jillstein.R

laurajakli

Fri Dec 9 00:57:02 2016

```
###This contains the entire code for data analysis and visualization for Jill Stein.

##SET WORKING DIRECTORY##

setwd("/Users/laurajakli/Desktop/231A_data")

####Package installations (for all sections) ######

library(tidyr)
library(viridis)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

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

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

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.2
```

```
library(scales)
```

```
## Warning: package 'scales' was built under R version 3.3.2
```

```
library(grid)
library(RColorBrewer)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(qdap)
```

```
## Loading required package: qdapDictionaries
```

```
## Loading required package: qdapRegex
```

```
##
## Attaching package: 'qdapRegex'
```

```
## The following object is masked from 'package:ggplot2':
##
```

```
##     %+%
## The following objects are masked from 'package:dplyr':
##
##     escape, explain
## Loading required package: qdapTools
##
## This data.table install has not detected OpenMP support. It will work but slower in single threaded m
##
## Attaching package: 'qdapTools'
## The following object is masked from 'package:dplyr':
##
##     id
##
## Attaching package: 'qdap'
## The following object is masked from 'package:dplyr':
##
##     %>%
## The following object is masked from 'package:tidyr':
##
##     %>%
## The following object is masked from 'package:base':
##
##     Filter
library(data.table)
## Warning: package 'data.table' was built under R version 3.3.2
## ------------------------------------------------------------------------------
## data.table + dplyr code now lives in dtplyr.
## Please library(dtplyr)!
## ------------------------------------------------------------------------------
##
## Attaching package: 'data.table'
## The following object is masked from 'package:qdapTools':
##
##     shift
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
library(scales)
library(tm)
## Loading required package: NLP
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:qdap':
##
##     ngrams

## The following object is masked from 'package:ggplot2':
##
##     annotate

##
## Attaching package: 'tm'

## The following objects are masked from 'package:qdap':
##
##     as.DocumentTermMatrix, as.TermDocumentMatrix
```

```
library(SnowballC)
library(wordcloud)
####read in data####

df <- read.csv("drjillstein_facebook.csv", fileEncoding="latin1")

###Section 1: Visualizing Facebook Reactions over Time#####

###I am interested in understanding the degree to which Facebook reactions are
###sarcastic versus "genuine." In other words, I want to see if most reactions
###are in a "trolling" style, or if generally, reactions actually translate to
###the actually intended emotion (via the status poster.) This is a difficult
###task and I will not achieve it with this project. But as a sanity check before
###the rest of my analysis, I wanted to see what produced the most anger (i.e.,
#most "anger" reactions towards each candidates' statuses.

#Here, we arrange the dataframe in descending order of num_angrys,
#subsetting to only the date, the status message, and the count of
#angry reactions.

dfangry<-arrange(df, desc(num_angrys))
dfangry<-(dfangry[1:10,])
dfangry<-select(dfangry, status_published, status_message, num_angrys)

#Now I save this as a table to a pdf using the grid.table function.

pdf("top_angrystatus_Stein.pdf", height=12, width=48)
grid.table(dfangry)
dev.off()
```

```
## pdf
##   2
```

```
##Here, I use the group by function to group by date
##rather than individual status, since there are
##multiple posts per day by different candidates.
##As such, we get the total reactions per day,
##by reaction type.

df_agg <- df %>% group_by(date = as.Date(substr(date, 1, 10))) %>%
  summarize(total_likes=sum(num_likes),
            total_loves=sum(num_loves),
```

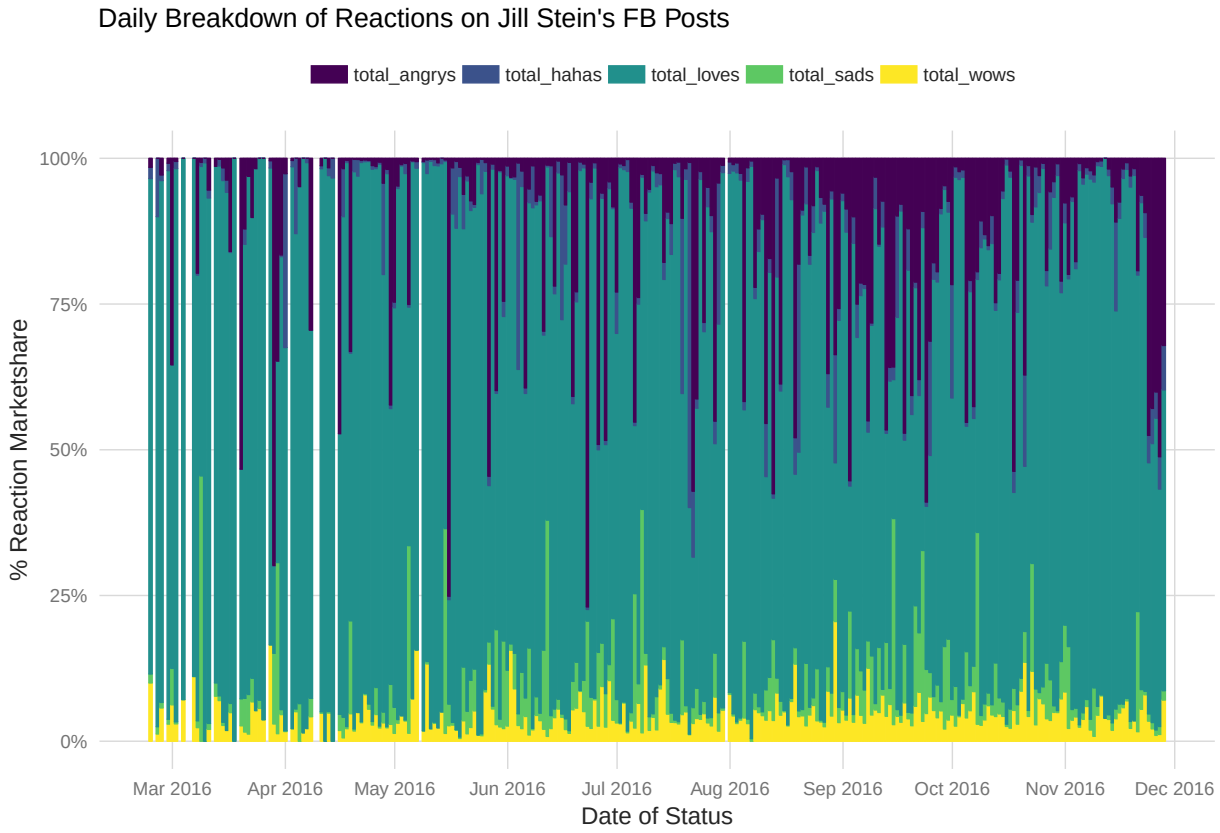
```
            total_wows=sum(num_wows),
            total_hahas=sum(num_hahas),
            total_sads=sum(num_sads),
            total_angrys=sum(num_angrys)) %>%
  arrange(date)

#Now, let's aggregate across all reaction types.

df_agg_long <- df_agg %>% gather(key=reaction, value=count, total_likes:total_angrys) %>%
  mutate(reaction=factor(reaction))

print(head(df_agg_long,10))
```

```
## # A tibble: 10 × 3
##          date    reaction count
##        <date>      <fctr> <int>
## 1  2016-02-24 total_likes  4262
## 2  2016-02-26 total_likes   867
## 3  2016-02-27 total_likes  3409
## 4  2016-02-29 total_likes  3529
## 5  2016-03-01 total_likes   705
## 6  2016-03-02 total_likes  8504
## 7  2016-03-04 total_likes  2352
## 8  2016-03-07 total_likes   460
## 9  2016-03-08 total_likes  9571
## 10 2016-03-09 total_likes  1699
```

```
react_theme <- function() {

  #Here, we're working with the Greys palette from RColorBrewer.
  palette <- brewer.pal("Greys", n=9)
  color.background = palette[1]
  color.grid.major = palette[3]
  color.axis.text = palette[6]
  color.axis.title = palette[8]
  color.title = palette[9]

  theme_bw(base_size=9) +

    #Here, we set the chart region to the background's light grey.
    theme(panel.background=element_rect(fill=color.background, color=color.background)) +
    theme(plot.background=element_rect(fill=color.background, color=color.background)) +
    theme(panel.border=element_rect(color=color.background)) +

    #The grid is a bit darker.
    theme(panel.grid.major=element_line(color=color.grid.major,size=.25)) +
    theme(panel.grid.minor=element_blank()) +
    theme(axis.ticks=element_blank()) +

    #The legend is hidden.
    theme(legend.position="none") +
    theme(legend.background = element_rect(fill=color.background)) +
    theme(legend.text = element_text(size=7,color=color.axis.title)) +
```

```
    #Here, we format the title and axis labels.
    theme(plot.title=element_text(color=color.title, size=10, vjust=1.25)) +
    theme(axis.text.x=element_text(size=7,color=color.axis.text)) +
    theme(axis.text.y=element_text(size=7,color=color.axis.text)) +
    theme(axis.title.x=element_text(size=8.5,color=color.axis.title, vjust=0)) +
    theme(axis.title.y=element_text(size=8.5,color=color.axis.title, vjust=1.25)) +

    #Finally, let's plot the margins
    theme(plot.margin = unit(c(0.35, 0.2, 0.3, 0.35), "cm"))
}


###I am filtering out the "likes" because they would effectively
###"drown out" all of the other reactions visuallt, as "likes" are still by far the
###most  popular reaction.

reactionsplot <- ggplot(df_agg_long %>% filter(reaction!="total_likes"), aes(x=date, y=count, color=rea
  geom_bar(size=0.25, position="fill", stat="identity") +
  react_theme() +
  scale_x_date(breaks = date_breaks("1 month"), labels = date_format("%b %Y")) +
  scale_y_continuous(labels=percent) +
  theme(legend.title = element_blank(),
        legend.position="top",
        legend.direction="horizontal",
        legend.key.width=unit(0.5, "cm"),
        legend.key.height=unit(0.25, "cm"),
        legend.spacing=unit(0,"cm")) +
  scale_color_viridis(discrete=T) +
  scale_fill_viridis(discrete=T) +
  labs(title="Daily Breakdown of Reactions on Jill Stein's FB Posts",
       x="Date of Status",
       y="% Reaction Marketshare")

reactionsplot
```

```
## Warning: Removed 5 rows containing missing values (position_stack).
```




```
#Now, let's aggregate but as percentages, not totals.
#I actually prefer the first visualization, but this is
#a slightly different way to help visualize the data.

df_percentagg <- df %>% group_by(date = as.Date(substr(date, 1, 10))) %>%
  summarize(total_reactions=sum(num_loves)+sum(num_wows)+sum(num_hahas)+sum(num_sads)+sum(num_angrys),
            perc_loves=sum(num_loves)/total_reactions,
            perc_wows=sum(num_wows)/total_reactions,
            perc_hahas=sum(num_hahas)/total_reactions,
            perc_sads=sum(num_sads)/total_reactions,
            perc_angrys=sum(num_angrys)/total_reactions) %>%
  select(-total_reactions) %>%
  arrange(date)


df_percentagg<-df_percentagg <- df_percentagg[-c(1), ]
df_percentagg_long <- df_percentagg %>% gather(key=reaction, value=count, perc_loves:perc_angrys) %>%
  mutate(reaction=factor(reaction))


reactionpercents <- ggplot(df_percentagg_long, aes(x=date, y=count, color=reaction)) +
  geom_line(size=0.5, stat="identity") +
  react_theme() +
  scale_x_date(breaks = date_breaks("1 month"), labels = date_format("%m %Y")) +
  scale_y_continuous(labels=percent) +
  theme(legend.title = element_blank(),
        legend.position="top",
```

```
        legend.direction="horizontal",
        legend.key.width=unit(0.5, "cm"),
        legend.key.height=unit(0.25, "cm"),
        legend.spacing=unit(0,"cm")) +
  scale_color_viridis(discrete=T) +
  scale_fill_viridis(discrete=T) +
  labs(title="Daily Breakdown of Reactions on Jill Stein's FB Posts",
       x="Date of Status",
       y="% Reaction Marketshare")
reactionpercents
```

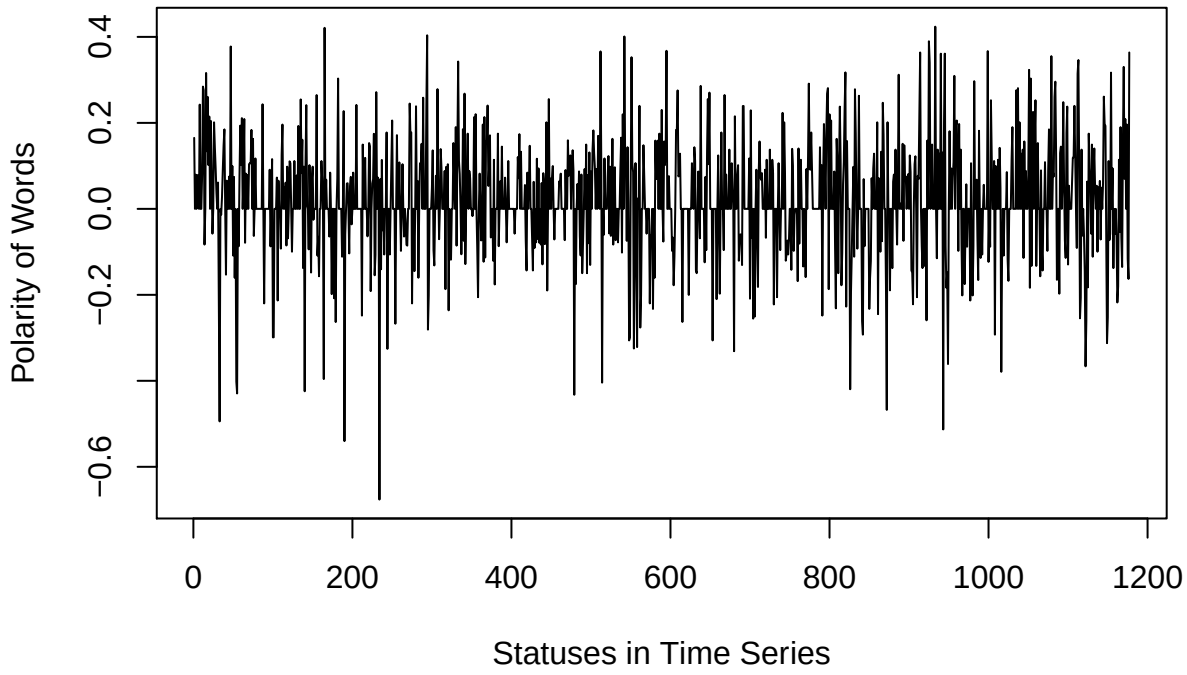
```
## Warning: Removed 5 rows containing missing values (geom_path).
```


```
####### Section 2. Examining Language Polarity####


##Here, I am tweaking code from the sentiment analysis lecture
##using the `qdap` function.  I am conducting a plot analysis
##of candidates' Facebook statuses using sentiments.
##As Rochelle mentioned, the `qdap` package is great for
##using dictionary methods to analyze text.
##One of the most popular of these methods is sentiment analysis,
##which calculates how "positive" or "negative" text is.
##Here, it's on a -1 (most negative) to 1 (most positive) scale.
##Note that we're not specifying UTF-8 (unicode) encoding here.

df <- read.csv("drjillstein_facebook.csv", fileEncoding="latin1")
```

```
# Then put the dataframe into a data.table
df.dat <- data.table(df)

# We now add columns for cumulative word counts and polarity scores,
# so that we measure sentiment over time.

# First, we add word counts
df.dat <-df.dat[, wc := wc(status_message,missing=0)]

# Next, we add cumulative word count and percent
# completes to proxy for progression
df.dat<- df.dat[, cumsum := cumsum(wc)]
df.dat<- df.dat[, pct.complete := df.dat$cumsum / sum(df.dat$wc)]
df.dat<- df.dat[, pct.complete.100 := pct.complete * 100]

# Here, we calculate polarity. We obtain a vector of polarity scores
df.dat <- with(df, polarity(status_message, date, constrain = TRUE))
```

```
## Warning in polarity(status_message, date, constrain = TRUE):
##   Some rows contain double punctuation.  Suggested use of `sentSplit` function.
```

```
polcount.dfdat <- na.omit(counts(df.dat)$polarity)

# Next, we put all of this polarity info into a data frame
len <- length(polcount.dfdat)
pol.df <- data.frame(polarity = polcount.dfdat, Time=1:len)


# Finally, we plot it. ggplot doesn't seem to add much here
# so I'll just use a simple plot.

plot(x = pol.df$Time, y = pol.df$polarity, type="l", xlab="Statuses in Time Series",
     ylab="Polarity of Words")
```

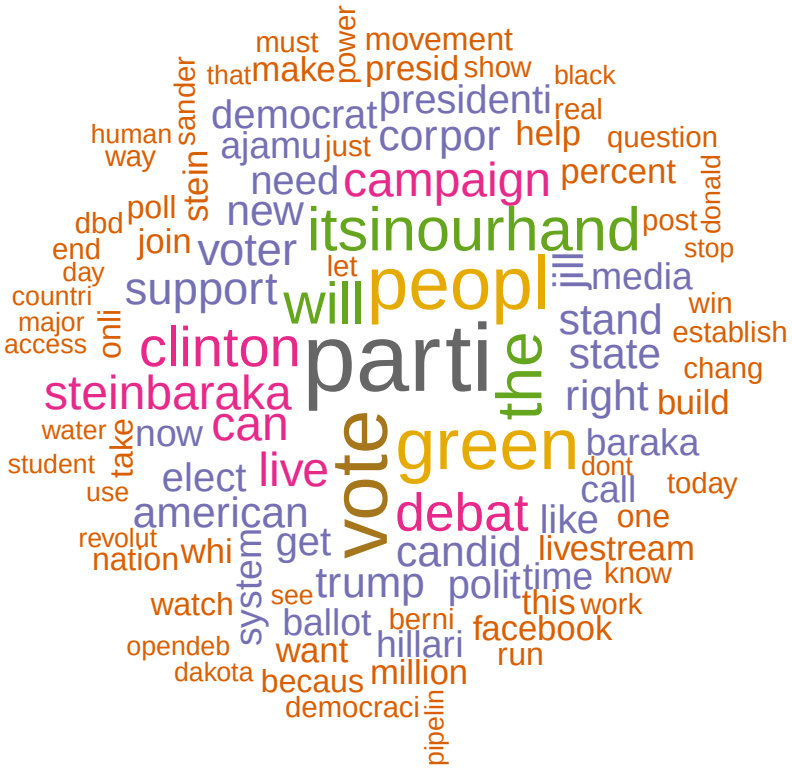



```
#### Section 3: Candidate Status WordCloud ####


jill <- read.csv('drjillstein_facebook.csv', stringsAsFactors = FALSE)

#First, prepare the "corpus."
#As we've discussed in lecture,
#A corpus is a collection of texts,
#usually stored electronically,
#and from which we perform our analysis.
#Here, the corpus is the candidates' status
#messages.

jillCorpus <- Corpus(VectorSource(jill$status_message))


#As we've noted in lecture, most text analysis
# applications follow a similar
```

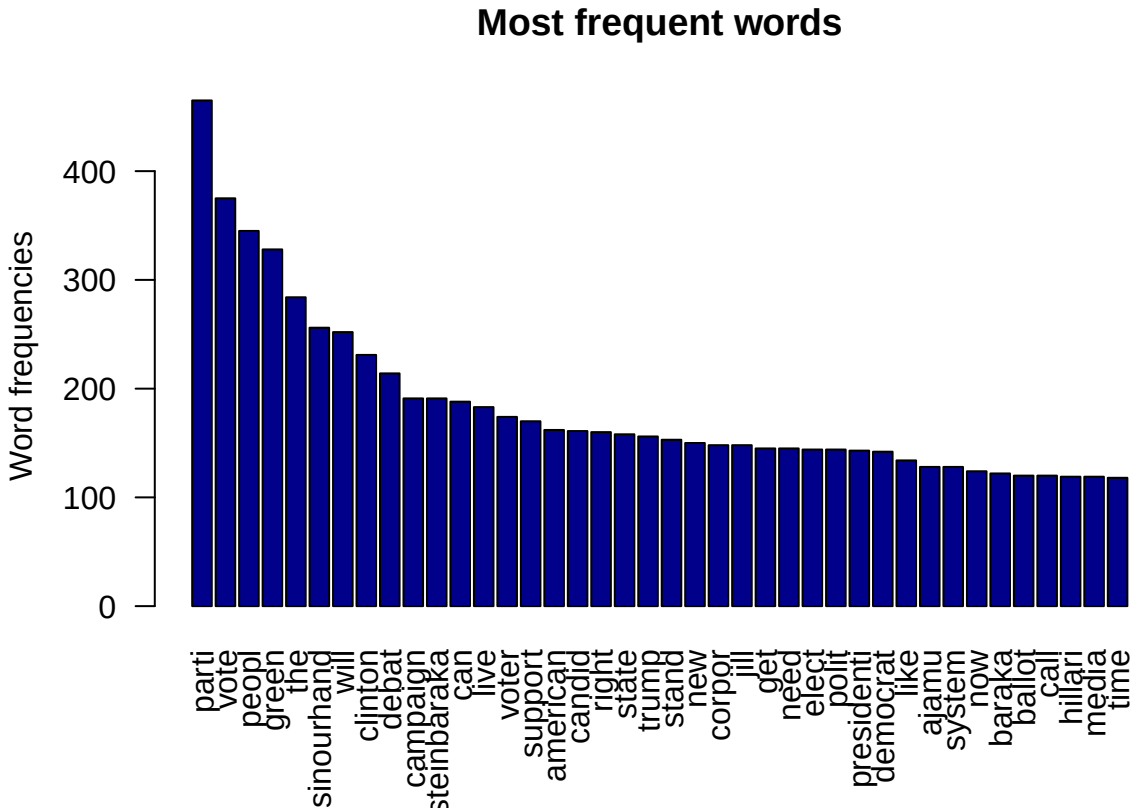
recipe' for preprecessing.

```
#In the following lines of code, we
#convert all characters to lowercase, remove
#punctuation, numbers, and stop words
#(including custom stop words). We also
#"stem" words. I also converted everything
#to utf-8 (unicode), because emoticons
#were giving me trouble in producing a
#wordcloud.

jillCorpus<- tm_map(jillCorpus,
                    content_transformer(function(x) iconv(x, to='UTF-8-MAC', sub='byte')),
                    mc.cores=1)
```

```
jillCorpus <- tm_map(jillCorpus, PlainTextDocument)
jillCorpus <- tm_map(jillCorpus, removePunctuation)
jillCorpus <- tm_map(jillCorpus, removeNumbers)
jillCorpus <- tm_map(jillCorpus, stemDocument)
jillCorpus <- tm_map(jillCorpus, removeWords, c('the', 'this', 'a', stopwords('english')))
wordcloud(jillCorpus, scale=c(3,0.5), max.words = 100, random.order = FALSE,
          colors=brewer.pal(8, 'Dark2'))
```




```
#Here, we create a Document-Term Matrix (DTM)

dtm <- TermDocumentMatrix(jillCorpus)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)

#And use this to create a simple bar plot of
#each candidates' 40 most frequent words,
#as a secondary visual.

barplot(d[1:40,]$freq, las = 2, names.arg = d[1:40,]$word,
        col ="darkblue", main ="Most frequent words",
        ylab = "Word frequencies")
```



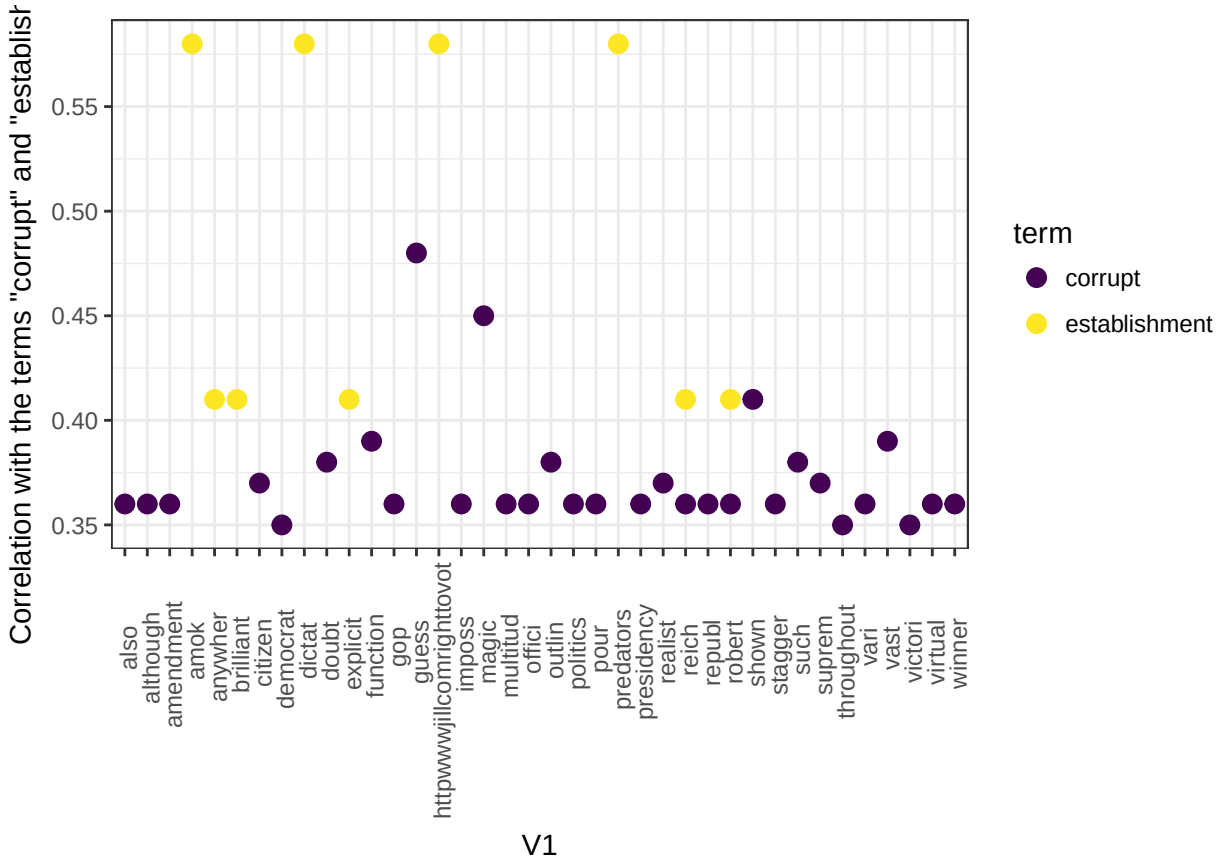

```
#####Section 4: Exploring Word Associations ######

###Although not the most intuitive method,
###one way to visualize word associations is through a
###scatterplot. To visualize multiple themes of
###interest, I plot 2 themes at once. The idea
###is to compute the term correlations and store
###them in a data frame.




######CORRUPTION & ESTABLISHMENT#######

###"toi" stands for "term of interest" here, and
###the corlimit specifies the lower
###correlation bound limit.

toi1 <- "corrupt"
toi2 <- "establishment"
corlimit <- 0.35

corr1 <-  findAssocs(dtm, toi1, corlimit)[[1]]
corr1 <- cbind(read.table(text = names(corr1), stringsAsFactors = FALSE), corr1)
corr2 <- findAssocs(dtm, toi2, corlimit)[[1]]
corr2 <- cbind(read.table(text = names(corr2), stringsAsFactors = FALSE), corr2)

##Here, we join the 2 correlations,
```

```
##and then gather them to plot.

two_terms_corrs <- full_join(corr1, corr2)
```

```
## Joining, by = "V1"
```

```
two_terms_corrs_gathered <- gather(two_terms_corrs, term, correlation, corr1:corr2)

#Here, we construct the legend, and then
#use ggplot2 to plot everything.
two_terms_corrs_gathered$term <- ifelse(two_terms_corrs_gathered$term  == "corr1", toi1, toi2)

ggplot(two_terms_corrs_gathered, aes(x = V1, y = correlation, colour =  term ) ) +
  geom_point(size = 3) +
  ylab(paste0("Correlation with the terms ", "\"", toi1,  "\"", " and ",  "\"", toi2, "\"")) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 90, hjust = .7, vjust = .7)) + scale_color_viridis(discrete="
```

```
## Warning: Removed 36 rows containing missing values (geom_point).
```




```
######TRUMP/CLINTON#######


toi1 <- "trump"
toi2 <- "clinton"
corlimit <- 0.5
```

```
corr1 <-  findAssocs(dtm, toi1, corlimit)[[1]]
corr1 <- cbind(read.table(text = names(corr1), stringsAsFactors = FALSE), corr1)
corr2 <- findAssocs(dtm, toi2, corlimit)[[1]]
corr2 <- cbind(read.table(text = names(corr2), stringsAsFactors = FALSE), corr2)

##Here, we join the 2 correlations,
##and then gather them to plot.

two_terms_corrs <- full_join(corr1, corr2)
```

```
## Joining, by = "V1"
```

```
two_terms_corrs_gathered <- gather(two_terms_corrs, term, correlation, corr1:corr2)

#Here, we construct the legend, and then
#use ggplot2 to plot everything.
two_terms_corrs_gathered$term <- ifelse(two_terms_corrs_gathered$term  == "corr1", toi1, toi2)

ggplot(two_terms_corrs_gathered, aes(x = V1, y = correlation, colour =  term ) ) +
  geom_point(size = 3) +
  ylab(paste0("Correlation with the terms ", "\"", toi1,  "\"", " and ",  "\"", toi2, "\"")) +
  theme_bw() +
  theme(axis.text.x = element_text(angle = 90, hjust = .7, vjust = .7)) + scale_color_viridis(discrete="
```

```
## Warning: Removed 37 rows containing missing values (geom_point).
```