Biden vs Trump

Andrew Chen

1/5/2021

Get data

The datasource for Trump's Twitter is from this amazing Twitter monitor website built by Brendan Brown and Biden's Tiwtter is a Kaggle dataset build by Vopani.

```
trump_tweets <- read_csv("tweets_11-06-2020.csv")
biden_tweets <- read_csv("JoeBidenTweets.csv")
```

First, we will combine Trump's and Biden's Twitter dataset, and we will exclude retweets.

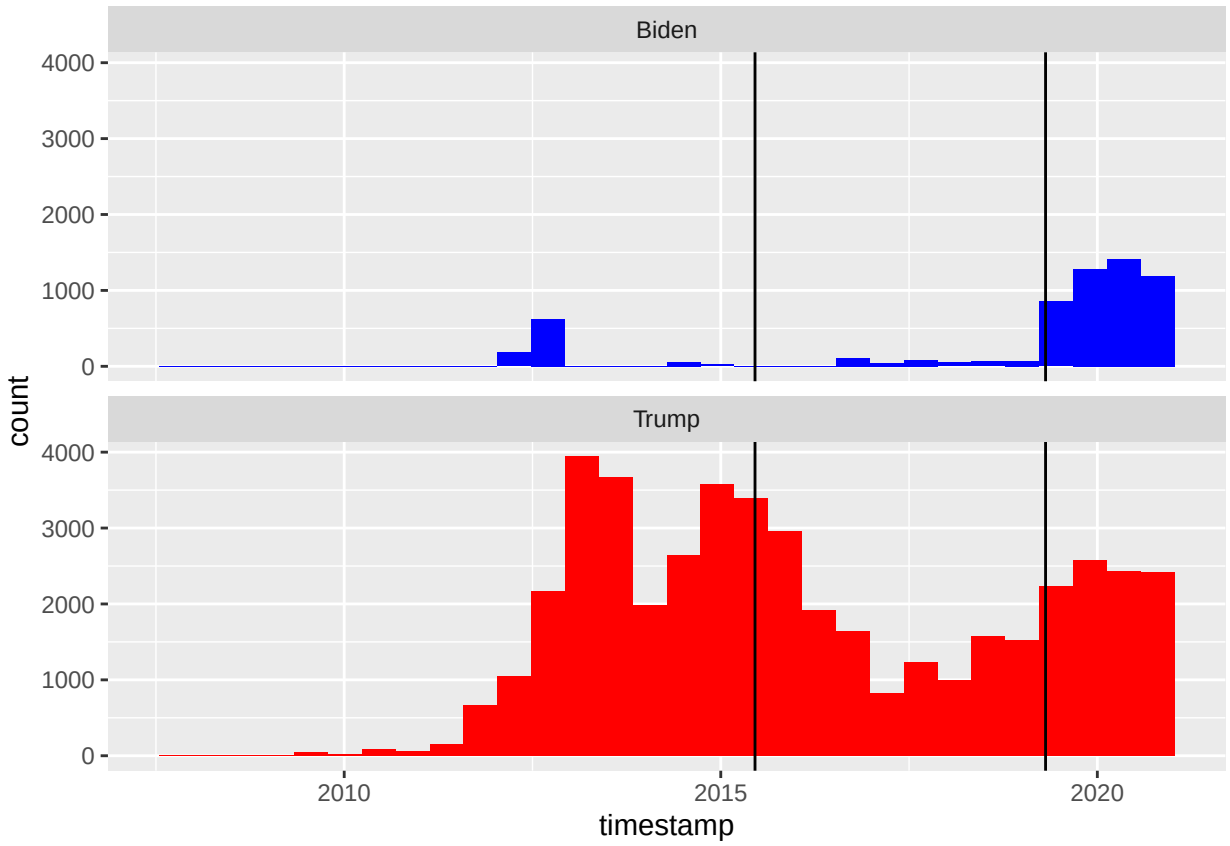
```
trump_tweets <- trump_tweets %>%
  mutate(timestamp = date, tweet = text, likes = favorites) %>%
  filter(!isRetweet) %>%
  select(id, timestamp, tweet, retweets, likes)

biden_tweets <- biden_tweets %>%
  select(-url, -replies, -quotes)

bt_tweets <- bind_rows(trump_tweets %>%
                         mutate(person = "Trump"),
                       biden_tweets %>%
                         mutate(person = "Biden")) %>%
  mutate(timestamp = ymd_hms(timestamp))

summary(bt_tweets)
```

```
##        id              timestamp                     tweet
##  Min.   :3.614e+08   Min.   :2007-10-24 22:45:00   Length:51819
##  1st Qu.:4.040e+17   1st Qu.:2013-11-22 20:27:13   Class :character
##  Median :6.646e+17   Median :2015-11-11 23:02:31   Mode  :character
##  Mean   :7.425e+17   Mean   :2016-06-13 01:01:13
##  3rd Qu.:1.144e+18   3rd Qu.:2019-06-26 18:41:20
##  Max.   :1.325e+18   Max.   :2020-11-06 17:38:17
##     retweets          likes            person
##  Min.   :     0   Min.   :      0   Length:51819
##  1st Qu.:    41   1st Qu.:     40   Class :character
##  Median :   827   Median :   1798   Mode  :character
##  Mean   :  7128   Mean   :  31276
##  3rd Qu.: 11586   3rd Qu.:  50812
##  Max.   :408866   Max.   :1890946
```

Tweet distribution

From the graph below you can see that Biden barely tweets and Trump is a lot more active on twitter. The two vertical line represent the announcement of entering presidential election, the first is Trump's 2016 campaign, the second is Biden's 2020 campaign.

```
group.colors <- c(Trump = colors()[552], Biden = colors()[26])

bt_tweets %>% ggplot(aes(timestamp, fill = person)) +
  geom_histogram(position = "identity", bins = 30, show.legend = FALSE) +
  geom_vline(xintercept = c(as.POSIXct("2015-06-16", tz = "UTC"),
                            as.POSIXct("2019-04-25", tz = "UTC"))) +
  scale_fill_manual(values = group.colors)+
  facet_wrap(~person, ncol = 1)
```



Word frequency

In order to make a tidy dataframe ready for text anaylsis in the tweets, which we will remove retweets, the common English stop words, and any http links.

```
remove_reg <- "&amp;|&lt;|&gt;" # to remove & < >
bt_tweets_tidy<- bt_tweets %>%
  filter(!str_detect(tweet, "^RT")) %>% # to remove retweet
  mutate(text = str_remove_all(tweet, remove_reg),
```

```
        text = str_remove_all(tweet,
                              "\\s?(f|ht)(tp)(s?)(://)([^\\.]*)[\\.|/](\\S*)")) %>% # to remove http l
  unnest_tokens(word, text, token = "tweets") %>%
  filter(!word %in% stop_words$word, # to remove stop words
         !word %in% str_remove_all(stop_words$word, "'"), # to remove ' in word
         !word %in% c("amp", "lt", "gt"), # remove amp/lt/gt in word
         str_detect(word, "[a-z]"))
```

Now we trying to create a dataframe that count each words' frequency for each person, first, we group by pearson, then count each words used by each person, then left join by a column with total words used by each person, then we can mutate a new column for frequency of each words.

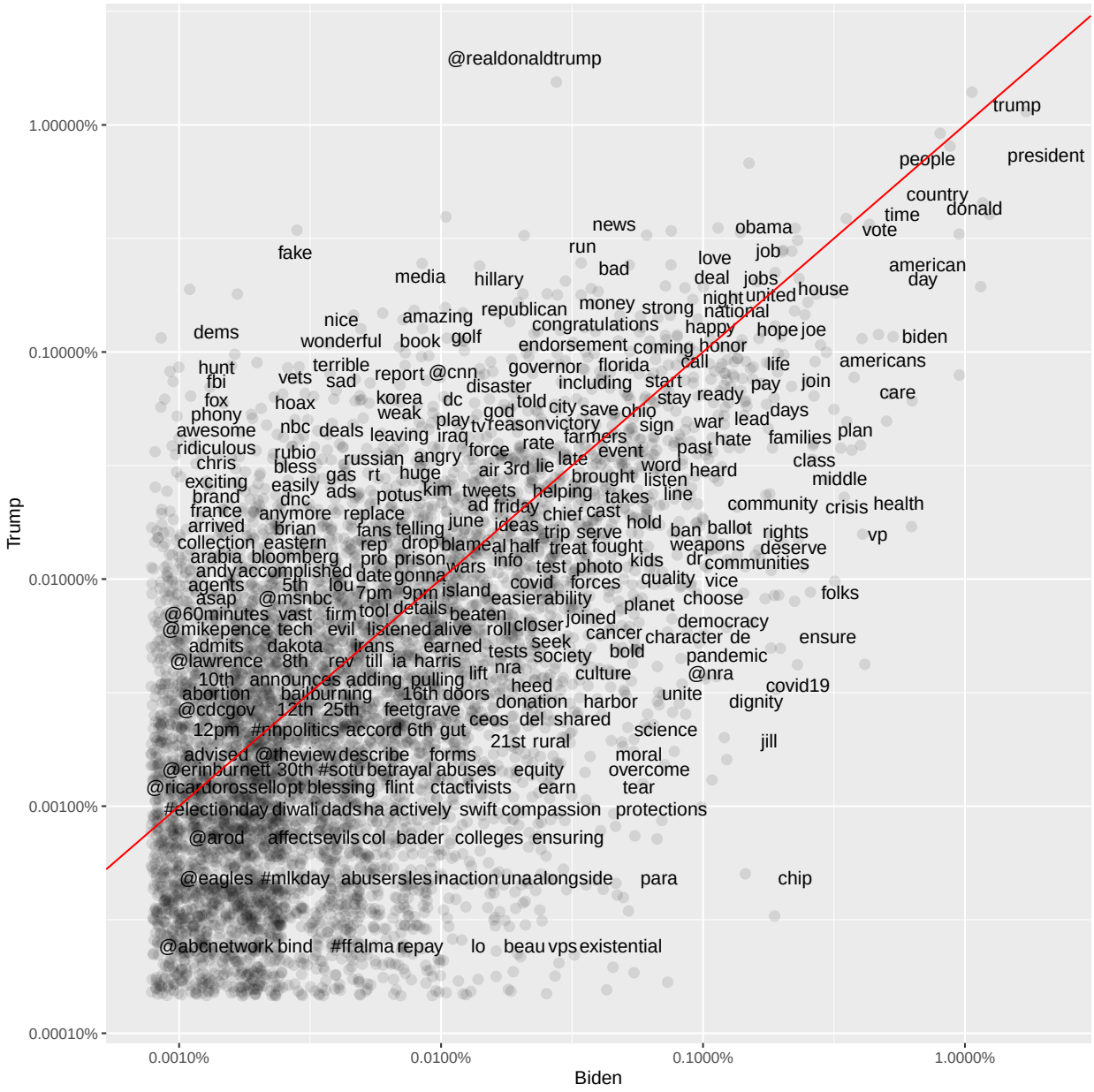
```
bt_frequency <- bt_tweets_tidy %>%
  group_by(person) %>%
  count(word, sort = TRUE) %>%
  left_join(bt_tweets_tidy %>%
              group_by(person) %>%
              summarise(total = n())) %>%
  mutate(freq = n/total)
bt_frequency
```

```
## # A tibble: 53,915 x 5
## # Groups:   person [2]
##    person word                n  total    freq
##    <chr>  <chr>           <int>  <int>   <dbl>
##  1 Trump  @realdonaldtrump  8121 384723 0.0211
##  2 Trump  trump             5075 384723 0.0132
##  3 Trump  president         3036 384723 0.00789
##  4 Trump  people            2941 384723 0.00764
##  5 Trump  country           2054 384723 0.00534
##  6 Trump  america           1850 384723 0.00481
##  7 Trump  donald            1778 384723 0.00462
##  8 Trump  time              1670 384723 0.00434
##  9 Trump  news              1511 384723 0.00393
## 10 Trump  obama             1476 384723 0.00384
## # ... with 53,905 more rows
```

```
bt_frequency <- bt_frequency %>% select(person, word, freq) %>%
          spread(person, freq) %>%
          arrange(Biden, Trump)
```

From this graph below, the words show up near the read line indicate similar frequency in both Twitter account, the points toward the top mean the words show up more in Trump's Twitter account, while the points toward the right mean the words show up more in Biden's Twitter account. From this graph alone, we can already spot some vocabulary differances, such as "fake" & "gender"

```
bt_frequency %>% ggplot(aes(Biden, Trump)) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.25, height = 0.25) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  geom_abline(color = "red")
```



However, Biden's Twitter account is barely active prior to his announcement of 2020 election, so let's see the frequency plot after 2019/04/25. Again, you see "fake" still falls above the redline and "climate" at the bottem right.

```
bt_tweets_tidy_campaign <- bt_tweets_tidy %>% filter(timestamp >= as.Date("2019-04-25"))
frequ_2019 <- bt_tweets_tidy_campaign %>%
  group_by(person) %>%
  count(word, sort = TRUE) %>%
  left_join(bt_tweets_tidy %>%
              group_by(person) %>%
              summarise(total = n())) %>%
  mutate(freq = n/total)

frequ_2019 <- frequ_2019 %>% select(person, word, freq) %>%
```

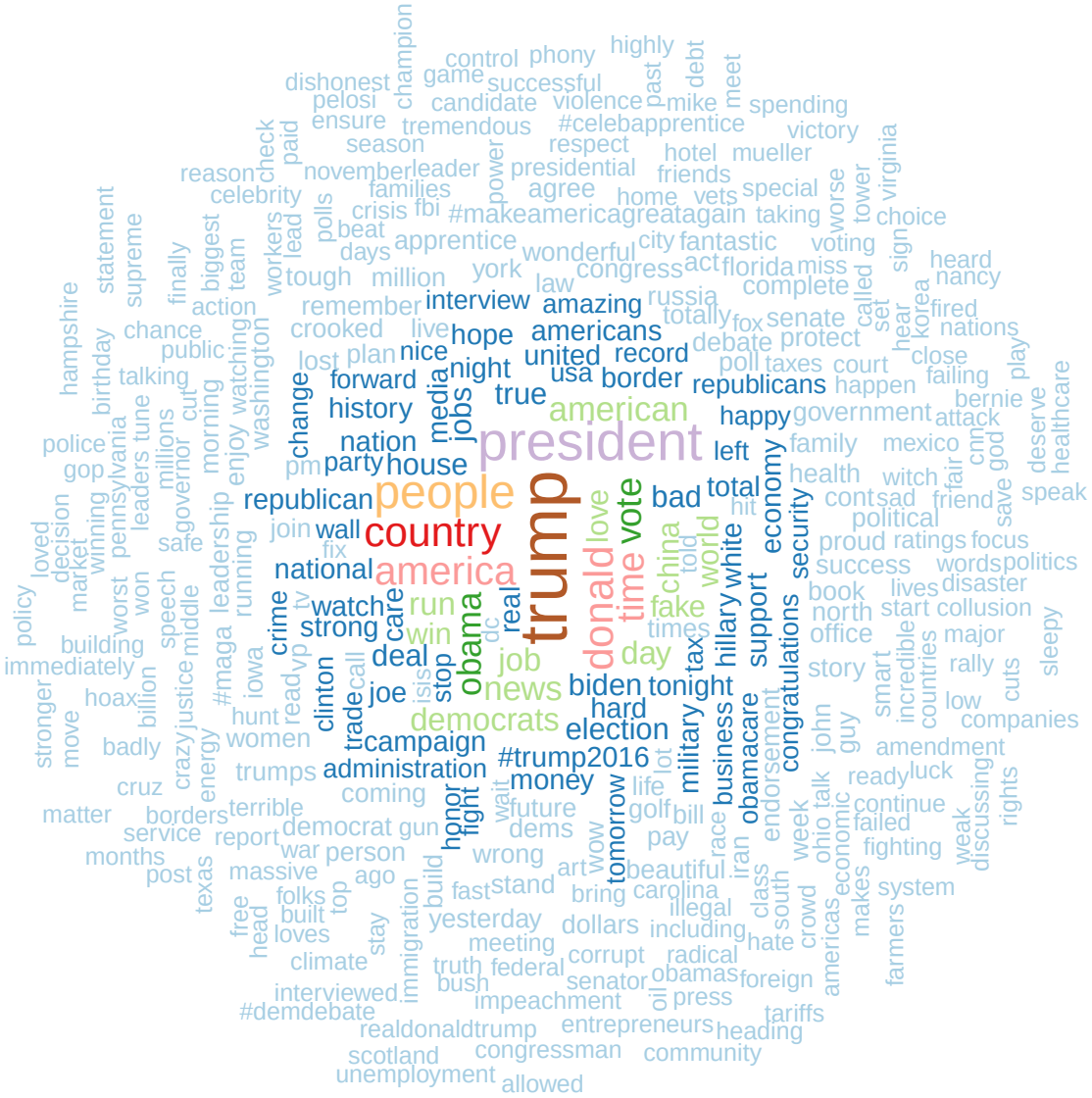
```
  spread(person, freq) %>%
  arrange(Biden, Trump)
```

```
frequ_2019 %>% ggplot(aes(Biden, Trump)) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.25, height = 0.25) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  geom_abline(color = "red")
```

Wordcloud

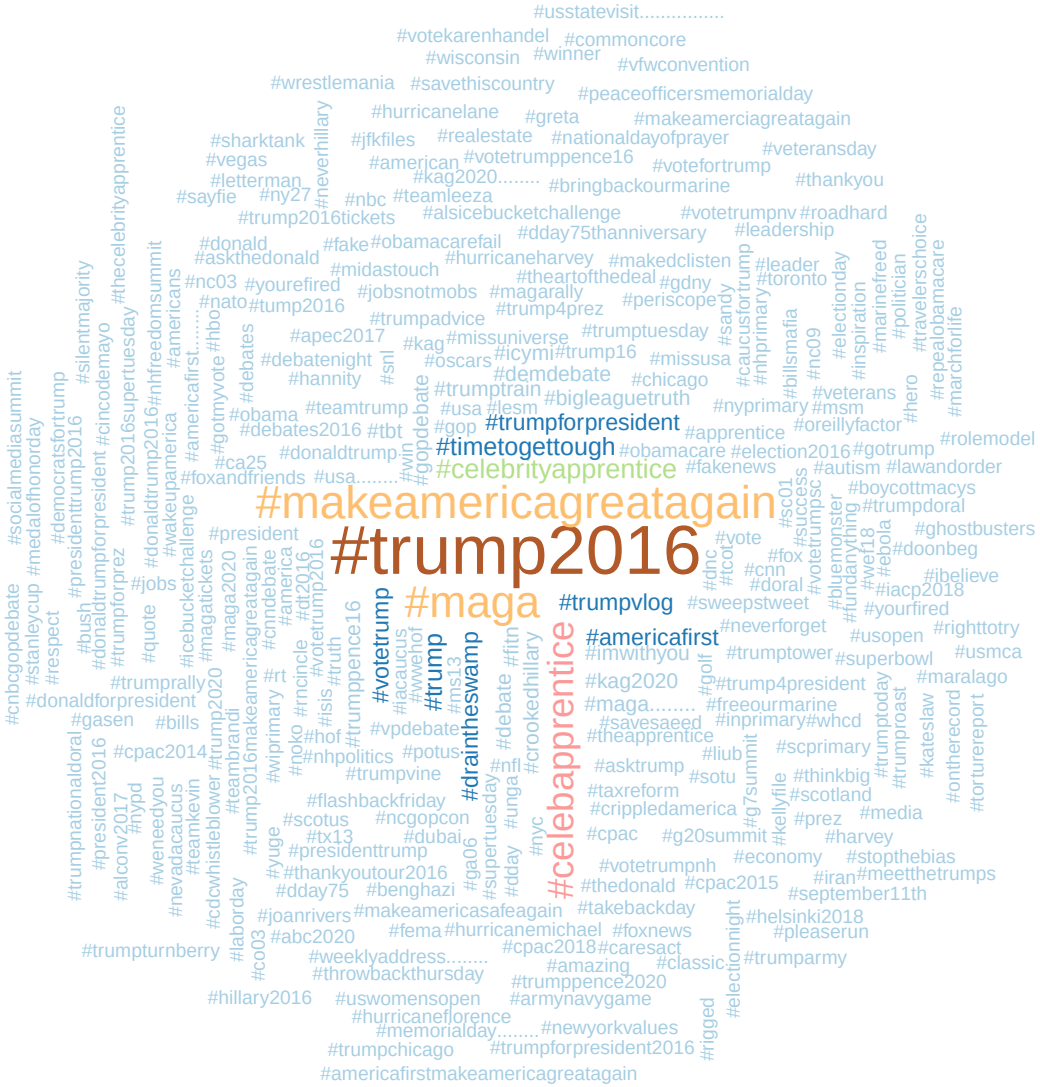
1. Total word counts This graph represents the overall word counts in the dataset, closer to the center means that particular word appear the more than the other and the size of the word shows the magnitude of the word appearance, in this case "trump" and "president" are the most used word.

```
bt_tweets_tidy %>%
  group_by(word) %>%
  filter(!str_detect(word, "^@")) %>%
  count() %>%
with(wordcloud(word, n, min.freq = 200, max.word = 400,
               rot.per =0.35, random.order = FALSE,
          colors = brewer.pal(12, "Paired"), scale = c(3,1)))
```

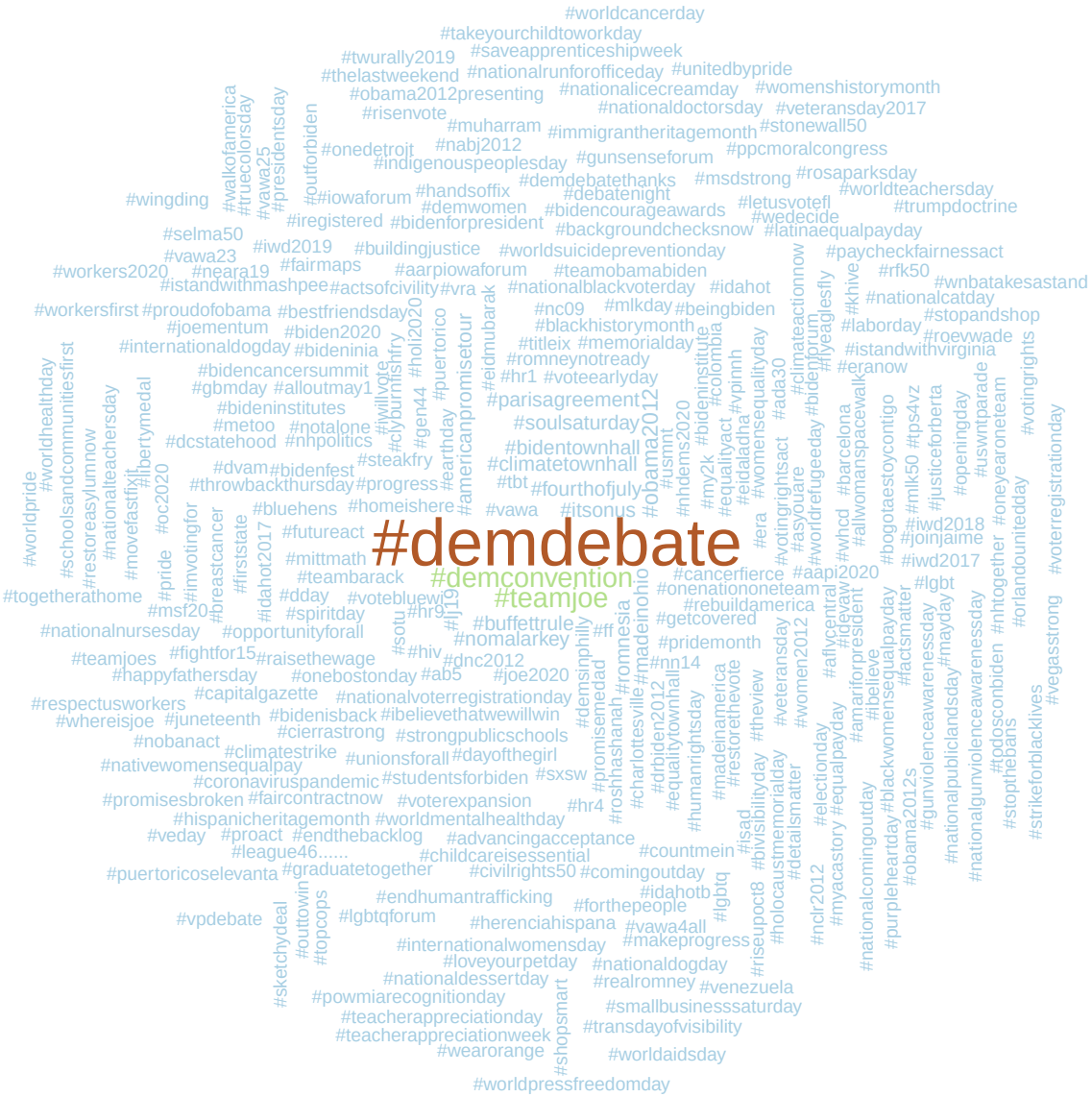


2. Hashtags These two graphs represent the hastag counts in Trump's and Biden's Twitter accounts, not surprisingly Trump's most used hashtag is his 2016 presidential campaign "#trump2016", whereas Biden is Democrat's debate "#demdebate", I would expect "teamjoe" be the most used hashtag.

```
bt_tweets_tidy %>%
  filter(person == "Trump") %>%
  group_by(word) %>%
  filter(str_detect(word, "#")) %>%
  count() %>%
  with(wordcloud(word,n, min.freq = 1, max.word = 300,
                 rot.per =0.35, random.order = FALSE,
                 colors = brewer.pal(12, "Paired"), scale = c(3,0.8)))
```

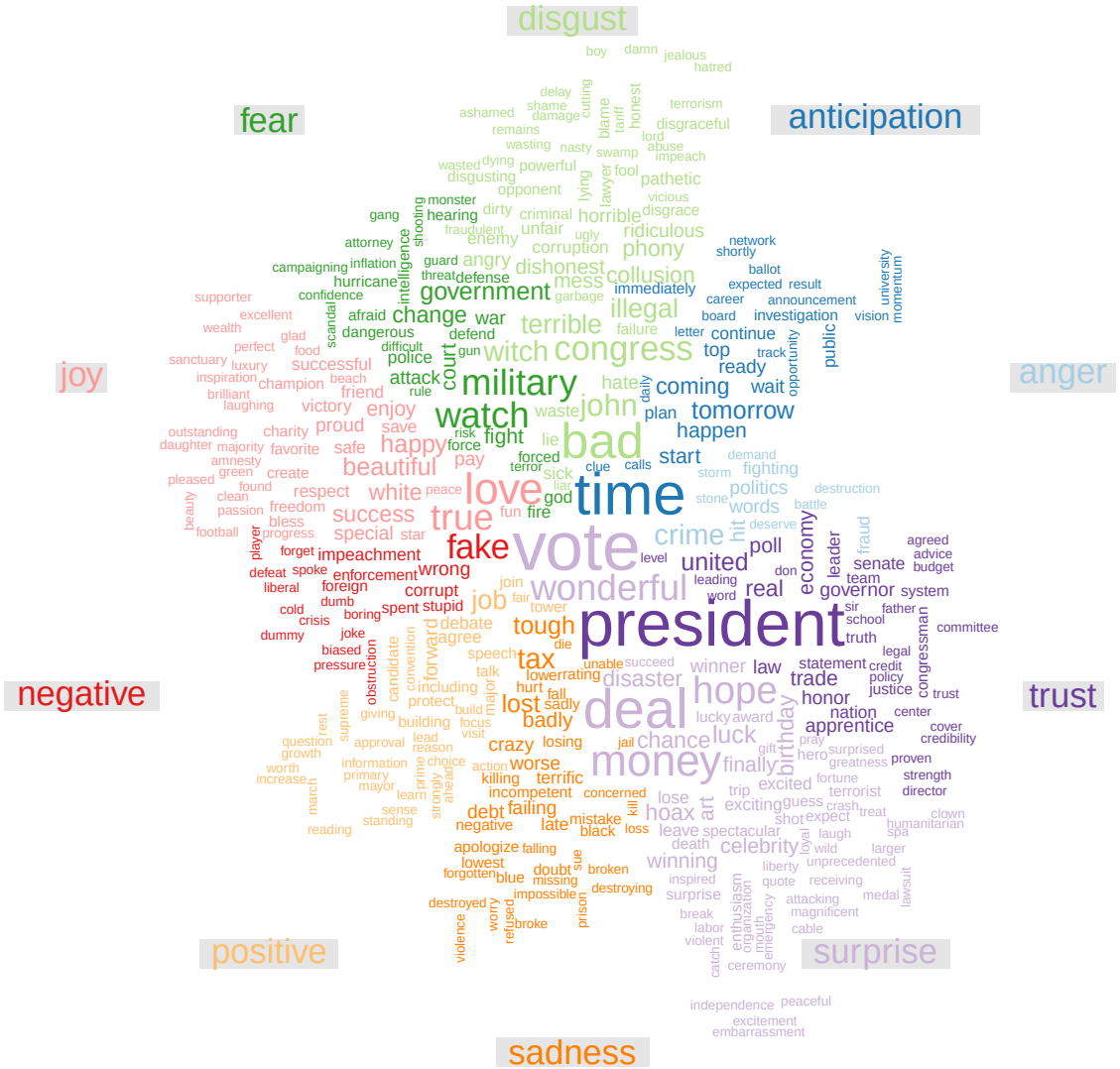


```
bt_tweets_tidy %>%
  filter(person == "Biden") %>%
  group_by(word) %>%
  filter(str_detect(word, "#")) %>%
  count() %>%
  with(wordcloud(word,n, min.freq = 1, max.word = 300,
                 rot.per =0.35, random.order = FALSE,
                 colors = brewer.pal(12, "Paired"), scale = c(3,0.8)))
```



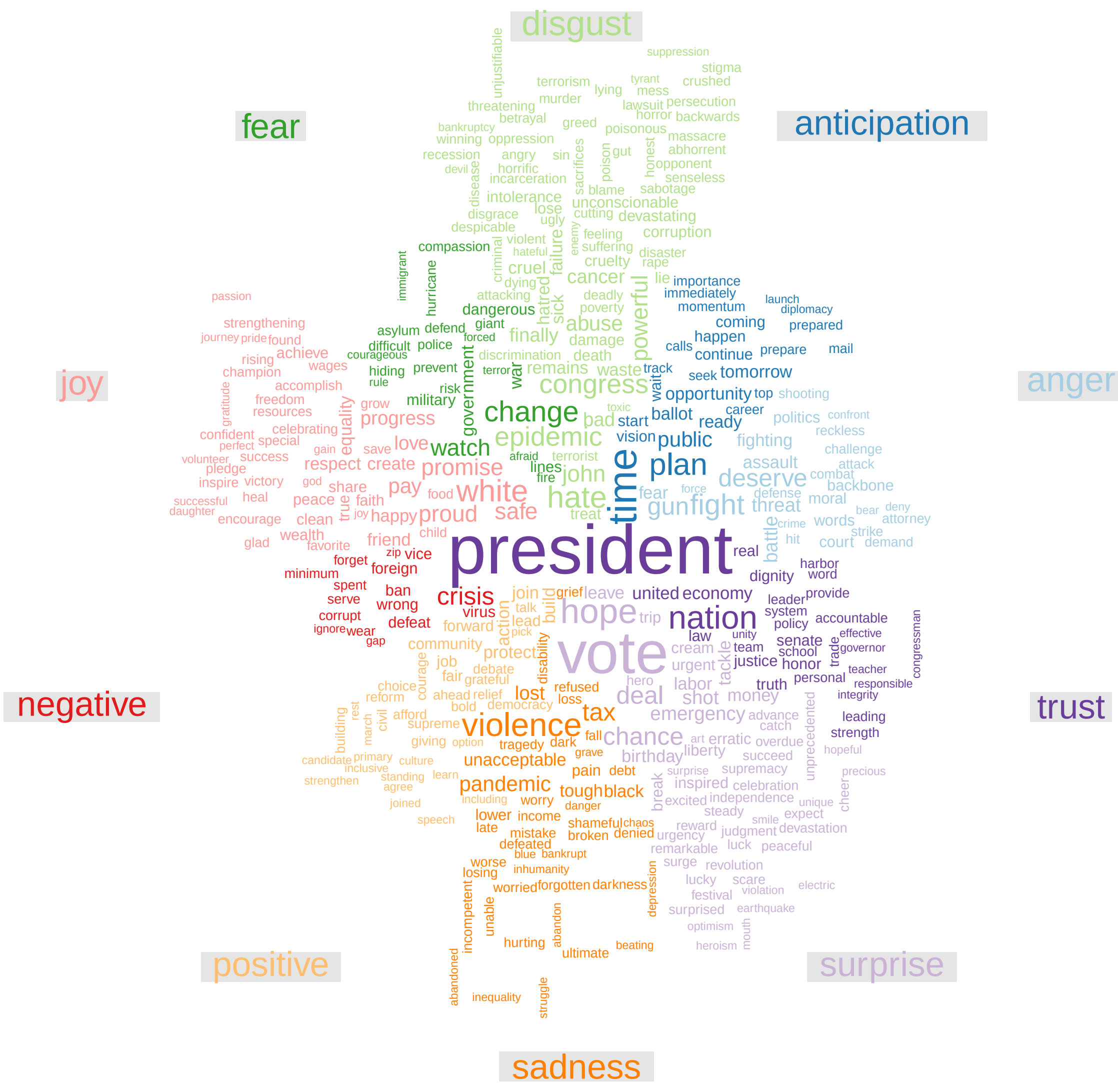
3. Separation in sentiments Here I tried a slightly fancier word cloud that separate the words base on the sentiments in NRC, graph 1 represents the sentiment word counts in Trump's account, and graph 2 represents Biden's. "vote", "president" and "time" are all quite relevant in both account, but why "vote" is assigned to surprise is beyond my understanding.

```
# trump words
bt_tweets_tidy %>%
  filter(person == "Trump" & !word == "trump") %>%
  inner_join(get_sentiments("nrc")) %>%
  count(word, sentiment, sort = TRUE) %>%
  acast(word ~ sentiment, value.var = "n", fill =0) %>%
  comparison.cloud(colors = brewer.pal(10, "Paired"), max.words = 400,
                   match.colors = TRUE, title.size = 1.5,
                   random.order = FALSE, scale = c(3,0.5))
```




```
# biden words
bt_tweets_tidy %>%
  filter(person == "Biden" & !word == "trump") %>%
```

```r
inner_join(get_sentiments("nrc")) %>%
count(word, sentiment, sort = TRUE) %>%
acast(word ~ sentiment, value.var = "n", fill =0) %>%
comparison.cloud(colors = brewer.pal(10, "Paired"), max.words = 400,
                 match.colors = TRUE, title.size = 1.5,
                 random.order = FALSE, scale = c(3,0.5))
```



Odds ratio

In order to know which word is more likely to be used in either Twitter account, we will find the odds ratio for each words and here we use the timeframe after Biden announced his presidential campaign. After ungroup, we spread the data by person.

```
bt_words_ratio <- bt_tweets_tidy_campaign %>%
  filter(!str_detect(word, "^@")) %>% # remove tags on individual accounts
  count(word, person) %>%
  group_by(word) %>%
  filter(sum(n) >= 10) %>% # only keep the words that are used more than 10 times
  ungroup() %>%
  spread(person, n, fill = 0) %>%
  mutate_if(is.numeric, list(~(. + 1) / (sum(.) + 1))) %>%
  mutate(logratio = log(Biden / Trump)) %>%
  arrange(desc(logratio))
```

Here it's just a demonstration of how to calculate the odds ratio manunally.

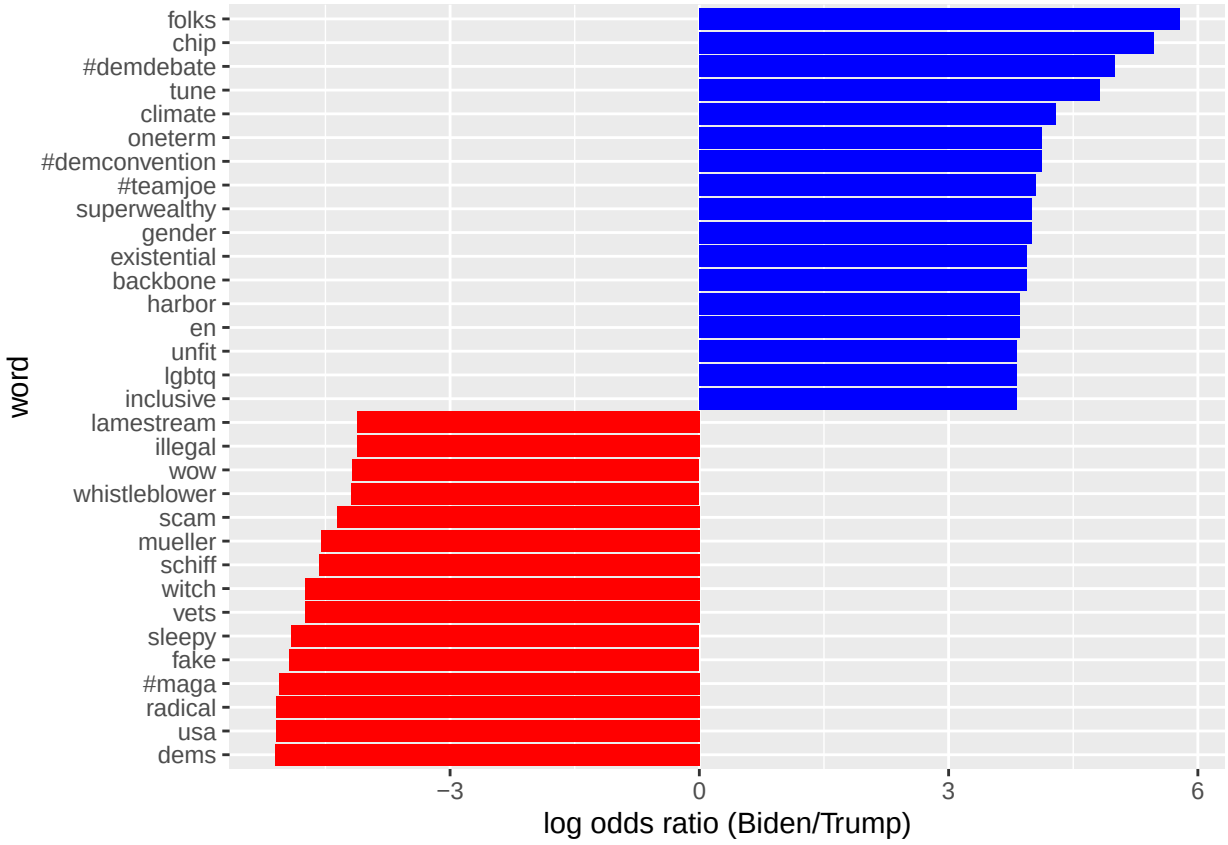
```
bt_tweets_tidy_campaign %>%
  filter(!str_detect(word, "^@")) %>%
  count(word, person) %>%
  group_by(word) %>%
  filter(sum(n) >= 10) %>%
  ungroup() %>%
  # use ungroup here is because if use summarise the data would automatically reduce the dimension
  spread(person, n, fill = 0) %>%
  mutate(biden_t = sum(Biden),
         trump_t = sum(Trump),
         br = (Biden+1)/(biden_t+1),
         tr = (Trump+1)/(trump_t+1),
         logr = log(br/tr)) %>%
  arrange(abs(logr))
```

```
## # A tibble: 2,488 x 8
##    word      Biden Trump biden_t trump_t       br       tr     logr
##    <chr>     <dbl> <dbl>   <dbl>   <dbl>    <dbl>    <dbl>    <dbl>
##  1 recovery     20    30   49467   72935 0.000425 0.000425 -0.00121
##  2 lot          43    64   49467   72935 0.000889 0.000891 -0.00194
##  3 words        44    66   49467   72935 0.000910 0.000919 -0.00977
##  4 happening    34    50   49467   72935 0.000708 0.000699  0.0118
##  5 forgotten    10    15   49467   72935 0.000222 0.000219  0.0136
##  6 july         10    15   49467   72935 0.000222 0.000219  0.0136
##  7 pushing      10    15   49467   72935 0.000222 0.000219  0.0136
##  8 refuse       10    15   49467   72935 0.000222 0.000219  0.0136
##  9 task         10    15   49467   72935 0.000222 0.000219  0.0136
## 10 fighting     50    73   49467   72935 0.00103  0.00101   0.0160
## # ... with 2,478 more rows
```

From the log odds ratio graph, I pick the top 15 words in both Twitter accounts, not surprisingly Biden's account is more likely to use vocabulary such as "climate", "gender", "lgbtq", "inclusive", where as Trump's account used a vocabulary like "wow", "lamestream", "sleepy", "fake", and most obviously "#maga".

```
## the words more likely from the otherside
group_colors <- c("FALSE" = colors()[26], "TRUE" = colors()[552])

bt_words_ratio %>%
  group_by(logratio < 0) %>%
```

```
  top_n(15, abs(logratio)) %>%
  ungroup() %>%
  mutate(word = reorder(word, logratio)) %>%
  ggplot(aes(word, logratio, fill = `logratio < 0`)) +
  geom_col(show.legend = FALSE) +
  scale_fill_manual(values = group_colors) +
  coord_flip() +
  ylab("log odds ratio (Biden/Trump)")
```



Retweets and likes

1. Retweets Let's see what sort of words that give the highest retweets for Trump and Biden. First, check the total rtweets from both accounts. The first group_by() and summarise() count how many retweets of each tweets id, the second group_by() and sumarise() sum up the whole retweets for each account.

```
bt_totals <- bt_tweets_tidy %>%
  group_by(person, id) %>%
  summarise(rts = first(retweets)) %>%
  group_by(person) %>%
  summarise(total_rts = sum(rts))

bt_totals
```

```
## # A tibble: 2 x 2
```

```
##   person total_rts
##   <chr>      <dbl>
## 1 Biden   37778753
## 2 Trump  306628537
```

Now we have the total retweets ready, we want to find the median of retweets of each word. The frist group_by() and summarise() show how many retweets does each word has for each tweet id and person. The second group_by() and summarise() show the median or retweets for each word and the total usage of each word. (so if the word is used only once, the median retweets is the same as the maximum and minum retweets.)

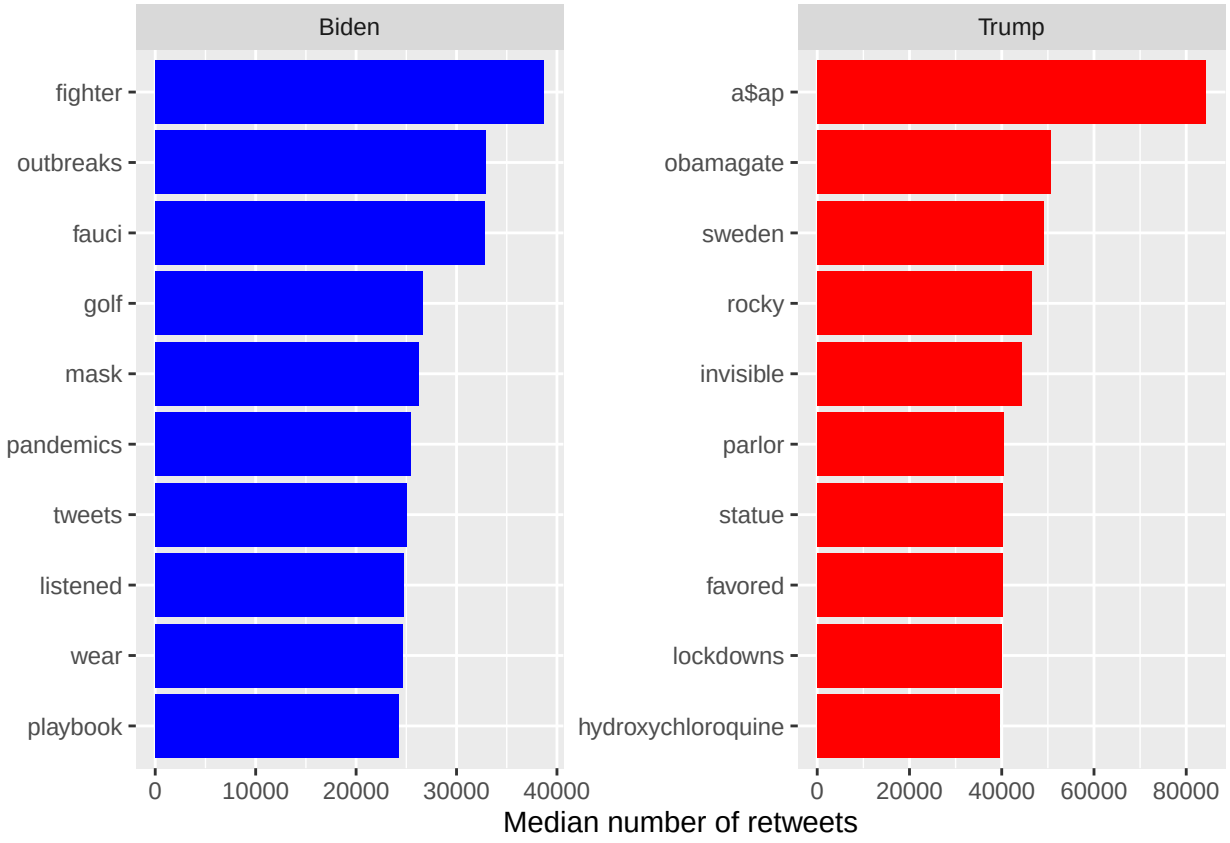
```
bt_word_by_rts <- bt_tweets_tidy %>%
  group_by(id, person, word) %>%
  summarise(rts = first(retweets)) %>%
  group_by(person, word) %>%
  summarise(retweets = median(rts), uses =n()) %>%
  left_join(bt_totals) %>%
  filter(retweets != 0) %>%
  ungroup()

bt_word_by_rts
```

```
## # A tibble: 53,589 x 5
##    person word            retweets  uses total_rts
##    <chr>  <chr>              <dbl> <int>     <dbl>
##  1 Biden  @32bjseiu            263     1  37778753
##  2 Biden  @60minutes           652     1  37778753
##  3 Biden  @abby4iowa          1000     1  37778753
##  4 Biden  @abc                9296     1  37778753
##  5 Biden  @abcnetwork          751     1  37778753
##  6 Biden  @abeshinzo          7127     1  37778753
##  7 Biden  @adamsmithtimes       19     1  37778753
##  8 Biden  @adybarkan          6589     1  37778753
##  9 Biden  @aflcio              423     1  37778753
## 10 Biden  @afscme               46     1  37778753
## # ... with 53,579 more rows
```

The following graph shows the median number of retweets for each word after filter the use for at least 5 times. It's funny that words related to A$AP Rocky incident in Sweden are the most retweets for Trump, in the other hand Biden's words are link with COVID19.

```
group.colors <- c(Trump = colors()[552], Biden = colors()[26])

bt_word_by_rts %>%
  filter(uses >= 5) %>%
  group_by(person) %>%
  top_n(10, retweets) %>%
  arrange(retweets) %>%
  ungroup() %>%
  mutate(word = factor(word, unique(word))) %>%
  ungroup() %>%
  ggplot(aes(word, retweets, fill = person)) +
  geom_col(show.legend = FALSE) +
```

```
  scale_fill_manual(values = group.colors) +
  facet_wrap(~person, scales = "free", ncol = 2) +
  coord_flip() +
  labs(x = NULL, y = "Median number of retweets")
```



2. Likes The strategy to get check the what word gives the highest median of retweets is the same as the one we used above.

```
bt_totals_fav <- bt_tweets_tidy %>%
  group_by(person, id) %>%
  summarise(fav = first(likes)) %>%
  group_by(person) %>%
  summarise(total_favs = sum(fav))

bt_totals_fav
```

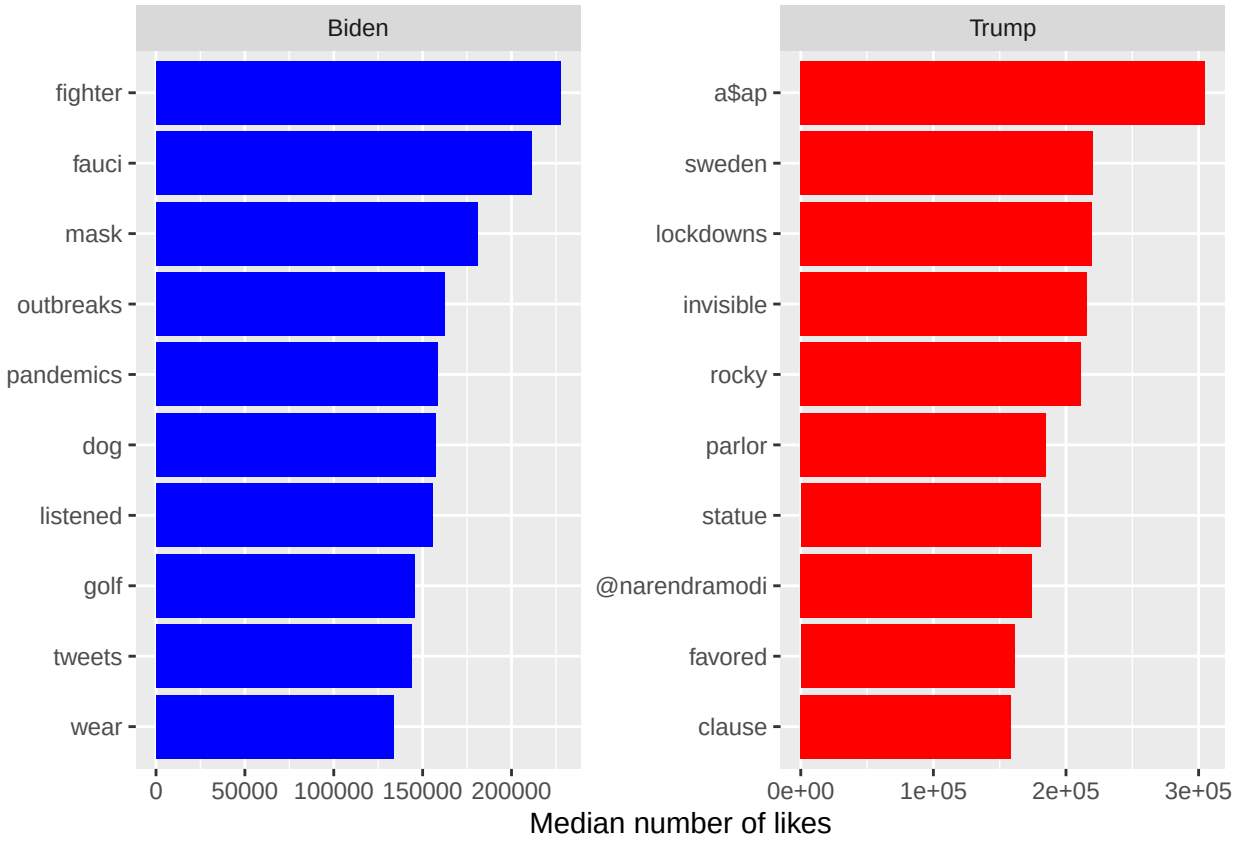
```
## # A tibble: 2 x 2
##   person total_favs
##   <chr>       <dbl>
## 1 Biden   206517140
## 2 Trump  1320390366
```

The results are more or less the same as retweets in both accounts, where words relate to A$ap Rocky earned the most likes for Trump, and COVID19 related words got the most likes for Biden.

```r
bt_word_by_fav <- bt_tweets_tidy %>%
  group_by(id, person, word) %>%
  summarise(fav = first(likes)) %>%
  group_by(person, word) %>%
  summarise(favs = median(fav), uses =n()) %>%
  left_join(bt_totals_fav) %>%
  filter(favs != 0) %>%
  ungroup()

bt_word_by_fav
```

```
## # A tibble: 53,149 x 5
##    person word             favs  uses total_favs
##    <chr>  <chr>           <dbl> <int>      <dbl>
##  1 Biden  @32bjseiu        1314     1  206517140
##  2 Biden  @60minutes       3705     1  206517140
##  3 Biden  @abby4iowa       6152     1  206517140
##  4 Biden  @abc            89310     1  206517140
##  5 Biden  @abcnetwork      7303     1  206517140
##  6 Biden  @abeshinzo      39912     1  206517140
##  7 Biden  @adamsmithtimes     7     1  206517140
##  8 Biden  @adybarkan      33607     1  206517140
##  9 Biden  @aflcio          1986     1  206517140
## 10 Biden  @afscme             7     1  206517140
## # ... with 53,139 more rows
```

```r
bt_word_by_fav %>%
  filter(uses >= 5) %>%
  group_by(person) %>%
  top_n(10, favs) %>%
  arrange(favs) %>%
  ungroup() %>%
  mutate(word = factor(word, unique(word))) %>%
  ungroup() %>%
  ggplot(aes(word, favs, fill = person)) +
  geom_col(show.legend = FALSE) +
  scale_fill_manual(values = group.colors) +
  facet_wrap(~person, scales = "free", ncol = 2) +
  coord_flip() +
  labs(x = NULL, y = "Median number of likes")
```



Changes in word use

1. Data prepping So now we would look at how words' frequencies change overtime. To do that, we first create a new time variable to get the unit of time to calculate the word in that time span. Here, I choose 1 month as the base, in floor_date() you can set your unit of time in different measures. The count() counts how many times the word is use in 1 month then follow by a group_by() to account unit of time in order to use a mutate() to attach the total amount of word used in 1 month, the second group_by() accounts word so that mutate() can attach a total count of word use by that person.

```
bt_words_by_time <- bt_tweets_tidy_campaign %>%
  filter(!str_detect(word, "^@")) %>% # to filter out account taging in Twitter
  mutate(time_floor = floor_date(timestamp, unit = "1 month")) %>%
  count(time_floor, person, word) %>%
  group_by(person, time_floor) %>%
  mutate(time_total = sum(n)) %>%
  group_by(person, word) %>%
  mutate(word_total = sum(n)) %>%
  ungroup() %>%
  rename(count = n) %>%
  filter(word_total > 50)
```

2. Nested Model Now the dataset is ready to be nested to run our model. Since we trying to find if the word useage change over time, it's ideal to use a nested dataset, so in each word it nested a samller dataset

that contains the “time_floor”, “count”, “time_total”, and “word_total”. Here we use glm and binomial for our model.

```
bt_nested_data <- bt_words_by_time %>%
  group_by(person, word) %>%
  nest()

bt_nested_data
```

```
## # A tibble: 507 x 3
## # Groups:   person, word [507]
##    person word       data
##    <chr>  <chr>      <list>
##  1 Biden  act        <tibble [19 x 4]>
##  2 Biden  address    <tibble [16 x 4]>
##  3 Biden  affordable <tibble [19 x 4]>
##  4 Biden  america    <tibble [20 x 4]>
##  5 Biden  american   <tibble [20 x 4]>
##  6 Biden  americans  <tibble [20 x 4]>
##  7 Biden  battle     <tibble [18 x 4]>
##  8 Biden  bring      <tibble [16 x 4]>
##  9 Biden  build      <tibble [20 x 4]>
## 10 Biden  campaign   <tibble [19 x 4]>
## # ... with 497 more rows
```

```
model <- function(x){
  glm(cbind(count, time_total) ~ time_floor, data = x, family = "binomial")
}

# map(bt_nested_data$data, model), I forgot how to use map() so put it here for reference

bt_nested_models <- bt_nested_data %>%
  mutate(models = map(data, model))

bt_nested_models
```

```
## # A tibble: 507 x 4
## # Groups:   person, word [507]
##    person word       data              models
##    <chr>  <chr>      <list>            <list>
##  1 Biden  act        <tibble [19 x 4]> <glm>
##  2 Biden  address    <tibble [16 x 4]> <glm>
##  3 Biden  affordable <tibble [19 x 4]> <glm>
##  4 Biden  america    <tibble [20 x 4]> <glm>
##  5 Biden  american   <tibble [20 x 4]> <glm>
##  6 Biden  americans  <tibble [20 x 4]> <glm>
##  7 Biden  battle     <tibble [18 x 4]> <glm>
##  8 Biden  bring      <tibble [16 x 4]> <glm>
##  9 Biden  build      <tibble [20 x 4]> <glm>
## 10 Biden  campaign   <tibble [19 x 4]> <glm>
## # ... with 497 more rows
```

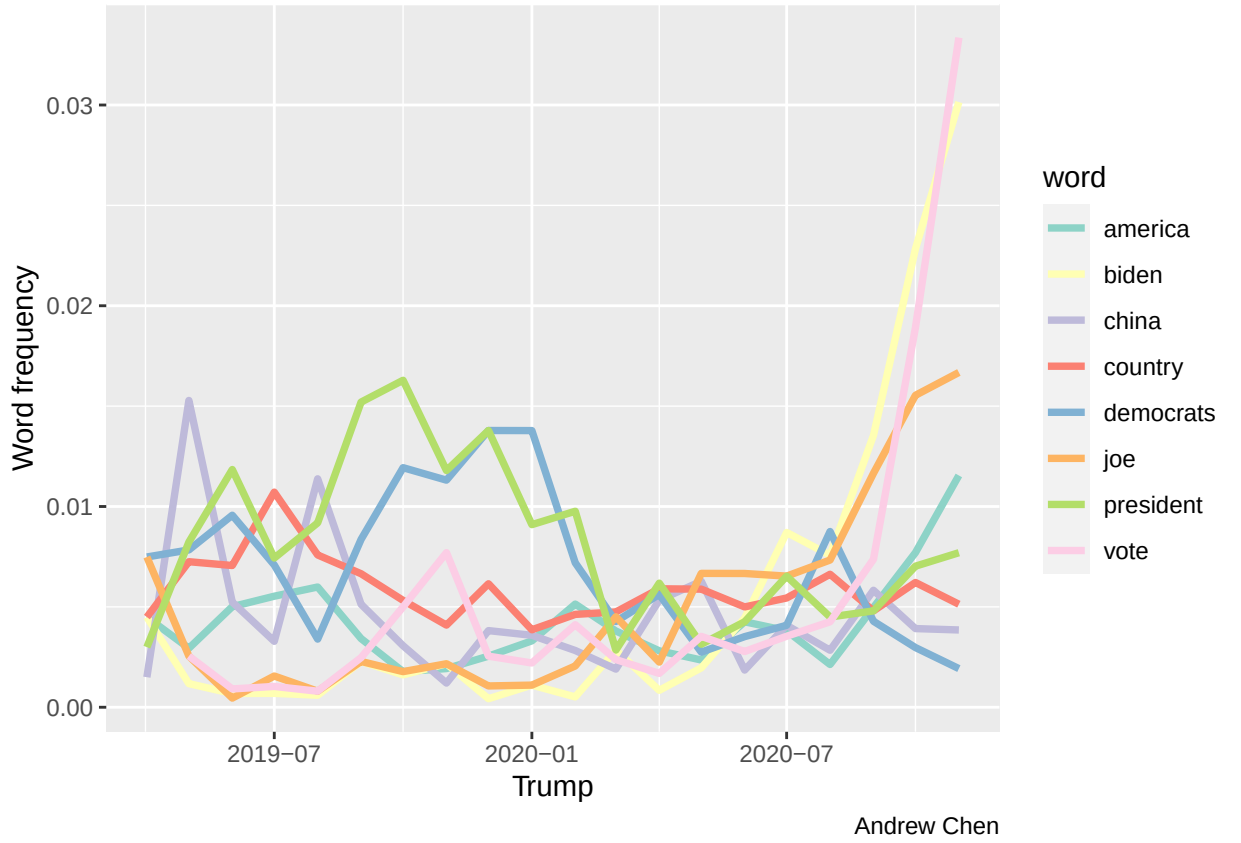
```
### different way to run glm over nested data
# bt_nested_models <- bt_nested_data %>%
#  mutate(models = map(data, ~glm(cbind(count, time_total) ~ time_floor,
#                       ., family = "binomial")))
###
```

From above the bt_nested_models is has another list attach in the dataframe label as models, now we want to extract the componets in the glm model, here we can use a powerful function tidy() to summarize information in the model then unnest that models column. Since we are comparing multiple p values best to adjust them!

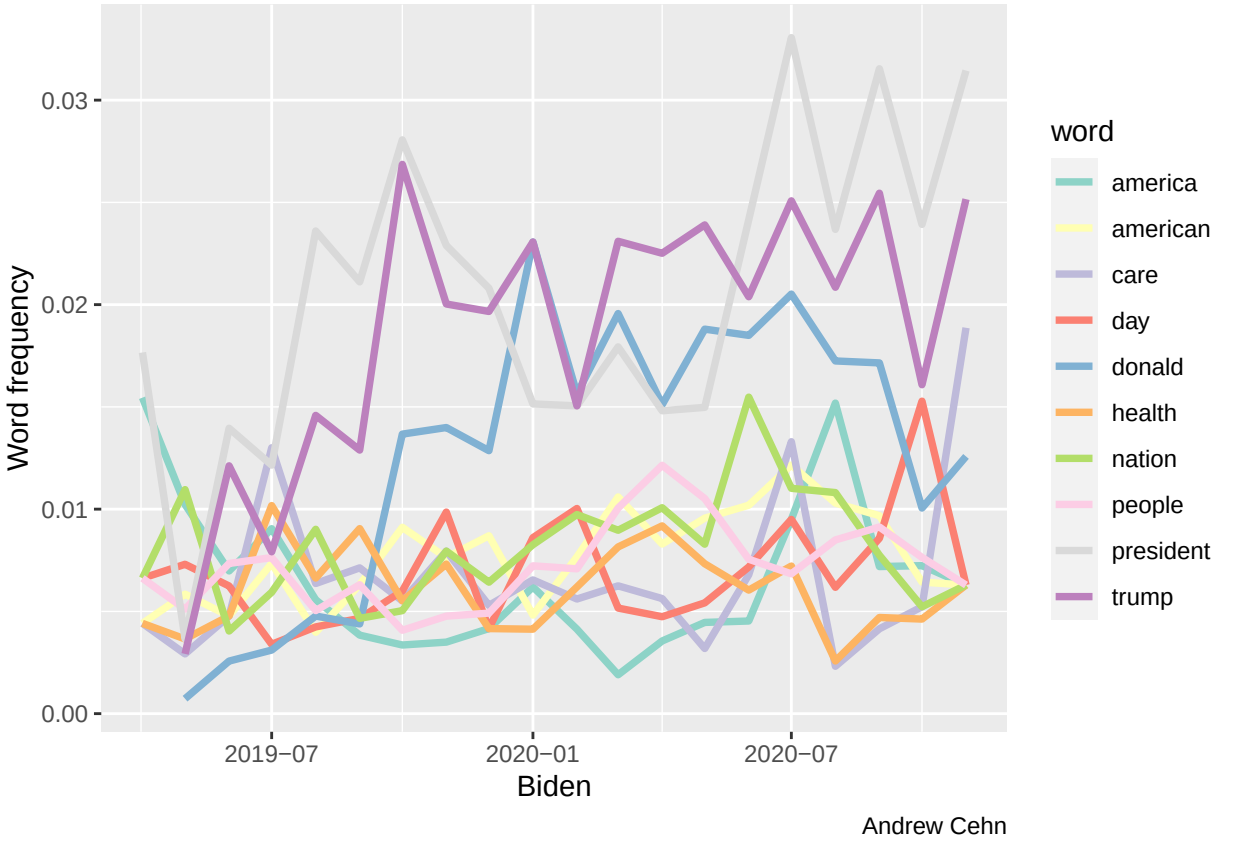
```
slopes <- bt_nested_models %>%
  mutate(models = map(models, tidy)) %>%
  unnest(cols = c(models)) %>%
  filter(term == "time_floor") %>%
  mutate(adjusted.p.value = p.adjust(p.value))

top_slopes <- slopes %>% filter(adjusted.p.value <0.05)
```

3. Graphing From the graphs below, both illustrate the trendy words that appeared more than 350 times in the period of 2019-04-25 to now. In the trump graph, it is obvious that closer to the election date the mentioning of biden and vote skyrocket compare to other words. In the other hand, biden graph shows that "trump" and "president" maintian rather stable high freqencies.

```
bt_words_by_time %>%
  inner_join(top_slopes, by = c("word", "person")) %>%
  filter(person == "Trump" &
         word_total > 350) %>%
  ggplot(aes(time_floor, count/time_total, color = word)) +
  scale_color_brewer(palette = "Set3") +
  geom_line(size = 1.3) +
  labs(x = "Trump", y = "Word frequency", caption = "Andrew Chen")
```




```
bt_words_by_time %>%
  inner_join(top_slopes, by = c("word", "person")) %>%
  filter(person == "Biden" &
           word_total > 350) %>%
  ggplot(aes(time_floor, count/time_total, color = word)) +
  scale_color_brewer(palette = "Set3") +
  geom_line(size = 1.3) +
  labs(x = "Biden", y = "Word frequency", caption = "Andrew Cehn")
```



Whole lotta sentiments

We briefly touch on sentiments in the wordcloud section, here we going to add in more sentiment analysis. First, I want to check what is the overall change of sentiment through time in both Twitter accounts. Second, I will show what are the vocabularies they use corrorspond to the positive and negative sentiments. Last, I will inspect the aggregate positive and negative sentiment over time.

1.Data prepping Since we want to see the how sentiment shifts since the beginning of the first tweet, we need to make some changes to the original bt_tweets_tidy dataset above. I arrange the dataset according to timestamp then create an index for each row, and here we used a custom stopword "trump" to filter out trump, because Bing sentiment take "trump" as an positive vocabulary, which would show overly positive bias in both account, becuase they both mentioned "Trump" a lot of times.

```
custom_stop_words <- bind_rows(tibble(word = c("trump"),
                                      lexicon = c("custom")), stop_words)

bt_tweets_tidy<- bt_tweets %>%
  arrange(timestamp) %>%
  mutate(tweetnumber = row_number()) %>%
  filter(!str_detect(tweet, "^RT")) %>%
  mutate(text = str_remove_all(tweet, remove_reg),
         text = str_remove_all(tweet, "\\s?(f|ht)(tp)(s?)(://)([^\\.]*)[\\.|/](\\S*)")) %>%
  unnest_tokens(word, text, token = "tweets") %>%
  filter(!word %in% custom_stop_words$word,
```

```
          !word %in% str_remove_all(custom_stop_words$word, "'"),
          !word %in% c("amp", "lt", "gt"),
          str_detect(word, "[a-z]"))
```

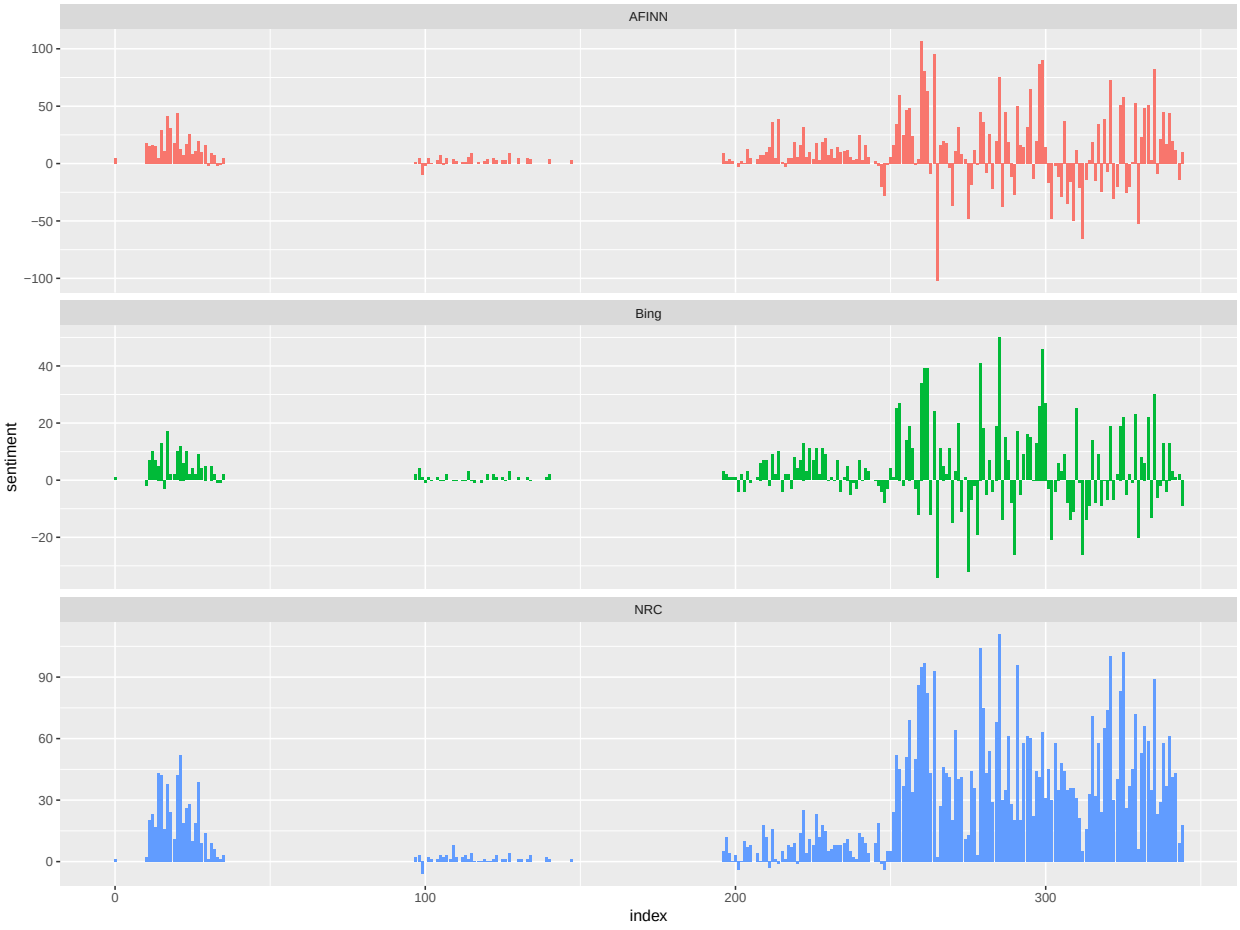
Before we get into graphing, we will need to prepare the dataset into a format we wanted. The procedure is fairly simply here, we left_join() each sentiment separately then we combine them together by rows using bind_rows(), here the index is create by uing %/%, which gaves you only the quotient, so the index take into account of 150 tweets as a bin to count the sentiment. One thing to keep in mind is that Afinn sentiment assigned words with a range of value from -5 to 5 (which is called value insted of sentiment, hence I renamed it to sentiment).

Bing sentiment is categorize in binary section of positive and negative, but Nrc has 10 different sentiments, therefore I filter for the wanted sentiments. After combining Bing and Nrc, there is an extra procedure to account for the overall sentiments of each index, which I achieved this by subtracting negative from positive.

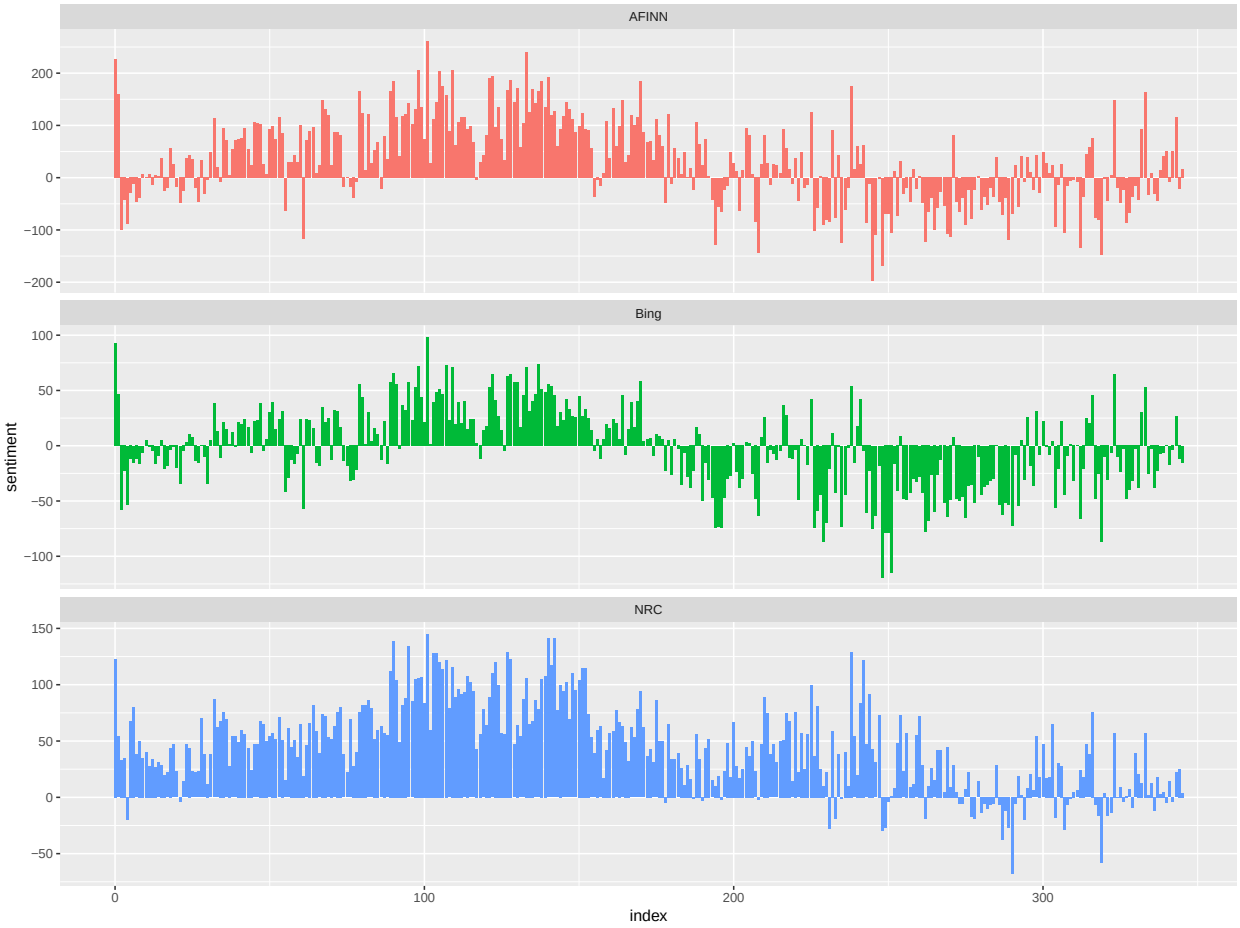
```
bt_afinn <- bt_tweets_tidy %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(person, index = tweetnumber %/% 150) %>%
  summarise(sentiment = sum(value)) %>%
  mutate(method = "AFINN")

bt_bing_and_nrc <- bind_rows(
  bt_tweets_tidy %>%
    inner_join(get_sentiments("bing")) %>%
    mutate(method = "Bing"),
  bt_tweets_tidy %>%
    inner_join(get_sentiments("nrc") %>%
                 filter(sentiment %in% c("positive",
                                         "negative"))
    ) %>%
    mutate(method = "NRC")) %>%
  count(person, method, index = tweetnumber %/% 150, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)

bt_all_sent <- bind_rows(bt_afinn, bt_bing_and_nrc)
```

2. Change of sentiment over time Now the dataset is ready to do some graph! From the graphs, Afinn and Bing both show simialr trends, however Nrc sentiment is over whelminingly positive, We can do a little check of that account. As you can see from the Nrc and Bing tibbles, you can tell NRC has less negative words and more positive words compare to Bing, which is probably the major reason why both Trump and Biden have a lot more positives sentiment when using Nrc measurement.

```
bt_all_sent %>% filter(person == "Biden") %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y")
```




```
bt_all_sent %>% filter(person == "Trump") %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y")
```




```
# checking the difference of NRC and Bing
get_sentiments("nrc") %>%
  filter(sentiment %in% c("positive", "negative")) %>%
  count(sentiment)
```

```
## # A tibble: 2 x 2
##   sentiment     n
##   <chr>     <int>
## 1 negative   3324
## 2 positive   2312
```

```
get_sentiments("bing") %>%
  count(sentiment)
```

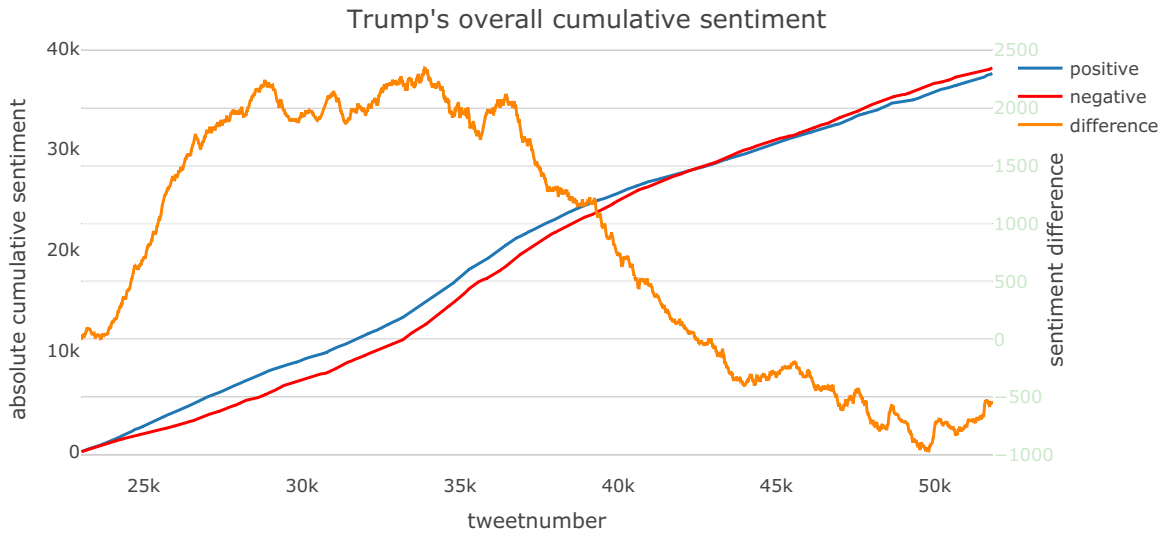
```
## # A tibble: 2 x 2
##   sentiment     n
##   <chr>     <int>
## 1 negative   4781
## 2 positive   2005
```

3. Overall cumulative sentiments This is a rather simple method, we just need to take the bt_tweets_tidy which is already prepared for graphing and inner_join() with sentiment. So how is Trump

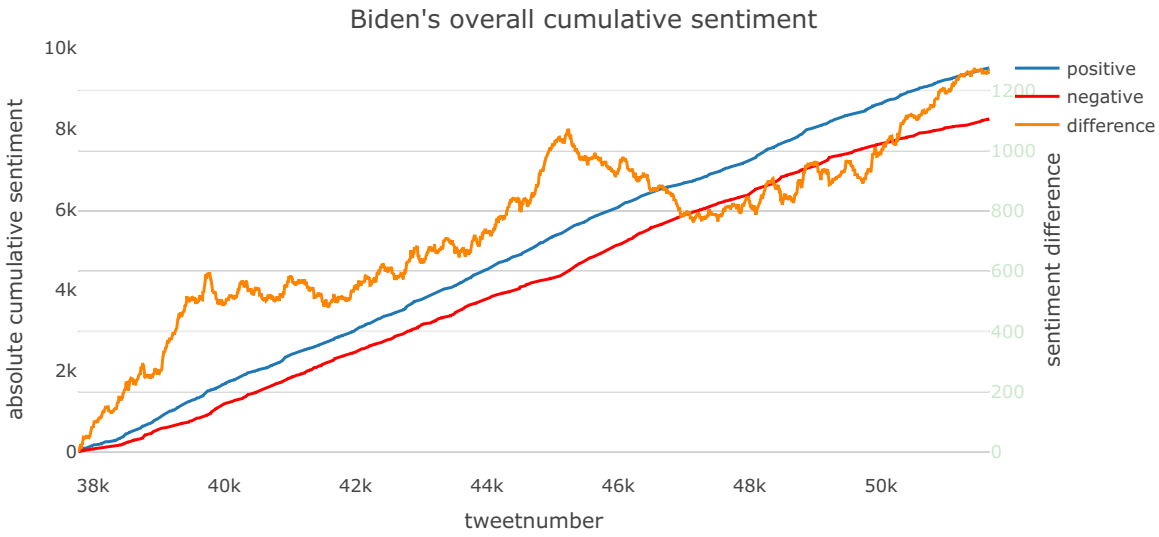
sentiments perform since his announcement of presidential campaign in 2015/06/16? It is quite obvious that over time his tweets are becoming more negative than positive where the difference between the two become negative as shown in the difference line. But what about Biden's sentiment after his announcement of his 2020 presidential campaign? The graph showed an opposite story, which the positive line diverge from the negative line early on and the positive difference is even larger in later tweets.

```
bt_tweets_sent <- bt_tweets_tidy %>%
  inner_join(get_sentiments("afinn")) %>%
  group_by(tweetnumber, timestamp, person, word) %>%
  summarise(value) %>%
  ungroup()
```

```
bt_tweets_sent %>%
  filter(person == "Trump" &
           !word %in% "trump" &
           timestamp >= "2015-06-16") %>%
  mutate(positivity = cumsum(if_else(value>0, value, 0)),
         negativity = cumsum(abs(if_else(value<0, value, 0)))) %>%
  plot_ly() %>%
  add_lines(x=~tweetnumber, y=~positivity, name='positive') %>%
  add_lines(x=~tweetnumber, y=~negativity, name='negative', color = I("red")) %>%
  add_lines(x=~tweetnumber, y=~positivity - negativity, name="difference",
            yaxis="y2", color = I("#ff8400")) %>%
  layout(
    title = "Trump's overall cumulative sentiment",
    yaxis = list(title='absolute cumulative sentiment'),
    yaxis2 = tb,
    width = 800
  ) %>%
  theme_plotly()
```



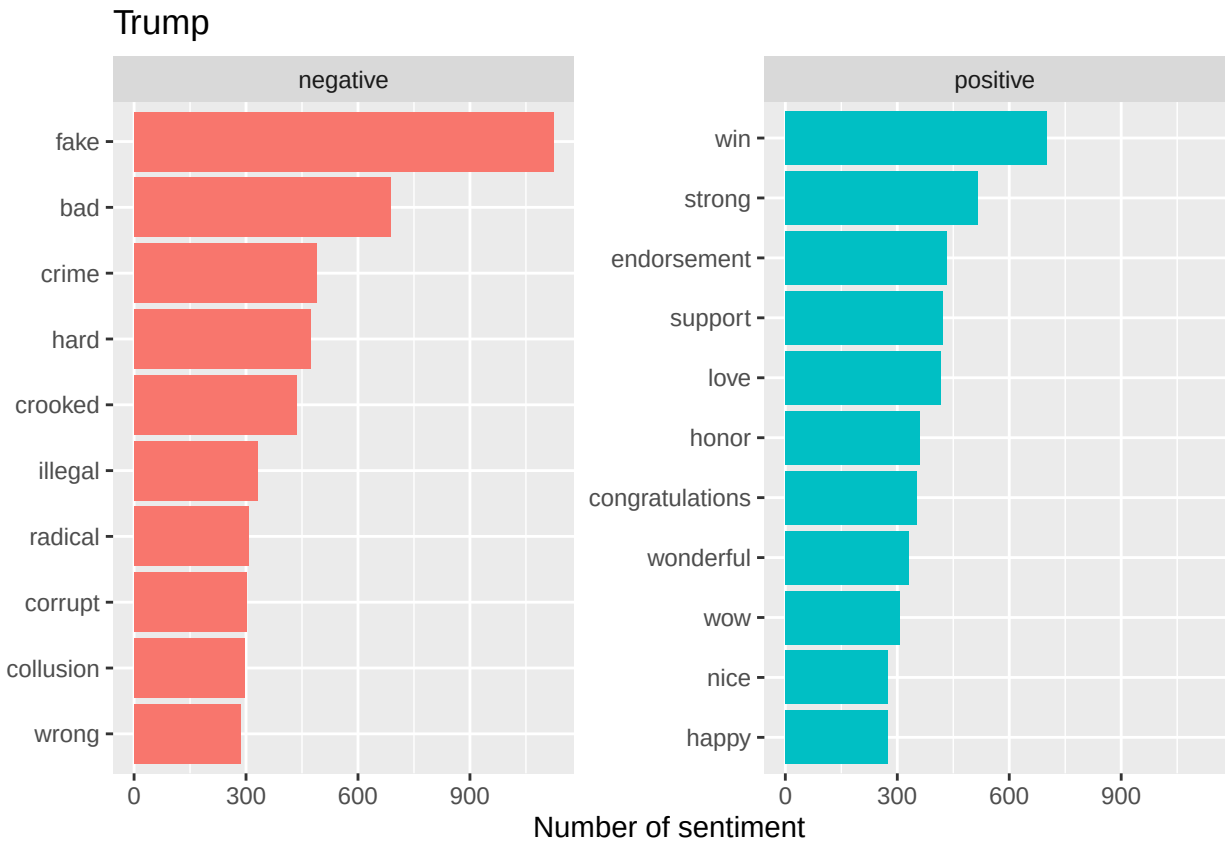

```
bt_tweets_sent %>%
  filter(person == "Biden" &
           !word %in% "trump" &
           timestamp >= "2019-04-25") %>%
  mutate(positivity = cumsum(if_else(value>0, value, 0)),
         negativity = cumsum(abs(if_else(value<0, value, 0)))) %>%
  plot_ly() %>%
  add_lines(x=~tweetnumber, y=~positivity, name='positive') %>%
  add_lines(x=~tweetnumber, y=~negativity, name='negative', color = I("red")) %>%
  add_lines(x=~tweetnumber, y=~positivity - negativity, name="difference",
            yaxis="y2", color = I("#ff8400")) %>%
  layout(
    title = "Biden's overall cumulative sentiment",
    yaxis = list(title='absolute cumulative sentiment'),
    yaxis2 = tb,
    width = 800
  ) %>%
  theme_plotly()
```



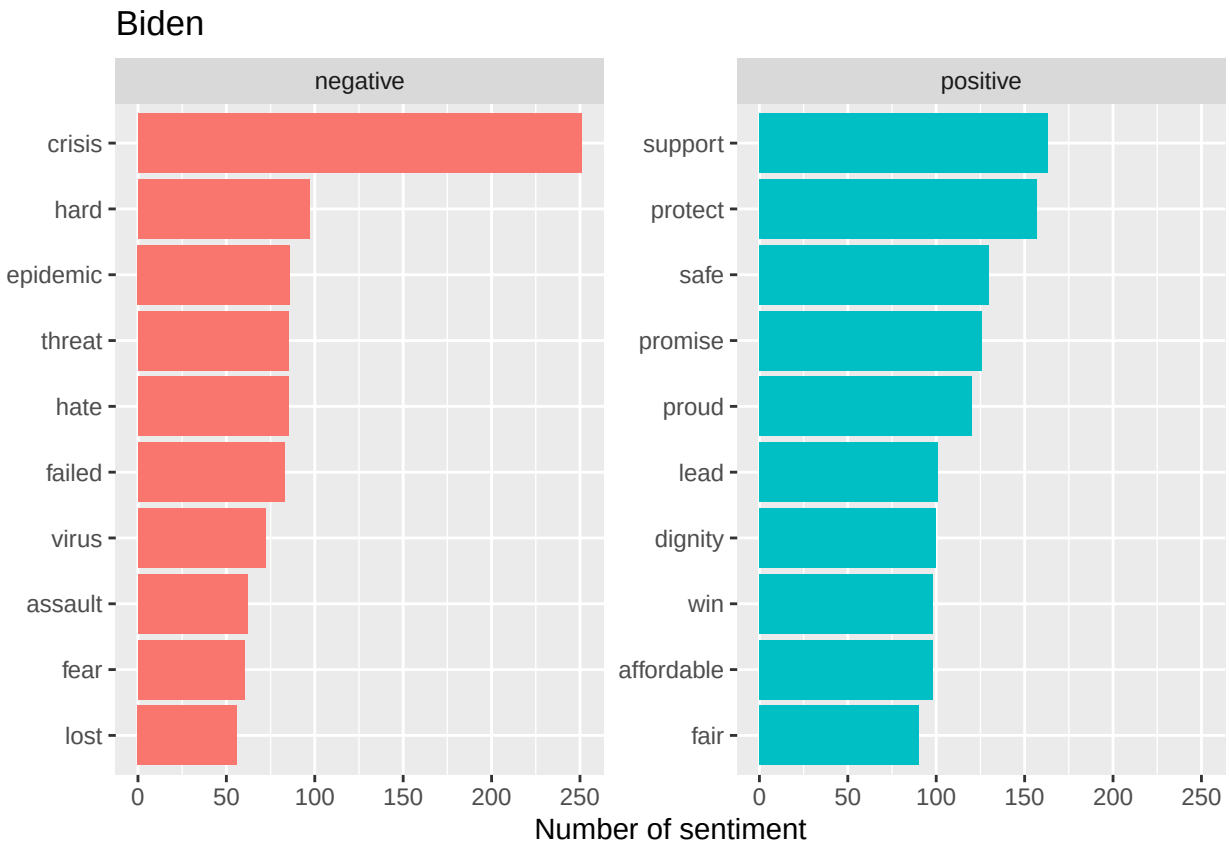
4. What words contribute to sentiments? Here, we going to check individual word contribution to the overall sentiment for both Twittier accounts, after the announcement of presidential camgaign. It is as expected that the majority of the negative sentiment in Trump's account is contributed by "fake", which is quite obvious by the constant "fake news" tweets, where as the positive sentiment is filled by "win". As for Biden, "crisis" contributed the most to the negative sentiment, which is the results of COVID19 tweets. "Support" and "protect" contributed to the positive sentiment relatively the same. Later, we will look on detail later on how words associate with eatch other.

```
wordcount_t <- bt_tweets_tidy %>%
  filter(timestamp >= "2015-06-16") %>%
  inner_join(get_sentiments("bing")) %>%
  count(person, word, sentiment, sort = TRUE) %>%
  ungroup()

wordcount_b <- bt_tweets_tidy %>%
  filter(timestamp >= "2019-04-25") %>%
  inner_join(get_sentiments("bing")) %>%
  count(person, word, sentiment, sort = TRUE) %>%
  ungroup()

wordcount_t %>%
           filter(person== "Trump" &
                    !word %in%custom_stop_words$word) %>%
           group_by(sentiment) %>%
           top_n(10) %>%
           ungroup() %>%
           mutate(word = reorder(word, n)) %>%
           ggplot(aes(n, word, fill = sentiment)) +
```

```
          geom_col(show.legend = FALSE) +
          facet_wrap(~sentiment, scales = "free_y") +
          labs(title = "Trump",
               x = "Number of sentiment",
               y = NULL)
```




```
wordcount_b %>%
          filter(person== "Biden" &
                   !word %in%custom_stop_words$word) %>%
          group_by(sentiment) %>%
          top_n(10) %>%
          ungroup() %>%
          mutate(word = reorder(word, n)) %>%
          ggplot(aes(n, word, fill = sentiment)) +
          geom_col(show.legend = FALSE) +
          facet_wrap(~sentiment, scales = "free_y") +
          labs(title = "Biden",
               x = "Number of sentiment",
               y = NULL)
```



Connecting words

1. Ngrams What if you want to check the what come after the word "ABC", what should you do? Ngrams would be a good tool in this sort of situation. The data wrangling process is relative the same as the very first section of Get data, unnest_tokens(), but here the token is "ngrams" and we will have to set "n" in order to tell how many consecutive words we want from the text. After unnesting, we will get a dataframe with every two consecutive words combinations, then separate(), filter(), and count() the number of "word1" "word2" combination, last we use graph_from_data_frame() so we could create an igraph (network).

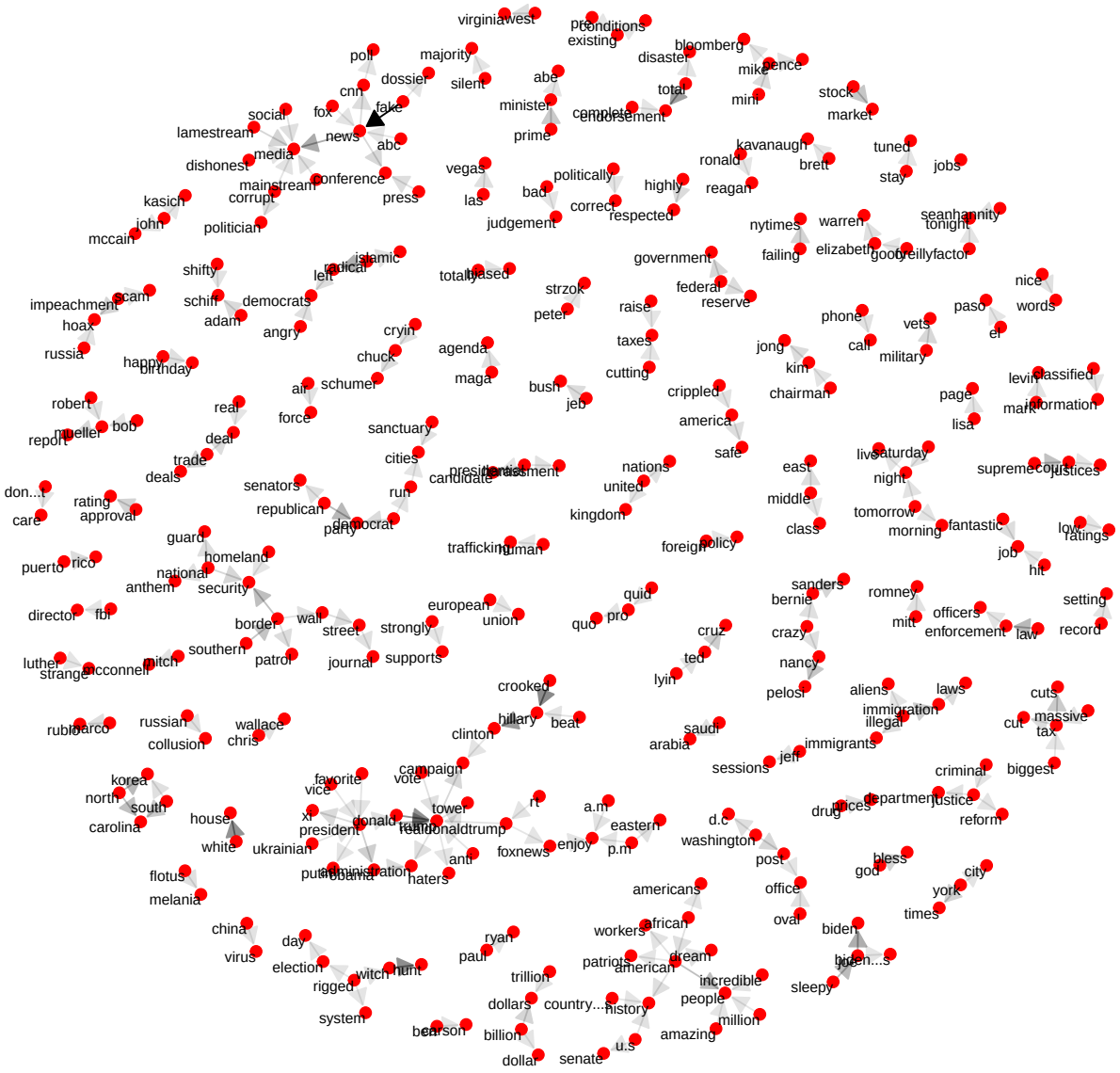
```
bt_bigrams <- bt_tweets %>%
  unnest_tokens(bigram, tweet, token = "ngrams", n = 2) %>%
  filter(!str_detect(bigram, "http") &
         !str_detect(bigram, "www.") &
         !str_detect(bigram, ".com") &
         !str_detect(bigram, ".org$") &
         !str_detect(bigram, "^t.co") &
         !str_detect(bigram, "\\d") &
         !str_detect(bigram, "^amp"))

bi_sep <- bt_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bi_filt <- bi_sep %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)
```

```
bi_graph_t <- bi_filt %>%
  filter(!word2 == "amp" &
         timestamp >= "2015-06-16" &
         person == "Trump") %>%
  count(word1, word2, sort = TRUE) %>%
  filter(n > 20) %>%
  graph_from_data_frame()

bi_graph_b <- bi_filt %>%
  filter(!word2 == "amp" &
           timestamp >= "2019-04-25" &
           person == "Biden") %>%
  count(word1, word2, sort = TRUE) %>%
  filter(n > 20) %>%
  graph_from_data_frame()
```

Now we are ready for graphing. In the Markov chain below, the individual node represents each word and the arrow represents the direction, the darker the color of the arrow means that the number of time of the bigrams combination appears in tweets is higher.

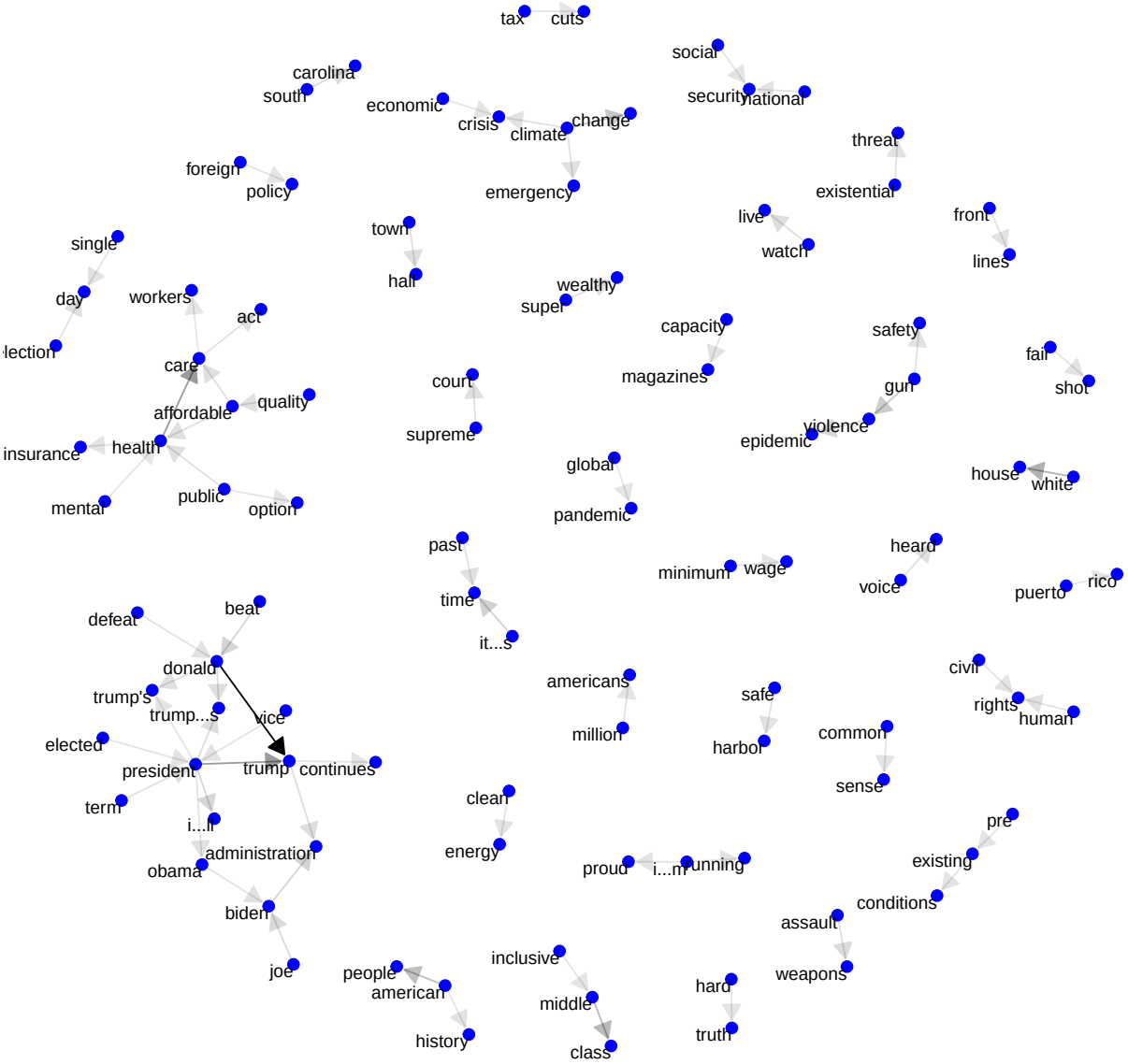
```
a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bi_graph_t, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "red", size = 3) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1, size = 3) +
  theme_void() +
  ggtitle("Trump")
```

Trump




```
ggraph(bi_graph_b, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "blue", size = 3) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void() +
  ggtitle("Biden")
```

Biden



2. Correlation between words We touched on the ngrams method to analze consecutive words in a text, now we going to look at the correlation of paired words. We gonna take the bt_tweets_tidy and create a new variable sec_by100 which we bind every words in each 100 tweets of each person. Once the dataset is tidy, we can filter() for timestamp and person. Since we want to assess how likely the paired words appear together, separately, or no appearance, which we can use pairwise_cor() from the widyr package.

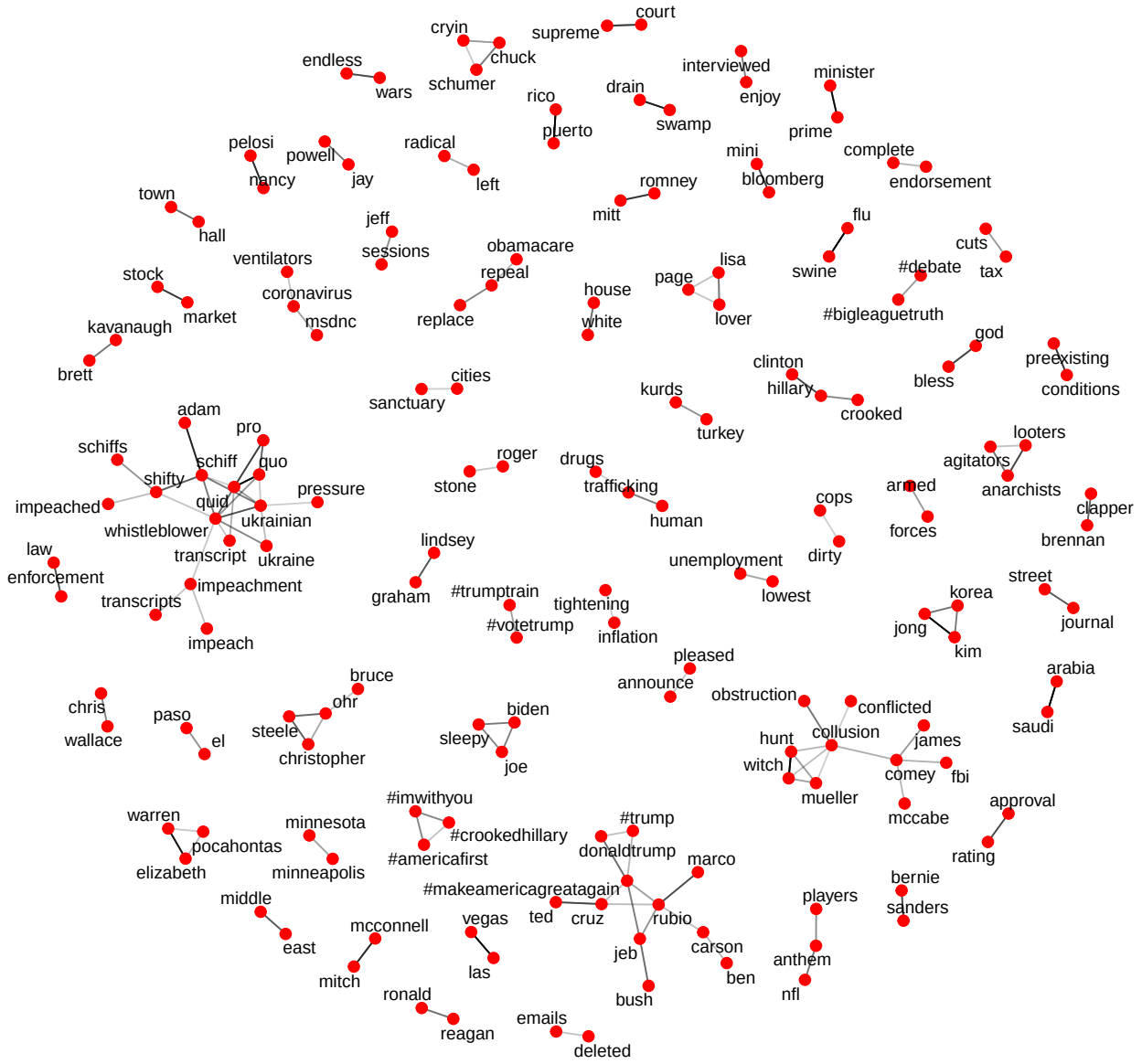
```
bt_tweets_tidy<- bt_tweets %>%
  arrange(timestamp) %>%
  mutate(sec_by100 = row_number()%/%100) %>%
  filter(!str_detect(tweet, "^RT") &
         sec_by100 > 0) %>%
  mutate(text = str_remove_all(tweet, remove_reg),
         text = str_remove_all(tweet, "\\s?(f|ht)(tp)(s?)(://)([^\\.]*)[\\.|/](\\S*)")) %>%
```

```
  unnest_tokens(word, text, token = "tweets") %>%
  filter(!word %in% custom_stop_words$word,
         !word %in% str_remove_all(custom_stop_words$word, "'"),
         !word %in% c("amp", "lt", "gt"),
         str_detect(word, "[a-z]") &
         !str_detect(word, "http") &
         !str_detect(word, "@") &
         !str_detect(word, "cont") &
         !str_detect(word, "www.") &
         !str_detect(word, ".com$") &
         !str_detect(word, ".org$") &
         !str_detect(word, "^tco") &
         !str_detect(word, "\\d") &
         !str_detect(word, "^amp") &
         !str_detect(word, "realdonald"))

word_cors_t <- bt_tweets_tidy %>%
  filter(timestamp >= "2015-06-16" &
         person == "Trump") %>%
  group_by(word) %>%
  filter(n() >= 20) %>%
  pairwise_cor(word, sec_by100, sort = TRUE)

word_cors_b <- bt_tweets_tidy %>%
  filter(timestamp >= "2019-04-25" &
           person == "Biden") %>%
  group_by(word) %>%
  filter(n() >= 20) %>%
  pairwise_cor(word, sec_by100, sort = TRUE)
```

We are all set for the correlation graph. The method here is the same as the Markov chain graph in the example above, but the major difference is that there is no direction in the paired words, because we are showing how likely the pairs show up in the every 100 tweets. We filtered the correlation to only showing the pairs that have a correlation above 0.5, and the darker the line means the higher the correlation. It is quite interesting that in Trump's tweets he mentioned a lot of names where as Biden's tweets was more about policy. I guess from the correlation graph is much more clear how different the two accounts are tweeting about.

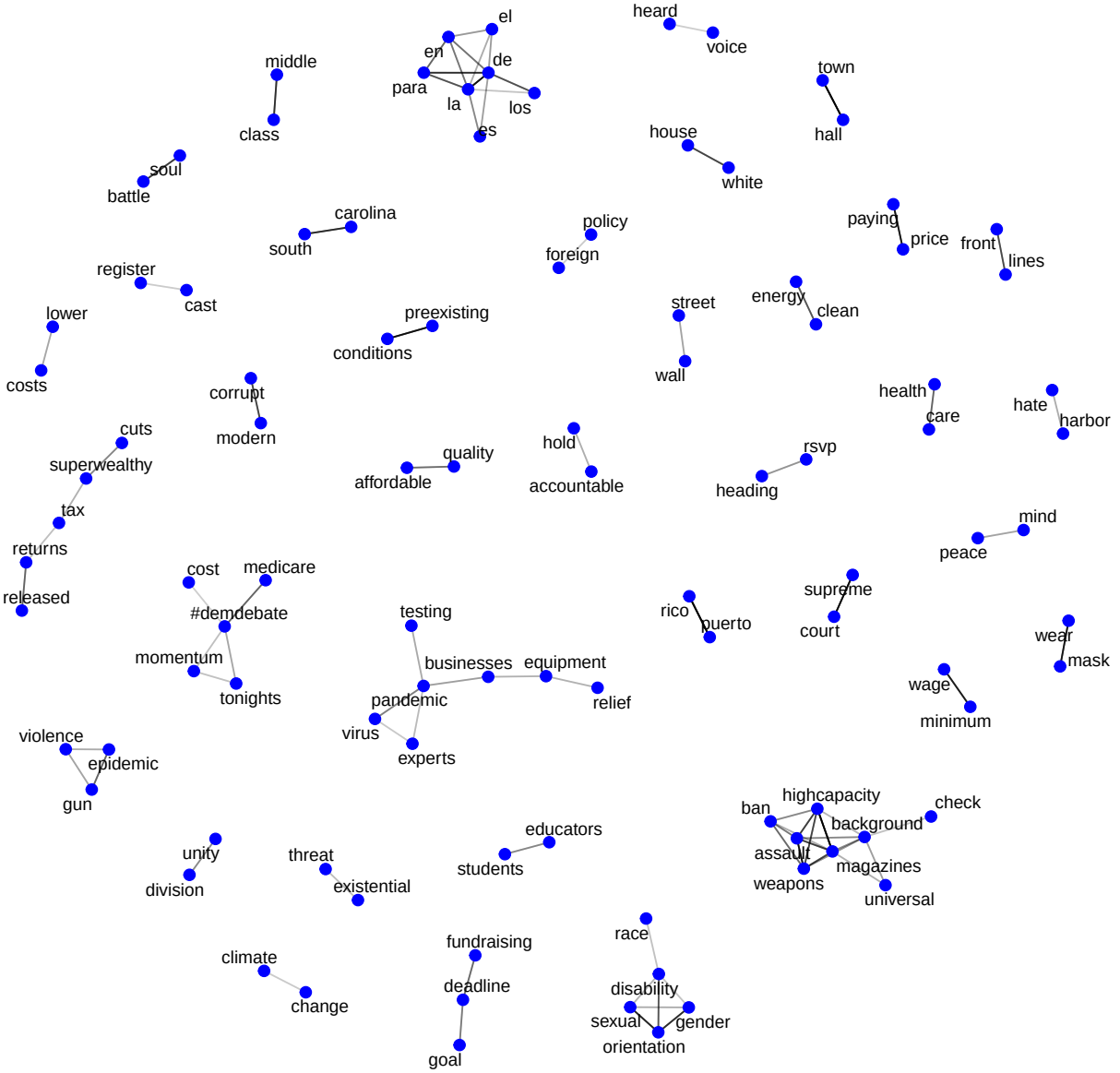
```
word_cors_t %>%
  filter(correlation > .50) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = correlation), show.legend = FALSE) +
  geom_node_point(color = "red", size = 3) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void() +
  ggtitle("Trump")
```

Trump




```
word_cors_b %>%
  filter(correlation > .50) %>%
  graph_from_data_frame() %>%
  ggraph(layout = "fr") +
  geom_edge_link(aes(edge_alpha = correlation), show.legend = FALSE) +
  geom_node_point(color = "blue", size = 3) +
  geom_node_text(aes(label = name), repel = TRUE) +
  theme_void() +
  ggtitle("Biden")
```

Biden



Model

My original plan is to use the tokenized text that I create above to predict what which text come from Trump or Biden, I successfully build a model using a logistic model with LASSO regularization, but I failed on building other models using text data to predict the "person" variable. I will first show the model using text data and later section will show a same dataset but using "retweets" and "likes" to predict the "person".

Text dataset

I am using a slightly different regular expression here, the reason being that it's simpler code do get what I did above, I haven't rewrite the above data cleaning, so in order to get the same result from above do please use the original regular expression.

So a crucial part in modeling a classification using text is to first separate the text to words, which I demonstrate throughout this project using unnest_tokens(), also since this is a Twitter data the text are sometimes embedded with html code like amp (&), lt (<), gt(>).

```
bt_tweets <- bt_tweets %>% mutate(id = row_number())
bt_text <- bt_tweets %>%
           mutate(text = str_remove_all(tweet, remove_reg),
           text = str_remove_all(tweet, "\\s?(f|ht)(tp)(s?)(://)([^\\.]*)[\\.|/](\\S*)"),
           person = as.factor(if_else(person=="Biden", "Biden", "Trump"))) %>%
           filter(!text =="") %>%
           select(-timestamp, tweet, -retweets, -likes) %>%
           unnest_tokens(word, text, token = "tweets") %>%
           filter(!word %in% stop_words$word & !word %in% c("amp", "lt", "gt"))
```

Here I only select words that been use more than 10 times by both Trump and Biden, I am actually not sure what would be a good measure here, I have not read enough to know what number to settle with but definitely not 1 because that would include way too many words and all these are going to be outliers since either person only use it once.

```
text_10 <- bt_text %>%
  group_by(word) %>%
  filter(n() >=10) %>%
  ungroup()
```

The following model I use are mentioned in Text classification with tidy data principles by Julia Silge, the initial_split() here by default split the data into 0.75 for training() and 0.25 for testing().

```
## building the model
library(rsample)
tweets_split <- bt_tweets %>%
                select(id) %>%
                initial_split()

train_data <- training(tweets_split)
test_data <- testing(tweets_split)
```

cast_sparse() is an easy way to transform a dataset into sparse matrix, since the train_data is basically just a column of "id" variable, first count how many times a word (word) is used in a tweet (id) then left_join() training_data back to the text_10, so the sparsed matrix will have "id" as row, "word" as column, and "n" (the number of time the word is used in one tweet) as the value in the matrix.

```
text_10 %>%
  count(id, word) %>%
  inner_join(train_data)
```

```
## # A tibble: 289,712 x 3
##       id word            n
##    <int> <chr>       <int>
##  1     1 created         1
##  2     1 democrats       1
##  3     1 economic        1
##  4     1 republicans     1
```

```
##  5     2 #kag2020        1
##  6     2 america         1
##  7     2 american        1
##  8     2 carolina        1
##  9     2 charlotte       1
## 10     2 cherish         1
## # ... with 289,702 more rows
```

```
words_spar <- text_10 %>%
  count(id, word) %>%
  inner_join(train_data) %>%
  cast_sparse(id, word, n)
```

With the sparse matrix above now the model just missing the y, or the classification that I am trying to predict. First, transform the row names of the sparse matrix which is "id" back to integer, then tibble so that the predictor "person" from original data (bt_tweets) can left_join back.

```
words_rown <- as.integer(rownames(words_spar))

tweets_joined <- tibble(id = words_rown) %>%
  left_join(bt_tweets %>%
              select(id, person))
```

The dependent and independent variables are ready for modeling, here will be using glmnet package to rung logistic regression and Lasso regularization. But you might ask why Lasso, what is it good for? It is a good method to minimize overfitting at the same time maintaining a simple model.

From the logist model graph shows that the minimum deviance is about -7 and calling model lambda.min confirm the plot.

In the glmnet.fit graph, the axis above shows number of coefficients at the current regularization λ, and this the degree of freedom of Lasso. Also, all the curves represents a variable. (This part might take some time to generate, for some reason this part doesn't run with Rmarkdown, but works in R, so I put the two following code chunk to eval =FALSE as well, but these code runs in R.script)

```
library(glmnet)
library(doMC)
registerDoMC(cores = 8) # parallel processing
length(is_biden)
length(words_spar)

is_biden <- tweets_joined$person == "Biden"
logist_model <- cv.glmnet(words_spar, is_biden,
                  family = "binomial",
                  parallel = TRUE, keep = TRUE
)

plot(logist_model)
log(logist_model$lambda.min)
plot(logist_model$glmnet.fit)
```

Now let's visualize the model, the graph show that these coefficients are the one has the most impact to probability, which alot of sense, as you can imagine @breitbartnews, #maga, @realdonaldtrump, @foxnews-freinds and riots are most likely comming from Trump's Twitter than Biden's, where as drbiden, @teamjoe, and @kamalaharris are very much unlikely to come from Trump's account.

```
library(broom)

logist_coefs <- logist_model$glmnet.fit %>%
  tidy() %>%
  filter(lambda == logist_model$lambda.1se)

logist_coefs %>%
  group_by(estimate > 0) %>%
  top_n(15, abs(estimate)) %>%
  ungroup() %>%
  ggplot(aes(fct_reorder(term, estimate), estimate, fill = estimate > 0)) +
  geom_col(alpha = 0.5, show.legend = FALSE) +
  coord_flip() +
  labs(
    x = "Words",
    title = "The coefs"
  ) + theme_minimal()
```

Now it is ready for prediction and model evaluations, logist_class is created in order to show the probability of the tweet coming from by Biden's account. The ROC curve seems pretty good, the AUC score is about 0.988, and the with probability of 0.5 as a cutoff point the confusion matrix shows that TP 1101, FP 134, FN 351, TN 10383.

```
int <- logist_coefs %>%
  filter(term == "(Intercept)") %>%
  pull(estimate)

logist_class <- text_10 %>%
  inner_join(test_data) %>%
  inner_join(coefs, by = c("word" = "term")) %>%
  group_by(id) %>%
  summarize(score = sum(estimate)) %>%
  mutate(probability = plogis(int + score))

logist_class

# model metric
library(yardstick)

logist_metrics <- logist_class %>%
  left_join(bt_tweets %>%
              select(person, id), by = "id") %>%
  mutate(person = as.factor(person))

logist_metrics %>%
  roc_curve(person, probability) %>%
  ggplot(aes(x = 1 - specificity, y = sensitivity)) +
  geom_line(
    color = "red",
    size = 2
  ) +
  geom_abline(
    lty = 2, alpha = 0.5,
```

```
    color = "black",
    size = 1.2
  ) +
  labs(
    title = "Was it Biden or Trump",
  ) + theme_minimal()

# AUC estimates
logist_metrics %>%
  roc_auc(person, probability)

# confusion metric
logist_metrics %>%
  mutate(
    prediction = case_when(
      probability > 0.5 ~ "Biden",
      TRUE ~ "Trump"
    ),
    prediction = as.factor(prediction)
  ) %>%
  conf_mat(person, prediction)
```

Retweets + likes

In this section I attempt to run 4 different models using the same function:

$$Y = \beta_1 X_1 + \beta_2 X_2$$

Here, I denote Y as person, X_1 as retweets, and X_2 as likes of that tweets.

```
library(caret)
train_index <- createDataPartition(bt_tweets$person, p = .8, list = FALSE)
training_bt <- bt_tweets[train_index,]
testing_bt  <- bt_tweets[-train_index,]
```

Data setup

Naive Bayes This model has an accuracy of 0.8663 but has a p-value of 1.

```
nb_bt <- train(person ~ likes + retweets,
               data = training_bt,
               method = "naive_bayes",
               trControl = trainControl(method = "none"),
               tuneGrid = data.frame(laplace = 0,
                                     usekernel = FALSE,
                                     adjust = FALSE))
nb_predict <- predict(nb_bt,
                      newdata = testing_bt)
nb_confm <- confusionMatrix(nb_predict, as.factor(bt_tweets[-train_index, ]$person))
nb_confm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Biden Trump
##      Biden    63   225
##      Trump  1149  8926
##
##                Accuracy : 0.8674
##                  95% CI : (0.8607, 0.8739)
##     No Information Rate : 0.883
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0409
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.051980
##             Specificity : 0.975413
##          Pos Pred Value : 0.218750
##          Neg Pred Value : 0.885955
##              Prevalence : 0.116955
##          Detection Rate : 0.006079
##    Detection Prevalence : 0.027791
##       Balanced Accuracy : 0.513696
##
##        'Positive' Class : Biden
##
```

LogitBoost This model compare to Naive Bayes perform a lot better with an accuracy of 0.8883 and a p-value of 0.05029.

```
library(caTools)
logitboost_bt <- train(person ~ likes + retweets,
                       data = training_bt,
                       method = "LogitBoost",
                       trControl = trainControl(method = "none"))

logitboost_predict <- predict(logitboost_bt,
                          newdata = testing_bt)

logitboost_confm <- confusionMatrix(logitboost_predict, as.factor(bt_tweets[-train_index, ]$person))
logitboost_confm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Biden Trump
##      Biden   124    20
##      Trump  1088  9131
##
##                Accuracy : 0.8931
##                  95% CI : (0.887, 0.899)
##     No Information Rate : 0.883
```

```
##     P-Value [Acc > NIR] : 0.0006863
##
##                   Kappa : 0.1621
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.10231
##             Specificity : 0.99781
##          Pos Pred Value : 0.86111
##          Neg Pred Value : 0.89353
##              Prevalence : 0.11695
##          Detection Rate : 0.01197
##    Detection Prevalence : 0.01390
##       Balanced Accuracy : 0.55006
##
##        'Positive' Class : Biden
##
```

Support Vector Machine This model has an accuracy of 0.8821 and a p-value of 0.6272, perform slightly worst than LogitBoost but better than Naive Bayes.

```
svm_bt <- train(person ~ likes + retweets,
                data = training_bt,
                method = "svmLinearWeights2",
                trControl = trainControl(method = "none"),
                tuneGrid = data.frame(cost = 1,
                                      Loss = 0,
                                      weight = 1))
svm_predict <- predict(svm_bt,
                   newdata = testing_bt)
svm_confm <- confusionMatrix(svm_predict, as.factor(bt_tweets[-train_index, ]$person))
svm_confm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Biden Trump
##      Biden   844  4625
##      Trump   368  4526
##
##                Accuracy : 0.5182
##                  95% CI : (0.5085, 0.5279)
##     No Information Rate : 0.883
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0757
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.69637
##             Specificity : 0.49459
##          Pos Pred Value : 0.15432
##          Neg Pred Value : 0.92481
```

```
##              Prevalence : 0.11695
##          Detection Rate : 0.08144
##    Detection Prevalence : 0.52774
##       Balanced Accuracy : 0.59548
##
##        'Positive' Class : Biden
##
```

Random Forest This model has an accuracy of 0.916 and a p-value < 2.2e-16, which is a better performance compare to other 3 models.

```
rf_bt <- train(person ~ likes + retweets,
               data = training_bt,
               method = "ranger",
               trControl = trainControl(method = "none"),
               tuneGrid = data.frame(mtry = floor(sqrt(dim(train_index)[2])),
                                     splitrule = "gini",
                                     min.node.size = 1))

rf_predict <- predict(rf_bt,
                  newdata = testing_bt)

rf_confm <- confusionMatrix(rf_predict, as.factor(bt_tweets[-train_index, ]$person))
rf_confm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Biden Trump
##      Biden   565   254
##      Trump   647  8897
##
##                Accuracy : 0.9131
##                  95% CI : (0.9075, 0.9184)
##     No Information Rate : 0.883
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5102
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.46617
##             Specificity : 0.97224
##          Pos Pred Value : 0.68987
##          Neg Pred Value : 0.93221
##              Prevalence : 0.11695
##          Detection Rate : 0.05452
##    Detection Prevalence : 0.07903
##       Balanced Accuracy : 0.71921
##
##        'Positive' Class : Biden
##
```

```r
model_results <- bind_rows(
  nb_confm$overall,
  logitboost_confm$overall,
  svm_confm$overall,
  rf_confm$overall,
) %>%
  as.data.frame() %>%
  mutate(model = c("Naive-Bayes", "LogitBoost", "SVM", "Random forest"))

model_results
```

```
##    Accuracy      Kappa AccuracyLower AccuracyUpper AccuracyNull AccuracyPValue
## 1 0.8674129 0.04092753     0.8607320     0.8738866    0.8830455   9.999995e-01
## 2 0.8930812 0.16207693     0.8869726     0.8989678    0.8830455   6.863414e-04
## 3 0.5181897 0.07567012     0.5085172     0.5278520    0.8830455   1.000000e+00
## 4 0.9130561 0.51017393     0.9074664     0.9184124    0.8830455   2.424920e-23
##   McnemarPValue         model
## 1 7.345151e-137   Naive-Bayes
## 2 1.875470e-225    LogitBoost
## 3  0.000000e+00           SVM
## 4  5.615880e-39 Random forest
```

Conclusion

This is my first attempt to do a text analysis and there is a lot more to explore in this section, such as unsupervise learning with LDA. This post is just a show case of what R can achieve in text mining on Twitter to examine more detail on what these politicians are actually saying, obviously text analysis can also be down on books, document or songs. However, I am still a bit skeptical about sentiment analysis because the way these methods could assigned words into odd categories.

From the supervise machine learning that I employed 2 different approaches to tackle classification problems. in this project, comparing the results of logistic regression with Lasso regularization on text sparse matrix to the classification and conventional way of using continuous variable to predict a discrete variable. I am surprised to find out that using text can actually get such a high estimate as 0.988 and with a better sensitivity which can be confirmed from the confusion matrix, but process pretty fast with an adequate sensitivity as well.

Text analysis has a lot of practical implications, such as detecting fake news, financial fraud, and easier to understand a large number of documentations. However, In this project I barely touch on much mathematical details, since I intend to write this as an application of R in text analysis. I might revisit this topic later and make an update in the future when I got more time and redo the classification problem using caret and try other modeling techniques.

Reference

1. Rafael A. Irizarry. (2019), Introduction to Data Science
2. Silge, Robinson (2017), Text Mining with R
3. Xie, Allaire, Grolemund (2020), R Markdown: The Definitive Guide
4. Michael ClarkAn (2018), Introduction to Text Processing and Analysis
5. Julia Silge (2018), Text classification with tidy data principles
6. Boehmke, Greenwell. (2020), Hands-On Machine Learning with R