# 1 Introduction

The goal of this project was to find the sentiment on Twitter for each of the presidential candidates for the United States 2020 Election. The sentiment analysis itself is rather rudimentary and only classifies tweets into two categories, "negative" and "positive", using the Bing lexicon for sentiment analysis. One of the main features of the project however is its applicability to real time twitter data. Using the rtweet package the algorithm is able to stream live data from twitter, categorize each tweet into the U.S state from which it came, find the sentiment of each individual tweet and finally output a graphical map and an animation, showing how sentiment of each candidate is changing with each additional tweet. The algorithm is executable in its entirety from start to finish. Various comments have been included along the way for descriptions and optional functionality which can be activated by uncommenting blocks of code.

# 2 Setup

Eight packages are used in addition to the Tidyverse. Below are each package with a short description of its use.

```
library(rtweet)
```

Used for data collection, it streams tweets from Twitter in real time. *Stream_tweets* is the main function used, which streams tweets in real time and stores them as a .json file.

```
library(textdata)
```

Contains the lexicons used for the sentiment analysis.

```
library(tidytext)
```

Allows for sentiment analysis, *unnest_tokens* splits sentences into individual words.

```
library(plotly)
```

Used for plotting the maps of the U.S.

```
library(extrafont)
```

Additional fonts for the map plots.

```
library(readxl)
```

Used for data import.

```
library(gganimate)
library(gifski)
```

Used for creating the animation. On most systems $gifski$ can be dropped.

# 3 Initial Parameters

Before commencing the user will have to decide on two things. First, the time for which to stream tweets for each candidate. This is stored as a variable *stream_time* with a minute multiplier.

```
# no upfront-parse, save as json file
filenames <- c("stream_data_unparsed_b.json", "stream_data_raw_unparsed_t.json")
q <- c("biden", "trump")
streamtime <- 0.2 * 60 #set time out in minutes
```

A reasonable lower threshold is about six seconds, which allows for the stream to boot, collect and end. In the making of this project however stream times of up to twelve minutes have been used successfully, collecting approximately twenty thousand observations per candidate. The $q$ variable is the search query.

Second, the user must decide on how many of the collected tweets per state that should be analysed and used for plotting and animating. The number of tweets successfully collected per state varies greatly due to natural reasons such as population and Twitter usage. As mentioned later on, redundancy is added so that there is no risk of setting this value too high or too low. The result in the case of too few tweets collected for an individual state is that the tibble is filled out with zeroes so that sentiment is simply not affected.

```
tweets_per_state <- 50 #number of tweets per state to be analyzed.
```

This is done via the *tweets_per_state* variable. Section 5, lines 134-144 and section 9, lines 273-283, contains an option for counting the amount of observations collected for Biden and Trump respectively. The code can be run up to only this point to get an idea of how to set the *tweets_per_state* variable. The result is a tibble with the number of viable observations per state that can be used for sentiment analysis. As a pointer, 6 seconds of streaming at one instance generated a mean of 1.68 tweets per state for Biden, who generates the least data on Twitter of the two. The minimum was 0 and maximum was 7. 12 seconds of streaming generated at another instance a mean of 3.68, with minimum still at 0 and a maximum at 15. This suggests that there are states that inherently generate little data, but also that the average can be significantly improved by increasing stream time, as would be expected.

The code is preset with $stream\_time = 0.2 * 60$ and $tweets\_per\_state = 5$. With these settings the entire code, excluding the last animation, takes about 50 seconds to run, out of which 24 seconds are used for streaming. The generation of the last animation takes about 3 minutes. It is worth noting that the *tweets_per_state* variable and the production of the animation in the last section are the main drivers of execution time.

# 4 Structure Of The Project

The project is divided into 16 sections as follows.

```
# Section One - Setup ----
# Section Two - Stream Data ----
# Sectiton Three - Collect Data (Biden)----
# Section Four - Clean Data (Biden)----
# Section Five - Classify Data (Biden)----
# Section Six - Find Sentiment in Data (Biden)----
# Section Seven - Collect Data (Trump)----
# Section Eight - Clean Data (Trump)----
# Section Nine - Classify Data (Trump)----
# Section Ten - Find Sentiment in Data (Trump)----
# Section Eleven - Aggregate Sentiments----
# Section Twelve - Gradient Map----
# Section Thirteen - Polarized Map----
# Section Fourteen - Data Wrangling for Animation----
# Section Fifteen - Graphing for Animation----
# Section Sixteen - Animation----
```

Sections 1-2 are for initialization and streams data from Twitter for both candidates. The $q$ parameter sets the queries to search for. Sections 3-6 and 7-10 are in all essence mirrored for each candidate and sections 11-15 creates graphs and section 16 produces the final animation.

# 5 Data Collection

Given the initial parameters, the data is streamed live, unparsed into .json files. This separation is intentional, so that when streaming longer sessions any eventual parsing errors won't disrupt the data collection process. Sequentially data is parsed into tibbles *data_b* and *data_t* for Biden and Trump respectively. These subscripts will denote the candidates throughout the project. Additional data is then requested in form of user data from Twitter. A block of code that can be activated checks where location information is most available, since there were two vectors for this, *location* and *quoted_location*. The collected data is then merged and only relevant variables are selected. Any missing observations are removed and only tweets in English and "undetermined" are kept. "Undetermined" are kept to increase the ratio of viable tweets for sentiment analysis later on. The majority of these undetermined tweets are in English and should they not be viable they will get sorted out later on.

Two things are worth mentioning regarding the data collection process. The first is that the *rtweet* package contains a function *search_tweets*, that can be used to search for historical tweets. This would most likely allow for some amount of increased accuracy in finding relevant tweets, which would mean less of a need for data wrangling. One of the key ideas with this project however was to build an algorithm that could stream tweets in real time and find the sentiment of users on Twitter

in relation to different search queries. In this case these were the two presidential candidates, but the generalization is easy to make by swapping search query. The second is that with *rtweet* there is no need to directly interact with the Twitter API. The library *rtweet* uses the Twitter's REST API and there is no need to go through the usual process of applying for a developer account. API keys, API secrets and authentication is handled automatically via $rstats2twitter$. This makes the data collection process more seamless.

There is however still some authentication that needs to be done. In an interactive R session a function call from the *rtweet* package (e.g. $stream\_tweets$) will trigger the embedded $rstats2twitter$ app and authentication will be done via a web browser popup. The token will be created and stored for future sessions. This should however require a Twitter account.

Should everything fail in the process of setting up the *rtweet* package, a fail safe solution is to go via the Twitter API directly. Instructions to follow are listed in the article below.

https://cran.r-project.org/web/packages/rtweet/vignettes/auth.html

Note however that this is only added as a precautionary measure. The *rtweet* package should work with the embedded authentication.

# 6 Data Cleaning

In cleaning the data the *stringr* package was indeed useful, especially in altering casing, removing white spaces and searching for keywords within the tweets. Mainly the *location*- and *text* vectors were cleaned. The *location* vector needed to exactly match the *state_ab* vector, which was constructed from the built in character vector *state.abb* containing each U.S state abbreviation code. The *text* vector contained the actual tweets and needed to be formatted to match the words in the Bing lexicon which is used for sentiment analysis. The *str_detect* function was used in combination with $filter()$ to sort out all tweets only containing the candidate in question, since there was several instances of tweets mentioning both candidates. This would otherwise have contaminated the sentiment analysis later on. *Distinct* was also used to acquire a set of unique users, since the ultimate goal is to use the sentiment analysis to find the presidential candidate each individual Twitter user would have voted for.

# 7 Data Classification

The process of data classification begins with a tibble called the $tweet\_matrix$. This is constructed with a $for$ loop that searches through the *location* vector for any of the U.S state names contained in *state_ab* or *state_name*. For the state of California this would thus be matched against "california" or "ca" in the *location* vector, with everything being converted from before to lower case. Since the data type we want to store in the tibble later on is "character" each cell is given a "y" for success

or ”n” for failure to identify location origin of the tweet.

| | al | ak | az | ar | ca | co | ct | de | fl |
|---|---|---|---|---|---|---|---|---|---|
| 1 | n | n | n | n | n | y | y | n | n |
| 2 | n | n | n | n | n | n | n | n | n |
| 3 | n | n | n | n | n | y | n | n | n |
| 4 | n | n | n | y | n | n | n | n | n |
| 5 | y | n | n | n | n | n | n | n | n |
| 6 | n | n | n | n | n | n | n | n | n |
| 7 | n | n | n | n | n | n | n | n | n |

The next *for* loop searches through the identification matrix just created for any cell containing ”y” and replaces it with the appropriate tweet from the *data_b* tibble for Biden.

| | al | ak |
|---|---|---|
| 1 | null | null |
| 2 | null | null |
| 3 | null | null |
| 4 | null | null |
| 5 | *biden loses south texas latino votes that bernie wo... | null |
| 6 | null | null |
| 7 | null | null |

At this point there is code that could be activated for calculating the number of viable observations collected for the candidate, lines 134-144.

# 8 Finding Sentiment

First off the word ”trump” is removed from the Bing lexicon to avoid excessive positive sentiment for Trump. The tibbles *column_local* and *tokens_local* are reused for extracting tokens and finding sentiment in the quite long *for* loop executing this all at once. This is arguably the most time consuming part of the code aside from generating the animation at the end. To enlighten the user a timer is put in place so that an evaluation of the setting of *tweets_per_state* can be made. This is as mentioned before the main driver of computing time.

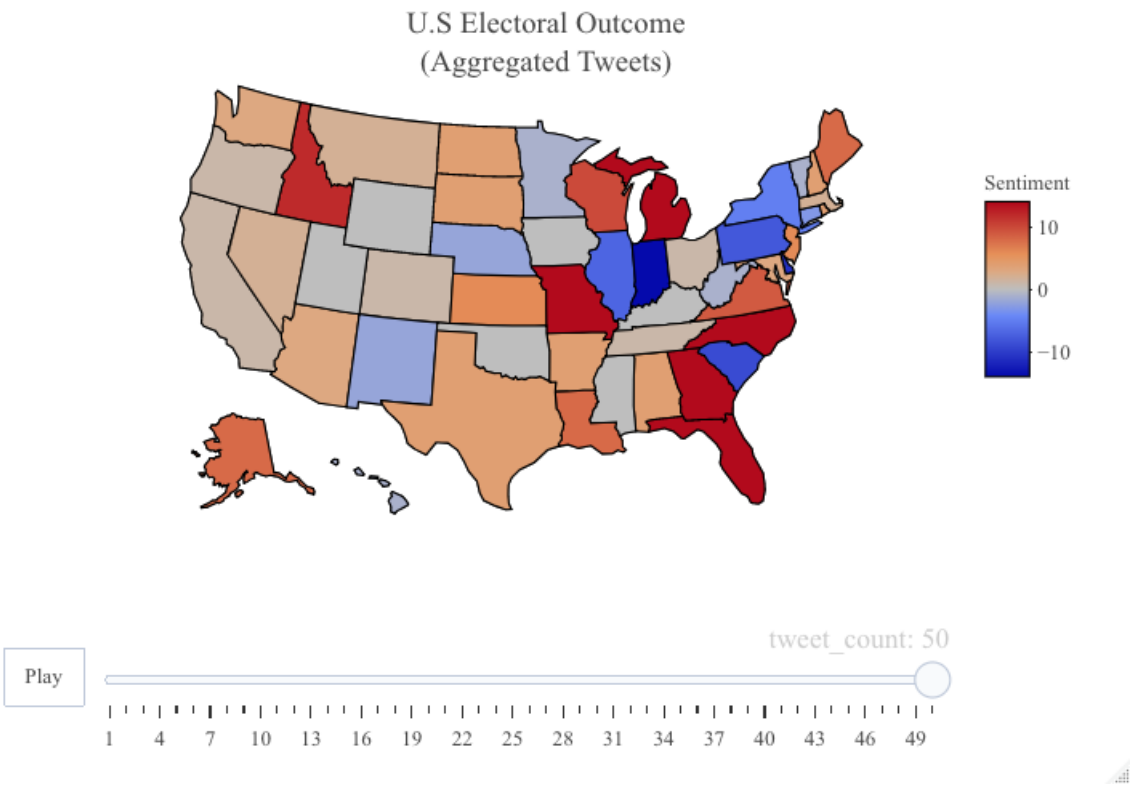
Conditional redundancy is added due to the great variability in tweets that the states generate. A *while* loop adds observations for all states having less observations than the set value

of *tweets_per_state*, thus avoiding an error message. Further redundancy is added for the cases when a state did not manage to generate any positive tweets, any negative tweets, or any tweets at all. The *if* statements assure that the tibble *tokens_local* always has three columns, "negative", "positive" and "sentiment", regardless of how few tweets that there could be found any sentiment in. Since the opinion of each individual voter was of interest, the sentiment of each observation was normalized to one. This is again because the purpose was to identify which presidential candidate the user would have voted for.

As for the sentiment analysis itself, it is done with *unnest_tokens* and *inner_join* with the Bing lexicon. The *unnest_tokens* function splits sentences into a vector of words, which then are used as the key argument for the *inner_join* with the Bing lexicon. The Bing lexicon has predefined the words that have a positive and negative sentiment. Section 11 aggregates the sentiment for both candidates, counting Trump as positive and Biden as negative. This is necessary for plotting the maps which rely on unit dimensional scaling of color. A *tooltip* vector is added to allow for hoovering with the cursor above individual states.

# 9 Maps

Two maps are created, one with a nuanced color gradient showing how far ahead or behind each candidate is, the other polarized to highlight the political party that will ultimately receive the electoral votes from the state. For brevity only the gradient map is shown below.




The maps are interactive in the sense that the slider at the bottom decides the number of counted tweets that should be shown. This is set via the *frame* parameter in the *plotly* package. The crucial difference between the two maps is in the *add_trace* argument. The min and max for $z$ is in the gradient case set to $min(sentiment\$Sentiment)$ and $-min(sentiment\$Sentiment)$, where as in the polarized case they are set to -1 and +1. This means the hoovering will still display the absolute difference between the candidates, but the scale coloring will be polarized.

A sort of finesse in the construction of these maps is that the underlying data accounts for most tweets being negative by taking differences between the two candidates. If e.g. Trump gets -50 in sentiment and Biden gets -51 in sentiment in a state the map will show that Trump is ahead by one vote. The accuracy of this logic relies on the fact that all voters who dislikes Trump votes for Biden and all voters who dislikes Biden votes for Trump, which can be questioned on an individual level. On the aggregate level however this is a nice way to circumvent the seemingly inherent nature of tweets to be overly represented in the negative dimension.

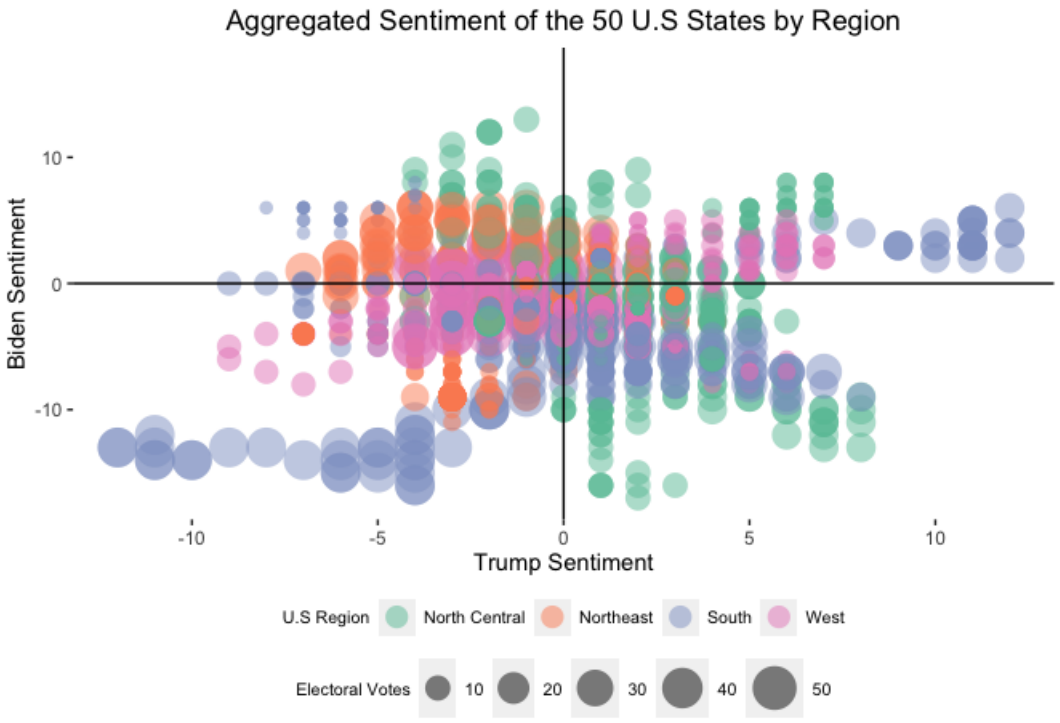
Both of these maps are based on the work presented by "dataslice" on YouTube, as part of the *plotly* package. Various visual additions and tweaks have been made, specifically the "colorbar" arguments, formatting the gradient bar on the right hand side have been improved. Adaption to data, color scaling and the $z$ arguments have all been modified. The bottom slider representing

*tweet_count* and not time has been changed and the idea of map polarization is also novel.

# 10 Animation

The following animation is also inspired by "dataslice". The data however needed to be wrangled a lot to fit the animation procedure. This is done in section 14, in a similar way to the wrangling done in section 11 when aggregating the sentiments. The animation is essentially different in that it has different centering, for which grid lines have been modified. The animation is now centered on (0,0) with a presidential candidate sentiment on each axis. Trump occupies the horizontal space and Biden the vertical space. External data is imported in excel format to adjust the size of each state moving around in the animation. Sentiment for each candidate is individually counted to make up the two axis.

In addition the *xlim*, *ylim* has been changed to be adaptive to the data input, *geom_vline* and *geom_hline* has been added for new centre, all of the legend formatting is new, allowing wrapping of multiple legends at the bottom. The *override.aes* command is new to allow for separate adjustment of the colored circles in the legend determining U.S region. All gridlines are removed with the theme being set to $panel.grid.minor = element\_blank()$ (also done for major lines). All the backgrounds (plot, panel, legend) are also set individually instead of as part of a theme, which allows for more flexibility. Below is an example of the static plot before being animated.




The coloring is divided by the U.S regions North Central, Northeast, South and West to visualize which regions voted for which candidate and if there are any trends to be found. Each individual circle is a U.S state for which the size is scaled by the number of electoral votes. In this static graph all trails for the states are shown which is why the number of circles are more than 50. This illustration is not useful in a static setting, only when animated. This exposition is only aimed at showing the general outlines of the animation.

# 11 Closing Words

In the production of this project there has been extensive data collection due to the relatively crude process of data refinement. Attached is therefore sets of data including 20,000 observations for each candidate as .rds files. These can be loaded with the *readRDS* function and then run by the algorithm without hindrance. Two .gif animations constructed of 50 tweets per state and 20 seconds animation time are also included, one consisting of tweets before the election and one of tweets after the election. It is food for thought to see how most states end up considerably more to the left end of the graph after the election, whereas before they were more balanced. As a final remark it is worth noting that the focus of this project was not to make a statistical survey, but to construct an algorithm that streams and analyses tweets in real time from the Twitter API. For any statistical purposes the sampling method would have to be refined, the sample size increased

and the population more carefully controlled.

The code is pasted in Appendix A, but for best formatting and readability it is advisable to retrieve the code from the supplied .zip file, which also contains .gif animations and .rds data files.

# 12 References

https://www.youtube.com/channel/UCBV194XNr6CIQCCuw1v2rMQ

# 13 Appendix A

```
streamtime <- 0.2 * 60 #set time out in minutes
tweets_per_state <- 5 #number of tweets per state to be analyzed.

# Section One - Setup ----
# install.packages("textdata")
# install.packages("tidytext")
# install.packages("rtweet")
# install.packages("tidyverse")
# install.packages("plotly")
# install.packages("extrafont")
# install.packages("gganimate")
# install.packages("gifski")

library(tidyverse)
library(rtweet)
library(textdata)
library(tidytext)
library(plotly)
library(extrafont)
library(gganimate)
library(gifski)
library(readxl)
# Section Two - Stream Data ----

# no upfront-parse, save as json file
filenames <- c("stream_data_unparsed_b.json", "stream_data_raw_unparsed_t.json")
q <- c("biden", "trump")

for (i in 1:2) {
  stream_tweets(
    q = q[i],
    #is_quote = "FALSE", #could be activated for removing quotes
    parse = FALSE,
```

```
    timeout = streamtime,
    file_name = filenames[i]
  )
}
# Section Three - Collect Data (Biden) ----

data_b <- parse_stream(filenames[1]) #parse from json file
user_data_b <- users_data(data_b) #get metadata for users

# code below was used to check for the unique identifier to use, we see that both
# screen_name and user_id can be used going forward, user_id was chosen.

# length(unique(data_b$screen_name))
# length(unique(data_b$user_id))

# code below was used to check where location info was most available,
# either in "location" or "quoted_location".

# location_availability_1 <- user_data_b %>%
#  select("user_id", "location")

# location_availability_2 <- data_b %>%
#  select("user_id", "quoted_location")

# location_availability_1_na <- remove_missing(location_availability_1)
# location_availability_2_na <- remove_missing(location_availability_2)

# retained_location_data_1 <- count(location_availability_1_na)/
# count(location_availability_1) #retains ~58% for location info not equal to N/A

# retained_location_data_2 <- count(location_availability_2_na)/
# count(location_availability_2) #retains ~23% for location info not equal to N/A

# merge tibbles to have a unified one with all variables
user_data_b <- user_data_b %>%
  distinct(user_id, .keep_all = TRUE)

data_b <- data_b %>%
  distinct(user_id, .keep_all = TRUE)

data_b <- inner_join(x=user_data_b, y=data_b, by=c("user_id" = "user_id")) %>%
  select("user_id", "location.x", "text", "lang") %>%
  rename(location = "location.x") %>%
  filter(lang %in% c("en", "und")) %>% #only tweets in english
  remove_missing() #only complete observations

# Section Four - Clean Data (Biden) ----

# clean location vector
location_vector_b <- pull(data_b, location)
```

```
location_vector_b <- location_vector_b %>%
  str_trim(side = "both") %>%
  str_to_lower() %>%
  str_squish()

data_b <- data_b %>%
  mutate(location = location_vector_b) #add back the cleaned vector

# clean text vector
data_b$text <- data_b$text %>% str_to_lower()
data_b$text <- gsub("http.*", "", data_b$text) #remove any links
data_b$text <- gsub("https.*", "", data_b$text) #remove any links
data_b$text <- gsub("&amp;", "&", data_b$text) #remove any links
data_b$text <- gsub("\\$", "", data_b$text) #remove "$" since special character in R

data_b <- data_b %>%
  filter(!str_detect(text, "trump")) #filter out tweets containing trump

# Section Five - Classify Data (Biden) ----

state_ab <- str_to_lower(state.abb)
state_name <- str_to_lower(state.name)

# initiating tibble
tweet_matrix_b <- as_tibble(data.frame(matrix(nrow=as.numeric(count(data_b)),
    ncol=length(state_ab))))
colnames(tweet_matrix_b) <- state_ab

# creates an identification matrix, showing where data is available
for (i in 1:length(tweet_matrix_b)) {
  tweet_matrix_b[i] <-
    if_else(
      str_detect(pull(data_b, location), state_name[i]) |
        str_detect(pull(data_b, location), state_ab[i])
      == TRUE,
      "y",
      "n"
    )
}

# uses the identification matrix as a key to build a matrix of tweets
for (j in 1:length(tweet_matrix_b)) {
  for (i in 1:as.numeric(count(tweet_matrix_b))) {
    tweet_matrix_b[i,j] <-
      if_else(
        as.character(tweet_matrix_b[i,j])
        == "y",
        as.character(data_b[i,3]), #the "3" will have to be changed if more
        #variables are selected in creation of "data_b"
```

```
        "null"
      )
  }
}

# the code below can be used to check the number of observations actually collected for
# the candidate, but is not essential.

# initiating tbl for number of observations.
# biden_observations <- as_tibble(data.frame(matrix(nrow=1,ncol=length(state_ab))))
# colnames(biden_observations) <- state_ab
#
# # this loop counts the number of observations for biden for each us state and puts
# # them into a matrix.
# for (i in 1:50) {biden_observations[,i] <-
#   as.numeric(count(local_var <- tweet_matrix_b %>%
#                      filter(tweet_matrix_b[,i] != "null")))
# }
# biden_observations
# Section Six - Find Sentiment in Data (Biden) ----

# adjusting bing lexicon
bing_lex <- get_sentiments("bing") %>%
  filter(word != "trump")

# initiating matrices for tokens and sentiment
sentiment_b <- as_tibble(data.frame(matrix(nrow=1,ncol=length(state_ab))))
sentiment_b[,1:50] <- 0
colnames(sentiment_b) <- state_ab
column_local <- tibble()
tokens_local <- tibble()

# extract tokens & evaluate sentiment
time_before <- Sys.time() #this loop tends to be time consuming for large amounts of tweets
for (j in 1:50) {
  column_local <- tweet_matrix_b[,j] #column_local is used to select each individual us
  #state from the tweet matrix.
  colnames(column_local) <- "column_name"
  column_local <- filter(column_local, column_name != "null") #remove "null" rows.
  if(as.integer(count(column_local)) < tweets_per_state){
    while (as.integer(count(column_local)) < tweets_per_state) { #this is for redundancy,
    #if the streamtime is short each state might not get enough tweets
    #for sentiment analysis.
      column_local <- add_row(column_local, column_name = "notaword")
    }
  }
  for (i in 1:tweets_per_state) {
    tokens_local <- column_local[i,1] #tokens_local is used to extract the tokens of
    #each individual tweet for each us state.
    colnames(tokens_local) <- "column_name"
```

```
    tokens_local <- unnest_tokens(tokens_local, word, "column_name")
    tokens_local <- inner_join(x = tokens_local, y= bing_lex, by = "word") #this is the
    #matching with the "bing" lexicon.
    tokens_local <- tokens_local %>%
      count(sentiment) %>%
      spread(sentiment, n, fill = 0)
    if(count(tokens_local) == 0){ #the following if statements are to ensure that
    #each tokens_local has the correct columns.
      tokens_local <- mutate(tokens_local, negative = 0, positive = 0); #if sentiment is
      #missing it is set to zero.
      tokens_local <- add_row(tokens_local, negative = 0, positive = 0)
    }
    if(!exists("negative", tokens_local)){
      tokens_local <- mutate(tokens_local, negative = 0)
    }
    if(!exists("positive", tokens_local)){
      tokens_local <- mutate(tokens_local, positive = 0)
    }
    tokens_local <- mutate(tokens_local, sentiment = positive - negative) #sentiment is
    #positive-negative.
    if(tokens_local$sentiment > 0){ #normalize each user's sentiment to    1
      tokens_local$sentiment = 1
    }
    if(tokens_local$sentiment < 0){ #normalize each user's sentiment to    1
      tokens_local$sentiment = -1
    }
    sentiment_b[i,j] <- tokens_local$sentiment #this is the final matrix with
    #individual sentiment.
  }
}
time_after <- Sys.time()
time_elapsed <- time_after - time_before
time_elapsed

# Section Seven - Collect Data (Trump) ----

data_t <- parse_stream(filenames[2]) #parse from json file
user_data_t <- users_data(data_t) #get metadata for users

# merge tibbles to have a unified one with all variables
user_data_t <- user_data_t %>%
  distinct(user_id, .keep_all = TRUE)

data_t <- data_t %>%
  distinct(user_id, .keep_all = TRUE)

data_t <- inner_join(x=user_data_t, y=data_t, by=c("user_id" = "user_id")) %>%
  select("user_id", "location.x", "text", "lang") %>% #possible extensions,
  #"created_at", "name.x"
  rename(location = "location.x") %>%
```

```
  filter(lang %in% c("en", "und")) %>% #only tweets in english
  remove_missing() #only complete observations

# Section Eight - Clean Data (Trump) ----

# clean location vector
location_vector_t <- pull(data_t, location)

location_vector_t <- location_vector_t %>%
  str_trim(side = "both") %>%
  str_to_lower() %>%
  str_squish()

data_t <- data_t %>%
  mutate(location = location_vector_t) #add back the cleaned vector

# clean text vector
data_t$text <- data_t$text %>% str_to_lower()
data_t$text <- gsub("http.*", "", data_t$text) #remove any links
data_t$text <- gsub("https.*", "", data_t$text) #remove any links
data_t$text <- gsub("&amp;", "&", data_t$text) #remove any links
data_t$text <- gsub("\\$", "", data_t$text) #remove "$" since special character in R

data_t <- data_t %>%
  filter(!str_detect(text, "biden")) #filter out tweets containing biden

# Section Nine - Classify Data (Trump) ----
# categorization of tweets into US States

# initiating tibble
tweet_matrix_t <-
as_tibble(data.frame(matrix(nrow=as.numeric(count(data_t)),ncol=length(state_ab))))
colnames(tweet_matrix_t) <- state_ab

# creates an identification matrix, showing where data is available
for (i in 1:length(tweet_matrix_t)) {
  tweet_matrix_t[i] <-
    if_else(
      str_detect(pull(data_t, location), state_name[i]) |
        str_detect(pull(data_t, location), state_ab[i])
      == TRUE,
      "y",
      "n"
    )
}

# uses the identification matrix as a key to build a matrix of tweets
for (j in 1:length(tweet_matrix_t)) {
  for (i in 1:as.numeric(count(tweet_matrix_t))) {
    tweet_matrix_t[i,j] <-
```

```
      if_else(
        as.character(tweet_matrix_t[i,j])
        == "y",
        as.character(data_t[i,3]), #the "3" will have to be changed if more variables
        #are selected in creation of "data_b"
        "null"
      )
  }
}
# the code below can be used to check the number of observations actually collected for
# the candidate, but is not essential.

# trump_observations <- as_tibble(data.frame(matrix(nrow=1,ncol=length(state_ab))))
# initiating tbl for number of observations.
# colnames(trump_observations) <- state_ab
#
# # this loop counts the number of observations for trump for each us state and puts them
# # into a matrix.
# for (i in 1:50) {
#   trump_observations[,i] <-
#     as.numeric(count(local_var <- tweet_matrix_t %>%
#                        filter(tweet_matrix_t[,i] != "null")))
# }
# trump_observations
# Section Ten - Find Sentiment in Data (Trump) ----

# adjusting bing lexicon
bing_lex <- get_sentiments("bing") %>%
  filter(word != "trump")

# initiating matrices for tokens and sentiment
sentiment_t <- as_tibble(data.frame(matrix(nrow=1,ncol=length(state_ab))))
sentiment_t[,1:50] <- 0
colnames(sentiment_t) <- state_ab
column_local <- tibble()
tokens_local <- tibble()

# extract tokens & evaluate sentiment
time_before <- Sys.time() #this loop tends to be time consuming for large amounts of tweets
for (j in 1:50) {
  column_local <- tweet_matrix_t[,j] #column_local is used to select each individual
  #us state from the tweet matrix.
  colnames(column_local) <- "column_name"
  column_local <- filter(column_local, column_name != "null") #remove "null" rows.
  if(as.integer(count(column_local)) < tweets_per_state){
    while (as.integer(count(column_local)) < tweets_per_state) { #this is for redundancy,
    #if the streamtime is short each state might not get enough tweets for
    #sentiment analysis.
      column_local <- add_row(column_local, column_name = "notaword")
    }
```

```
  }
  for (i in 1:tweets_per_state) {
    tokens_local <- column_local[i,1] #tokens_local is used to extract the tokens of
    #each individual tweet for each us state.
    colnames(tokens_local) <- "column_name"
    tokens_local <- unnest_tokens(tokens_local, word, "column_name")
    tokens_local <- inner_join(x = tokens_local, y= bing_lex, by = "word") #this is the
    #matching with the "bing" lexicon.
    tokens_local <- tokens_local %>%
      count(sentiment) %>%
      spread(sentiment, n, fill = 0)
    if(count(tokens_local) == 0){ #the following if statements are to ensure that
    #each tokens_local has the correct columns.
      tokens_local <- mutate(tokens_local, negative = 0, positive = 0); #if sentiment is
      #missing it is set to zero.
      tokens_local <- add_row(tokens_local, negative = 0, positive = 0)
    }
    if(!exists("negative", tokens_local)){
      tokens_local <- mutate(tokens_local, negative = 0)
    }
    if(!exists("positive", tokens_local)){
      tokens_local <- mutate(tokens_local, positive = 0)
    } #sentiment is positive-negative.
    tokens_local <- mutate(tokens_local, sentiment = positive - negative)
    if(tokens_local$sentiment > 0){ #normalize each user's sentiment to    1
      tokens_local$sentiment = 1
    }
    if(tokens_local$sentiment < 0){ #normalize each user's sentiment to    1
      tokens_local$sentiment = -1
    }
    sentiment_t[i,j] <- tokens_local$sentiment #this is the final matrix.
  }
}
time_after <- Sys.time()
time_elapsed <- time_after - time_before
time_elapsed

# Section Eleven - Aggregate Sentiments ----

sentiment <- sentiment_b*0 #initiating matrix for sentiment

# this loop aggregates the sentiments of both candidates.
# from before, the sentiment of each tweet has been normalized to   1.
for (j in 1:50) {
  for (i in 1:as.numeric(count(sentiment_b))) {
    sentiment[i,j] <- sentiment[i,j] + sentiment_t[i,j] - sentiment_b[i,j]
    sentiment[i+1,j] <- sentiment[i,j]
  }
}
sentiment <- sentiment[-as.numeric(count(sentiment)),] #remove last duplicate row
```

```
sentiment <- sentiment %>%
  mutate(tweet_count = 1:as.numeric(count(sentiment))) %>%
  pivot_longer(state_ab[], names_to = "state") %>%
  rename(Sentiment = "value") %>%
  mutate(tooltip = paste0(str_to_title(state_name), "\n", Sentiment))

sentiment$state <- sentiment$state %>%
  str_to_upper()

# Section Twelve - Gradient Map ----
fontstyle = list(
  family = "Space␣Mono",
  size = 10,
  color = "black"
)

label = list(
  bgcolor = "#FFFFFF",
  bordercolor = "transparent",
  font = fontstyle
)

us_graph = plot_geo(sentiment,
                    locationmode = "USA-states",
                    frame = ~tweet_count) %>% #set the bottom slider for the map.
  add_trace(locations = ~state,
            z = ~Sentiment,
            zmin = min(sentiment$Sentiment),
            zmax = -min(sentiment$Sentiment),
            color = ~Sentiment,
            colorscale = "RdBu", #alternative: Bluered, RdBu
            text = ~tooltip,
            hoverinfo = "text") %>%
  layout(geo = list(scope = "usa"),
            font = list(family = "Space␣Mono"),
            title = "U.S␣Electoral␣Outcome\n(Aggregated␣Tweets)") %>%
  style(hoverlabel = label) %>%
  colorbar(
            outlinecolor = "#000000",
            y = 0.8,
            thickness = 25,
            len = 0.5,
            title = list(text = "Sentiment"))

us_graph
# Section Thirteen - Polarized Map ----
us_graph_polarized = plot_geo(sentiment,
                    locationmode = "USA-states",
                    frame = ~tweet_count) %>% #add bottom slider for map.
```

```
  add_trace(locations = ~state,
            z = ~Sentiment,
            zmin = -1,
            zmax = 1,
            color = ~Sentiment,
            colorscale = "RdBu", #alternative: Bluered, RdBu
            text = ~tooltip,
            hoverinfo = "text") %>%
  layout(
            geo = list(scope = "usa"),
            font = list(family = "Space Mono"),
            title = "U.S Electoral Outcome\n(Aggregated Tweets)") %>%
  style(hoverlabel = label) %>%
  colorbar(
            outlinecolor = "#000000",
            y = 0.8,
            thickness = 25,
            len = 0.5,
            tickmode = "array",
            tickvals = c(-0.7,0,0.7),
            ticktext = c("Biden",0,"Trump"),
            title = list(text = "Sentiment"))

us_graph_polarized
# Section Fourteen - Data Wrangling for Animation ----

# data is imported for number of electoral votes per state, which will be the size in
# the animation.
electoral_votes <-
read_excel("~/Documents/UiO/Autumn 2020/Data Science.ECON4170/Project/electoral_votes.xlsx")
electoral_votes <- electoral_votes %>%
  rename(state = State) %>%
  mutate(region = as.character(state.region))
electoral_votes$state <- str_to_upper(state_ab)

sentiment_aggregated_b <- sentiment_b*0 # initiating matrix for aggregated biden series
sentiment_aggregated_t <- sentiment_t*0 # initiating matrix for aggregated trump series
sentiment_animation <- tibble()

# this loop is replacing the loop in section eleven.
# we now want to keep the sentiment of each of the candidate separate to see the
# evolution over time.
# it will be used as animation axis.
for (j in 1:50) {
  for (i in 1:as.numeric(count(sentiment_b))) {
    sentiment_aggregated_b[i,j] <- sentiment_aggregated_b[i,j] + sentiment_b[i,j]
    sentiment_aggregated_b[i+1,j] <- sentiment_aggregated_b[i,j]
  }
}
```

```
for (j in 1:50) {
  for (i in 1:as.numeric(count(sentiment_t))) {
    sentiment_aggregated_t[i,j] <- sentiment_aggregated_t[i,j] + sentiment_t[i,j]
    sentiment_aggregated_t[i+1,j] <- sentiment_aggregated_t[i,j]
  }
}

#remove last duplicate row
sentiment_aggregated_b <-
sentiment_aggregated_b[-as.numeric(count(sentiment_aggregated_b)),]

sentiment_aggregated_t <-
sentiment_aggregated_t[-as.numeric(count(sentiment_aggregated_t)),]

sentiment_aggregated_b <- sentiment_aggregated_b %>%
  mutate(tweet_count = 1:as.numeric(count(sentiment_aggregated_b))) %>%
  pivot_longer(state_ab[], names_to = "state") %>%
  rename(Biden = "value")

sentiment_aggregated_t <- sentiment_aggregated_t %>%
  mutate(tweet_count = 1:as.numeric(count(sentiment_aggregated_t))) %>%
  pivot_longer(state_ab[], names_to = "state") %>%
  rename(Trump = "value")

sentiment_aggregated_b$state <- sentiment_aggregated_b$state %>%
  str_to_upper()
sentiment_aggregated_t$state <- sentiment_aggregated_t$state %>%
  str_to_upper()

sentiment_animation <- sentiment_aggregated_b %>%
  mutate(Trump = sentiment_aggregated_t$Trump)

sentiment_animation <- left_join(
  x = sentiment_animation,
  y = electoral_votes,
  by = "state")

# Section Fifteen - Graphing for Animation ----

candidate_graph = sentiment_animation %>%
  ggplot(aes(x = Trump,
             y = Biden,
             color = region,
             size = votes)) +
  geom_point(alpha = 0.5, stroke = 0) +
  xlim(min(sentiment_animation$Trump), -min(sentiment_animation$Trump)) +
  ylim(min(sentiment_animation$Biden), -min(sentiment_animation$Biden)) +
  scale_size_continuous(name = "Electoral␣Votes", range = c(3,10)) +
  scale_color_brewer(name = "U.S␣Region", guide = "legend", palette = "Set2") +
  labs(title = "Aggregated␣Sentiment␣of␣the␣50␣U.S␣States␣by␣Region",
```

```
       x = "Trump Sentiment",
       y = "Biden Sentiment") +
  guides(color = guide_legend(override.aes = list(size = 5))) +
  geom_vline(xintercept = 0, color = "black") +
  geom_hline(yintercept = 0, color = "black") +
  theme(panel.grid.minor = element_blank(),
        panel.grid.major = element_blank(),
        legend.position = "bottom",
        legend.box = "vertical",
        legend.margin = margin(),
        legend.title = element_text(size = 8),
        legend.text = element_text(size = 8),
        plot.title = element_text(hjust = 0.5),
        plot.background = element_rect(fill = "#FFFFFF"),
        panel.background = element_rect(fill = "#FFFFFF"),
        legend.background = element_rect(fill = "#FFFFFF"))

candidate_graph
# Section Sixteen - Animation ----

candidate_graph_anim <- candidate_graph +
  transition_time(tweet_count) +
  labs(subtitle = "Number of Tweets: {frame_time}") +
  shadow_wake(wake_length = 0.1)

animate(candidate_graph_anim, height = 500, width = 800, fps = 30,
        duration = 10, end_pause = 60, res = 100)

# optional for saving the rendered animation as a .gif file.
# anim_save("candidate_sentiment.gif")
```