Final Exam

Christoffer Mondrup Kramer

08-12-2020

Student number: 201704767

Cultural Data Science

Final exam

Aarhus University

Contents

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Note: Originally made in overleaf.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

1 25-09-2020: Shell

Your supervisor has shared a folder of photos on Sciencedata.dk with you (password is 2020CDS, folder is 500Mb and contains 189 images) and needs your help with a couple diagnostics:

1.1 Part 1: Identify the names and format of the 3 biggest files. Can you come up with a command to generate a numerically ordered list of 3 biggest files? (hint: consider using wc to gauge image size)

- **Answer:** The three biggest files, ordered numerically by size, are
  - "9240_Overview_S.RW2" (14.761.472 bytes)
  - "9247_Overview_SW.RW2" (14.733.312 bytes)
  - "9237_Overview_W.RW2" (14.713.856 bytes)

1.1.1 Steps

- **Action:** I started out by navigating to the HW folder, which is located in my data-shell folder on my desktop

```
cd OneDrive/Skrivebord/data-shell/HW
```

cd is the command for moving through the directories. Since my pc is backed up with OneDrive I need to move to my OneDrive folder first, then Dekstop (Skrivebord), then data-shell and lastly HW.

- **Action:** I then listed what was in the current directory, to see what files it contained:

```
ls
```

ls is the command for listing the contents of a directory. Since I'm currently in the HW directory, I do not need to specify the path.

- **Result:** There where a lot of .jpgs but also some other file-formats.
- **Action:** I needed to list the file-size. I knew that the wc command could do it, but I couldn't remember what flag to use, so I consulted the manual:

```
wc --help
```

- **Result:** I figured out, that I could use the -c flag to get the file-sizes.
- **Action:** I knew that I had to sort it numerically and in reverse. By doing this I could use the head command to get the first three lines:

```
wc -c * | sort -n -r | head -n3
```

The asterisk after -c indicates that I want to count the size of everything in this folder. With the second pipe I'm sorting the output from the previous command nummerically and in reverse (the largest numbers appear on top of the list). The last pipe takes the three first lines from the output of the second pipe.

- **Result:** It almost worked, however, the first line is the combined size of every file. I therefore needed to get the second, third and fourth line of the output, rather than the first, second and third line.
- **Action:** I changed head -n3 to head n-4 and added another pipe:

```
wc -c * | sort -n -r | head -n4 | tail -n3
```

The last pipe indicates that i want the 3 last lines from the previous output.

- **Action:** It worked, I now had a list of the three biggest files. Sorted numerically by size.

1.2 Part 2: Some of the image files are empty, a sign of error in the data processing or corruption. Can you find the empty photo files (0 kb size), count them, and generate a list of their filenames to make their later replacement easier?

- **Action:** I knew that I needed to somehow find files according to how big they were. I, therefore, thought that the command "find" would be the best place to start. Since, If I used the answer from part 1, I would have to count zero sized files manually. I used –help to see what flags I could use with find:

```
find --help
```

- **Result:** It turns out that I can use the flag -size to find files according to their size.
- **Action:** I used find -size 0 to find all file containing zero bytes. Moreover, I also used the flag -type f to indicate, that I only wanted files (not directories). The last part wasn't that important, but it would make sure, that I only matched files.

```
find -size 0 -type f
```

- **Result:** It worked. Now I only needed to count the amount of files. I noticed that all files was separated by line-breaks. Therefore, I could just use a pipe containing wc -l to count the number of lines from the previous output:

```
find -size 0 -type f | wc -l
```

- **Result:** It worked. I had 73 files with 0 bytes. Now I only needed a list for later replacement.
- **Action:** Luckily this is easy. I can just use the "larger than" character to make a text-file containing the output:

```
find -size 0 -type f > list_zero_byte_files.txt
```

Result: It worked. I now have a file containing all file names with zero bytes called list_zero_byte_files.txt in the HW folder.

1.3 Appendix

1.3.1 Source code for part 1 (Copied from my terminal)

```
484  cd OneDrive/Skrivebord/data-shell/HW/
485  ls
486  wc --help
487  wc -c * | sort -n -r | head -n3
488  wc -c * | sort -n -r | head -n4 | tail -n3
489 history
```

1.3.2 Source code for part 2 (Copied from my terminal)

```
500  find --help
501  find -size 0 -type f
502  find -size 0 -type f | wc -l
503  find -size 0 -type f > list_zero_byte_files.txt
504  history
```

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

1 02-10-2020: Start with R

For this assignment, please submit a page of your journal showing the following solutions:

1.1 Part 1: Use R to figure out how many elements in the vector below are greater than 2

```
rooms <- c(1, 2, 1, 3, 1, NA, 3, 1, 3, 2,
           1, NA, 1, 8, 3, 1, 4, NA, 1, 3,
           1, 2, 1, 7, 1, NA)
```

1.1.1 Steps

- **Action:** I created new project called "assignment_start_with_R". Then I created a folder called "scripts" for keeping track of my scripts. I also created a folder called "data" for keeping track of my data. Lastly, I copy-pasted the code above in to a script called "assignment1_2".

```
rooms <- c(1, 2, 1, 3, 1, NA, 3, 1, 3,
   2, 1, NA, 1, 8, 3, 1, 4, NA, 1,
   3, 1, 2, 1, 7, 1, NA)
```

- **Action:** Then I created a variable, which would be a subvector of "rooms" but excluding all NA's.

```
rooms_NA_removed <- rooms[!is.na(rooms)]
```

"rooms_NA_removed" is my new variable. rooms[!is.na(rooms)] shows that I want a subsection of rooms but without NA's. This is done by using the operator ! which means NOT. If I removed the ! I would only get NA's.

- **Result:** It worked, I now had a new vector without any NA's.

- **Action:** I then used the logical operator "larger than" to only get a vector containing numbers larger than 2.

```
rooms_NA_removed[rooms_NA_removed > 2]
```

This worked since I got the following output:

```
3 3 3 8 3 4 3 7
```

Therefore, I just needed to count the length of this output.

4

- **Action:** I, therefore, used the function length() with my previous command as the parameter. This should count the length of the list.

```
length(rooms_NA_removed[rooms_NA_removed > 2])
```

- **Result:** This worked. The result is 8.

1.2 Part 2: What is the result of running median() function on the above 'rooms' vector? (again, best remove the NAs)

- **Action:** Since I already had the variable "rooms_NA_removed" without NA's I could just use the median() function with this variable as the parameter:

```
median(rooms_NA_removed)
```

- **Result:** This worked. The median of the rooms vector is 1.5.

1.3 Part 3

Inside your R Project (.Rproj), install the 'tidyverse' package and use the download.file() and read_csv() function to read the SAFI_clean.csv dataset into your R project as 'interviews' digital object. Take a screenshot of your RStudio interface showing a) the script you used to create the object b) the 'interviews' object in the Environment and the c) structure of your R project in the bottom right Files pane. Save the screenshot as an image and put it in your AUID_lastname_firstname repository inside our Github organisation (github.com/Digital-Methods-HASS). Place here the URL leading to the screenshot in your repository.

- **Click here for the screenshots of my script, the interview object, and my file structure**

1.4 Appendix

1.4.1 Source code for part 1 and 2

```
 #1) Use R to figure out how many elements in the vector below are greater than 2 .
#(You need to filter out the NAs first)

rooms <- c(1, 2, 1, 3, 1, NA, 3, 1, 3,
           2, 1, NA, 1, 8, 3, 1, 4, NA, 1,
           3, 1, 2, 1, 7, 1, NA)



#Create variable excluding NA from the vector
rooms_NA_removed <- rooms[!is.na(rooms)]

#Extract numbers over 2
rooms_NA_removed[rooms_NA_removed > 2]

#show the length of previous output
length(rooms_NA_removed[rooms_NA_removed > 2])

#What is the result of running median() function on the above 'rooms' vector?
#(again, best remove the NAs)
median(rooms_NA_removed)
```

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. 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
```

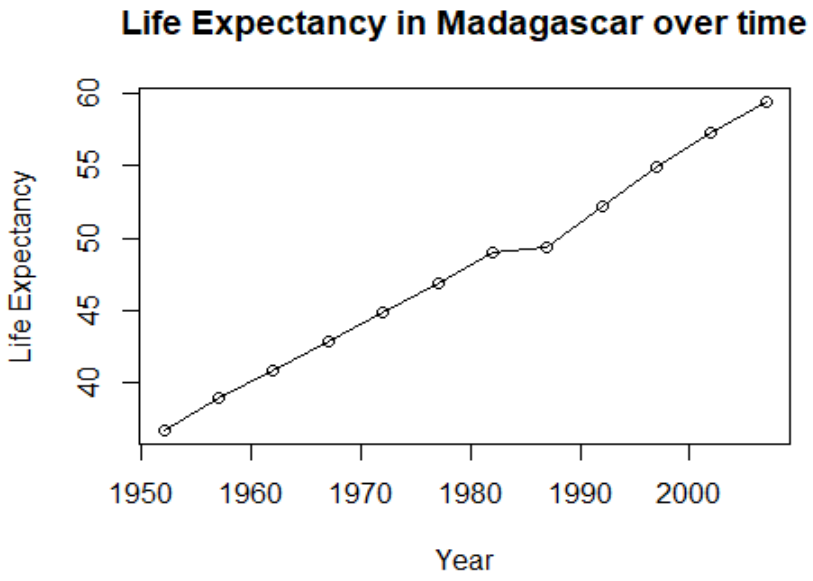
The second csv file is a summary of the average life expectancy of each country (based on the previous csv file), with column a called "mean_life". This will be used in order to check whether the average life expectancy is under 50.

```
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
```

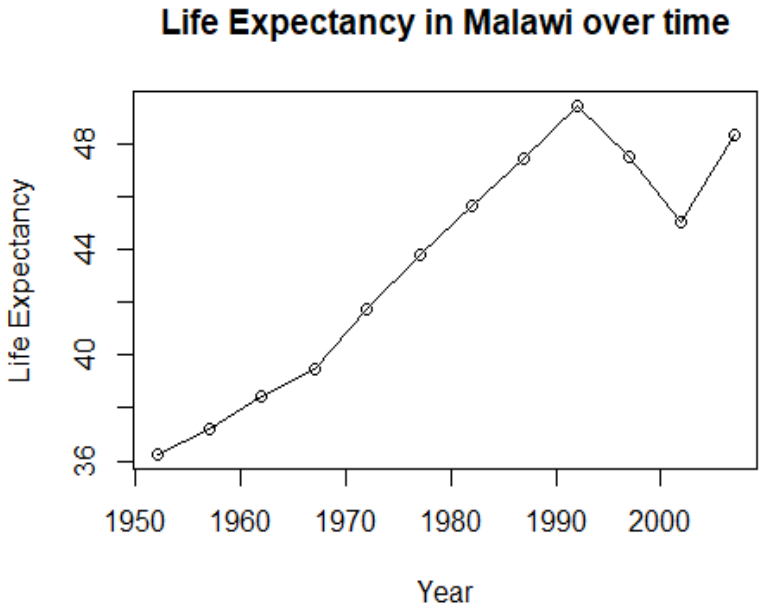
```
## `summarise()` ungrouping output (override with `.groups` argument)

  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.

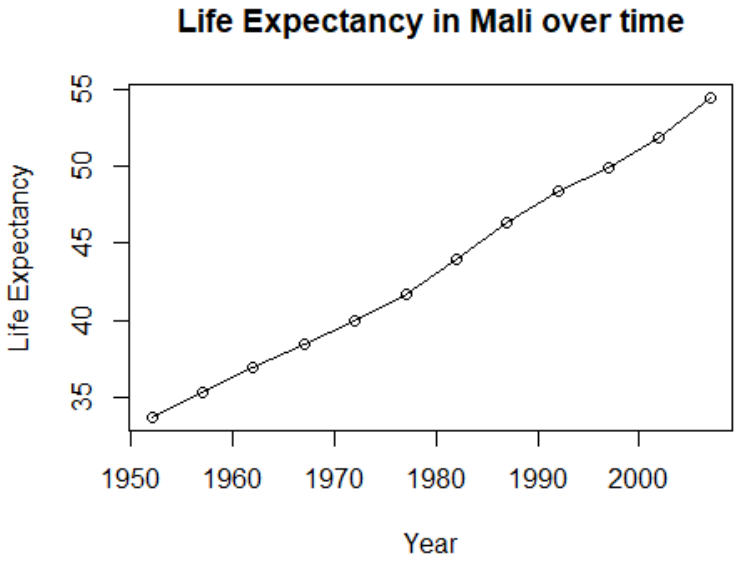
```
for (i in mean_life_country_m$country) { #for every country in "mean_life_country"

  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
```



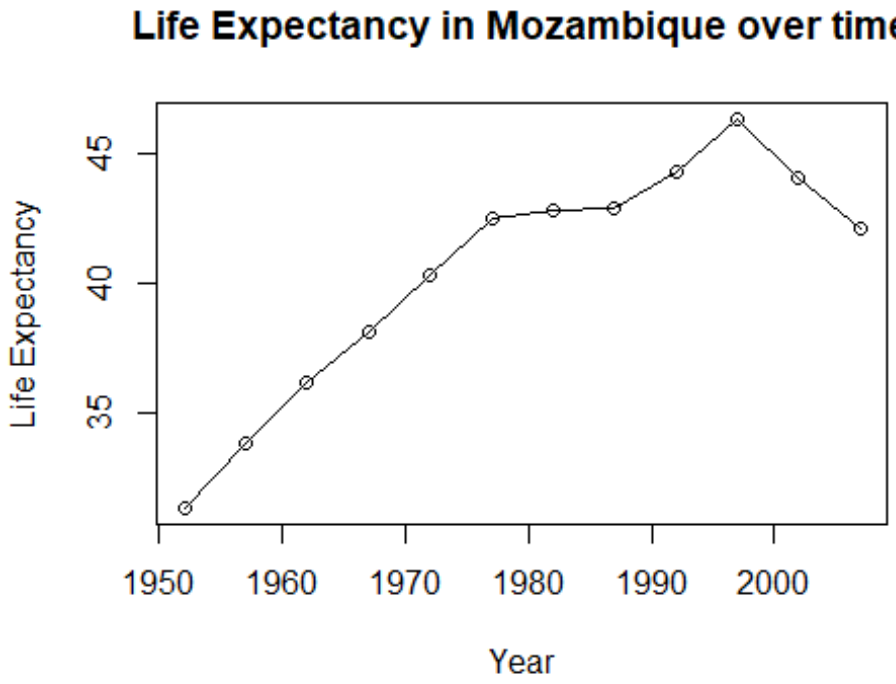

```
## The life expectancy of Malawi is smaller than 50 ( 43.35158 ) plotting life
expectancy
```




```
## 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 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/)*
- *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.

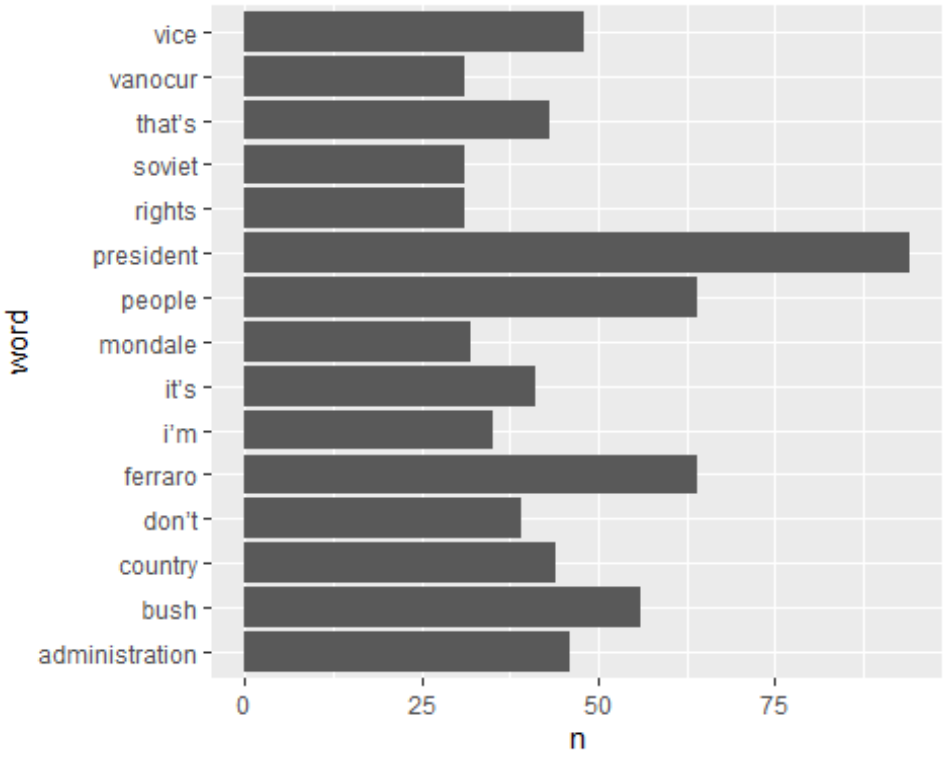
```
`debate_october-11-1992`
```

```
## # 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

US Presidential and Vice-Presidential Debates

- **Research Question:** *How have the political discourse during presidential and vice-presidential debates in the US changed since 1960?*
- **Data source:** Web-scraping transcripts from www.debates.org
- **Data cleaning:** Tokenize by word
  - **Packages**: Tidytext, Tidyverse
- **Analysis:** tf-idf, sentiment analysis, stop word filtering
- **Visualization:** Word Clouds, bar-charts
  - **Packages:** ggplot2

Relevancy

TRADITIONAL DISCOURSE ANALYSIS

CLOSE READING AND QUALITATIVE CODING

THIS METHOD

DISTANT READING AND DATA VISUALIZATION

ADVANTAGES

NEW INSIGHTS AND TIME SAVING

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:

```
# 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 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, 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")
```

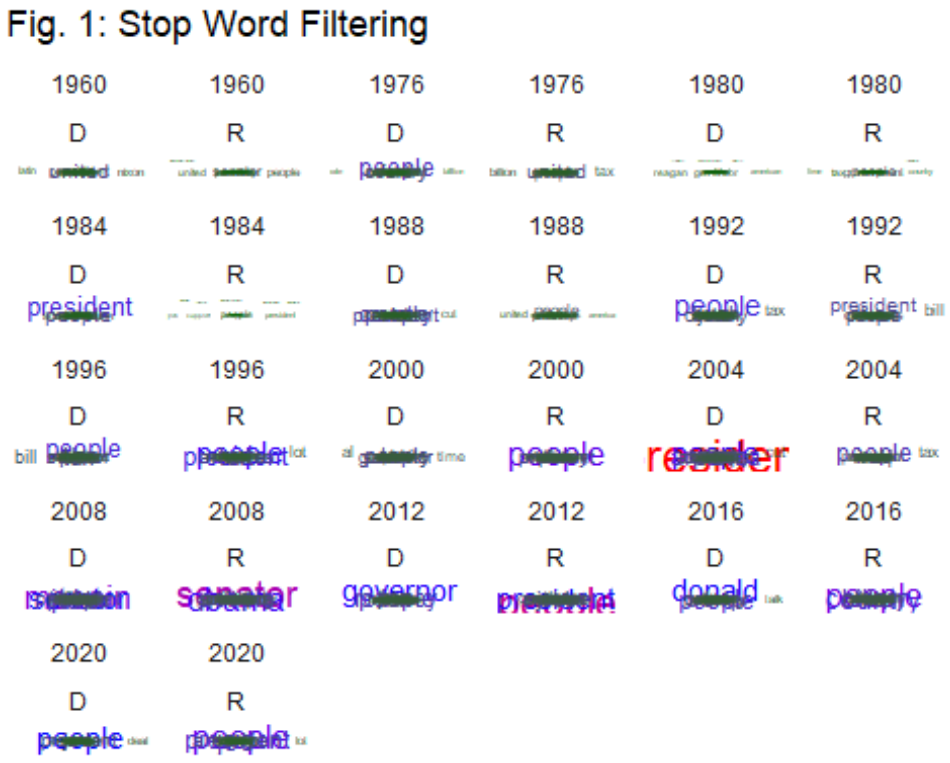


Fig. 1: Stop Word Filtering

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")
```

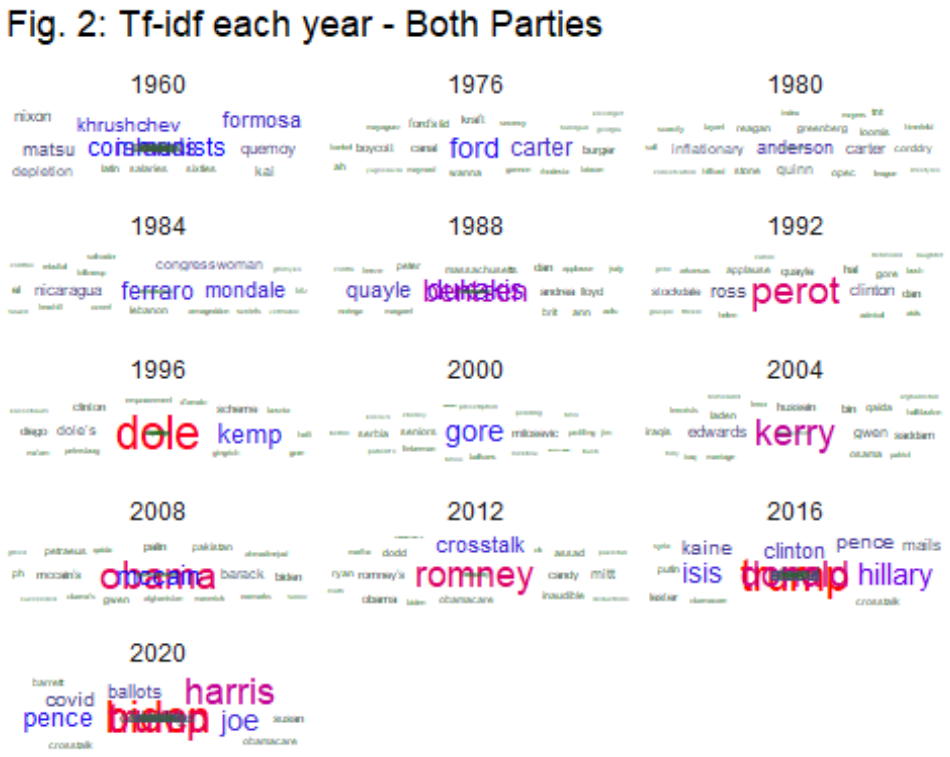


Fig. 2: TF-IDF each year - Both Parties

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")
```

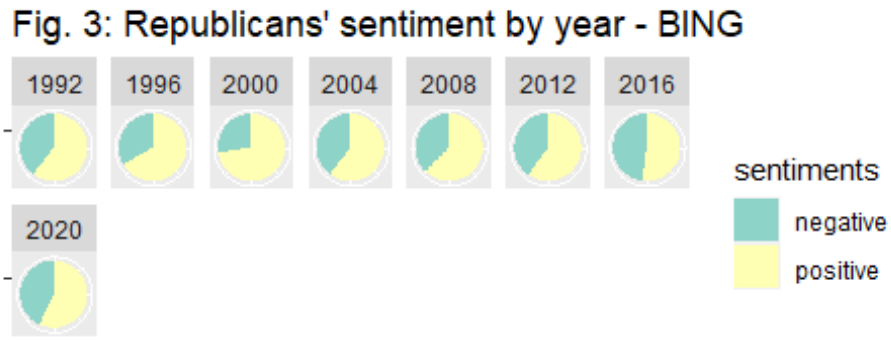


Fig. 3: Republicans' sentiment for each year - BING

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")
```

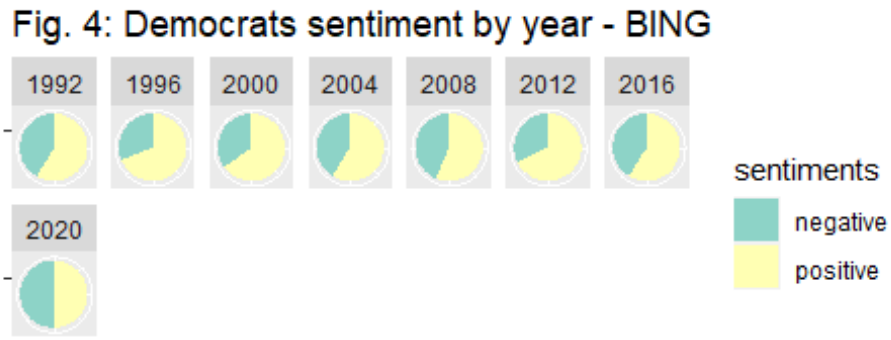


Fig. 4: Democrats' sentiment for each year - BING

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