McDermott-Final-Project-Report

Amanda McDermott

5/7/2019

Research Objective

This was a continuation of a project I started last semester analyzing different journals and their articles using text analysis. This semester I wanted to delve deeper and wished to accomplish three things: gathering more journals, incorporating news articles like the New York Times or Washington Post, and further analyze the contents of these pieces. I sought to better understand how different texts use langauge, for example if it is more polarizing, negative, or positive language, what types of subjects journals focus on, are certain journals more popular than others, and if there is an incentive to discuss certain subjects over others.

Scraping the New York Times

I first started by gathering data from the NYT which requires an API key to gather article information. To get an API key, create a developer account and create an API key. Go to here for more information. Create an API key under "Archive API" which allows users to get all NYT articles for a given month. Once your app is created, copy the API key and paste into the function below.

```
api <- rstudioapi::askForSecret("Api")
```

The data was formated in JSON which made grabbing the data far easier compared to web scraping journal articles. I was interested in collecting NYT data from 2000 to the present to compare differences in language in topics over time. To do this, I used the format the NYT recommends to gain access to the data and simply pasted together the url, date, and API key (`paste0('https://api.nytimes.com/svc/archive/v1/', dates, '.json?api-key=', api)`).

```
# create vector of dates to look at, so we're looking at NYT articles from 2000 to 2018
dates <- paste(rep(2000:2018, each = 12), rep(1:12, 19), sep = "/")

# add dates in from 2018 since there isn't a complete archive of 2018 yet
dates[229:231] <- c("2019/1", "2019/2", "2019/3")

# create urls unique to the given date and api key
url <- paste0('https://api.nytimes.com/svc/archive/v1/', dates, '.json?api-key=', api)

# I'm only using two json links for demonstration purposes and so it doesn't take forever to run - DON'
url <- url[1:2]
```

This gets all the information from the JSON file.

```
call_urls <- function(x){
  fromJSON(url[x])
}

urls <- map_df(seq_along(url), call_urls)
```

This filters for the column sections I want and converts the info into a dataframe.

```
# Sections to include in apis
sections_interest <- c("World", "U.S.", "World; Washington", "World; Front Page",
```

```
                        "Washington", "Front Page; U.S.", "World; Washington",
                        "World; Front; Washington", "U.S.; Washington",
                        "Front Page; U.S.; Washington", "Education; U.S.", "World; Education",
                        "Technology; U.S.; Washington",
                        "Technology; World", "Technology; Education",
                        "Technology; Science; Health")

# Convert first group of apis to dataframes
select_element <- function(x){
  temp <- test_url[[x]][["response"]][["docs"]]

  temp <- temp %>%
  select(-c(blog, multimedia, headline, keywords, byline, word_count,
            slideshow_credits, subsection_name)) %>%
  filter(section_name %in% sections_interest)
}

ny_api <- map(seq_along(test_url), select_element) %>% rbindlist()

ny_api <- ny_api %>%
  select(web_url, pub_date) %>%
  mutate(pub_date = str_extract(pub_date, "\\d{4}-\\d{2}-\\d{2}"))
```

Unfortunately there were many articles (roughly 7,900) so web scraping would take a long time. A problem I had last semester while web scraping using my old method was an error where the my code timed out. Upon further research I read that this type error occurred among web scraping to prevent over-burdening the servers. One package I found to workaround this problem was `polite`, which takes a base url like "https://www.nytimes.com", and then can string on specific, individual, ending urls to an article. It also prevents timing out by automatically waiting a few seconds between scraping. So while this process still took a long time, it was more robust since it not result in errors constantly.

While I was web scraping the data, still very far from fully web scraping roughly all 7,900 articles, I realized that all the unique, individual urls I collected were not from 2000 to the present, but instead were representative of about two months worth of articles in 2000. Given that my dataset for all the journal articles I previously collected seemed like a small sample compared to the amount of NYT articles, I thought this would be all the NYT data I filtered for. As a result, I had to sadly scrap this part of the project because it was simply not efficient to leave my computer web scraping potentially hundreds upon thousands of NYT articles when I needed to use R for other things and because it would take a very long time to use text analysis on massive amounts of data. In the future, I want to see if there's a way to speed up the process or at least know what tools to use to tackle these types of problems.

Journal Articles

Afterwards, I focused my attention on just journal articles. I expanded my dataset to encompass four different journals: the American Political Science Association, the American Journal of Political Science, the Political Science Quarterly, and Political Analysis. I selected these since they are widely known in the political science community and prestigious authors are usually published in these.

To measure popularity I used the Altmetric score, which essentially scores a journal based on how much it is discussed. It takes into account different measures for this, for example, where the article is posted such as Facebook, a blog, Twitter, or news article, and weighs the amount of times it was posted on these sites by how many people the post is likely to reach (so Twitter would receive a higher rating than say a blogpost). To gain access to a journal article's Altmetric information, I needed the journal's digitial object infomration, or doi for short. This is a way to identify individual objects on the web, and in this case all journal articles have a doi. Given that I was expanding my data in this aspect and needed all doi's, I needed to go back and redo my journal text data from last semester and web scrape again to incorporate this doi information. While it was unfortunate that I needed to redo the data, it allowed me to see the time effeciency difference in the method I used last semester, which was a lot of for-loops and automatically writing the journal's text data into individual files on my computer, versus this semester where I could use `purrr` and mapping functions to expedite the process. While the time difference was tremendous, web scraping was still very tedious.

```
# base link - this will be used to complete partial links throughout the scraping process
base_link <- "https://www.cambridge.org"

# Let's get the volume links first
vol_links <- read_html("https://www.cambridge.org/core/journals/political-analysis/all-issues")

# combine with base_link to get the full link
full_vol_links <- map2_chr(base_link, vol_links, paste0)


# these links give us access to the text
get_html_urls <- function(x){
  url <- read_html(full_vol_links[x])

 url <-  url %>%
    html_nodes(".links > li > a") %>%
    html_attr("href") %>%
    unlist() %>%
    as.vector()

 # links with "core/product" in their string will lead us to the scrapable pages
 keep_url <- str_detect(url, "core/product")
 url <- url[keep_url == T]

}

html_links <- map(seq_along(full_vol_links), get_html_urls)
# combine into a character vector
html_links <- unlist(html_links)

# combine main link with each partial link
html_links <- map2_chr(base_link, html_links, paste0)


# function to scrape text
scrape <- function(x){
  read_links <- read_html(html_links[x])

  read_links %>%
    html_nodes("#contentContainer") %>%
    html_text() %>%
    str_trim() %>%
    toString() %>%
    str_squish() %>%
    as_tibble()
}
```

```
df <- map_df(seq_along(html_links), scrape)

# grabbing dates (dates are from when the article was published online)
grab_dates <- function(x){
  url <- read_html(html_links[x])

  url %>%
    html_nodes(".source+ .published .date") %>%
    html_text() %>%
    as_tibble()
}


dates <- map_df(seq_along(html_links), grab_dates)
dates <- dates[-72,1]

# Grab the article titles
scrape_titles <- function(x){
  read_links <- read_html(html_links[x])

  read_links %>%
    html_node(".row .title") %>%
    html_text() %>%
    str_trim() %>%
    toString() %>%
    str_squish() %>%
    as_tibble()
}

titles <- map(seq_along(html_links), scrape_titles)
titles <- unlist(titles)
```

Political Analysis Scraping

```
# I want to find the impact score for each of these articles so first I need to get the doi
get_doi <- function(x){
  urls <- read_html(html_links[x])

  urls <- urls %>%
    html_nodes(".doi") %>%
    html_text() %>%
    str_replace("https://", "") %>%
    str_replace(".org", "") %>%
    paste0("https://api.altmetric.com/v1/", .)

}


dois <- map(seq_along(html_links), get_doi)
dois <- unlist(dois)

remove_dois <- c("https://api.altmetric.com/v1/10.1017/psrm.2014.8", "https://api.altmetric.com/v1/doi/
```

```r
ifelse(dois %in% remove_dois, 0, 1)

final_dois <- dois[!(dois %in% remove_dois)]


# combine data and rename columns
df_1 <- cbind(titles, df, dates, dois)
names(df_1) <- c("title", "text", "date", "doi")
df_1 <- df_1 %>% mutate(date = dmy(date))

df_1$source <- "PA"
```

American Poliical Science Scraping

```r
url <- read_html("https://www.cambridge.org/core/journals/american-political-science-review/all-issues")

partial_links <- url %>%
  html_nodes(".fourth .row") %>%
  html_attr("href")

# these are the scrapable ones
partial_links <- partial_links[1:34]
full_links <- paste0("https://www.cambridge.org", partial_links)


# titles
grab_titles <- function(x){
test <- read_html(full_links[x])
test %>%
  html_nodes(".part-link") %>%
  html_text() %>%
  gsub("\n", "", .) %>%
  as_tibble()
}

apsa <- map(seq_along(full_links), grab_titles)
apsa_titles <- unlist(apsa)

# article links
article_links <- function(x){
  test <- read_html(full_links[x])

  test %>%
  html_nodes(".part-link") %>%
  html_attr("href")
}

apsa_article_links <- map(seq_along(full_links), article_links)
apsa_article_links <- unlist(apsa_article_links)
apsa_article_links <- paste0("https://www.cambridge.org", apsa_article_links)

article_links2 <- function(x){
```

```r
test <- read_html(apsa_article_links[x])
test %>%
  html_nodes(".core-reader") %>%
  html_attr("href")
}

apsa_article_links <- map(seq_along(apsa_article_links), article_links2)
apsa_article_links <- paste0("https://www.cambridge.org", apsa_article_links)
apsa_article_links <- apsa_article_links[1:561]
apsa_article_links <-  apsa_article_links[apsa_article_links != "https://www.cambridge.orgcharacter(0)"]

# grab the text
scrape <- function(x){
test <- read_html(apsa_article_links[x])
tryCatch({
test %>%
  html_nodes("#contentContainer") %>%
  html_text() %>%
  gsub("\n", "", .) %>%
  toString() %>%
  as_tibble()
  }, error = function(e){cat("Error:", x, "failed", "\n")})
}

apsa_text <- map_df(1:502, scrape)

scrape <- function(x){
test <- read_html(apsa_article_links[x])
tryCatch({
test %>%
  html_nodes(".doi") %>%
  html_text() %>%
  gsub("\n", "", .) %>%
  toString() %>%
  as_tibble()
  }, error = function(e){cat("Error:", x, "failed", "\n")})
}

# grab the dates
grab_dates <- function(x){

  test <- read_html(apsa_article_links[x])
test %>%
  html_nodes(".source+ .published .date") %>%
  html_text() %>%
  as_tibble()
}

apsa_dates <- map_df(seq_along(apsa_article_links), grab_dates)
beep(1)

# grab titles
grab_titles <- function(x){
```

```
test <- read_html(apsa_article_links[x])

test %>%
  html_node(".row .title") %>%
  html_text()
}

apsa_titles <- map(seq_along(apsa_article_links), grab_titles)
apsa_titles <- unlist(apsa_titles)


# dois for altmetric
grab_dois <- function(x){
test <- read_html(apsa_article_links[x])
test %>%
  html_nodes(".doi") %>%
  html_attr("href") %>%
  as_tibble()
}

apsa_dois <- map_df(seq_along(apsa_article_links), grab_dois)

# bring together values
apsa_full <- cbind(apsa_dates$value, apsa_titles, apsa_text$value)
colnames(apsa_full) <- c("date", "title", "text")
apsa_full <- as.data.frame(apsa_full)
apsa_full <- apsa_full %>%
  mutate(date = dmy(as.character(date)))
```

American Journal of Political Science Scraping

```
# Get links to other pages
get_links <- function(x){
  paste0("https://onlinelibrary.wiley.com/action/doSearch?SeriesKey=15405907&content=articlesChapters&c
}

url_list <- map_chr(0:53, get_links)

# Get the unique html links and create full link version
get_unique_links <- function(x){
  temp <- read_html(url_list[x])

  temp %>%
    html_nodes(".visitable") %>%
    html_attr("href")
}

unique_links <- map(1:54, get_unique_links)
unique_links <- unlist(unique_links)
unique_links <- paste0("https://onlinelibrary.wiley.com", unique_links)
unique_links <- unique_links[1:28]
```

```r
# Titles
grab_titles <- function(x){
  unique_links_read <- read_html(unique_links[x])

  unique_links_read %>%
    html_nodes(".citation__title") %>%
    html_text() %>%
    as_tibble()
}

titles <- map(seq_along(unique_links), grab_titles)
titles <- unlist(titles)

# Dates
grab_dates <- function(x){
  unique_links_read <- read_html(unique_links[x])

  unique_links_read %>%
    html_nodes(".epub-date") %>%
    html_text() %>%
    as_tibble()
}

dates <- map(seq_along(unique_links), grab_dates)
dates <- unlist(dates)

# Grab_text
grab_text <- function(x){
  unique_links_read <- read_html(unique_links[x])

  unique_links_read %>%
  html_nodes("#article__content") %>%
  html_text() %>%
  str_trim() %>%
  str_squish() %>%
  toString() %>%
  as_tibble()
}

text <- map_df(seq_along(unique_links), grab_text)
text <- unlist(text)

# Create a dataframe for the text to go into
ajps <- cbind(dates, text, titles)
ajps$source <- "AJPS"
ajps <- ajps %>%
  mutate(dates = dmy(dates))

full_ajps <- full_txt %>%
  unnest_tokens(full_text, text, token = "sentences") %>%
  filter(source != "PSQ") %>%
  transform(source = case_when(source == "APSA" ~ "AJPS")) %>%
  select(date, full_text, name, source) %>%
```

```
  rename(dates = date, value = full_text, titles = name) %>%
  rbind(ajps) %>% arrange(desc(dates))
```

Political Science Quarterly Scraping and Collecting DOI Information

This is building on the dataset from last semester

```
url <- read_html("https://onlinelibrary.wiley.com/loi/1538165x")

# issue base page
issue_base_pages <- paste0("https://onlinelibrary.wiley.com/loi/1538165x/year/", 1998:2019)

grab_issue_links <- function(x){
issue_link <- read_html(issue_base_pages[x])

issue_link %>%
  html_nodes(".visitable") %>%
  html_attr("href")
}

issue_links <- map(seq_along(issue_base_pages), grab_issue_links)
issue_links <- unlist(issue_links)
issue_links <- paste0("https://onlinelibrary.wiley.com", issue_links)

# issue info
grab_dois <- function(x){
test <- read_html(issue_links[x])
test %>%
  html_nodes("div.issue-item") %>%
  html_nodes("a.issue-item__title.visitable") %>%
  html_attr("href")
}

psq_dois <- map(seq_along(issue_links), grab_dois)
psq_dois <- unlist(psq_dois)


# reformat PSQ - formating from the old dataset
psq_full <- full_txt %>%
  unnest_tokens(full_text, text, token = "sentences") %>%
  filter(source != "APSA") %>%
  select(date, full_text, name, source) %>%
  rename(dates = date, value = full_text, titles = name)

psq_dois <- str_replace(psq_dois, "/doi/", "")

alm <- function(x){
  tryCatch({
    altmetrics(doi = psq_dois[x]) %>%
  altmetric_data() %>%
  select(title, score, context.all.rank, context.all.count, context.all.pct, context.similar_age_3m.rank
    }, error = function(e){cat("Error:", x, "failed", "\n")})
}
```

```r
results <- map_df(seq_along(psq_dois), alm)

psq_full <- psq_full %>%
  rename(title = titles) %>%
  left_join(results, by = "title")
```

Once I web scraped all four journals with their doi information (roughly 1700 articles were collected), I could begin using the `rAltmetric` package that takes an article's doi and outputs many facets of information: the Altmetric score, how popular the article is among all journals, its specific journal, how popular it is compared to other articles of the same age, and how many times the article was tweeted.

DOI Infomattion for Political Analysis

```r
# Altmetric search
test_dois <- str_replace(df_1$doi, "https://api.altmetric.com/v1/doi/", "")

alm <- function(x){
  altmetrics(doi = test_dois[x]) %>%
  altmetric_data() %>%
  select(title, score, context.all.rank, context.all.count, context.all.pct, context.similar_age_3m.ranl
}

# These articles don't have scores
test_dois[2] <- NA # 10.1017/pan.2018.11
test_dois[10] <- NA
test_dois[11] <- NA # "10.1017/pan.2018.43"
test_dois[24] <- NA # "10.1017/pan.2018.16"
test_dois[34] <- NA # "10.1017/pan.2017.31"
test_dois[37] <- NA

results <- map_df(seq_along(test_dois), alm)

df_1 <- df_1 %>%
  left_join(results, by = "title")
```

DOI Information for the American Political Science Association

```r
# get altmetric info for apsa
apsa_dois <- str_replace(apsa_dois$value, "https://doi.org/", "")

alm <- function(x){
  tryCatch({
    altmetrics(doi = apsa_dois[x]) %>%
  altmetric_data() %>%
  select(title, score, context.all.rank, context.all.count, context.all.pct, context.similar_age_3m.ranl
    }, error = function(e){cat("Error:", x, "failed", "\n")})
}

results <- map_df(seq_along(apsa_dois), alm)
beep(1)
```

```
# failed dois - remove twitter count column
failed_dois <- "Error: 21 failed
Error: 41 failed
Error: 61 failed
Error: 84 failed
Error: 100 failed
Error: 116 failed
Error: 118 failed
Error: 122 failed
Error: 147 failed
Error: 151 failed
Error: 168 failed
Error: 183 failed
Error: 184 failed
Error: 201 failed
Error: 203 failed
Error: 206 failed
Error: 208 failed
Error: 214 failed
Error: 219 failed
Error: 221 failed
Error: 222 failed
Error: 229 failed
Error: 232 failed
Error: 252 failed
Error: 261 failed
Error: 268 failed
Error: 274 failed
Error: 276 failed
Error: 281 failed
Error: 283 failed
Error: 288 failed
Error: 290 failed
Error: 291 failed
Error: 294 failed
Error: 295 failed
Error: 309 failed
Error: 311 failed
Error: 314 failed
Error: 316 failed
Error: 324 failed
Error: 327 failed
Error: 329 failed
Error: 335 failed
Error: 339 failed
Error: 341 failed
Error: 343 failed
Error: 345 failed
Error: 346 failed
Error: 350 failed
Error: 358 failed
Error: 360 failed
Error: 361 failed
```

```
Error: 362 failed
Error: 377 failed
Error: 380 failed
Error: 381 failed
Error: 390 failed
Error: 394 failed
Error: 396 failed
Error: 403 failed
Error: 414 failed
Error: 415 failed
Error: 421 failed
Error: 423 failed
Error: 424 failed
Error: 430 failed
Error: 431 failed
Error: 432 failed
Error: 446 failed
Error: 448 failed
Error: 450 failed
Error: 452 failed
Error: 453 failed
Error: 465 failed
Error: 467 failed
Error: 471 failed
Error: 475 failed
Error: 480 failed
Error: 482 failed
Error: 484 failed
Error: 486 failed
Error: 492 failed
Error: 494 failed
Error: 495 failed
Error: 505 failed
Error: 508 failed
Error: 512 failed
Error: 515 failed
Error: 517 failed
Error: 526 failed
Error: 531 failed
Error: 532 failed
Error: 533 failed
Error: 534 failed
Error: 539 failed
Error: 540 failed
Error: 543 failed
Error: 544 failed
Error: 545 failed
Error: 546 failed
Error: 547 failed
Error: 549 failed
Error: 555 failed
Error: 557 failed
Error: 559 failed
```

```
Error: 560 failed
Error: 563 failed
Error: 565 failed
Error: 567 failed
Error: 568 failed
Error: 569 failed
Error: 572 failed
Error: 575 failed
Error: 577 failed
Error: 581 failed
Error: 584 failed"

# take the failed dois, convert them into numerical values
failed_dois <- as.numeric(unlist(str_extract_all(failed_dois, "[0-9]{1,4}")))

# index apsa_dois by the failed ones so we just grab those dois
failed_dois <- apsa_dois[failed_dois]

failed_alm <- function(x){
  tryCatch({
    altmetrics(doi = failed_dois[x]) %>%
  altmetric_data() %>%
 select(title, score, context.all.rank, context.all.count, context.all.pct, context.similar_age_3m.rank
    }, error = function(e){cat("Error:", x, "failed", "\n")})
}

# grabbed all the dois I could
failed_results <- map_df(seq_along(failed_dois), failed_alm)

failed_results$cited_by_tweeters_count <- NA
failed_results <- failed_results %>% select(1, 2, 3, 4,5, 6, 7, 8, 10,9)

apsa_altmetric_dois <- rbind(results, failed_results)

# merge with apsa_full
apsa_full <- apsa_full %>%
  filter(title != "Notes from the Editors") %>%
  left_join(apsa_altmetric_dois, by = "title")
apsa_full$source <- "APSA"
```

```
# Altmetric
grab_doi <- function(x){
test <- read_html(paste0("https://onlinelibrary.wiley.com/doi/", ajps_dois[x]))
test %>%
  html_nodes(".epub-doi") %>%
  html_attr("href")
}

ajps_dois <- map_chr(seq_along(ajps_dois), grab_doi)

ajps_dois <- str_replace(ajps, "https://doi.org/", "")

alm <- function(x){
  tryCatch({
```

```r
    altmetrics(doi = ajps_dois[x]) %>%
  altmetric_data() %>%
  select(title, score, context.all.rank, context.all.count, context.all.pct, context.similar_age_3m.ran
    }, error = function(e){cat("Error:", x, "failed", "\n")})
}

results <- map_df(seq_along(ajps_dois), alm)
```

```r
# either these actually failed or don't have the "cited_by_tweeters_count" columns
failed_dois <- "Error: 2 failed
Error: 18 failed
Error: 20 failed
Error: 34 failed
Error: 35 failed
Error: 51 failed
Error: 60 failed
Error: 71 failed
Error: 108 failed
Error: 117 failed
Error: 126 failed
Error: 130 failed
Error: 137 failed
Error: 155 failed
Error: 157 failed
Error: 161 failed
Error: 164 failed
Error: 165 failed
Error: 174 failed
Error: 178 failed
Error: 179 failed
Error: 187 failed
Error: 209 failed
Error: 227 failed
Error: 235 failed
Error: 247 failed
Error: 259 failed
Error: 260 failed
Error: 278 failed
Error: 455 failed
Error: 476 failed
Error: 486 failed
Error: 509 failed
Error: 512 failed
Error: 524 failed
Error: 528 failed
Error: 529 failed
Error: 530 failed
Error: 536 failed
Error: 541 failed
Error: 542 failed
Error: 543 failed
Error: 544 failed
Error: 545 failed
Error: 547 failed
```

```
Error: 551 failed
Error: 554 failed
Error: 556 failed
Error: 557 failed
Error: 558 failed
Error: 559 failed
Error: 560 failed
Error: 562 failed
Error: 567 failed
Error: 568 failed
Error: 570 failed
Error: 571 failed
Error: 575 failed
Error: 576 failed
Error: 577 failed
Error: 578 failed
Error: 581 failed
Error: 582 failed
Error: 583 failed
Error: 584 failed
Error: 585 failed
Error: 587 failed
Error: 589 failed
Error: 590 failed
Error: 591 failed
Error: 592 failed
Error: 594 failed
Error: 595 failed
Error: 596 failed
Error: 598 failed
Error: 599 failed
Error: 601 failed
Error: 603 failed
Error: 604 failed
Error: 606 failed
Error: 607 failed
Error: 609 failed
Error: 610 failed
Error: 611 failed
Error: 613 failed
Error: 614 failed
Error: 616 failed
Error: 617 failed
Error: 619 failed
Error: 620 failed
Error: 621 failed
Error: 622 failed
Error: 623 failed
Error: 625 failed
Error: 629 failed
Error: 632 failed
Error: 633 failed
Error: 634 failed
```

```
Error: 636 failed
Error: 637 failed
Error: 638 failed
Error: 639 failed
Error: 640 failed
Error: 642 failed
Error: 643 failed
Error: 644 failed
Error: 652 failed
Error: 658 failed
Error: 660 failed
Error: 661 failed
Error: 662 failed
Error: 664 failed
Error: 665 failed
Error: 666 failed
Error: 667 failed
Error: 668 failed
Error: 669 failed
Error: 670 failed
Error: 671 failed
Error: 672 failed
Error: 673 failed
Error: 674 failed
Error: 676 failed
Error: 677 failed
Error: 678 failed
Error: 679 failed
Error: 680 failed
Error: 682 failed
Error: 683 failed
Error: 685 failed
Error: 686 failed
Error: 687 failed
Error: 688 failed
Error: 690 failed
Error: 692 failed
Error: 693 failed
Error: 694 failed
Error: 696 failed
Error: 697 failed
Error: 698 failed
Error: 699 failed
Error: 700 failed
Error: 702 failed
Error: 704 failed
Error: 707 failed
Error: 708 failed
Error: 709 failed
Error: 711 failed
Error: 712 failed
Error: 713 failed
Error: 714 failed
```

```
Error: 716 failed
Error: 718 failed
Error: 721 failed
Error: 722 failed
Error: 723 failed
Error: 724 failed
Error: 725 failed
Error: 727 failed
Error: 728 failed
Error: 729 failed
Error: 730 failed
Error: 731 failed
Error: 732 failed
Error: 733 failed
Error: 735 failed
Error: 736 failed
Error: 739 failed
Error: 740 failed
Error: 741 failed
Error: 742 failed
Error: 744 failed
Error: 746 failed
Error: 747 failed
Error: 748 failed
Error: 749 failed
Error: 750 failed
Error: 755 failed
Error: 756 failed
Error: 757 failed
Error: 758 failed
Error: 759 failed
Error: 760 failed
Error: 761 failed
Error: 764 failed
Error: 768 failed
Error: 769 failed
Error: 773 failed
Error: 774 failed
Error: 775 failed
Error: 778 failed
Error: 779 failed
Error: 780 failed
Error: 781 failed
Error: 784 failed
Error: 785 failed
Error: 789 failed
Error: 791 failed
Error: 793 failed
Error: 795 failed
Error: 796 failed
Error: 797 failed
Error: 798 failed
Error: 800 failed
```

```
Error: 801 failed
Error: 802 failed
Error: 803 failed
Error: 804 failed
Error: 805 failed
Error: 806 failed
Error: 807 failed
Error: 808 failed
Error: 810 failed
Error: 813 failed
Error: 814 failed
Error: 815 failed
Error: 816 failed
Error: 819 failed
Error: 820 failed
Error: 822 failed
Error: 827 failed
Error: 828 failed
Error: 829 failed
Error: 830 failed
Error: 832 failed
Error: 833 failed
Error: 834 failed
Error: 836 failed
Error: 839 failed
Error: 840 failed
Error: 841 failed
Error: 842 failed
Error: 843 failed
Error: 845 failed
Error: 847 failed
Error: 849 failed
Error: 850 failed
Error: 852 failed
Error: 854 failed
Error: 856 failed
Error: 857 failed
Error: 858 failed
Error: 859 failed
Error: 860 failed
Error: 861 failed
Error: 862 failed
Error: 864 failed
Error: 865 failed
Error: 866 failed
Error: 868 failed
Error: 869 failed
Error: 871 failed
Error: 875 failed
Error: 877 failed
Error: 878 failed
Error: 879 failed
Error: 881 failed
```

```
Error: 884 failed
Error: 885 failed
Error: 886 failed
Error: 888 failed
Error: 889 failed
Error: 890 failed
Error: 891 failed
Error: 893 failed
Error: 894 failed
Error: 895 failed
Error: 896 failed
Error: 898 failed
Error: 899 failed
Error: 900 failed
Error: 901 failed
Error: 902 failed
Error: 903 failed
Error: 904 failed
Error: 905 failed
Error: 906 failed
Error: 907 failed
Error: 909 failed
Error: 910 failed
Error: 911 failed
Error: 913 failed
Error: 914 failed
Error: 915 failed
Error: 916 failed
Error: 917 failed
Error: 918 failed
Error: 920 failed
Error: 921 failed
Error: 922 failed
Error: 923 failed
Error: 924 failed
Error: 928 failed
Error: 929 failed
Error: 931 failed
Error: 932 failed
Error: 933 failed
Error: 934 failed
Error: 935 failed
Error: 936 failed
Error: 937 failed
Error: 938 failed
Error: 939 failed
Error: 940 failed
Error: 941 failed
Error: 942 failed
Error: 943 failed
Error: 944 failed
Error: 945 failed
Error: 946 failed
```

```
Error: 947 failed
Error: 948 failed
Error: 949 failed
Error: 950 failed
Error: 951 failed
Error: 953 failed
Error: 954 failed
Error: 955 failed
Error: 956 failed
Error: 957 failed
Error: 959 failed
Error: 960 failed
Error: 962 failed
Error: 963 failed
Error: 964 failed
Error: 965 failed
Error: 966 failed
Error: 967 failed
Error: 968 failed
Error: 970 failed
Error: 971 failed
Error: 972 failed
Error: 973 failed
Error: 974 failed
Error: 975 failed
Error: 976 failed
Error: 977 failed
Error: 978 failed
Error: 979 failed
Error: 980 failed
Error: 981 failed
Error: 984 failed
Error: 985 failed
Error: 986 failed
Error: 987 failed
Error: 988 failed
Error: 989 failed
Error: 990 failed
Error: 991 failed
Error: 994 failed
Error: 995 failed
Error: 996 failed
Error: 997 failed
Error: 999 failed
Error: 1001 failed
Error: 1002 failed
Error: 1003 failed
Error: 1005 failed
Error: 1006 failed
Error: 1007 failed
Error: 1008 failed
Error: 1009 failed
Error: 1010 failed
```

```
Error: 1011 failed
Error: 1012 failed
Error: 1013 failed
Error: 1016 failed
Error: 1017 failed
Error: 1018 failed
Error: 1020 failed
Error: 1021 failed
Error: 1022 failed
Error: 1023 failed
Error: 1024 failed
Error: 1025 failed
Error: 1026 failed
Error: 1027 failed
Error: 1028 failed
Error: 1029 failed
Error: 1030 failed
Error: 1032 failed
Error: 1033 failed
Error: 1034 failed
Error: 1035 failed
Error: 1036 failed
Error: 1037 failed
Error: 1039 failed
Error: 1040 failed
Error: 1041 failed
Error: 1042 failed
Error: 1043 failed
Error: 1044 failed
Error: 1045 failed
Error: 1046 failed
Error: 1047 failed
Error: 1048 failed
Error: 1049 failed
Error: 1050 failed
Error: 1052 failed
Error: 1053 failed
Error: 1054 failed
Error: 1055 failed
Error: 1056 failed
Error: 1057 failed
Error: 1059 failed
Error: 1061 failed
Error: 1062 failed
Error: 1063 failed"

# take the failed dois, convert them into numerical values
failed_dois <- as.numeric(unlist(str_extract_all(failed_dois, "[0-9]{1,4}")))

# index ajps_dois by the failed ones so we just grab those dois
failed_dois <- ajps_dois[failed_dois]

failed_alm <- function(x){
```

```r
  tryCatch({
    altmetrics(doi = failed_dois[x]) %>%
  altmetric_data() %>%
 select(title, score, context.all.rank, context.all.count, context.all.pct, context.similar_age_3m.rank
    }, error = function(e){cat("Error:", x, "failed", "\n")})
}

# grabbed all the dois i could
failed_results <- map_df(seq_along(failed_dois), failed_alm)
```

```r
# combine altmetric results
failed_results$cited_by_tweeters_count <- NA
results <- rbind(results, failed_results)
ajps <- ajps %>%
  rename(title = titles) %>%
  left_join(results, by = "title")
```

Here I combine all the data.

```r
# change class of columns
ajps$doi <- NA
cols_to_change <- c(5:12)
ajps[,cols_to_change] <- apply(ajps[,cols_to_change], 2, function(x) as.numeric(as.character(x)))

cols_to_change <- c(6:13)
political_analysis_text[,cols_to_change] <- apply(political_analysis_text[,cols_to_change], 2, function

political_analysis_text[,5:13]

psq_full$doi <- NA
cols_to_change <- c(5:12)
psq_full[,cols_to_change] <- apply(psq_full[,cols_to_change], 2, function(x) as.numeric(as.character(x)

ajps <- ajps %>% dplyr::select("dates", "title", "text", "source", "doi", everything()) %>% rename(text
psq_full <- psq_full %>% dplyr::select("dates", "title", "text", "source", "doi", everything()) %>% rena
political_analysis_text <- political_analysis_text %>% dplyr::select("date", "title", "text", "doi", eve

# combine AJPS, PSQ, and PA
full_texts <- rbind(ajps, political_analysis_text, psq_full)

# combine with APSA
apsa_full$doi <- NA
apsa_full <- apsa_full %>% select(1, 2, 3, 13, 14, everything())
cols_to_change <- c(5:12)
apsa_full[,cols_to_change] <- apply(apsa_full[,cols_to_change], 2, function(x) as.numeric(as.character(
apsa_full <- apsa_full %>% mutate(date = dmy(as.character(date)))

full_texts <- full_texts %>% rbind(apsa_full)
```

I had multiple attempts in collecting information this way and ran into a couple of problems. First, not all articles were tweeted about even though they might have been posted somewhere else. I wanted to strictly stick to tweets regarding articles for this project, so if a doi call on an article failed, it was due to two reasons. The second problem, building off the first, is that any doi that failed to return information on an article was possibly due to a lack of Twitter information on the article. The function I made to collect Altmetric information had this Twitter filter built in so the function would fail to work if a doi call failed if this information did not exist. To solve this, I used `tryCatch` that would still run the code while outputting results showing which indexed doi value failed. I copied these values, and ran them through another function that did not include the Twitter information. This resulted in a few more Altmetric information showing up. The final issue was that sometimes the doi information I had collected from articles failed to work with my function and the Altmetric function at all. I think this was due to the wrong doi being collected from web scraping because when double checking I saw that articles that did not return Altmetric information did still have Altmetric information online. In the future, I want to find a way to grab all the accurate doi information so I can have more robust results.

Findings

Text Analysis

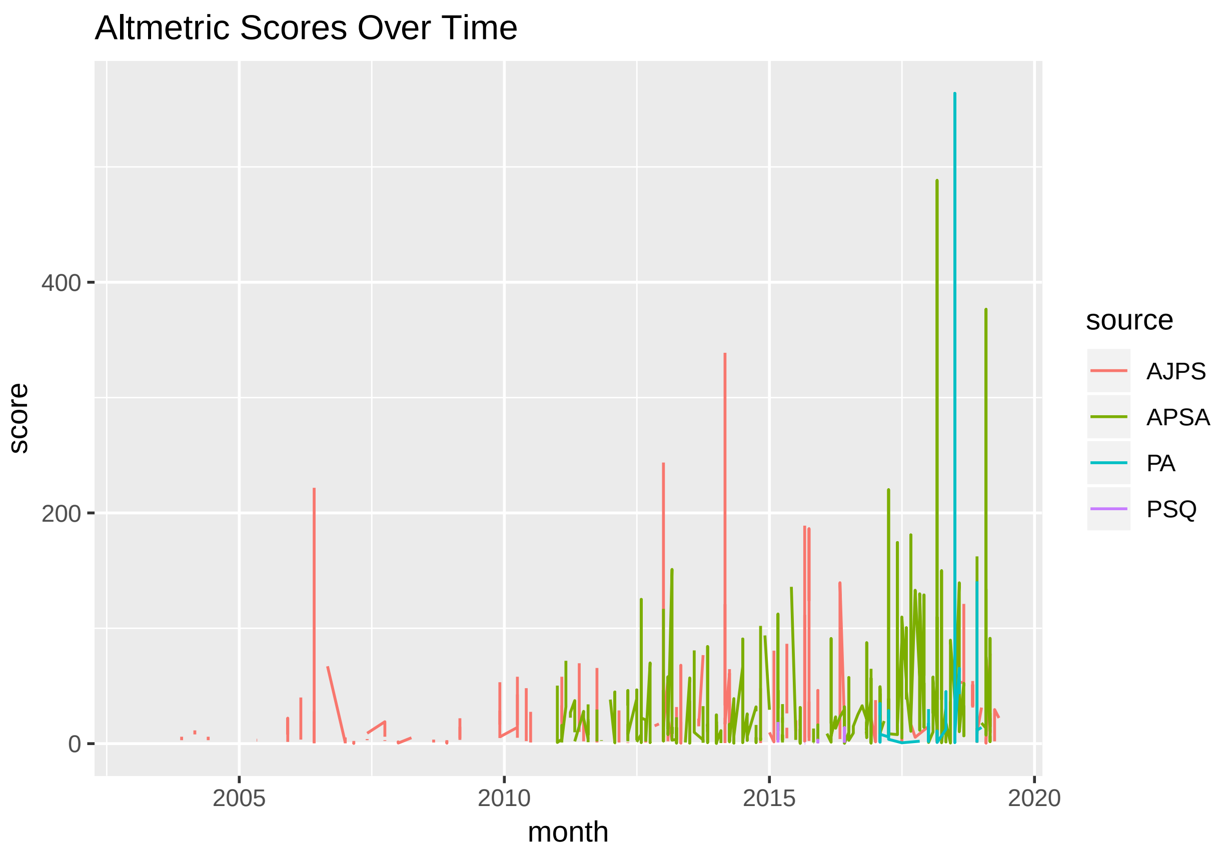
```
# Positive and Negative Political Lexicon
# Source: https://rstudio-pubs-static.s3.amazonaws.com/338458_3478e1d95ccf49bf90b30abdb4e3bd40.html
url <- read_html("https://rstudio-pubs-static.s3.amazonaws.com/338458_3478e1d95ccf49bf90b30abdb4e3bd40.h

words <- url %>%
  html_nodes("#full-lexicon td") %>%
  html_text() %>%
  as_tibble() %>%
  .[seq(1, nrow(.), 2), ]

score <- url %>%
  html_nodes("#full-lexicon td") %>%
  html_text() %>%
  as_tibble() %>%
  .[seq(2, nrow(.), 2), ]

poli_lexicon <- cbind(words, score)
colnames(poli_lexicon) <- c("word", "sent_score")
poli_lexicon <- poli_lexicon %>%
  transform(sent_score = as.double(sent_score))
```

```
my_stopwords <- tibble(word = c("stix", "1", "2", "3", "4", "0", "5", "x1d6fc", "e.g", "al", "6", "x_",


tidytexts <- full_texts %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words) %>%
  anti_join(my_stopwords) %>%
  left_join(arm_poli_lexicon, by = "word") %>%
  left_join(poli_lexicon, by = "word")
```

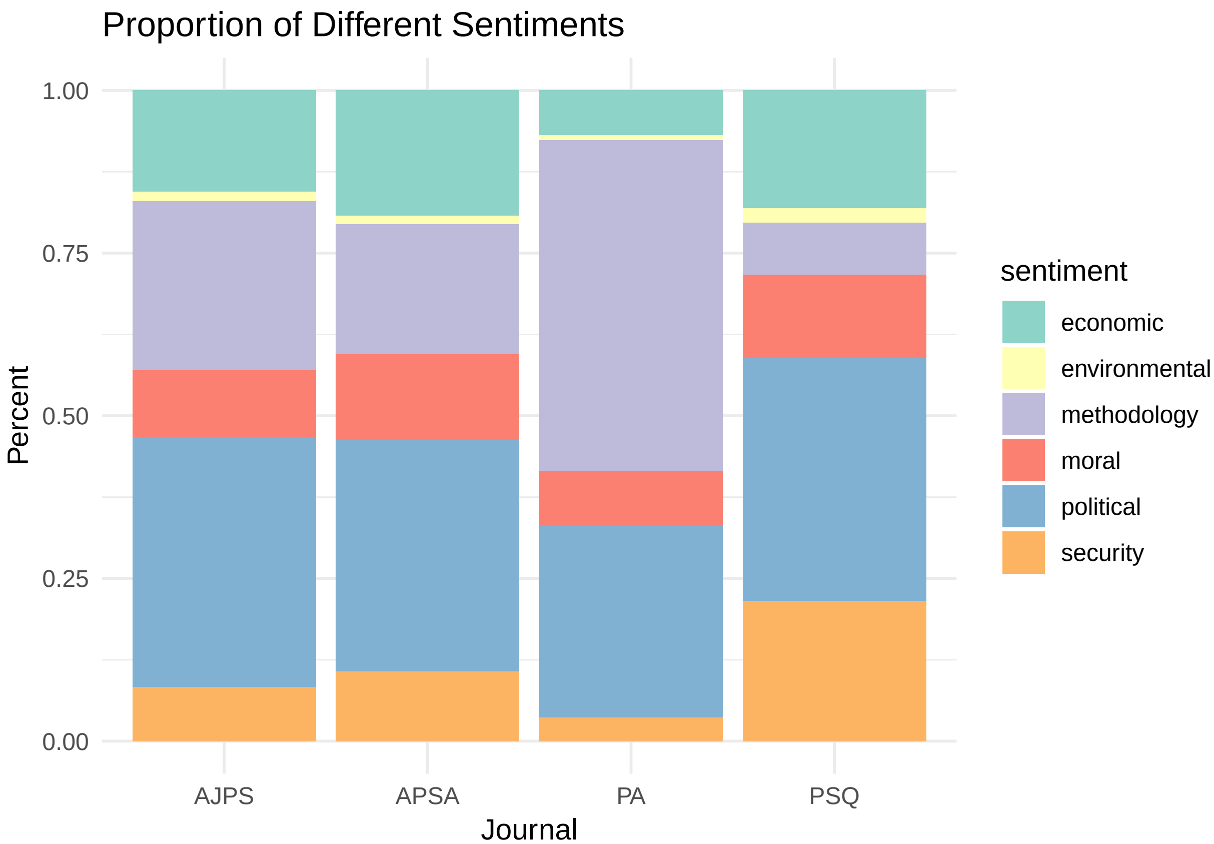
```
## Joining, by = "word"
## Joining, by = "word"
```

```
full_texts %>%
  group_by(month = floor_date(date, "month"), source) %>%
  ggplot(., aes(month, score)) +
  geom_line(aes(color = source))+
  ggtitle("Altmetric Scores Over Time")
```

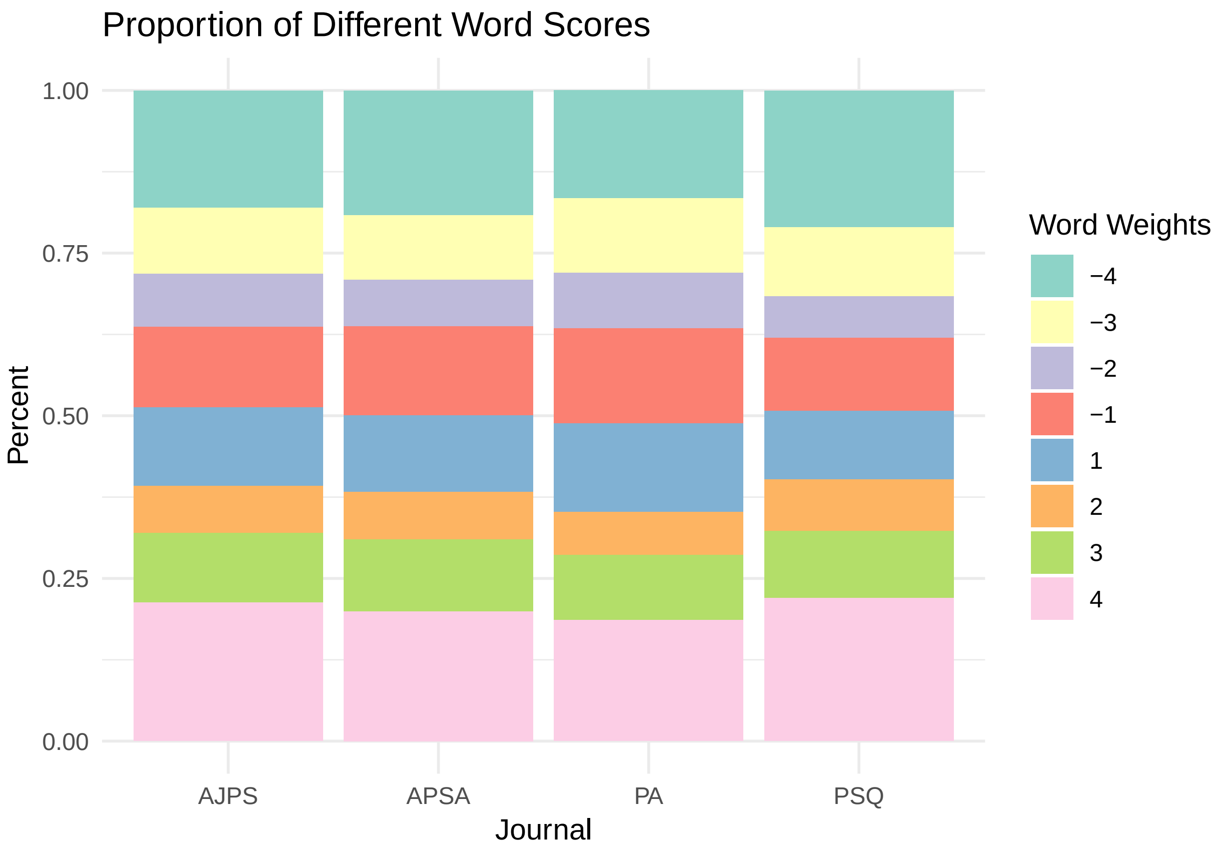
```
## Warning: Removed 11 rows containing missing values (geom_path).
```




```r
# prop of dfferent sentiments
tidytexts %>%
  filter(!is.na(sentiment)) %>%
  ggplot(., aes(source, fill = sentiment)) +
  geom_bar(position = "fill") +
  scale_fill_brewer(palette = "Set3") +
  xlab("Journal") +
  ylab("Percent") +
  ggtitle("Proportion of Different Sentiments") +
  theme_minimal()
```

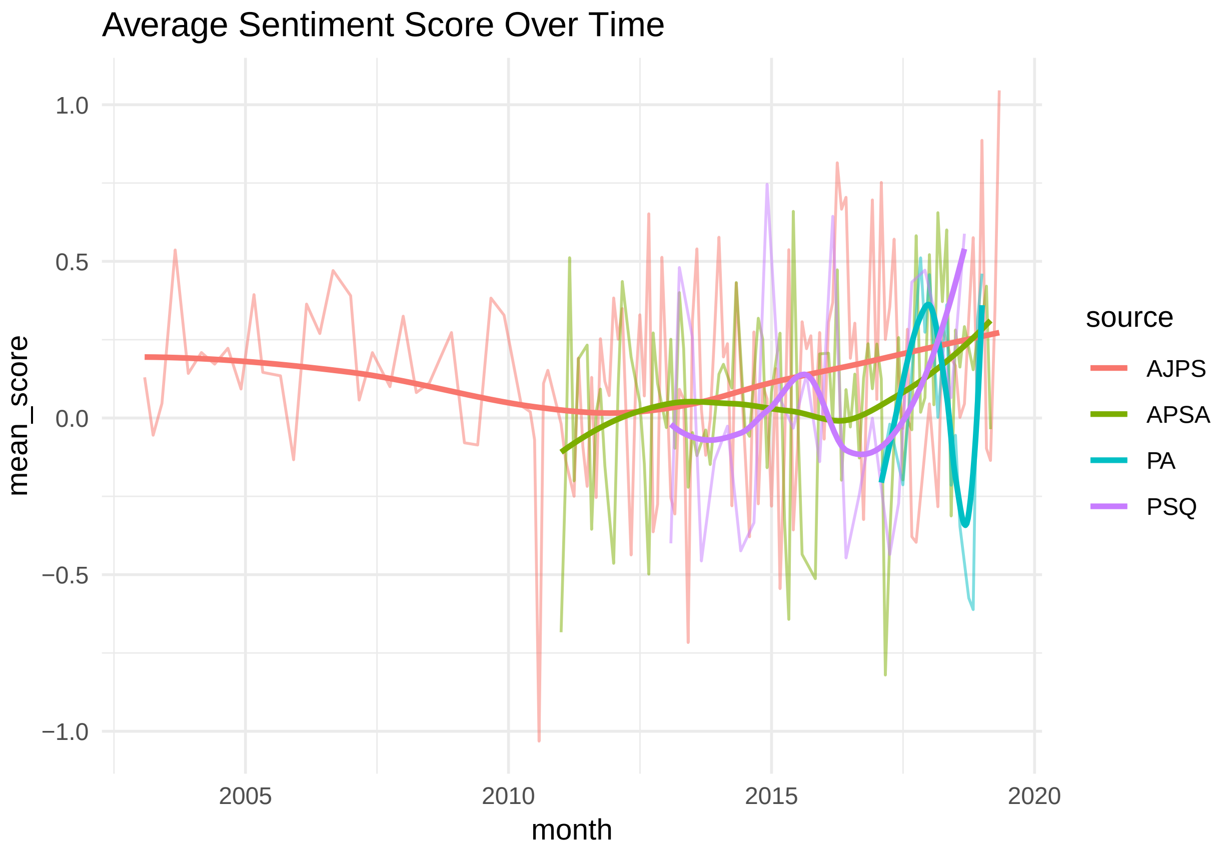



```r
# Prop of Different Word Weights
tidytexts %>%
  filter(!is.na(sent_score)) %>%
  group_by(source) %>%
 # summarize(mean_sent_score = mean(sent_score)) %>%
  ggplot(., aes(source, fill = as.factor(sent_score))) +
  geom_bar(position = "fill") +
  scale_fill_brewer(palette = "Set3") +
  xlab("Journal") +
  ylab("Percent") +
  ggtitle("Proportion of Different Word Scores") +
  theme_minimal() +
  guides(fill = guide_legend(title = "Word Weights"))
```



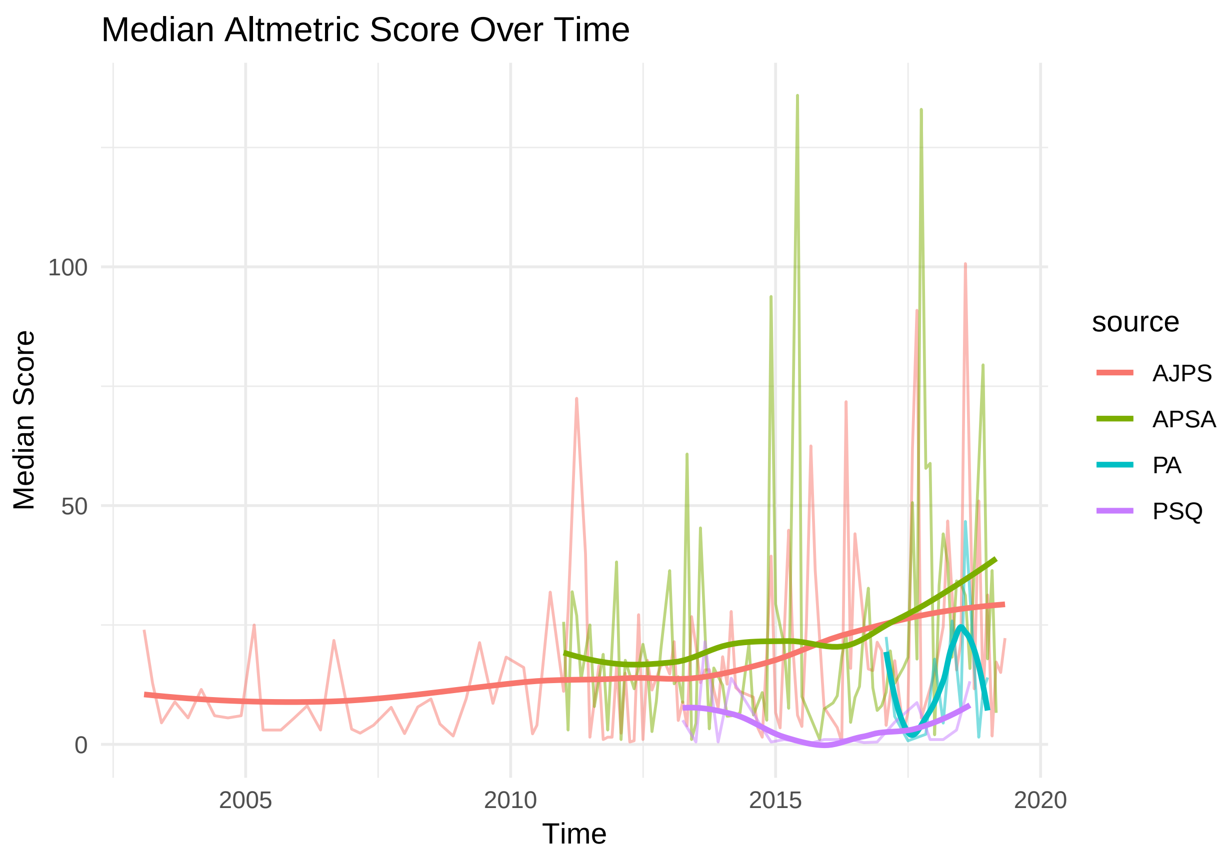

```r
# avg sentitment score over time
tidytexts %>%
  filter(!is.na(sent_score)) %>%
  group_by(month = floor_date(date, "month"), source) %>%
  summarize(mean_score = mean(sent_score)) %>%
  ggplot(., aes(month, mean_score))+
  geom_line(aes(group = source, color = source), alpha = .5) +
  geom_smooth(aes(group = source, color = source, weight = 2), se = F)+
  theme_minimal() +
    ggtitle("Average Sentiment Score Over Time")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



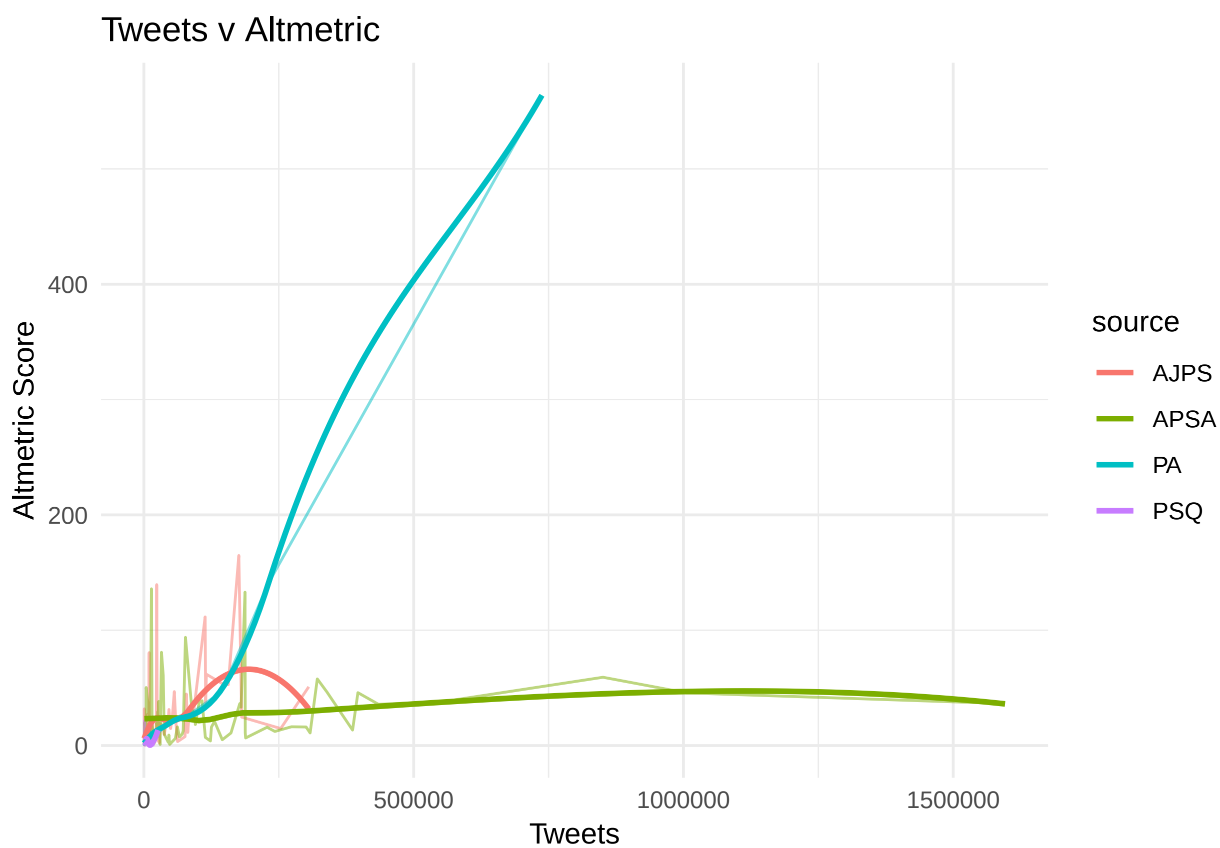

```r
#altmetric score over time
full_texts %>%
  filter(!is.na(score)) %>%
  group_by(month = floor_date(date, "month"),  source, year = floor_date(date, "year")) %>%
  summarize(med_score = median(score),
            total_tweets = sum(cited_by_tweeters_count)) %>%
  ggplot(., aes(month, med_score)) +
  geom_line(aes(color = source), alpha = .5) + theme_minimal() +
  geom_smooth(aes(color = source), se = F) +
  ggtitle("Median Altmetric Score Over Time") +
  xlab("Time") +
  ylab("Median Score")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



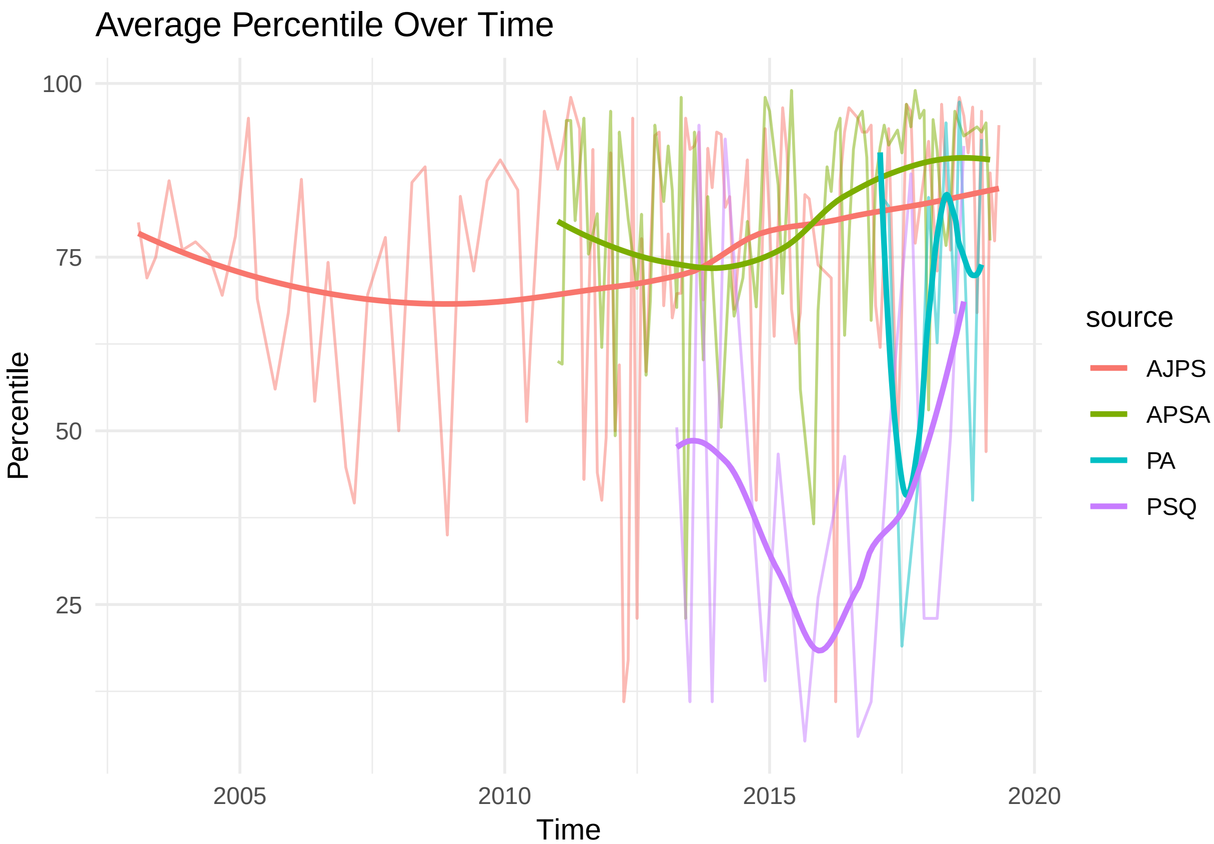

```r
# sentiment score versus altmetric score
tidytexts %>%
  filter(!is.na(sent_score)) %>%
  filter(!is.na(score)) %>%
  group_by(month = floor_date(date, "month"),  source, year = floor_date(date, "year")) %>%
  summarize(med_score = median(score),
            total_tweets = sum(cited_by_tweeters_count),
            mean_sent_score = mean(sent_score)) %>%
  ggplot(., aes(total_tweets, med_score)) +
  geom_line(aes(color = source), alpha = .5) +
  geom_smooth(aes(color = source, weight = 2), se = F)+
  theme_minimal()+
  ggtitle("Tweets v Altmetric") +
  ylab("Altmetric Score")+
  xlab("Tweets")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'

## Warning: Removed 46 rows containing non-finite values (stat_smooth).

## Warning: Removed 46 rows containing missing values (geom_path).
```



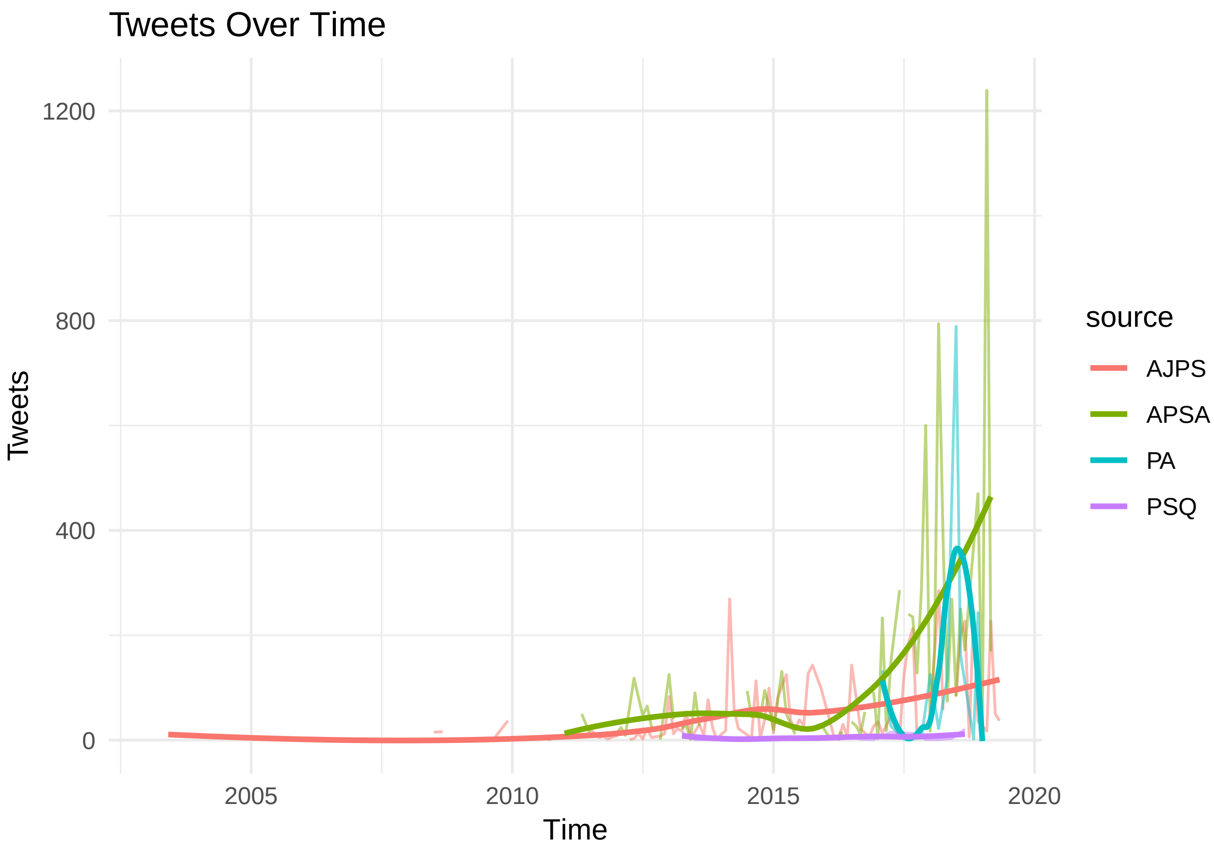

```
# percentile all time over time
full_texts %>%
  filter(!is.na(score)) %>%
  group_by(month = floor_date(date, "month"),  source, year = floor_date(date, "year")) %>%
  summarize(med_score = median(score),
            avg_pct = mean(context.all.pct),
            ) %>%
  ggplot(., aes(month, avg_pct)) +
  geom_line(aes(color = source), alpha = .5) +
  geom_smooth(aes(color = source, weight = 2), se = F) +
  theme_minimal() +
  ggtitle("Average Percentile Over Time") +
  ylab("Percentile") +
  xlab("Time")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```r
# Tweets over time
full_texts %>%
  filter(!is.na(score)) %>%
  group_by(month = floor_date(date, "month"),  source, year = floor_date(date, "year")) %>%
  summarize(med_score = median(score),
            sum_tweets = sum(cited_by_tweeters_count),
            ) %>%
  ggplot(., aes(month, sum_tweets)) +
  geom_line(aes(color = source), alpha = .5) +
  geom_smooth(aes(color = source, weight = 2), se = F) +
  theme_minimal() +
  ggtitle("Tweets Over Time") +
  ylab("Tweets") +
  xlab("Time")
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'

## Warning: Removed 46 rows containing non-finite values (stat_smooth).

## Warning: Removed 2 rows containing missing values (geom_path).
```




```r
# Top articles
full_texts %>%
  select(-text) %>%
  select(date, title, source, score) %>%
  arrange(desc(score)) %>%
  head()
```

```
## # A tibble: 6 x 4
##   date       title                                         source score
##   <date>     <chr>                                         <chr>  <dbl>
## 1 2018-07-31 Gendered Citation Patterns across Political Scie~ PA       564.
## 2 2018-03-22 Bias in Perceptions of Public Opinion among Poli~ APSA     488.
## 3 2019-02-19 Local News and National Politics              APSA     376.
## 4 2014-03-05 Conspiracy Theories and the Paranoid Style(s) of~ AJPS     339.
## 5 2014-09-02 Assortative Mating on Ideology Could Operate Thr~ AJPS     272.
## 6 2013-01-22 When Are Women More Effective Lawmakers Than Men? AJPS     244.
```

```r
# Average altmetric score
full_texts %>%
  filter(!is.na(score)) %>%
 # count(source)
  group_by(source) %>%
  summarize(mean_score = mean(score))
```

```
## # A tibble: 4 x 2
##   source mean_score
##   <chr>       <dbl>
## 1 AJPS         23.3
```

```
## 2 APSA         28.6
## 3 PA           33.5
## 4 PSQ           4.35
```

```r
# average sentiment score
tidytexts %>%
  filter(!is.na(sent_score)) %>%
  group_by(source) %>%
  summarize(mean_score = mean(sent_score))
```

```
## # A tibble: 4 x 2
##   source mean_score
##   <chr>       <dbl>
## 1 AJPS      0.128
## 2 APSA      0.0503
## 3 PA       -0.00869
## 4 PSQ       0.0567
```

Finally, I could do text analysis on my data. I built upon the lexicon I used last semester to include more terms and categories which now consist of environmental, moral, economic, security, political, and methodology related terms. While my lexicon is not exhuastive since it indexes roughly 300 terms, it still provides interesting results when used visually. In a barplot showing the proportion of different subjects used, the American Journal of Political Science and the American Association of Political Science show roughly the same distribution in terms used. The Political Analysis, as expected, contains many methodology related terms. As seen from last semester still, the Political Science Quarterly contains many security related terms. I expected to see more varying distributions but that does not seem the case.

Next, looking at positive and negative sentiment across articles I found that the proportion of positive and negative sentiment is roughly the same throughout all articles, which would initially be expected since journals tend to be tame in their language. The Political Analysis had the lowest sentiment score but the highest Altmetric score. The Political Science Quarterly had one of the highest sentiment scores and the lowest Altmetric score, making it the journal scoring in the lowest percentile on average. The American Journal of Political Science and American Political Science Association had many popular articles overtime, possibly considered as outliers. Overall, there are no extremes in positie-negative sentiment scores.

Unsupervised topic modeling can possibly be used in the future to analyze what journal articles are tweeted or talked about the most, as shown the tweets over time plot, the Political Analysis had articles frequently discussed. However, as shown by the American PS Association in the Tweets versus Altmetric score plot, an article that's tweeted more doesn't necessarily mean the article will have a higher score. Furthermore, articles that are discussed more does not mean they are always good articles, they could be discussed because they are very controversial or poorly researched.

Shiny

I was having issues uploading my Shiny to shinyapps.io for some reason but have worked out an alternative. The Shiny is on my GitHub so using the code below will compile the app and will automatically pop up once finished compiling. It takes a bit because there are a lot of parts and the tokenized dataset is huge, but it works.

```r
runGitHub("Glacieus/DataSci", "Glacieus", subdir = "Shiny/")
```

Future Work

Overall, I want to find a way to streamline my web scraping process because it is very time-consuming and I was not able to obtain all doi information for all articles. I also want to incorporate regular news articles like the New York Times, which I initially started doing but then realized the massive volume of articles put out by the NYT. Roughly 7,900 articles equated to about two months of articles. This massive amount of text would be computationally difficult to process and parse for contextual information. I additionally want to find a way to contextualize the content of articles further so I know what authors discuss. LDA can potentially help here. Lastly, I'm attending the "Text as Data" workshop on May 28th to hopefully gain insight on how to quantitatively analyze text.