”. This will be used for plotting.

country\_m <- gapminder %>% #Create a new object called "country\_m"  
 group\_by(country) %>% #group by country  
 filter(str\_detect(country, "^M")) #filter out rows where the first letter of the column "country" is not M  
  
 write.csv(country\_m, "country\_m.csv") #create a csv document called "country\_m.csv  
 country\_m <- read.csv("country\_m.csv") #let the object "country\_m" contain the csv file

The second csv file is a summary of the average life expectancy of each country (based on the previous csv file), with column a called “mean\_life”. This will be used in order to check whether the average life expectancy is under 50.

mean\_life\_country\_m <- country\_m %>% #Create a new object called "mean\_life\_country\_m"  
 group\_by(country) %>% #group by country  
 summarise(mean\_life = mean(lifeExp)) #create a new column called "mean\_life" with the average life expectancy

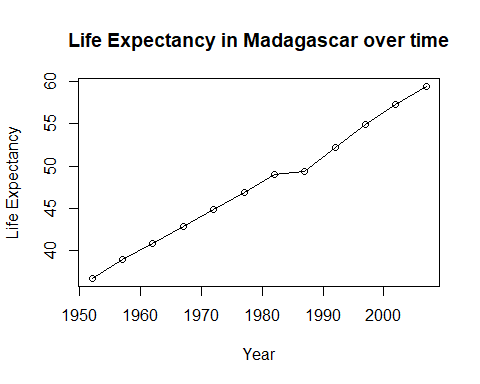
## `summarise()` ungrouping output (override with `.groups` argument)

write.csv(mean\_life\_country\_m, "mean\_life\_country\_m.csv") #create a csv file called "mean\_life\_country.csv  
 mean\_life\_country\_m <- read.csv("mean\_life\_country\_m.csv") #let the object "mean\_life\_country" contain the csv file

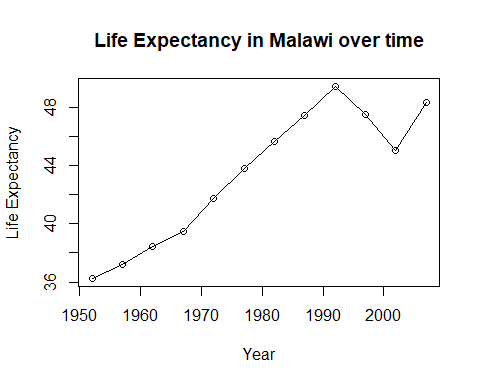
I then create a loop, which essentially works as the previous loop. It loops over each country in the csv file “mean\_life\_country\_m”. It then creates an object called “mean\_life\_i” which is a sub vector of “mean\_life\_m” containing the value of the column “mean\_life”. It then checks whether or not ”mean\_life\_i" is below 50. If this is true it plots out the life expectancy for each year.

for (i in mean\_life\_country\_m$country) { #for every country in "mean\_life\_country"  
   
 mean\_life\_i <- mean\_life\_country\_m[mean\_life\_country\_m$country == i, "mean\_life"] #create an object called "mean\_life\_i, which is a subvector of "mean\_life\_country"   
   
 if(mean\_life\_i < 50) { #if "mean\_life\_i is smaller than 50  
 cat("The life expectancy of", i, "is smaller than 50", "(",mean\_life\_i,") plotting life expectancy\n") #print this out  
   
 with(subset(country\_m, country==i), #with a subset of data from country\_m  
 plot(year, lifeExp, #plot year on x axis and life lifeExp on the y axis  
 type="o", #Make it at line and point plot  
 main = paste("Life Expectancy in", i, "over time"), #Create Title  
 ylab = "Life Expectancy", #Rename y axis  
 xlab = "Year" #Rename x axis  
 ) # end plot  
 ) # end with  
 }  
}

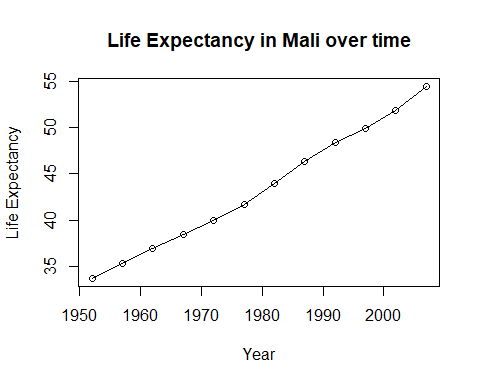
## The life expectancy of Madagascar is smaller than 50 ( 47.77058 ) plotting life expectancy



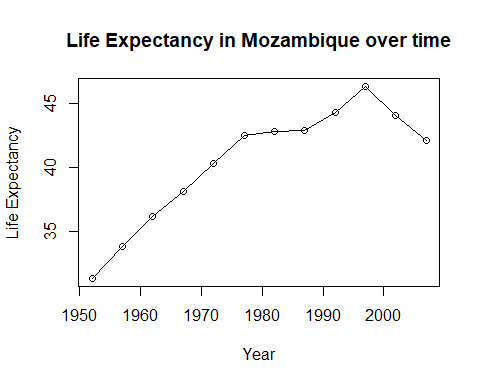
## The life expectancy of Malawi is smaller than 50 ( 43.35158 ) plotting life expectancy



## The life expectancy of Mali is smaller than 50 ( 43.4135 ) plotting life expectancy



## The life expectancy of Mozambique is smaller than 50 ( 40.3795 ) plotting life expectancy



# Assignment week 44: Web-Scraping

Christoffer M. Kramer

30/10/2020

My repo: <https://github.com/Digital-Methods-HASS/au590388_Christoffer_Kramer/tree/master/learning_journal_and_assignments/week_44_webscrape>

**Take the repository at** [**https://github.com/adivea/KilledbyPolice2020.git**](https://github.com/adivea/KilledbyPolice2020.git) **and depending on your familiarity with R, either**

* *adapt the web scraping example to scrape homicide data from FBI site and produce a meaningful report on how homicide trends evolve around US in relation to this urban unrest*
* *use the rvest library to scrape data of your interest (football statistics in Wikipedia? global population by country in* [*https://www.worldometers.info/world-population/population-by-country/*](https://www.worldometers.info/world-population/population-by-country/) *)*
* *produce data visualisations that shed light on another interesting aspect of the police killing data*

I will do the second item on the list. I will scrape transcripts of presidential debates, since these will be useful for my final project.

I start by loading the relevant libraries:

library(rvest)

## Loading required package: xml2

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(tidyr)

## Warning: package 'tidyr' was built under R version 4.0.3

library(stringr)  
library(janitor)

## Warning: package 'janitor' was built under R version 4.0.3

##   
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':  
##   
## chisq.test, fisher.test

library(tidyverse)

## Warning: package 'tidyverse' was built under R version 4.0.3

## -- Attaching packages ------------------------------------------------------------------------------------------------ tidyverse 1.3.0 --

## v ggplot2 3.3.2 v purrr 0.3.4  
## v tibble 3.0.3 v forcats 0.5.0  
## v readr 1.3.1

## Warning: package 'ggplot2' was built under R version 4.0.3

## -- Conflicts --------------------------------------------------------------------------------------------------- tidyverse\_conflicts() --  
## x dplyr::filter() masks stats::filter()  
## x readr::guess\_encoding() masks rvest::guess\_encoding()  
## x dplyr::lag() masks stats::lag()  
## x purrr::pluck() masks rvest::pluck()

library(tidytext)

## Warning: package 'tidytext' was built under R version 4.0.3

library(ggplot2)

Then I create a function for web scraping. This makes it possible to automate the web scraping process. The parameter is the website I wish to scrape. The function reads the html elements of the website. It only reads the “p” elements, since this is where the text is located. It then parses the elements as text. Lastly, it saves the output in a vector, which is useful for text-mining and data-cleaning.

scrape\_debates <- function(website) { #Create a function called scrape\_debates  
 p\_html <- read\_html(website) %>% #Create an object which contains the parameter "website" and is read as a html file  
 html\_nodes("p") %>% #Grab all <p> elements in the html file.  
 html\_text() #Parse the result as text.  
   
vect\_p\_html <- c(p\_html) #Save the output in a vector.  
}

Let’s test it:

head(scrape\_debates("https://www.debates.org/voter-education/debate-transcripts/2008-debate-transcript/"))

## [1] "September 26, 2008"   
## [2] "The First McCain-Obama Presidential Debate"   
## [3] "FIRST PRESIDENTIAL CANDIDATES’ DEBATETHE UNIVERSITY OF MISSISSIPPI, OXFORD MISSISSIPPI"  
## [4] "SPEAKERS:"   
## [5] "U.S. SENATOR JOHN MCCAIN (AZ)REPUBLICAN PRESIDENTIAL NOMINEE"   
## [6] "U. S. SENATOR BARACK OBAMA (IL)DEMOCRATIC PRESIDENTIAL NOMINEE"

tail(scrape\_debates("https://www.debates.org/voter-education/debate-transcripts/2008-debate-transcript/"))

## [1] "For now, from Oxford, Mississippi, thank you, senators, both. I’m Jim Lehrer. Thank you, and good night."  
## [2] "(APPLAUSE)"   
## [3] "END"   
## [4] "Transcription by: CQ Transcriptions/Morningside"   
## [5] "Â "   
## [6] "© COPYRIGHT 2020 THE COMMISSION ON PRESIDENTIAL DEBATES. ALL RIGHTS RESERVED."

It works, but I still have 40 transcripts that I need to scrape. I will, therefore, use a loop to automate the process.

The links to all transcripts are located here: <https://www.debates.org/voter-education/debate-transcripts/> Therefore, I need to grab all the relevant links (without getting links from the nav-bar).

Luckily the links follow an easy structure. Every link is contained within an “a” element, which is inside of of “p” element, which inside a “blockqoute” element. So, by navigating to the “blockqoute” element, I can grab the “child” (which is “p”) and then navigate to the relevant “a” element and then select all a elements that contain a link (a href attribute).

link\_html <- read\_html("https://www.debates.org/voter-education/debate-transcripts/") %>%   
 html\_nodes("blockquote") %>% #Grab the blockqoute elements  
 html\_children() %>% #Navigate to the blockqoute's children.  
 html\_node("a") %>% #Grab <a> elements  
 html\_attr("href") #Grab <a> elements which contains a hyperlink.  
  
vect\_link <- c(link\_html) #save the output in a vector

Let’s see if it works.

vect\_link

## [1] "https://www.debates.org/voter-education/debate-transcripts/september-29-2020-debate-transcript/"   
## [2] "http://debates.org/voter-education/debate-transcripts/vice-presidential-debate-at-the-university-of-utah-in-salt-lake-city-utah/"  
## [3] "https://www.debates.org/voter-education/debate-transcripts/october-22-2020-debate-transcript/"   
## [4] "/voter-education/debate-transcripts/september-26-2016-debate-transcript/"   
## [5] "/voter-education/debate-transcripts/october-4-2016-debate-transcript/"   
## [6] "/voter-education/debate-transcripts/october-9-2016-debate-transcript/"   
## [7] "/voter-education/debate-transcripts/october-19-2016-debate-transcript/"   
## [8] "/voter-education/debate-transcripts/october-3-2012-debate-transcript/"   
## [9] "/voter-education/debate-transcripts/october-11-2012-the-biden-romney-vice-presidential-debate/"   
## [10] "/voter-education/debate-transcripts/october-16-2012-the-second-obama-romney-presidential-debate/"   
## [11] "/voter-education/debate-transcripts/october-22-2012-the-third-obama-romney-presidential-debate/"   
## [12] "/voter-education/debate-transcripts/2008-debate-transcript/"   
## [13] "/voter-education/debate-transcripts/2008-debate-transcript-2/"   
## [14] "/voter-education/debate-transcripts/october-7-2008-debate-transcrip/"   
## [15] "/voter-education/debate-transcripts/october-15-2008-debate-transcript/"   
## [16] "/voter-education/debate-transcripts/october-13-2004-debate-transcript/"   
## [17] "/voter-education/debate-transcripts/october-8-2004-debate-transcript/"   
## [18] "/voter-education/debate-transcripts/october-5-2004-transcript/"   
## [19] "/voter-education/debate-transcripts/september-30-2004-debate-transcript/"   
## [20] "/voter-education/debate-transcripts/october-3-2000-transcript/"   
## [21] "/voter-education/debate-transcripts/october-5-2000-debate-transcript/"   
## [22] "/voter-education/debate-transcripts/october-11-2000-debate-transcript/"   
## [23] "/voter-education/debate-transcripts/october-17-2000-debate-transcript/"   
## [24] "/voter-education/debate-transcripts/2000-debate-transcripts-translations/"   
## [25] "/voter-education/debate-transcripts/october-6-1996-debate-transcript/"   
## [26] "/voter-education/debate-transcripts/october-9-1996-debate-transcript/"   
## [27] "/voter-education/debate-transcripts/october-16-1996-debate-transcript/"   
## [28] NA   
## [29] "/voter-education/debate-transcripts/october-11-1992-first-half-debate-transcript/"   
## [30] "/voter-education/debate-transcripts/october-13-1992-debate-transcript/"   
## [31] NA   
## [32] "/voter-education/debate-transcripts/october-15-1992-first-half-debate-transcript/"   
## [33] "/voter-education/debate-transcripts/october-19-1992-debate-transcript/"   
## [34] "/voter-education/debate-transcripts/september-25-1988-debate-transcript/"   
## [35] "/voter-education/debate-transcripts/october-5-1988-debate-transcripts/"   
## [36] "/voter-education/debate-transcripts/october-13-1988-debate-transcript/"   
## [37] "/voter-education/debate-transcripts/october-7-1984-debate-transcript/"   
## [38] "/voter-education/debate-transcripts/october-11-1984-debate-transcript/"   
## [39] "/voter-education/debate-transcripts/october-21-1984-debate-transcript/"   
## [40] "/voter-education/debate-transcripts/september-21-1980-debate-transcript/"   
## [41] "/voter-education/debate-transcripts/october-28-1980-debate-transcript/"   
## [42] "/voter-education/debate-transcripts/september-23-1976-debate-transcript/"   
## [43] "/voter-education/debate-transcripts/september-26-1960-debate-transcript/"   
## [44] "/voter-education/debate-transcripts/october-7-1960-debate-transcript/"   
## [45] "/voter-education/debate-transcripts/october-13-1960-debate-transcript/"   
## [46] "/voter-education/debate-transcripts/october-21-1960-debate-transcript/"

It works. Every element in this vector is a link to a relevant debate transcript. There are two NA’s, which (I believe) is from two “p” elements, that does not contain any “a” elements. This is easily dealt with and is, therefore, not a problem.

Now I just need to automate the process. This is done with a loop which will create an object for each transcript. The loop will not go through NA values. I start by creating an object called “debate\_names”, this will contain the name of each debate object. Then I loop through each element in my previous vector “vect\_link”. I then create an object called “name”. This object contains a substring from the current link. This is used for naming the objects. If the link does not match the regex, it will just contain the full link. Then I create an object called website. It contains the first part of the URL and the current link which are concatenated. Then I push the name into my empty vector (debate\_names). I then create an object called “debate”. This contains the scraped data. The debate object is then transformed to a tibble, which is easier to text-mine. Lastly, I create a unique object with the function assign(). This object has the same name as the name in the vector “debate\_names”, and it contains the value of the object “debate”.

debate\_names = NULL #Create an empty object  
  
for (link in vect\_link[!is.na(vect\_link)]) { #For every link in the vector "vect\_link" (Na excluded")  
   
 name <- str\_extract(link, "[A-Za-z]+-\\d+-\\d+") #Create an object called name, set the value to be the regex match.  
   
 if(is.na(name)) { #If name is NA  
 name <- str\_extract(link, ".+") #Set name's value to be the same as the current link  
 }  
  
 debate\_names = append(debate\_names, paste("debate\_",name, sep = "")) #Add the object "name" to the vector "debate\_names"  
   
 website <- paste0("https://www.debates.org/", link) #Create an object called "website".  
 debate <- scrape\_debates(website) #Scrape data from the website and save it in an object called "debate"  
 debate <- tibble(line = 1:length(debate), text = debate) #Make "debate" a tibble, so it can be text-mined  
 assign(paste("debate\_", name, sep = ""), debate) #Create an object with the same name as the element in "debate\_names" which contains the value of debate.  
   
  
}

By now I should have the name for each object in the vector “debate\_names”, let’s check the vector out.

debate\_names

## [1] "debate\_september-29-2020"   
## [2] "debate\_/voter-education/debate-transcripts/vice-presidential-debate-at-the-university-of-utah-in-salt-lake-city-utah/"  
## [3] "debate\_october-22-2020"   
## [4] "debate\_september-26-2016"   
## [5] "debate\_october-4-2016"   
## [6] "debate\_october-9-2016"   
## [7] "debate\_october-19-2016"   
## [8] "debate\_october-3-2012"   
## [9] "debate\_october-11-2012"   
## [10] "debate\_october-16-2012"   
## [11] "debate\_october-22-2012"   
## [12] "debate\_/voter-education/debate-transcripts/2008-debate-transcript/"   
## [13] "debate\_/voter-education/debate-transcripts/2008-debate-transcript-2/"   
## [14] "debate\_october-7-2008"   
## [15] "debate\_october-15-2008"   
## [16] "debate\_october-13-2004"   
## [17] "debate\_october-8-2004"   
## [18] "debate\_october-5-2004"   
## [19] "debate\_september-30-2004"   
## [20] "debate\_october-3-2000"   
## [21] "debate\_october-5-2000"   
## [22] "debate\_october-11-2000"   
## [23] "debate\_october-17-2000"   
## [24] "debate\_/voter-education/debate-transcripts/2000-debate-transcripts-translations/"   
## [25] "debate\_october-6-1996"   
## [26] "debate\_october-9-1996"   
## [27] "debate\_october-16-1996"   
## [28] "debate\_october-11-1992"   
## [29] "debate\_october-13-1992"   
## [30] "debate\_october-15-1992"   
## [31] "debate\_october-19-1992"   
## [32] "debate\_september-25-1988"   
## [33] "debate\_october-5-1988"   
## [34] "debate\_october-13-1988"   
## [35] "debate\_october-7-1984"   
## [36] "debate\_october-11-1984"   
## [37] "debate\_october-21-1984"   
## [38] "debate\_september-21-1980"   
## [39] "debate\_october-28-1980"   
## [40] "debate\_september-23-1976"   
## [41] "debate\_september-26-1960"   
## [42] "debate\_october-7-1960"   
## [43] "debate\_october-13-1960"   
## [44] "debate\_october-21-1960"

It appears to be working. Now I should be able to access each debate by just typing the name of one of the objects above.

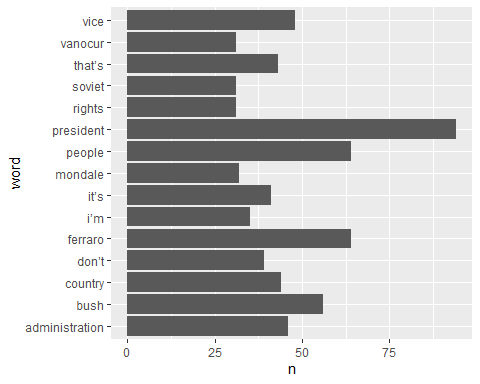
`debate\_october-11-1992`

## # A tibble: 105 x 2  
## line text   
## <int> <chr>   
## 1 1 "\nOctober 11, 1992\n"   
## 2 2 "The First Clinton-Bush-Perot Presidential Debate(First Half of Debate~  
## 3 3 "This is a transcript of the first half of the first presidential deba~  
## 4 4 "Â "   
## 5 5 "LEHRER: Good evening, and welcome to the first of 3 debates among the~  
## 6 6 "PEROT: I think the principal that separates me is that 5 and a half m~  
## 7 7 "LEHRER: Governor Clinton, a one minute response."   
## 8 8 "CLINTON: The most important distinction in this campaign is that I re~  
## 9 9 "LEHRER: President Bush, one minute response, sir."   
## 10 10 "PRESIDENT BUSH: Well, I think one thing that distinguishes is experie~  
## # ... with 95 more rows

It works! Now each debate is scraped, saved in an object and is in a dataframe. Let’s do a very simple text-analysis on one of the debates. First, I need to get a list of stop words, which is included in the tidytext package. Then I tokenize the debate by word so I can do an analysis. I do this with the function unnest\_tokens. I then remove all stop words from the debate with anti\_join(). Then I count each word and sort it. Then I filter out all words that does not appear more than 30 times. Lastly, I pipe it directly in to a ggplot:

data("stop\_words")#Get all stopwords  
  
 `debate\_october-11-1984` %>% #Chose the debate of october 11th. 1984.  
 unnest\_tokens(word, text) %>% #Tokenize by word  
 anti\_join(stop\_words) %>% #Remove stop words  
 count(word, sort = TRUE) %>% #Count each word and sort the result.  
 filter(n > 30) %>% #Remove words, that does not appear more than 30 times  
 ggplot(aes(word, n)) + #Make a ggplot  
 geom\_col() + #Make it a bar chart where the height represents the value  
 coord\_flip() #Flip the bar chart

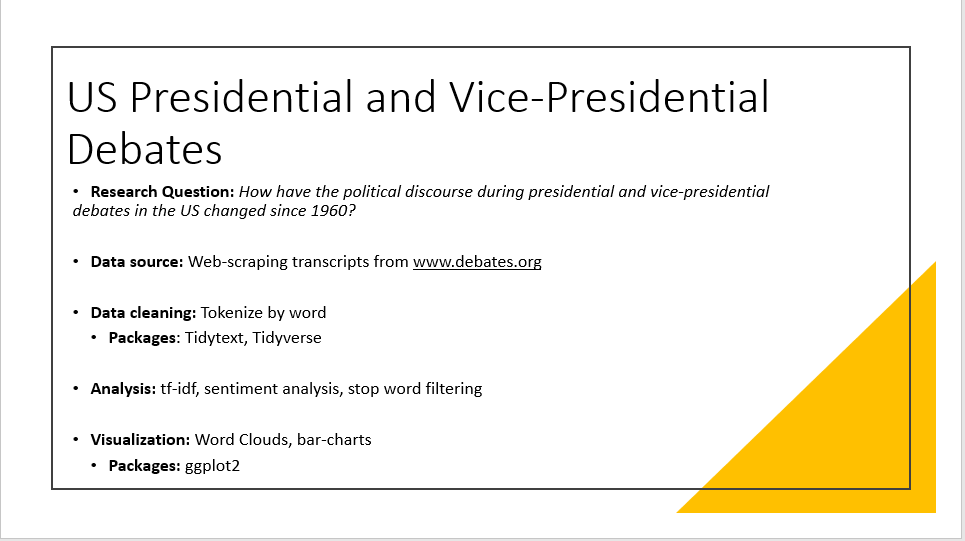
## Joining, by = "word"



As you can see, there is still a lot of cleaning to be done, but it is a great start.

# Lightning Talk - Slides

30-11-2020





# Final Project: Text-mining US Election Debates

*Christoffer Mondrup Kramer*

*08-12-2020*

*Student number: 201704767*

*Cultural Data Science*

*Final exam*

*Aarhus University*

***Words****: 1.899*

Practical Information

* *Repository:* <https://github.com/Digital-Methods-HASS/au590388_Christoffer_Kramer/tree/master/final_project>
* *Figures:* Also provided in the appendix.
* *Data source:* <https://www.debates.org/voter-education/debate-transcripts/>
* *License*:
  + Transcripts: In the public domain.
  + My repository: Licensed under *creative commons*

## Introduction

Ever since the 2016 presidential election, which ended with Donald Trump becoming president, American politics and the American populace have become more polarized than ever. However, Donald Trump’s victory was not a sudden event. It was the culmination of increasing political polarization in the US, which has been documented by countless studies from the Pew research center (pewresearch.org, n.d.). But can this polarization also be seen in election debates? I’m going to do an exploratory analysis of the debates to answer this question. This is done with simple text-mining techniques, such as sentiment analysis, stop-word filtering, term frequency-inverse document frequency, and data-visualizations.

This analysis is only preliminary, and I will conclude this paper by suggesting, how polarization among presidential and vice-presidential candidates can be further investigated based on the results.

## Web Scraping

I need to web-scrape each transcript since 1960. This is done in the code chunk below. For an explanation of this code see pp. 17-24 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 transcript, 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

To make my data reproducible, I will save my results from the web-scraping in a CSV-file. By doing this, I can ensure, that my data stays intact even if the website [www.debates.org](http://www.debates.org) ceases to exist. This 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

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. Before using the data set, I’m mutating all last names to uppercase. This is done to easier match the last names in the transcripts with 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

### Cleaning Dates

I will start the data wrangling by cleaning the dates. This is done by creating a function, that 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 aren’t formatted properly. I can do this by inverting a regex with the function *filter* from the *tidyverse* package (Wickham et al., 2019) and *grepl*. I then store the output in an object called *wrong\_dates*:

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

There 4 wrong dates. One of them is a link to the translations of the debate transcripts from 2008. I don’t want this transcript, I will, therefore, remove it:

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

The last 3 dates aren’t properly formatted. I will, therefore, replace them with the correct date manually 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")

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 Speaking?

In each transcript, different speakers are marked at the start of a paragraph with their last name in uppercase followed by a semicolon. Some speakers have lowercase letters in their names (e.g. the moderator “McGee”) and in older debates, speakers are referred to as “MR.” or “MS”. I will, therefore, use a series *str\_extract* and regexes to match all speakers, and gradually remove semicolon and “MR.” or “MS.”. The output is saved in a tibble 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

I then combine *all\_debates* and *last\_name*:

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

Every time a speaker is not mentioned, it is the last known speaker who is talking. By replacing all NA’s with the previously known value, it is, therefore, possible to assign a speaker to each row. 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.

I will then transform the column *last\_name* to uppercase so I can do a left\_join:

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

I then left\_join *all\_debates* with *candidates* by column *last\_name* and *year*. I’m also replacing NA’s with the string “not\_a\_candidate”. This is done to follow Broman and Woo’s principles of data organization in spreadsheets. One of these 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 transcripts:

# 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 aesthetic 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 is clean and ready to be analyzed.

## Text-mining: Preparation

I will do three types of text-mining: *tf-idf*, *sentiment analysis*, and *stop word filtering*. I'm mainly inspired by Julia Silge and David Robinson’s approach to text-mining in their book: "Text Mining with R: A Tidy Approach" (2017). Therefore, I'm going to use the *tidytext* package (Silge & Robinson, 2016).

I start by creating a custom function for plotting word clouds. Most parameters have a default value, which will make it a lot easier to plot, but still give me room for customization:

# Function for plotting word clouds

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()   
 }

I then create a tibble called *my\_stop\_words*, which contains stop words that are not included in the stop word lexicon. Afterward, I tokenize the text by word and use a regex to filter out digits. Lastly, by using anti\_join, I filter out rows that match *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)

Now I’m ready for text-mining.

## Text-mining: Stop Word Filtering

Most transcripts use a single right quotation mark instead of an apostrophe. This causes problems when using the anti\_join since my stop word lexicon contains proper apostrophes. This is a problem caused by the font in the transcripts, which does not differentiate between right single quotation marks and apostrophes. Luckily, I found an answer to the problem in [this stack overflow question](https://stackoverflow.com/questions/47209828/r-tidytext-stop-words-are-not-filtering-consistently-from-gutenbergr-downloads), which is used on the second line. I then do an anti\_join to remove stop words. The output is saved in the object *stop\_words\_removed*:

stop\_words\_removed <- debate\_words %>%   
 mutate(word = gsub("\u2019", "'", word)) %>% #Read right single quotation mark as an apostrophe   
 anti\_join(stop\_words)

Now I’m ready to plot my results.

### Plotting Stop Word Filtering

I will plot my results by filtering out rows where the candidate does not belong to the republican or democratic party, from the object *stop\_words\_removed*. Then I count by word and year, and then pipe the result directly into my function *plot\_wordclouds*, which will be faceted by year and party:

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")

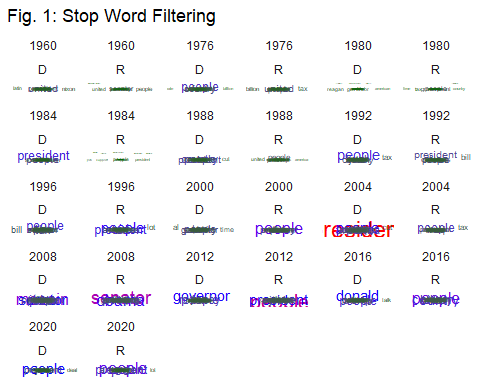


Fig. 1: Stop Word Filtering

As fig. 1 shows, the candidates’ primary talking points from 2008-2016 appears to be the opposite candidate. It is quite astonishing that the opponent’s name appears so often, that otherwise common themes, such as national debt, jobs, wars, or budgets, are pushed aside. However, this result might be a consequence of changes to the debate format, such as longer debates, which naturally would increase how often a candidate is mentioned.

By using *term frequency-inverse document frequency*, which looks at the word frequency across all documents, it should be clearer whether this is a general phenomenon or the result of changes to the debate format.

## Text-mining: Term Frequency-Inverse Document Frequency

I start the tf-idf analysis by counting how often a word in the data set *debate\_words* appear 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)

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. 3: Tf-idf each year - Both Parties")

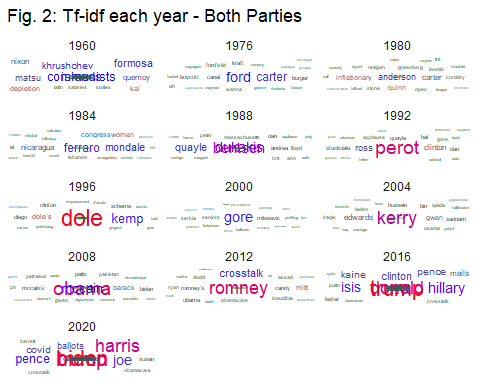


Fig. 2: TF-IDF each year - Both Parties

The stop word filtering suggests that each candidate increasingly talks about their opponent. The tf-idf analysis substantiates this point, since it shows, that each debate increasingly revolves around the candidates themselves rather than political issues. This tendency started back in 1992 and was especially pronounced in 1996 and from 2004 to 2020. These results certainly point towards increasing polarization, since I doubt that the candidates are praising their opponents. But what about their sentiment, has it changed over the last couple of years, and does it differ between the parties? This can be explored through a sentiment analysis.

## Text-mining: Sentiment Analysis

To get reliable results, I will only focus on debates after 1992. I will start by saving the words “Trump” and “vice” in a tibble called *distortion\_words*. This tibble is used to anti\_join these words since they distort the result. Trump is, in this case, a name, but is defined as a word with a positive sentiment in most sentiment lexicons. Vice refers to vice-presidents, but it has a negative sentiment in most lexicons:

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

I then filter out all debates before 1992, remove the distortion words, and inner\_joins the binary sentiment lexicon “bing”. The result is saved in the object *sentiment\_debate\_words*:

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

Now I should be ready to plot the sentiments.

### Plotting Sentiments

I want to make a pie-chart to visualize the bing values. Many authors reject pie-charts, but, as Claus Wilke points out, they work well when showing simple fractions such as one-half, one-third, one-quarter, and so on (Wilke, 2019). Moreover, 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 calculates how large a share each sentiment makes up of the total words and use *coord\_polar* to make the bar chart round rather than rectangular:

#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 republicans. 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. 5: Republicans' sentiment by year - BING",  
 x = "",  
 y = "",  
 fill = "sentiments")

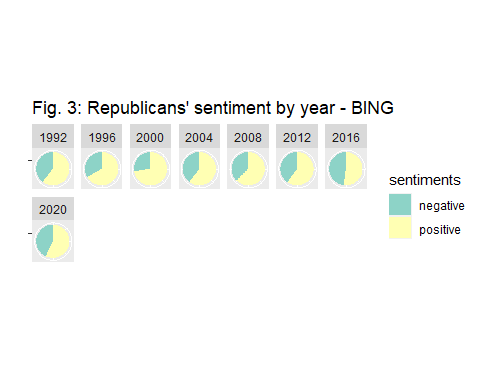


Fig. 3: Republicans’ sentiment for each year - BING

The year 2000 was the most positive debate for the republicans, and all subsequent years have since then 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. 6: Democrats sentiment by year - BING",  
 x = "",  
 y = "",  
 fill = "sentiments")

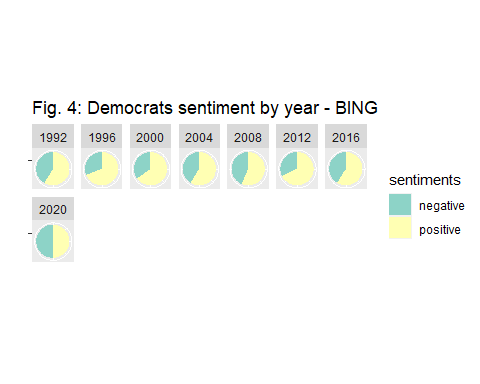


Fig. 4: Democrats’ sentiment for each year - BING

Democrats almost follow the same pattern, with the debate in 2000 having an extremely positive sentiment, and subsequent debates being much more negative culminating in 2020, where about half of the counted words were negative. The 2012 debate is also interesting since democrats have a surprisingly positive sentiment.

## Conclusion

This preliminary exploratory analysis already points towards some interesting tendencies. Based on the tf-idf analysis it does look like debates increasingly revolve around the individual candidates rather than policy issues. This tendency appears to have started in 1992 but was especially pronounced in 1996, 2004, 2008, 2016, and 2020. The stop word filtering points towards the same results and suggests that candidates are primarily mentioned by their opponent rather than by themselves or their vice-presidential candidate. The sentiment analysis shows that both parties have generally developed a more negative sentiment since 2000 culminating in 2016 for the republicans and 2020 for democrats. The results, therefore, point towards an increased polarization, where each candidate spends more time attacking their opponent. This suggests that rather than looking at differences in political topics or opinions, further research into polarization among political candidates could also investigate how often and in what context each candidate addresses their opponent.

## Literature

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

- Pewreseach.org (n.d.): ”Political Polarization”. *Pew Research Center*. URL: <https://www.pewresearch.org/topics/political-polarization/>

- 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>

- Wilke, C. O. (2019). “Chapter 10: Visualizing proportions” in *Fundamentals of Data Visualization*. *O’Reilly Media Inc.*, URL: <https://clauswilke.com/dataviz/visualizing-proportions.html>

## Packages

- Grolemund, G. & Wickham, H. (2011). ”Dates and Times Made Easy with lubridate”. *Journal of Statistical Software*, 40(3), 1-25. URL http://www.jstatsoft.org/v40/i03/

- Silge, J. & Robinson, D. (2017). “Text Mining with R: A Tidy Approach”. *O’reilly Media, Inc.* URL: <https://www.tidytextmining.com/>

- Wickham, H (2016). “ggplot2: Elegant Graphics for Data Analysis.” Springer-Verlag New York. ISBN 978-3-319-24277-4, https://ggplot2.tidyverse.org

- Wickham, H (2016). “ggplot2: Elegant Graphics for Data Analysis.” *Springer-Verlag New York.* ISBN 978-3-319-24277-4, <https://ggplot2.tidyverse.org>

- Wickham, H. (2020). “rvest: Easily Harvest (Scrape) Web Pages”. Located at CRAN here: <https://cran.r-project.org/web/packages/rvest/index.html>

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

- 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