Chapter 3 Text Analysis Tools Part 1, Text Mining

Now that we have some text data to work with, we will begin to learn some methods for analyzing text. This section will cover text mining. In future weeks we will discuss sentiment and language models.

3.1 Working with Text Data in R

Let's dig in a little more to into how to handle text data in R. We'll first introduce a concept called **tidytext**. Silge and Robinson define tidytext as text data in which each row is a *token*. Tokens are meaningful groups of text, such as words, sentences, or n-grams (groups of n words or letters). This concept is borrowed from Wickam's concept of tidy data, which says that each observation should be a row, each variable a column, and each type of observational unit a table.

In other words, tidy data could look something like this:

Row	Person	Birthday	Occupation
1	Joe	12/3/1963	Carpenter
2	Malik	6/8/1978	Architect
3	Suzanna	4/3/2001	Student

Or this:

Row	County	Temperature	PM2.5
1	Santa Clara	78.1	12.1
2	San Mateo	82.3	32.1
3	San Francisco	65.4	44.7

While tidy text would look something like this:

Row	Paper	Article	Text
1	New York Times	Study Compares Gas Stove Pollu…	Using
2	New York Times	Study Compares Gas Stove Pollu…	a
3	New York Times	Study Compares Gas Stove Pollu…	single

Or this:

Row	Paper	Article	Text
1	New York Times	Study Compares Gas Stove Pollution to Secondhand Cigarette Smoke	Using a single gas-stove burner can raise indoor concentrations of benzene, which is linked to cancer risk, above levels that have prompted investigations when detected outdoors.
2	New York Times	Study Compares Gas Stove Pollution to Secondhand Cigarette Smoke	For the peer-reviewed study, researchers at Stanford’s Doerr School of Sustainability measured benzene emissions from stoves at 87 homes in California and Colorado and found that natural gas and propane stoves emitted benzene at rates that frequently reached indoor concentrations above health benchmarks set by the World Health Organization and other public agencies.

Row	Paper	Article	Text
3	New York Times	Study Compares Gas Stove Pollution to Secondhand Cigarette Smoke	In about a third of the homes, a single gas burner on high or an oven set at 350 degrees for 45 minutes raised benzene levels above the upper range of indoor concentrations seen in secondhand tobacco smoke, researchers found.

What about the Guardian news data that we pulled earlier - is this in tidy text format? Read it in to your R environment, and if not, try to convert it into a tidy text table.

```
library(readr)
ca_wf <- read_csv("Data/ca_wf.csv")
```

One of the first questions that comes up when trying to convert a document into tidytext format is *what is our unit of analysis?* This is an important question for any research project, and should be given careful thought.

Perhaps we want to study whether there are differences in how blogs and news articles write about climate events. Using the Guardian dataset from earlier, we can create two dataframes of the following forms:

```
library(tidytext)
library(dplyr)

# first, set up liveblog dataframe
tidy_blogs <- ca_wf %>%
  filter(type == "liveblog")

# unnest tokens
tidy_blogs %<>%
  unnest_tokens(word, body_text) %>%
  anti_join(stop_words)

# look at examples
tidy_blogs %>%
  select(type, word) %>%
  head()
```

```
## # A tibble: 6 × 2
##   type     word
##   <chr>    <chr>
## 1 liveblog 6pm
## 2 liveblog york
## 3 liveblog city
## 4 liveblog skies
## 5 liveblog shrouded
## 6 liveblog thick
```

```r
library(tidytext)
library(dplyr)

# first, set up liveblog dataframe
tidy_blogs <- ca_wf %>%
  filter(type == "liveblog")

# unnest tokens
tidy_blogs %<>%
  unnest_tokens(word, body_text) %>%
  anti_join(stop_words)

# look at examples
tidy_blogs %>%
  select(type, word) %>%
  head()
```

```
## # A tibble: 6 × 2
##   type     word
##   <chr>    <chr>
## 1 liveblog 6pm
## 2 liveblog york
## 3 liveblog city
## 4 liveblog skies
## 5 liveblog shrouded
## 6 liveblog thick
```

```
tidy_articles <- ca_wf %>%
  filter(type == "article")
```

```
tidy_articles %<>%
  unnest_tokens(word, body_text) %>%
  anti_join(stop_words)
```

```
tidy_articles %>%
  select(type, word) %>%
  head()
```

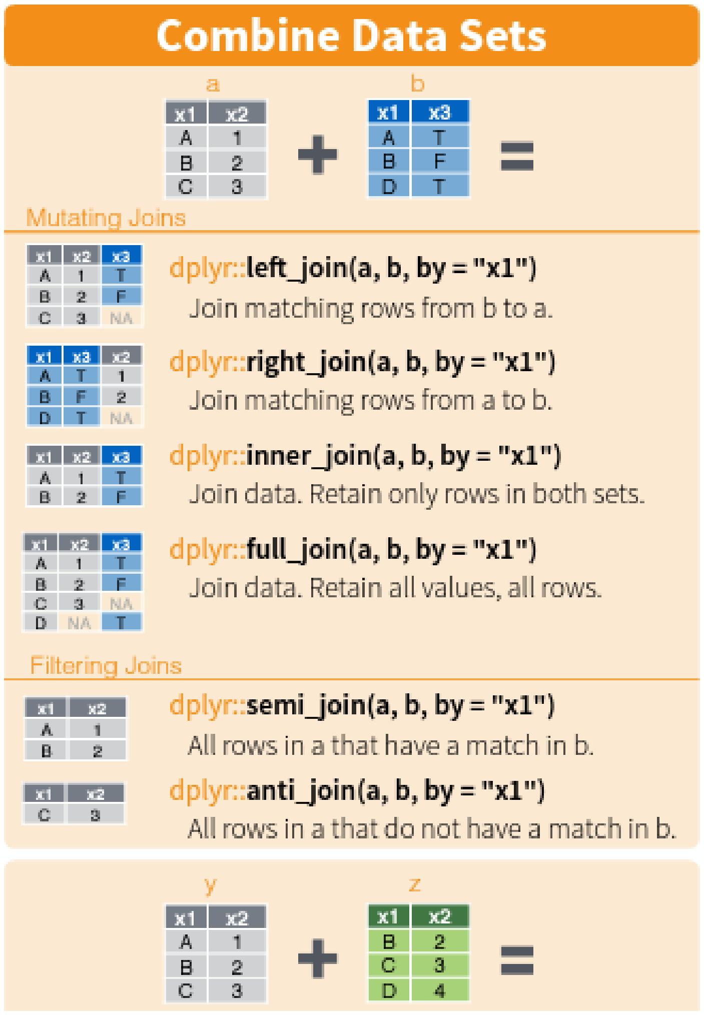
```
## # A tibble: 6 × 2
##   type    word
##   <chr>   <chr>
## 1 article poor
## 2 article air
## 3 article quality
## 4 article returned
## 5 article north
## 6 article east
```

We've used `unnest_tokens()` before. But notice the `anti_join(stop_words)` ? What do you think is happening here?

First, let's take a look at `stop_words` . You can run `View(stop_words)` , once `tidytext` is already in your library, to do the same on your machine. We see that these stop words come free three different lexicons: SMART, snowball, and onix. Stop words are essentially words that are not useful for our analyses, such as "the." Are there any words in these lists that you are surprised to see?

word	lexicon
a	SMART
a’s	SMART
able	SMART
about	SMART
above	SMART
according	SMART
accordingly	SMART
across	SMART
actually	SMART
after	SMART
afterwards	SMART
again	SMART
against	SMART
ain’t	SMART
all	SMART
allow	SMART
allows	SMART
almost	SMART
alone	SMART
along	SMART
already	SMART
also	SMART

Next, let's make sure we understand what the `anti_join` is doing. When merging two different dataframes, it is common to use "join" functions. There are two major types of joins in R: *mutating* joins and *filtering* joins. We'll use mutating joins for most instances where we want to combine two sets of information, but in this case, the filtering `anti_join()` allows us to remove all the stop words from our dataframe. The table below is a helpful guide (this is taken from a `dplyr` cheat sheet, you can see cheat sheets for various R packages here.


Combine Data Sets

a

x1	x2
A	1
B	2
C	3

+ b

x1	x3
A	T
B	F
D	T

=

Mutating Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NA

dplyr::**left_join(a, b, by = "x1")**
Join matching rows from b to a.

x1	x3	x2
A	T	1
B	F	2
D	T	NA

dplyr::**right_join(a, b, by = "x1")**
Join matching rows from a to b.

x1	x2	x3
A	1	T
B	2	F

dplyr::**inner_join(a, b, by = "x1")**
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NA
D	NA	T

dplyr::**full_join(a, b, by = "x1")**
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

dplyr::**semi_join(a, b, by = "x1")**
All rows in a that have a match in b.

x1	x2
C	3

dplyr::**anti_join(a, b, by = "x1")**
All rows in a that do not have a match in b.

y

x1	x2
A	1
B	2
C	3

+ z

x1	x2
B	2
C	3
D	4

=

Now that we have removed stop words and organized our data, we can do things like examine the word frequencies in articles and blogs.

```
# look at blog word frequencies
tidy_blogs %>%
  count(word, sort = TRUE)
```

```
## # A tibble: 2,379 × 2
##    word          n
##    <chr>     <int>
##  1 air         111
##  2 quality      69
##  3 smoke        68
##  4 wildfires    64
##  5 trump        59
##  6 pence        58
##  7 york         58
##  8 canada       55
##  9 president    55
## 10 city         46
## # ℹ 2,369 more rows
```

```
# look at article frequencies
tidy_articles %>%
  count(word, sort = TRUE)
```

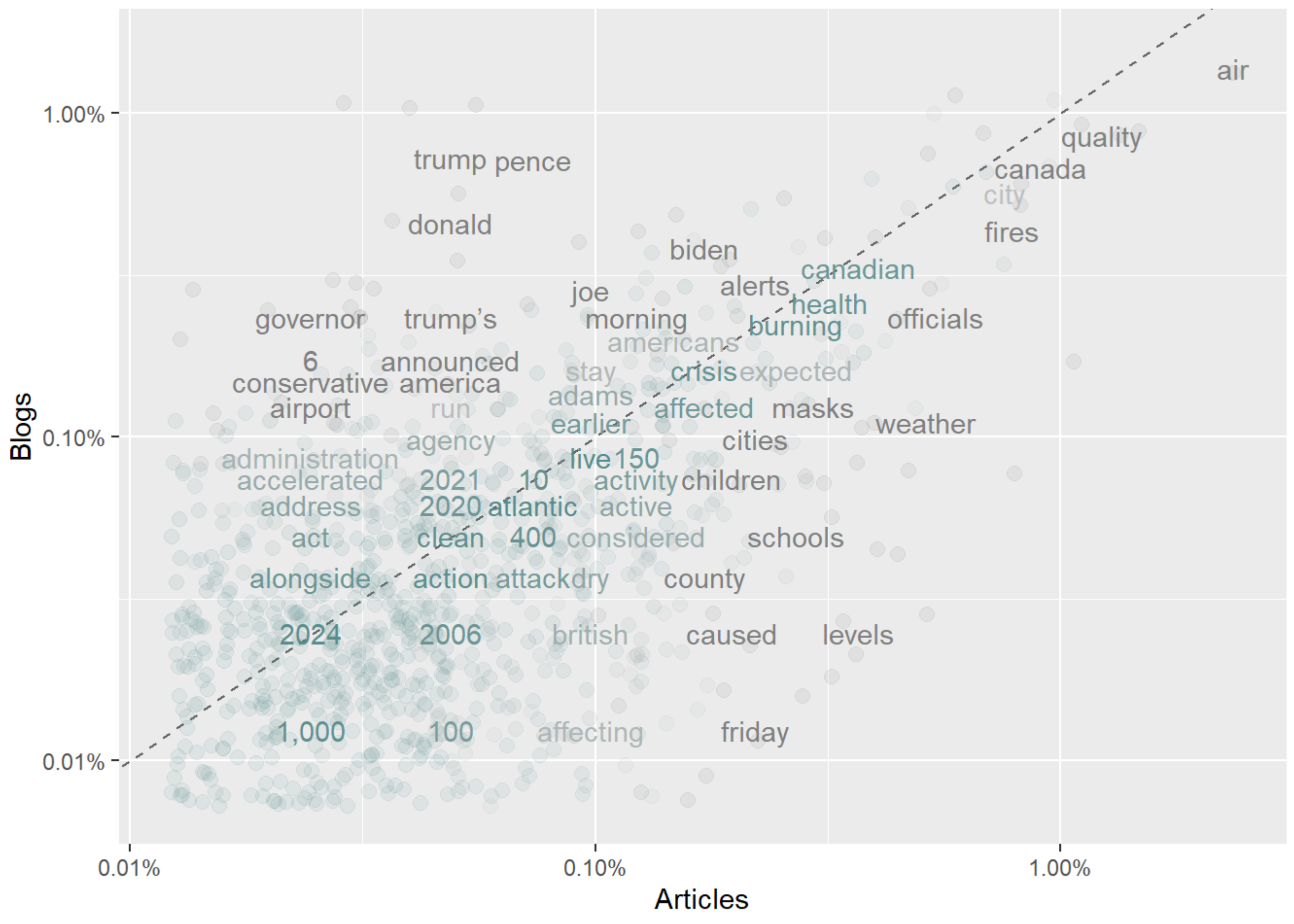
```
## # A tibble: 1,808 × 2
##    word          n
##    <chr>     <int>
##  1 air          96
##  2 smoke        58
##  3 quality      50
##  4 york         40
##  5 canada       37
##  6 wildfires    34
##  7 climate      33
##  8 fires        32
##  9 city         31
## 10 wednesday    30
## # ℹ 1,798 more rows
```

Both samples contain words like “air,” “quality,” and “fire,” which are not surprising, since our searches included these terms. In this small sample, the blogs contain more words like “trump,” “pence,” and “president,” suggesting that these news stories may be more political than the articles.

```
library(tidyr)
frequency <- bind_rows(tidy_blogs,
                       tidy_articles) %>%
  count(type, word) %>%
  group_by(type) %>%
  mutate(proportion = n / sum(n)) %>%
  select(-n) %>%
  pivot_wider(names_from = type, values_from = proportion)

frequency %>%
  head()
```

```
## # A tibble: 6 × 3
##   word     article  liveblog
##   <chr>      <dbl>     <dbl>
## 1 1,000   0.000245  0.000144
## 2 1,300   0.000245 NA
## 3 1,624   0.000245  0.000144
## 4 10      0.000735  0.000864
## 5 100     0.000490  0.000144
## 6 100,000 0.000490 NA
```

```r
library(scales)
library(ggplot2)

# expect a warning about rows with missing values being removed
ggplot(frequency, aes(x = article, y = liveblog,
                      color = abs(article - liveblog))) +
  geom_abline(color = "gray40", lty = 2) +
  geom_jitter(alpha = 0.1, size = 2.5, width = 0.3, height = 0.3) +
  geom_text(aes(label = word), check_overlap = TRUE, vjust = 1.5) +
  scale_x_log10(labels = percent_format()) +
  scale_y_log10(labels = percent_format()) +
  scale_color_gradient(limits = c(0, 0.001),
                       low = "darkslategray4", high = "gray75") +
  theme(legend.position="none") +
  labs(x = "Articles", y = "Blogs")
```



3.2 Hurricane Katrina and the National Flood Insurance Program

Before jumping into the models and coding, we'll take a look at the costliest climate disaster to hit the U.S.: Hurricane Katrina. We focus on this event because of the scale of its destruction, but also because of what it reveals about race, class, and politics in the U.S., and what this may mean for a climate-changed future.

The readings for this week cover Katrina in a bit more detail. If you have the time, I highly recommend listening to the entire Floodlines podcast series, or watching the movie Katrina Babies. For now, we will mostly focus on various governmental responses to the disaster.

Hurricane Katrina provided a political opening for reforms to the National Flood Insurance Program (NFIP), which insures flood losses in the United States. The program was created in 1968, a few years after Hurricane Betsy brought widespread destruction to Louisiana. For various reasons, the NFIP had endured financial losses that it likely would never be able to repay. Therefore, when Katrina hit Louisiana in 2005, another window of opportunity for change in policy opened. The discourse and lobbying would ultimately lead to changes enacted in the Biggert-Waters Flood Insurance Reform Act of 2012.

We are going to look at congressional hearing records, conducted in the wake of Katrina, to get a sense of how the disaster was portrayed among lawmakers and through official government channels. The congressional hearing records that we are using only go until 2010, so we will not be able to analyze everything in the run-up to Biggert-Waters. Still, we can look at how Katrina was discussed in congress in the seven years following the storm.

3.3 Gathering Data from ProQuest

Last week, we learned how to gather text data ourselves from APIs and using web scrapers. This week, I want to introduce another useful resource for gathering text data: the Pro Quest Text and Data Mining Studio. Pro Quest is one of the largest databases out there on news articles, scholarly articles, government reports, and dissertations. The TDM Studio allows users to create their own databases of news articles or congressional hearings, and to either visualize the information directly or work with the data in Jupyter Notebooks.

To get started with ProQuest TDM Studio, start an account here. Try producing a visualization. Make sure that you check all three boxes for "geographic analysis," "sentiment analysis," and "topic modeling" (we will learn these in more detail in the next three weeks). Then spend some time thinking about the precise search terms you want to use and the types of media you want to include or exclude. Then submit your request! These can take hours to complete, so we you probably want to check back in on them later.

For now, we'll try the other application in ProQuest TDM Studio: the workbench. Try clicking "create new database" and choosing between publication titles, ProQuest databases, and congressional hearings. Here, we'll look at data from the most recent congressional hearings (2003-2010).

From the workbench dashboard, you can click "Open Jupyter Notebook" to begin. Within the Jupyter Notebook, you can click the "data" folder to see the name of your custom dataset. Then go to Getting Started R -> 2022.5.13 -> ProQuest TDM Studio R Samples. In "R Display Document Counts" and "R Convert to Dataframe," change `SAMPLENAME` to your database name to get the number of records and save your file to a .csv.

Now, you can begin a new Jupyter Notebook by following the instructions in Getting Started/2022.05.25/ProQuest TDM Studio Manuals/TDM_Studio_Manual.ipynb. You could also click "New" and "R" (or whatever program you want to use) in the filepath that you want to save your notebook (e.g. Getting Started R/2022.05.13/ProQuest TDM Studio R Samples/), although if you set it up this way you will not be able to load in some key packages such as `tidytext` .

Once you have your own notebook, you can run the following code to read in your csv file:

```
########################################
## title: proquest exploratory analysis
## author: you!
## purpose: experiment with proquest tdm studio
## date:
########################################
library(readr)
library(dplyr)
library(stringr)

# check that your file is there (may need to change filepath depending on where your notebo
list.files("../output_files/")

## read in file (change the name to your file!)
congress_hearings <- read_csv("../output_files/Congressional_Hearings_for_NFIP_text_analysi
```

Great, now we have a dataset of congressional hearings from 2003-2010. But let's say we only are interested in those that have the words "Hurricane" and "Katrina" in the title. We can limit our data to these by running the following:

```
# filter to only hearings with "Katrina" and "Hurricane" in title
katrina_hearings <- congress_hearings %>%
  filter(str_detect(Title, "(?=.*Hurricane)(?=.*Katrina)"))

# check names of titles
katrina_hearings$Title
```

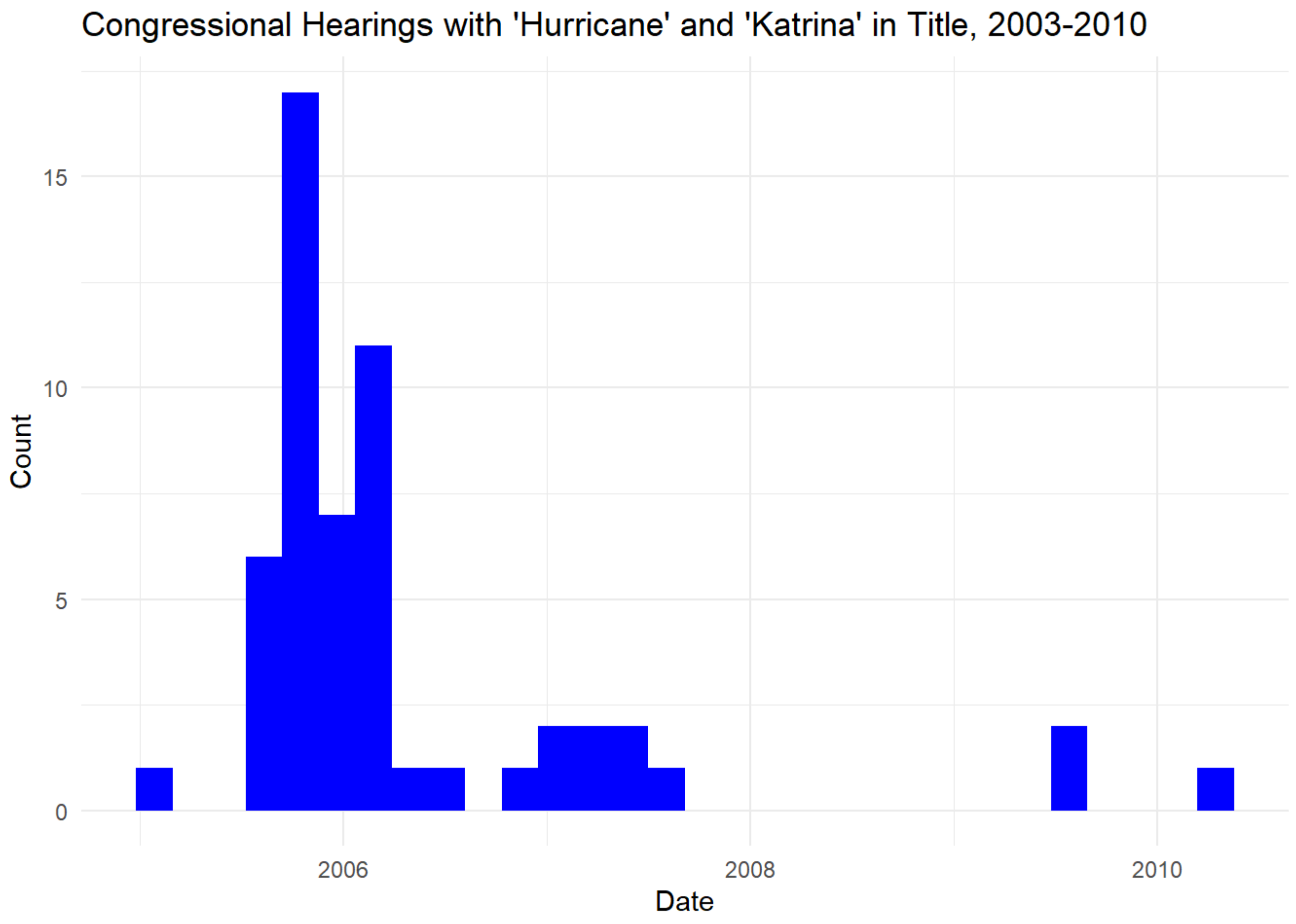
We could now do various text analyses on this dataset. If we wanted to export our results, there is a specific process for that. We can't export full text, so we will limit our data to only the dates and titles of hearings, and save these to the output file folder.

```
# limit to dates and titles
hearings_dates <- katrina_hearings %>%
  select(Title, Date)

# save to output folder
write_csv(hearing_dates, "../output_files/hearing_dates.csv")
```

Finally, we can make a copy of the Getting Started/2022.05.25/ProQuest TDM Studio Manuals/Export_Instructions.ipynb (click File -> Make a Copy…). It may help to move our export file (hearing_dates.csv) to Getting Started/2022.05.25/ProQuest TDM Studio Samples/output_files.

Once it is exported, we can read it into our local machine.

```
# read in data
hearing_dates <- read_csv("Data/hearing_dates.csv")

# histogram of dates
ggplot(hearing_dates, aes(x = Date))+
  geom_histogram(fill = "blue")+
  labs(x = "Date", y = "Count", title = "Congressional Hearings with 'Hurricane' and 'Katr
  theme_minimal()
```



In short, the pros and cons of working in ProQuest are:

- Pro: can analyze data on the cloud. This is great when working with large datasets, as it frees up your local machine to do whatever else you want to use it for.
- Con: Slow and finicky. This could change, but as of now, the ProQuest TDM system is not super fast, and may freeze or crash on you. Just something to be aware of.
- Con: Time required to output results. You can't just save your outputs from ProQuest TDM to your desktop and use them, you need to have them approved by the ProQuest team. This process shouldn't take too long (usually less than an hour), but it could take longer, so it's important to budget this into your project timeline if you use these data!

3.4 Combining Text Mining with In-Depth Reading

Sometimes, the most effective way to gather meaning from text is to combine qualitative and computational approaches. By this, I mean that you can use your machine to mine large swaths of text, but you can use other sources of knowledge to figure out how to classify data, or to refine your approach and gather meanings from the most interesting parts of your data. In practice, we are all always using some qualitative processes to determine how to advance our research questions, whether we acknowledge this or not.

As an example, we might be interested in how congressional hearings played out following Hurricane Katrina. We could use our corpus of text data from all the hearings to sort and filter to only the most relevant ones, and then read these in detail.

We can start by looking at congressional hearings here. Ideally, we would *systematically* download hearings that met a certain criteria. For example, all the hearings that come up when we search “Hurricane Katrina,” as well as possibly some other search terms, possibly from a specific committee. For now, we will look broadly, at ten different hearings on different topics related to Katrina. All of these showed up in the first three pages of search results for “Hurricane Katrina.”

We will download PDF documents for each of the hearings of interest, and put them in the same folder. To begin, we can read in the hearing called “Why did the Levees Break?” we will use a package called `pdftools` to get the text from the PDF files into R.

```
library(pdftools)
# read hearing into R
levees_hearing <- pdf_text("Data/Katrina_hearings/katrina_hearing_levees.pdf") %>%
  as.data.frame()

# set column names
colnames(levees_hearing) <- c("text")
```

Great! Let’s take a look at our data.

text
S. Hrg. 109–526 HURRICANE KATRINA: WHY DID THE LEVEES FAIL? HEARING BEFORE TI COMMITTEE ON HOMELAND SECURITY AND GOVERNMENTAL AFFAIRS UNITED STATES SENATE ONE HUNDRED NINTH CONGRESS FIRST SESSI NOVEMBER 2, Printed for the use of Committee on Homeland Security and Gove (VerDate 0ct 09 2002 09:37 Aug 31, 2006 Jkt 024446 PO 00000 Frm 00001 Fmt 6011 Sfmt 60
HURRICANE KATRINA: WHY DID THE LEVEES FAIL? VerDate 0ct 09 2002 09:37 Aug 31, 2006 Jkt 024446 PO 00000 Frm 00002 Fmt 6019 Sfmt 60

We have a dataframe with one row per page of the congressional hearing document.

```
# unnest tokens
tidy_hearing <- levees_hearing %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)

# take a look
head(tidy_hearing)
```

```
##        word
## 1       hrg
## 2       109
## 3       526
## 4 hurricane
## 5   katrina
## 6    levees
```

Let's look at the top words from the hearing on levees.

```
# look at top words
tidy_hearing_counts <- tidy_hearing %>%
  count(word, sort = TRUE)
```

```
# now we can look at the top 20
head(tidy_hearing_counts, 20)
```

```
##          word   n
## 1          09 726
## 2        6601 617
## 3        2006 365
## 4          37 365
## 5        2002 364
## 6          31 364
## 7       00000 363
## 8      024446 363
## 9         0ct 363
## 10      24446 363
## 11        aug 363
## 12       docs 363
## 13        fmt 363
## 14        frm 363
## 15        jkt 363
## 16        pat 363
## 17         po 363
## 18        psn 363
## 19   saffairs 363
## 20       sfmt 363
```

Hmmm, our top three words all look like junk text. This doesn't seem right! We can search for some of these "words" in our PDF file (just using ctrl+f), and realize that they are mostly invisible document markers. Therefore, we should remove them. We can do this by adding all of these "junk words" to our `stop_words` dataset.

We can do this by first using the `View()` function to examine the words and `n` values for `tidy_hearing`. We notice that all of the words that appear over 300 times are junk words (as a side note: why do we think there are so many words that appear 363 times?).

Next, we can use the `filter` and `select` functions to limit our data to the words that appear over 300 times. These functions are incredibly useful, so be sure to familiarize yourself with these if you are unsure what is going on here. We can also search the document for other words that we may want to remove. For example, we will include the word "6633" in our new stop words.

```
library(magrittr)
# vector for additional stop words
addl_stop_words <- tidy_hearing_counts %>%
  filter(n > 300 ) %>%
  select(word)

# include 6633 as well
addl_stop_words %<>%
  bind_rows(data.frame(word = "6633"))
```

Now we have the additional stop words, and we want to add these to our `stop_words` data. We can do this by creating a custom dataframe of stop words:

```
# add additional stop words
custom_stop_words <- bind_rows(data.frame(word = addl_stop_words,
                                          lexicon = c("custom")),
                               stop_words)

# examine new stop words dataset
head(custom_stop_words)
```

```
##   word lexicon
## 1   09  custom
## 2 6601  custom
## 3 2006  custom
## 4   37  custom
## 5 2002  custom
## 6   31  custom
```

Let's remove these from our hearing data as well, and then look at the most frequent words.

```
# remove custom stop words
tidy_hearing %<>%
  anti_join(custom_stop_words)

# look at top words
tidy_hearing_counts <- tidy_hearing %>%
  count(word, sort = TRUE)



# now we can look at the top 20
head(tidy_hearing_counts, 20)
```

```
##         word   n
## 1     levees 127
## 2    senator 126
## 3      corps 121
## 4      levee  92
## 5    orleans  91
## 6      slide  86
## 7         dr  80
## 8       seed  76
## 9  hurricane  72
## 10  chairman  62
## 11     water  62
## 12 engineers  60
## 13       van  57
## 14   heerden  56
## 15     level  56
## 16     canal  55
## 17     storm  54
## 18     surge  48
## 19 lieberman  47
## 20 nicholson  44
```

Great! We see that words like “levees,” “senator,” and “corps” are the most prominent. These are all fitting with the topic and the setting (corps probably being used as in “the Army Corps of Engineers”, who built the levees). Are there any surprises in our top 20?

It is wise to qualitatively explore our text documents as we learn more about them. We notice that the word “van” appears 57 times, but upon inspection we see that van Heerden was the Head of the Louisiana State Data Forensics Gathering Team and a key witness for the hearing. “Seed” is also a name, and “slide” is used in the context of van Heerden and Nicholson saying “next slide” many times. We can also learn more about what happened during Hurricane Katrina by reading the hearing testimony. For example, Joe Lieberman states that “if the levees had done what they were designed to do, a lot of the flooding of New Orleans would not have occurred, and a lot of the suffering that occurred as a result of the flooding would not have occurred,” which is confirmed by Dr. Seed, who was leading the National Science Foundation’s investigation of the levees.

Finally, we can get details by qualitatively examining documents that are impossible through text analysis alone. For example, we can look at photos of distressed or damaged levees like the one below to see more details on the structural failures.

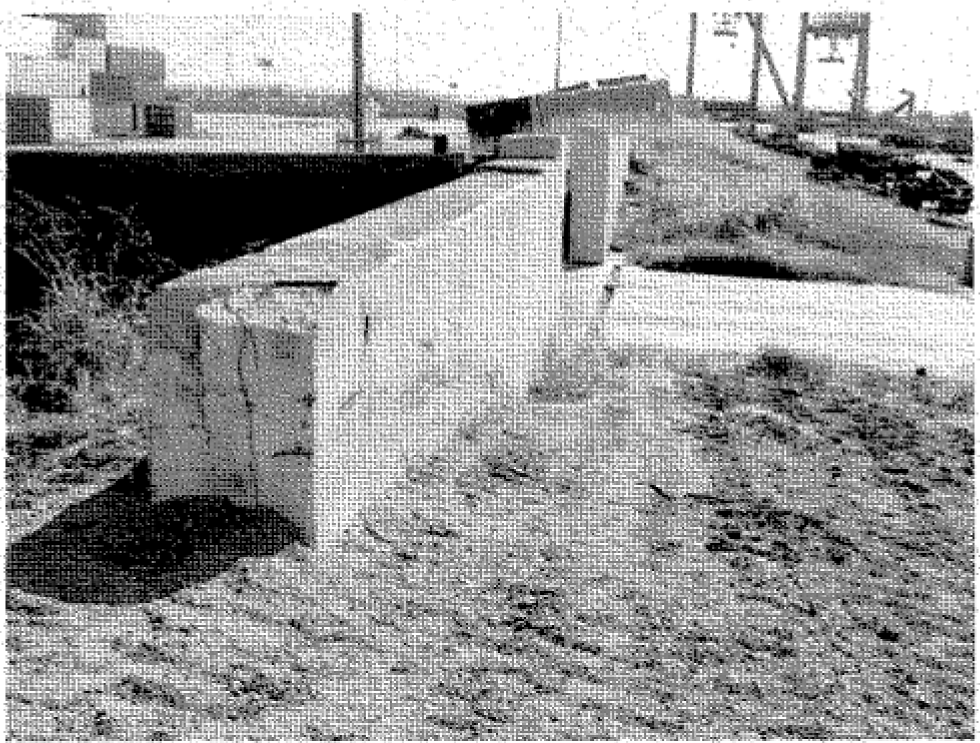


Figure 2.32: Common structural wall (with flood gate) and earthen levee transition problem with erosion at the contact between the earth and concrete sections.

3.5 Comparing Multiple Text Documents

To get a broader understanding of how text is used in this hearing, we might want to compare it to another. Let's download and use the hearing on the National Flood Insurance Program. We can combine the two hearings into a single dataframe using the `bind_rows` function (and we also add a variable called `hearing` to each to specify which hearing it is).

```
# read hearing into R
nfip_hearing <- pdf_text("Data/Katrina_hearings/katrina_hearing_nfip.pdf") %>%
  as.data.frame()

# set column names
colnames(nfip_hearing) <- c("text")

# set dataframe with both hearings
df_hearings <- bind_rows(nfip_hearing %>%
                           mutate(hearing = "nfip"),
                         levees_hearing %>%
                           mutate(hearing = "levees"))
```

Like before, we will use `unnest_tokens` to get our data in `tidytext` format. I've also added some custom stop words that are relevant to the NFIP hearing, which I don't display here, because the process is the same.

```
# get our data in tidytext format
df_hearings_tidy <- df_hearings %>%
  unnest_tokens(word, text) %>%
  anti_join(custom_stop_words)
```

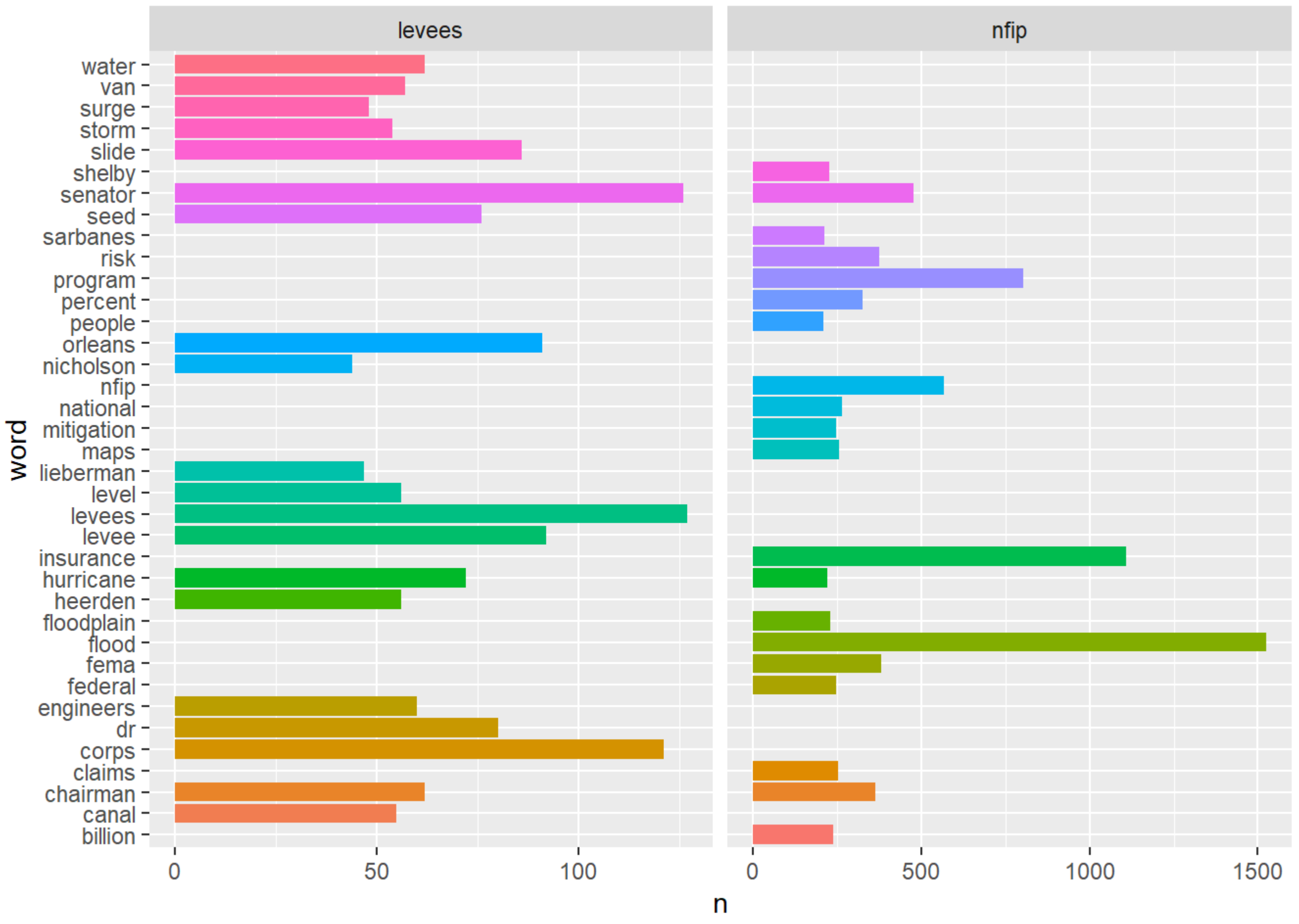
```
##         word
## 1         09
## 2       6601
## 3       2006
## 4         37
## 5       2002
## 6         31
## 7      00000
## 8     024446
## 9        0ct
## 10     24446
## 11       aug
## 12      docs
## 13       fmt
## 14       frm
## 15       jkt
## 16       pat
## 17        po
## 18       psn
## 19  saffairs
## 20      sfmt
## 21       txt
## 22   verdate
## 23      6633
## 24        12
## 25        35
## 26        19
## 27      2007
## 28    000000
## 29     33994
## 30       jul
## 31     kevin
## 32    sbank4
## 33      6621
```

Like before, we can gather counts of words in the two documents. However this time, we will use `group_by()` to specify that we want *separate* counts for each of the hearings.

```
# look at top words for each hearing
tidy_hearing_counts <- df_hearings_tidy %>%
  group_by(hearing) %>%
  count(word, sort = TRUE)



# limit to top 20 words in each hearing
tidy_hearing_counts %<>%
  filter(row_number() < 21)

tidy_hearing_counts %>%
  head(20)
```

```
## # A tibble: 20 × 3
## # Groups:   hearing [1]
##    hearing word           n
##    <chr>   <chr>      <int>
##  1 nfip    flood       1524
##  2 nfip    insurance   1109
##  3 nfip    program      803
##  4 nfip    nfip         567
##  5 nfip    senator      476
##  6 nfip    fema         381
##  7 nfip    risk         375
##  8 nfip    chairman     362
##  9 nfip    percent      326
## 10 nfip    national     265
## 11 nfip    maps         257
## 12 nfip    claims       253
## 13 nfip    federal      248
## 14 nfip    mitigation   248
## 15 nfip    billion      237
## 16 nfip    floodplain   228
## 17 nfip    shelby       226
## 18 nfip    hurricane    222
## 19 nfip    sarbanes     211
## 20 nfip    people       208
```

And we’ll do a side-by-side plot of the top words for each hearing.



Perhaps unsurprisingly, there is not too much overlap between the two hearings. Words like “senator,” “chairman,” and “hurricane” appear in the top 20 words in both hearings, but otherwise these words are distinct. It may be that if we looked at a deeper list of words for each we would see substantially more overlap.

3.6 Comparing Word and Document Frequencies

Extending the previous example, let’s say we have several documents and we want to compare word frequencies in each. We will use 11 different hearings related to Hurricane Katrina.

```r
# create vector with names for each hearing
katrina_hearings <- c("crime",
                      "environment",
                      "evacuation",
                      "finance",
                      "flood",
                      "fraud",
                      "housing",
                      "military",
                      "nextphase")



# use a for loop to gather them all
for(hearing in katrina_hearings){

  # read hearing into R
new_hearing <- pdf_text(paste0("Data/Katrina_hearings/katrina_hearing_",
                               hearing,
                               ".pdf")) %>%
  as.data.frame()

# set column names
colnames(new_hearing) <- c("text")

# specify which hearing it is
new_hearing %<>%
  mutate(hearing = paste(hearing))

df_hearings <- bind_rows(df_hearings,
                         new_hearing)

}
```

We've looked at word frequencies in the previous sections, but these frequencies do not account for words that are common *across* documents, such as "senator" or "chairman." We might instead be interested in words that are uniquely prevalent in each document. To examine this, we will use Text Frequency-Inverse Document Frequency, or TF-IDF. The IDF part of this concept can be calculated as follows:

$$IDF(term) = ln\left(\frac{n_{documents}}{n_{documentscontainingterm}}\right)$$

And the TF part of the equation, which we have already used, can be represented as $TF(term) = \frac{n_{term}}{n_{alltermsindocument}}$, or the proportion of all terms in the document that are the term of interest. When we combine these, we have:

$$TFIDF = TF \cdot IDF$$

Intuitively, we can think of this as a measure of popularity of a term in a document, which adjusts (inversely) for how popular this term is in other documents. So if a term like "insurance" is popular in the NFIP hearing and does not appear very much in other hearings, we would expect this to have a high TF-IDF score, whereas a term like "senator," which appears in all the documents, would have a lower TF-IDF score. This allows us, in theory, to isolate the words that convey the most meaning about the individual content of each document.

```
hearing_words <- df_hearings %>%
  unnest_tokens(word, text) %>%
  count(hearing, word, sort = TRUE)

total_words <- hearing_words %>%
  group_by(hearing) %>%
  summarize(total = sum(n))

hearing_words <- left_join(hearing_words, total_words)
```

Similarly to above, there are a lot of "junk words" in our `hearing_words` dataframe now. Note that we have not removed stop words here. I will look through `hearing_words` to define the junk words, and I'll remove these junk words as follows:

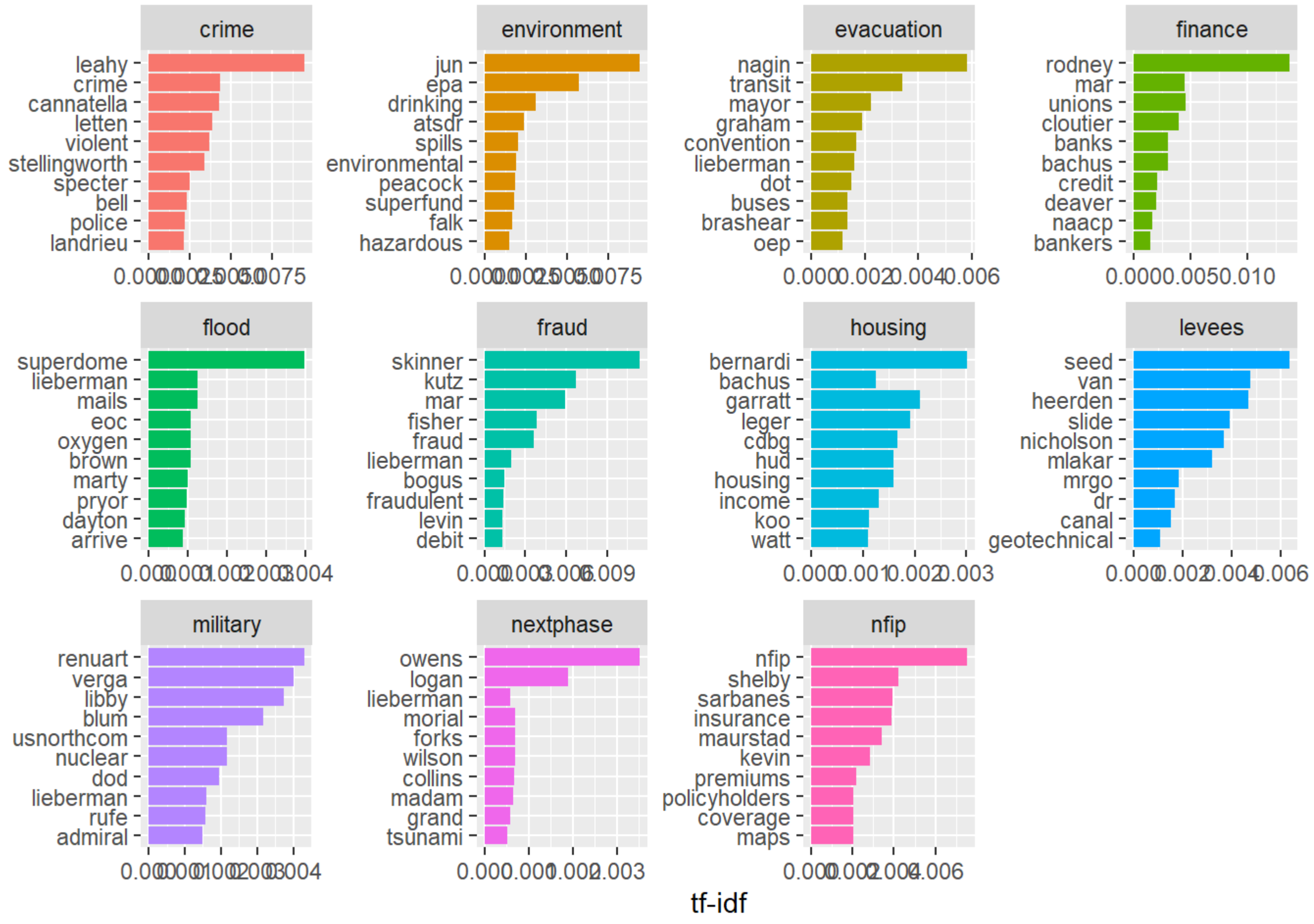
```r
junk_words_hearings <- c("6601", "09", "30", "2007", "15",
                         "po", "27", "2002", "00000", "034671",
                         "0ct", "34671", "apr", "fmt", "frm",
                         "hfin", "jkt", "k", "psn", "sfmt", "terrie",
                         "txt", "verdate", "2006", "37", "024446",
                         "24446", "aug", "c", "docs", "pat",
                         "saffairs", "sfmt", "6602", "s", "12", "35",
                         "19", "6601", "00000", "000000", "33994",
                         "jul", "sbank4", "6633", "6621", "18",
                         "34", "f", "07", "24251", "hcom1", "joep",
                         "bahamonde", "oct", "037361", "37361",
                         "027030", "27030", "040461", "40461",
                         "cmorc", "gpo", "sjud1", "109.53", "08",
                         "027023", "27023", "p", "2008", "06",
                         "024244", "24244", "feb", "g", "01",
                         "024442", "24442")

# turn to datafrmae
junk_words_hearings <- data.frame(word = junk_words_hearings)

# anti join with dataset
hearing_words %<>%
  anti_join(junk_words_hearings)
```

Finally, we can calculate the TF-IDF for this group of documents.

```r
hearing_tf_idf <- hearing_words %>%
  bind_tf_idf(word, hearing, n)
```

We can then plot the top 10 (or any number) of words for each document according to the TF-IDF.

```
library(forcats)

# limit to 10 words
hearing_tf_idf_10 <- hearing_tf_idf %>%
  group_by(hearing) %>%
  slice_max(tf_idf, n = 10) %>%
  ungroup()

# plot results
ggplot(hearing_tf_idf_10,
       aes(tf_idf, fct_reorder(word, tf_idf), fill = hearing)) +
geom_col(show.legend = FALSE) +
facet_wrap(~hearing, ncol = 4, scales = "free") +
labs(x = "tf-idf", y = NULL)
```



What do we notice? There are a lot of proper nouns - names of people and places specific to each hearing. This is how TF-IDF works. It identifies the words that are uniquely important to each document, which, in these cases, tend to be senators, names of government organizations, and programs. This is no replacement for in-depth reading and analysis, but it does give us a cursory summary of what each hearing focused on.

3.7 Problem Set 3

Due: July 17th, 2023

Recommended Resources:

Text Mining with R: A Tidy Approach

ProQuest LibGuides

1. Using the guardian_katrina.csv dataset (from Canvas Files), create a graph that compares word frequencies for different types of news stories (for example, World news vs. U.S. news). Motivate your comparison with some intuition or evidence, and tell us why this comparison is interesting. If you have more than two categories, try using the `facet_wrap()` command to display all the different graphs (an example of this can be found in the Text Mining with R book).Explain what you see in your visualization(s) and results.
2. Create an account on ProQuest DTM, and create a visualization for news related to Hurricane Katrina. Explain what news sources you included and why. Produce the three types of visuals, and say a little bit about what each might mean (or whether there is any significance to these). Note: you can take a screenshot of these and include the image in your markdown with the following code:

```
knitr::include_graphics("image.png")
```

But if you are unable to visually display these in your output, it is ok to just describe them in words!

3. Now, try creating a dataset in the workbench. See section 3.3 for instructions on saving the dataset as a .csv file and beginning to work with it in ProQuest's Jupyter Notebooks. Create some simple frequencies of dates or news sources, and try to export these as a csv file.

(Remember that you will not be able to export the full text). Then load the csv into your local R and create a graph using `ggplot2` . BONUS: If you are feeling ambitious, you can try installing the `tidytext` package in ProQuest TDM Studio, and exporting some word frequencies or other summary statistics. Just FYI, installing `tidytext` in ProQuest requires some additional effort. For instructions, see Getting Started -> 2022.05.25 -> ProQuest TDM Studio Manuals -> TDM_Studio_Manual.ipynb.

4. Download at least two of the files from the “Katrina Hearings” folder in Canvas (within the Data folder), or choose your own congressional hearings here and download PDFs. Convert the documents to `tidytext` style using the `unnest_tokens()` function, and remove stop words. Show the first few rows of each dataset using the `head()` function.

5. Compare the frequencies of words between the two hearings. First, use the `count()` function to generate counts of words in each document, and compare the two using tables or a plot. Then, read portions of each document in more detail to explain some of the trends that you see. Are you able to say something meaningful about the congressional discourse? Why or why not?

3.8 Final Project Proposal

Final projects can be completed in groups of up to four students. The proposals can also be collaborative, but each student should turn in one.

Proposals should answer the following questions:

1. What is your research question? Why does it matter? How does it relate to climate change and society?

2. What data will you use to answer the question? Do you have access to these data?

- Ideally, you are able to show access to at least a portion of the data that you will need.
- Beyond the APIs and other data sources we have used in class, there are some great options on Kaggle. For example, this dataset on reddit comments, this dataset of tweets, and these data on urban climate equity would all be potentially good sources for a final project.

3. What methods will you use to answer your research question?

- Most projects will use some of the methods that we haven’t yet covered. This is great, just show that you have an idea of where you are heading!

4. What kind of product will you create? How do you want others to read or engage with your project?

- In most cases, projects will be R Markdown documents or Jupyter Notebooks, both of which can be published as websites.