Text Analysis with R

Claudia Engel, Scott Bailey

Last updated: July 05, 2019

Contents

Prerequisites

- You should have some **basic knowledge** of R, and be familiar with the topics covered in the Introduction to R.
- Have a **recent** version of R and RStudio installed.
- Packages needed:
  - `tidyverse`
  - `tidytext`
  - `readtext`
  - `sotu`
  - `SnowballC`
  - `widyr`
  - `igraph`
  - `ggraph`
  - `tm`

It is recommended that you not only intall, but also load the packages, to make sure the respective versions get along with your R version.

References

Feinerer, I., Hornik, K., and Meyer, D. (2008). Text Mining Infrastructure in R. Journal of Statistical Software, 25(5), 1 - 54. doi:http://dx.doi.org/10.18637/jss.v025.i05

Gries, Stefan Thomas, 2009: Quantitative Corpus Linguistics with R: A Practical Introduction. Routledge.

Silge, J and D. Robinson, 2017: Text Mining with R: A Tidy Approach

Kasper Welbers, Wouter Van Atteveldt & Kenneth Benoit (2017) Text Analysis in R, Communication Methods and Measures, 11:4, 245-265, DOI: 10.1080/19312458.2017.1387238

CRAN Task View: Natural Language Processing

Chapter 1

Preparing Textual Data

Learning Objectives

- read textual data into R using `readtext`
- use `stringr` package to manipulate strings
- use `tidytext` functions to tokenize texts and remove stopwords
- use `SnowballC` to stem words

We'll use several R packages in this section:

- `sotu` will provide the metadata and text of State of the Union speeches ranging from George Washington to Barack Obama.
- `tidyverse` is a collection of R packages designed for data science, including `dplyr` with a set of verbs for common data manipulations and `ggplot2` for visualization.
- `tidytext` provides specific functions for a "tidy" approach to working with textual data, where one row represents one "token" or meaningful unit of text, for example a word.
- `readtext` provides a function well suited to reading textual data from a large number of formats into R, including metadata.

```
library(sotu)
library(tidyverse)
library(tidytext)
library(readtext)
```

1.1 Reading text into R

First, let's look at the data in the `sotu` package. The metadata and texts come separately. Below is what the metadata look like. Can you tell how many speeches we have?

```
# Let's take a quick look at the state of the union metadata
str(sotu_meta)
```

```
#> Classes 'tbl_df', 'tbl' and 'data.frame':    236 obs. of  5 variables:
#>  $ president   : chr  "George Washington" "George Washington" "George Washington" "George Washingt
#>  $ year        : int  1790 1790 1791 1792 1793 1794 1795 1796 1797 1798 ...
#>  $ years_active: chr  "1789-1793" "1789-1793" "1789-1793" "1789-1793" ...
#>  $ party       : chr  "Nonpartisan" "Nonpartisan" "Nonpartisan" "Nonpartisan" ...
#>  $ sotu_type   : chr  "speech" "speech" "speech" "speech" ...
```

In order to work with the speech texts and to later practice reading text files from disk we're going to use a function `sotu_dir` to write the texts out. This function by default writes to a temporary directory with one speech in each file. It returns a character vector where each element is the name of the path to the individual speech file. We save this vector into the `file_paths` variable.

```
# sotu_dir writes the text files to disk in a temporary dir,
# but you could specific where you want them.
file_paths <- sotu_dir()
head(file_paths)
```

```
#> [1] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp2sxDMZ/file3c061f5b124b/george-washington-1790a.tx
#> [2] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp2sxDMZ/file3c061f5b124b/george-washington-1790b.tx
#> [3] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp2sxDMZ/file3c061f5b124b/george-washington-1791.txt
#> [4] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp2sxDMZ/file3c061f5b124b/george-washington-1792.txt
#> [5] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp2sxDMZ/file3c061f5b124b/george-washington-1793.txt
#> [6] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp2sxDMZ/file3c061f5b124b/george-washington-1794.txt
```

Now that we have the files on disk and a vector of filepaths, we can pass this vector directly into `readtext` to read the texts into a new variable.

```
# let's read in the files with readtext
sotu_texts <- readtext(file_paths)
head(sotu_texts)
```

```
#> readtext object consisting of 6 documents and 0 docvars.
#> # Description: df[,2] [6 x 2]
#>   doc_id                   text
#> * <chr>                    <chr>
#> 1 abraham-lincoln-1861.txt "\"\n\n Fellow-\"..."
#> 2 abraham-lincoln-1862.txt "\"\n\n Fellow-\"..."
#> 3 abraham-lincoln-1863.txt "\"\n\n Fellow-\"..."
#> 4 abraham-lincoln-1864.txt "\"\n\n Fellow-\"..."
#> 5 andrew-jackson-1829.txt  "\"\n\n Fellow \"..."
#> 6 andrew-jackson-1830.txt  "\"\n\n Fellow \"..."
```

To work with a single tabular dataset, we combine the text and metadata into a single tibble. You can see that our `sotu_texts` are organized by alphabetical order, so first we'll need to sort our metadata to match.

```
sotu_whole <-
  sotu_meta %>%
  arrange(president) %>% # sort metadata
  bind_cols(sotu_texts) # combine with texts

glimpse(sotu_whole)
```

```
#> Observations: 236
#> Variables: 7
#> $ president    <chr> "Abraham Lincoln", "Abraham Lincoln", "Abraham Li...
#> $ year         <int> 1861, 1862, 1863, 1864, 1829, 1830, 1831, 1832, 1...
#> $ years_active <chr> "1861-1865", "1861-1865", "1861-1865", "1861-1865...
#> $ party        <chr> "Republican", "Republican", "Republican", "Republ...
#> $ sotu_type    <chr> "written", "written", "written", "written", "writ...
#> $ doc_id       <chr> "abraham-lincoln-1861.txt", "abraham-lincoln-1862...
#> $ text         <chr> "\n\n Fellow-Citizens of the Senate and House of ...
```

Now that we have our data, we need to think about cleaning it. Depending on the quality of your data, you might need to explicitly replace certain characters or words, remove urls or types of numbers, such as phone numbers, or otherwise clean up misspellings or errors. There are several ways to handle this sort of cleaning, we'll show a few examples for string manipulation and replacement.

1.2 String operations

R has many functions available to manipulate strings including functions like `grep` and `paste`, which come with the R base install.

Here we will here take a look at the `stringr` package, which is part of the `tidyverse`. Under the hood it wraps a lot of the functions from the `stringi` package which is perhaps one of the most comprehensive string manipulation packages.

Below are examples for a few functions that might be useful.

`str_count` takes a characer vector as input and by default counts the number of pattern matches in a string.

How man times does the word "citizen" appear in each of the speeches?

```
sotu_whole %>%
  pull(text) %>% # extract texts vector
  str_count("citizen")
```

```
#>   [1]  9  7 15  3 19 14 23 19 14 25 10  9 11 10 11 12  3  6  3  6  7  3  2
#>  [24]  8 14 13 17 15 13  3  5  6  9  7 14  9 20 17 14 17 23  1  8  6  0  6
#>  [47]  4  3  3  1  2  2  6  1  3  2  1  1  6  2  3 12 17 17 29  2  3  4  1
#>  [70]  5  9  9  6  7  9 11 10  2  4  2  6  4 10  3  5  0  8  6 43 42  5 37
#>  [93] 19 16 21 16  7  5 10  6  8  4  2 11  9  3  4  1 13 41 30 35 29 42 34
#> [116] 15  3  3  4  4  4  2  3  5  7  8  6  3  6  1  7  9  4  9  3 15  4 24
#> [139] 25  8  2  3  1  2  7  6 10  6 11  8 13 13 11  9  5  3  2  6  2  2 14
#> [162] 27 17 13 13 16 14  0  0  0  8  2 10  2  4  3  4  5  2  3  0 15 17 27
#> [185] 20 13  1 19 27 31 28 18 10 10  6  7  3  9  6  5  8 15 16 17 22 20 28
#> [208] 29 22  4  5  9 10 10 27  1  2 21 12 10  9  3  8 20 12 26 13  4  2  8
#> [231]  0  0  0  0  0 11
```

It is possible to use regular expressions, for example, this is how we would check how many times either "citizen" or "Citizen" appear in each of the speeches:

```
sotu_whole %>%
  pull(text) %>% # extract texts vector
  str_count("[C|c]itizen")
```

```
#>   [1] 10  8 16  4 20 15 24 20 15 26 11 10 12 11 12 13  3  6  3  6  7  6  2
#>  [24]  8 14 13 17 15 13  3  5  6  9  7 14  9 20 17 14 17 23  2  8  6  0  6
#>  [47]  4  3  3  1  2  2  6  1  3  2  1  1  6  2  3 13 18 18 30  2  3  4  1
#>  [70]  5  9 10  6  7  9 11 10  3  5  3  7  5 11  4  6  0  8  6 43 42  5 37
#>  [93] 19 16 21 16  7  5 10  6  8  4  2 11  9  3  4  1 15 42 31 36 30 43 35
#> [116] 16  4  4  5  5  5  3  4  6  8  9  7  4  7  2  8 10  4  9  3 15  4 24
#> [139] 25  8  2  3  1  2  7  6 11  7 12  9 13 14 11  9  5  3  2  6  2  2 15
#> [162] 28 18 14 15 17 15  0  0  0  8  2 10  2  4  3  4  5  2  3  0 16 18 28
#> [185] 21 13  1 19 27 31 28 18 10 11  6  7  3  9  6  5  8 15 16 17 22 20 28
#> [208] 29 22  4  5  9 10 10 27  1  2 22 12 11  9  3  8 20 12 26 13  4  2  8
#> [231]  0  0  0  0  0 12
```

When used with the `boundary` argument `str_count` can count different entities like "character", "line_break", "sentence", or "word". Here we add a new column to the dataframe indicating how many words are there in each speech:

```
sotu_whole %>%
  mutate(n_words = str_count(text, boundary("word")))
```

```
#> # A tibble: 236 x 8
#>   president   year years_active party  sotu_type doc_id  text       n_words
#>   <chr>      <int> <chr>        <chr>  <chr>     <chr>   <chr>        <int>
#> 1 Abraham L~  1861 1861-1865    Repub~ written   abraha~ "\n\n Fe~     6998
```

```
#>  2 Abraham L~  1862 1861-1865    Repub~ written   abraha~ "\n\n Fe~    8410
#>  3 Abraham L~  1863 1861-1865    Repub~ written   abraha~ "\n\n Fe~    6132
#>  4 Abraham L~  1864 1861-1865    Repub~ written   abraha~ "\n\n Fe~    5975
#>  5 Andrew Ja~  1829 1829-1833    Democ~ written   andrew~ "\n\n Fe~   10547
#>  6 Andrew Ja~  1830 1829-1833    Democ~ written   andrew~ "\n\n Fe~   15109
#>  7 Andrew Ja~  1831 1829-1833    Democ~ written   andrew~ "\n\n Fe~    7198
#>  8 Andrew Ja~  1832 1829-1833    Democ~ written   andrew~ "\n\n Fe~    7887
#>  9 Andrew Ja~  1833 1833-1837    Democ~ written   andrew~ "\n\n Fe~    7912
#> 10 Andrew Ja~  1834 1833-1837    Democ~ written   andrew~ "\n\n Fe~   13472
#> # ... with 226 more rows
```

> CHALLENGE: Use the code above and add another column `n_sentences` where you calculate the number of sentences per speech. Then create a third column `avg_word_per_sentence`, where you calculate the number of words per sentence for each speech. Finally use `filter` to find which speech has shortest/longest average sentences length and what is the avderage length.

`str_detect` also looks for patterns, but instead of counts it returns a logical vector (TRUE/FALSE) indiciating if the pattern is or is not found. So we typically want to use it with the `filter` "verb" from `dplyr`.

What are the names of the documents where the words "citizen" and "Citizen" do **not** occur?

```
sotu_whole %>%
  filter(!str_detect(text, "[C|c]itizen")) %>%
  select(doc_id)
```

```
#> # A tibble: 11 x 1
#>    doc_id
#>    <chr>
#>  1 dwight-d-eisenhower-1958.txt
#>  2 gerald-r-ford-1975.txt
#>  3 richard-m-nixon-1970.txt
#>  4 richard-m-nixon-1971.txt
#>  5 richard-m-nixon-1972a.txt
#>  6 ronald-reagan-1988.txt
#>  7 woodrow-wilson-1916.txt
#>  8 woodrow-wilson-1917.txt
#>  9 woodrow-wilson-1918.txt
#> 10 woodrow-wilson-1919.txt
#> 11 woodrow-wilson-1920.txt
```

The `word` function extracts specific words from a character vector of words. By default it returns the first word. If for example we wanted to extract the first 5 words of each speech by Woodrow Wilson we provide the `end` argument like this:

```
sotu_whole %>%
  filter(president == "Woodrow Wilson") %>%  # sample a few speeches as demo
  pull(text) %>% # extract character vector
  word(end = 5) # end = 5 to extract words 1 - 5.
```

```
#> [1] "\n\nGentlemen of the Congress:\n\nIn pursuance"
#> [2] "\n\nGENTLEMEN OF THE CONGRESS: \n\nThe"
#> [3] "GENTLEMEN OF THE CONGRESS: \n\nSince"
#> [4] "\n\nGENTLEMEN OF THE CONGRESS: \n\nIn"
#> [5] "Gentlemen of the Congress:\n\nEight months"
#> [6] "\n\nGENTLEMEN OF THE CONGRESS: \n\nThe"
#> [7] "\n\nTO THE SENATE AND HOUSE"
#> [8] "\n\nGENTLEMEN OF THE CONGRESS:\n\nWhen I"
```

To clean this up a little we will first remove the newline characters (`\n`). We use the `str_replace_all` function to replace all the ocurrences of the `\n` pattern with a white space `" "`. We need to add the escape character `\` in front of our pattern to be replaced so the backslash before the **n** is interpreted correctly.

```
sotu_whole %>%
  filter(president == "Woodrow Wilson") %>%
  pull(text) %>%
  str_replace_all("\\n", " ") %>% # replace newline
  word(end = 5)
```

```
#> [1] "  Gentlemen of the"           "  GENTLEMEN OF THE"
#> [3] "GENTLEMEN OF THE CONGRESS: " "  GENTLEMEN OF THE"
#> [5] "Gentlemen of the Congress: " "  GENTLEMEN OF THE"
#> [7] "  TO THE SENATE"              "  GENTLEMEN OF THE"
```

This looks better, but we still have a problem to extract exactly 5 words because of the whitespaces. So let's get rid of any whitespaces before and also of repeated whitespaces within the string with the convenient `str_squish` function.

```
sotu_whole %>%
  filter(president == "Woodrow Wilson") %>%
  pull(text) %>%
  str_replace_all("\\n", " ") %>%
  str_squish() %>%  # remove whitespaces
  word(end = 5)
```

```
#> [1] "Gentlemen of the Congress: In"    "GENTLEMEN OF THE CONGRESS: The"
#> [3] "GENTLEMEN OF THE CONGRESS: Since" "GENTLEMEN OF THE CONGRESS: In"
#> [5] "Gentlemen of the Congress: Eight" "GENTLEMEN OF THE CONGRESS: The"
#> [7] "TO THE SENATE AND HOUSE"          "GENTLEMEN OF THE CONGRESS: When"
```

(For spell checks take a look at https://CRAN.R-project.org/package=spelling or https://CRAN.R-project.org/package=hunspell)

1.3 Tokenize, lowercase

A very common part of preparing your text for analysis involves tokenization. Currently our data contains in each each row a single text with metdata, so the entire speech text is the unit of observation. When we tokenize we break down the text into "tokens" (most commonly single words), so each row contains a single word with its metadata as unit of observation.

`tidytext` provides a function `unnest_tokens` to convert our speech table into one that is tokenized. It takes three arguments:

- a tibble or data frame which contains the text;
- the name of the newly created column that will contain the tokens;
- the name of the column within the data frame which contains the text to be tokenized.

In the example below we name the new column to hold the tokens `word`. Remember that the column that holds the speech text is called `text`.

```
tidy_sotu <- sotu_whole %>%
  unnest_tokens(word, text)

tidy_sotu
```

```
#> # A tibble: 1,965,212 x 7
#>    president     year years_active party   sotu_type doc_id        word
#>    <chr>        <int> <chr>        <chr>   <chr>     <chr>         <chr>
#>  1 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ fellow
#>  2 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ citizens
#>  3 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ of
```

```
#>  4 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ the
#>  5 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ senate
#>  6 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ and
#>  7 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ house
#>  8 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ of
#>  9 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ represe~
#> 10 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ in
#> # ... with 1,965,202 more rows
```

Note that the `unnest_tokens` function didn't just tokenize our texts at the word level. It also lowercased each word and stripped off the punctuation. We can tell it not to do this, by adding the following parameters:

```
# Word tokenization with punctuation and no lowercasing
sotu_whole %>%
  unnest_tokens(word, text, to_lower = FALSE, strip_punct = FALSE)
```

```
#> # A tibble: 2,157,777 x 7
#>    president     year years_active party   sotu_type doc_id        word
#>    <chr>        <int> <chr>        <chr>   <chr>     <chr>         <chr>
#>  1 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ Fellow
#>  2 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ -
#>  3 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ Citizens
#>  4 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ of
#>  5 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ the
#>  6 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ Senate
#>  7 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ and
#>  8 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ House
#>  9 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ of
#> 10 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ Represe~
#> # ... with 2,157,767 more rows
```

We can also tokenize the text at the level of ngrams or sentences, if those are the best units of analysis for our work.

```
# Sentence tokenization
sotu_whole %>%
  unnest_tokens(sentence, text, token = "sentences", to_lower = FALSE) %>%
  select(sentence)
```

```
#> # A tibble: 69,158 x 1
#>    sentence
#>    <chr>
#>  1 Fellow-Citizens of the Senate and House of Representatives:   In the mi~
#>  2 You will not be surprised to learn that in the peculiar exigencies of t~
#>  3 A disloyal portion of the American people have during the whole year be~
#>  4 A nation which endures factious domestic division is exposed to disresp~
#>  5 Nations thus tempted to interfere are not always able to resist the cou~
#>  6 The disloyal citizens of the United States who have offered the ruin of~
#>  7 If it were just to suppose, as the insurgents have seemed to assume, th~
#>  8 If we could dare to believe that foreign nations are actuated by no hig~
#>  9 The principal lever relied on by the insurgents for exciting foreign na~
#> 10 Those nations, however, not improbably saw from the first that it was t~
#> # ... with 69,148 more rows
```

```
# N-gram tokenization
sotu_whole %>%
  unnest_tokens(trigram, text, token = "ngrams", n = 3) %>%
  select(trigram)
```

```
#> # A tibble: 1,964,740 x 1
#>    trigram
```

```
#>   <chr>
#>  1 fellow citizens of
#>  2 citizens of the
#>  3 of the senate
#>  4 the senate and
#>  5 senate and house
#>  6 and house of
#>  7 house of representatives
#>  8 of representatives in
#>  9 representatives in the
#> 10 in the midst
#> # ... with 1,964,730 more rows
```

1.4 Stopwords

Another common task of preparing text for analysis is to remove stopwords. Stopwords are common words that are considered to provide non-relevant information about the content of a text.

Let's look at the stopwords that come with the `tidytext` package to get a sense of what they are.

```
stop_words
```

```
#> # A tibble: 1,149 x 2
#>    word        lexicon
#>    <chr>       <chr>
#>  1 a           SMART
#>  2 a's         SMART
#>  3 able        SMART
#>  4 about       SMART
#>  5 above       SMART
#>  6 according   SMART
#>  7 accordingly SMART
#>  8 across      SMART
#>  9 actually    SMART
#> 10 after       SMART
#> # ... with 1,139 more rows
```

Depending on the type of analysis you're doing, you might leave these words in or alternatively use your own curated list of stopwords. Stopword lists exist for many languages. For now we will remove the English stopwords as suggested here.

There are a number of ways how to do this, here we use `anti_join` from `dplyr`. We can use it to return all rows from our table of tokens `tidy_sotu` where there are not matching values in our list of stopwords. Both of these tables have one column name in common `word` so by default the join will be on that colunmn, and dplyr will tell us so.

```
tidy_sotu_words <- tidy_sotu %>%
  anti_join(stop_words)

tidy_sotu_words
```

```
#> # A tibble: 778,161 x 7
#>    president     year years_active party   sotu_type doc_id         word
#>    <chr>        <int> <chr>        <chr>   <chr>     <chr>          <chr>
#>  1 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ fellow
#>  2 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ citizens
#>  3 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ senate
#>  4 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ house
```

```
#>  5 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ represe~
#>  6 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ midst
#>  7 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ unprece~
#>  8 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ politic~
#>  9 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ troubles
#> 10 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ gratitu~
#> # ... with 778,151 more rows
```

If we compare this with `tidy_sotu` we see that the records with words like "of", "the", "and", "in" are now removed.

So we went from 1965212 to 778161 rows, which means we had a lot of stopwords in our corpus. This is a huge removal, so for serious analysis, we might want to scrutinize the stopword list carefully and determine if this is feasible.

1.5 Word Stemming

Another way you may want to clean your data is to stem your words, that is, to reduce them to their word stem or root form, for example reducing *fishing*, *fished*, and *fisher* to the stem *fish*.

`tidytext` does not implement its own word stemmer. Instead it relies on separate packages like `hunspell` or `SnowballC`.

We will give an example here for the `SnowballC` package which comes with a function `wordStem`. (`hunspell` appears to run much slower, and it also returns a list instead of a vector, so in this context `SnowballC` seems to be more convenient.)

```
library(SnowballC)
tidy_sotu_words %>%
        mutate(word_stem = wordStem(word))
```

```
#> # A tibble: 778,161 x 8
#>    president   year years_active party  sotu_type doc_id   word   word_stem
#>    <chr>      <int> <chr>        <chr>  <chr>     <chr>    <chr>  <chr>
#>  1 Abraham L~  1861 1861-1865    Repub~ written   abraham~ fellow fellow
#>  2 Abraham L~  1861 1861-1865    Repub~ written   abraham~ citiz~ citizen
#>  3 Abraham L~  1861 1861-1865    Repub~ written   abraham~ senate senat
#>  4 Abraham L~  1861 1861-1865    Repub~ written   abraham~ house  hous
#>  5 Abraham L~  1861 1861-1865    Repub~ written   abraham~ repre~ repres
#>  6 Abraham L~  1861 1861-1865    Repub~ written   abraham~ midst  midst
#>  7 Abraham L~  1861 1861-1865    Repub~ written   abraham~ unpre~ unpreced
#>  8 Abraham L~  1861 1861-1865    Repub~ written   abraham~ polit~ polit
#>  9 Abraham L~  1861 1861-1865    Repub~ written   abraham~ troub~ troubl
#> 10 Abraham L~  1861 1861-1865    Repub~ written   abraham~ grati~ gratitud
#> # ... with 778,151 more rows
```

Lemmatization takes this another step further. While a stemmer operates on a single word without knowledge of the context, lemmatization attempts to discriminate between words which have different meanings depending on part of speech. For example, the word "better" has "good" as its lemma, something a stemmer would not detect.

For lemmatization in R, you may want to take a look a the `koRpus` package, another comprehensive R package for text analysis. It allows to use TreeTagger, a widely used part-of-speech tagger. For full functionality of the R package a local installation of TreeTagger is recommended.

Chapter 2

Analyzing Texts

Learning Objectives

- perform different frequency counts and generate plots
- use the `widyr` package to calculate co-ocurrance
- use `igraph` and `ggraph` to plot a co-ocurrance graph
- import and export a Document-Term Matrix into `tidytext`
- use the `sentiments` dataset from `tidytext` to perform a sentiment analysis

---

Now that we've read in our text and metadata, tokenized and cleaned it a little, let's move on to some analysis.

First, we'll make sure we have loaded the libraries we'll need.

```
library(tidyverse)
library(tidytext)
```

Let's remind ourselves of what our data looks like.

```
tidy_sotu_words
```

```
#> # A tibble: 778,161 x 7
#>    president     year years_active party   sotu_type doc_id        word
#>    <chr>        <int> <chr>        <chr>   <chr>     <chr>         <chr>
#>  1 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ fellow
#>  2 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ citizens
#>  3 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ senate
#>  4 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ house
#>  5 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ represe~
#>  6 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ midst
#>  7 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ unprece~
#>  8 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ politic~
#>  9 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ troubles
#> 10 Abraham Lin~  1861 1861-1865    Republ~ written   abraham-linc~ gratitu~
#> # ... with 778,151 more rows
```

2.1 Frequencies

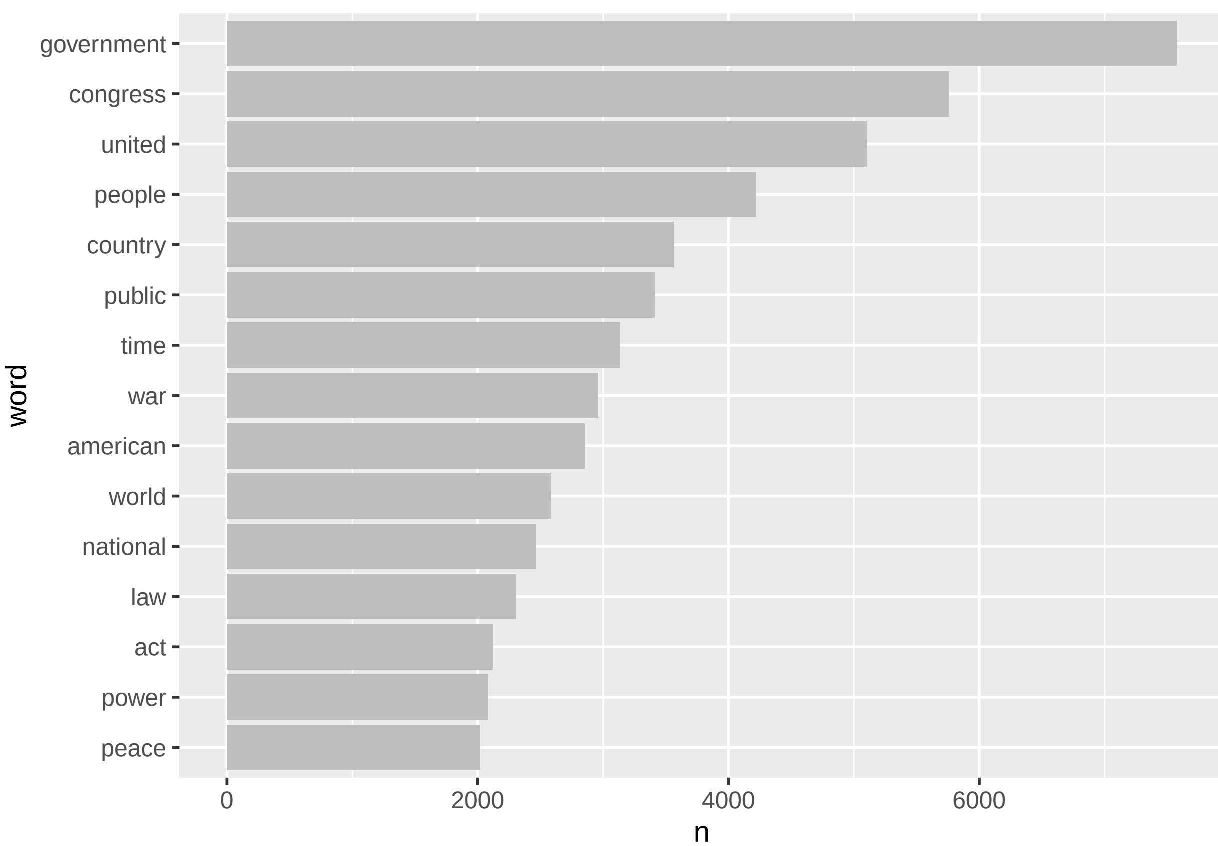
Since our unit of analysis at this point is a word, let's do some straightforward counting to figure out which words occur most frequently in the corpus as a whole.

```
tidy_sotu_words %>%
  count(word, sort = TRUE)
```

```
#> # A tibble: 29,558 x 2
#>    word            n
#>    <chr>       <int>
#>  1 government  7573
#>  2 congress    5759
#>  3 united      5102
#>  4 people      4219
#>  5 country     3564
#>  6 public      3413
#>  7 time        3138
#>  8 war         2961
#>  9 american    2853
#> 10 world       2581
#> # ... with 29,548 more rows
```

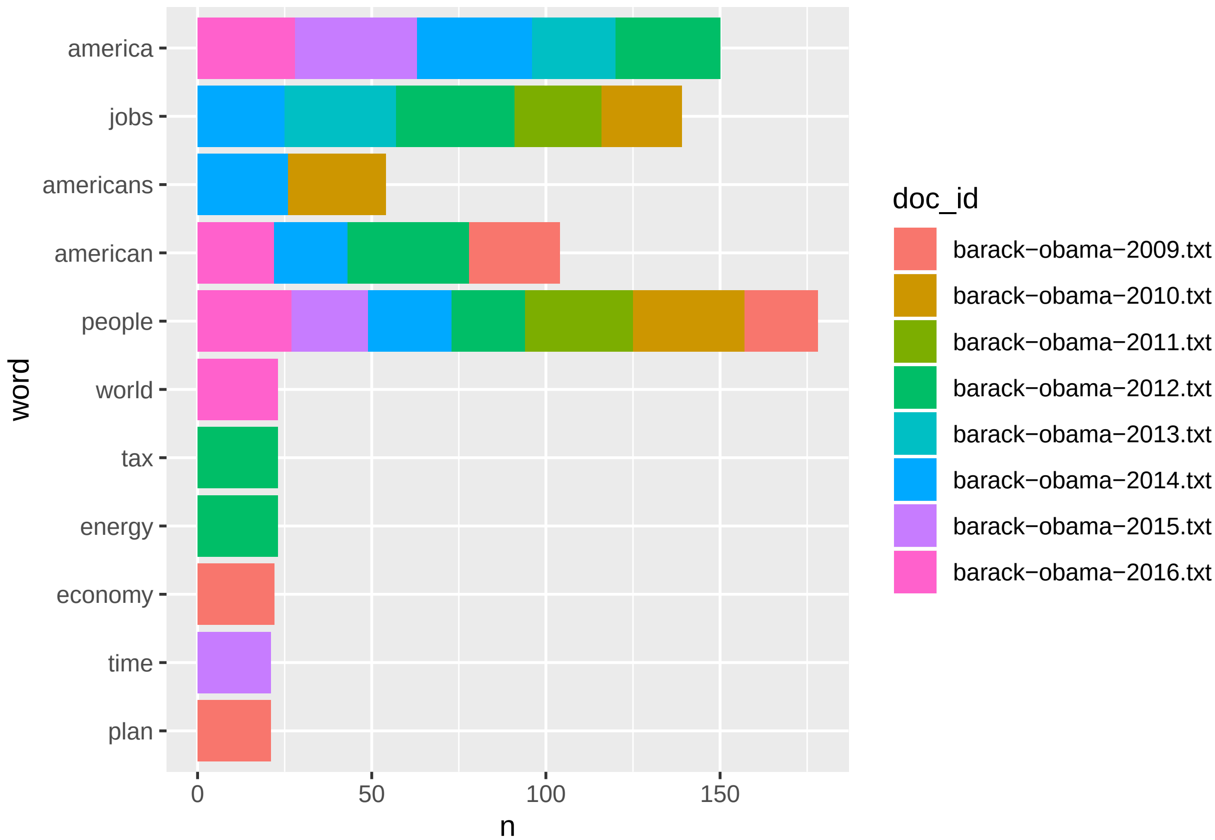
We can pipe this into ggplot to make a graph of the words that occur more that 2000 times. We count the words and use `geom_col` to represent the n values.

```
tidy_sotu_words %>%
  count(word) %>%
  filter(n > 2000) %>%
  mutate(word = reorder(word, n)) %>%  # reorder values by frequency
  ggplot(aes(word, n)) +
     geom_col(fill = "gray") +
     coord_flip()  # flip x and y coordinates so we can read the words better
```



What if we want to check the most common words per speech for a single president and see which of the top words apppear in which speech?

```
tidy_sotu_words %>%
  filter(president == "Barack Obama") %>%
  count(doc_id, word) %>%
  filter(n > 20) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill=doc_id)) +
    geom_col() +
    coord_flip()
```



CHALLENGE: In any given year, how often is the word ‘peace’ used and how often is the word ‘war’ used? Make a bar chart that shows for each year the proportion of each of these two words out of the total of how often both words are used. Bonus: use decades instead of years.

As another example let us calculate the average number of words per speech for each president. How long was the average speech of each president and who are the top ‘wordiest’ presidents?

First we summarize the words per president per speech

```
tidy_sotu_words %>%
  count(president, doc_id)
```

```
#> # A tibble: 236 x 3
#>    president       doc_id                       n
#>    <chr>           <chr>                    <int>
#>  1 Abraham Lincoln abraham-lincoln-1861.txt  2578
#>  2 Abraham Lincoln abraham-lincoln-1862.txt  3088
#>  3 Abraham Lincoln abraham-lincoln-1863.txt  2398
#>  4 Abraham Lincoln abraham-lincoln-1864.txt  2398
#>  5 Andrew Jackson  andrew-jackson-1829.txt   3849
#>  6 Andrew Jackson  andrew-jackson-1830.txt   5428
```

```
#>  7 Andrew Jackson  andrew-jackson-1831.txt   2612
#>  8 Andrew Jackson  andrew-jackson-1832.txt   2881
#>  9 Andrew Jackson  andrew-jackson-1833.txt   2869
#> 10 Andrew Jackson  andrew-jackson-1834.txt   4952
#> # ... with 226 more rows
```

Then we use the output table and group it by president. That allows us to calculate the average number of words per speech.

```
tidy_sotu_words %>%
  count(president, doc_id)  %>%
  group_by(president) %>%
  summarize(avg_words = mean(n)) %>%
  arrange(desc(avg_words))
```

```
#> # A tibble: 41 x 2
#>    president           avg_words
#>    <chr>                   <dbl>
#>  1 William Howard Taft     9126.
#>  2 William McKinley        7797
#>  3 Jimmy Carter            7673.
#>  4 Theodore Roosevelt      7356
#>  5 James K. Polk           6920.
#>  6 Grover Cleveland        5736.
#>  7 James Buchanan          5409
#>  8 Benjamin Harrison       5308.
#>  9 Rutherford B. Hayes     4411
#> 10 Martin Van Buren        4286.
#> # ... with 31 more rows
```

2.2 Term frequency

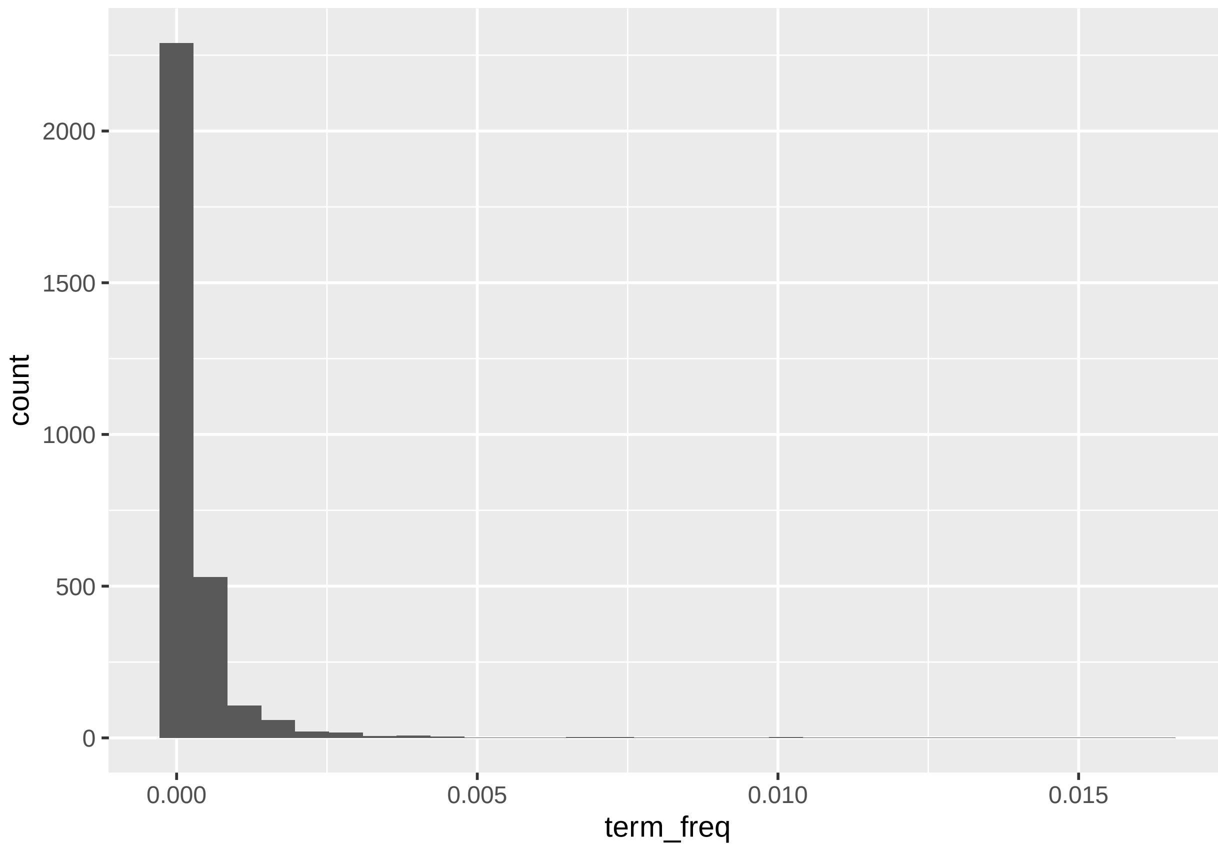
Often a raw count of a word is less important than understanding how often that word appears relative to the total number of words in a text. This ratio would be the **term frequency**. We can use `dplyr` to calculate it like this:

```
tidy_sotu_words %>%
  count(doc_id, word, sort = T)  %>%  # count n for each word
  group_by(doc_id) %>%
  mutate(n_tot = sum(n), # count total number of words per doc
         term_freq = n/n_tot)
```

```
#> # A tibble: 352,846 x 5
#> # Groups:   doc_id [236]
#>    doc_id                       word               n n_tot term_freq
#>    <chr>                        <chr>          <int> <int>     <dbl>
#>  1 harry-s-truman-1946.txt      dollars          207 12614   0.0164
#>  2 jimmy-carter-1980b.txt       congress         204 16128   0.0126
#>  3 harry-s-truman-1946.txt      war              201 12614   0.0159
#>  4 william-howard-taft-1910.txt government       164 11178   0.0147
#>  5 james-k-polk-1846.txt        mexico           158  7023   0.0225
#>  6 richard-m-nixon-1974b.txt    federal          141  9996   0.0141
#>  7 harry-s-truman-1946.txt      million          138 12614   0.0109
#>  8 harry-s-truman-1946.txt      fiscal           129 12614   0.0102
#>  9 jimmy-carter-1981.txt        administration   129 16595   0.00777
#> 10 william-howard-taft-1912.txt government       129 10215   0.0126
#> # ... with 352,836 more rows
```

Let's graph the term frequency for one of these speeches so we can understand the frequency distribution of words over a text.

```
tidy_sotu_words %>%
  filter(doc_id == "harry-s-truman-1946.txt") %>%
  count(doc_id, word)  %>%  # count n for each word
  group_by(doc_id) %>%
  mutate(n_tot = sum(n), # count total number of words per doc
         term_freq = n/n_tot) %>%
  ggplot(aes(term_freq)) +
    geom_histogram()
```



This distribution makes sense. Most words are used relatively rarely in a text. Only a few have a high term frequency.

We could keep filtering this data to see which terms have high frequency, thus maybe increased significance, for different presidents and different particular speeches.

> CHALLENGE: Pick one president. For each of his speeches, which is the term with highest term frequency? Create a table as output. (Hint: `top_n`might be useful)

2.3 Tf-idf

So far we've been looking at term frequency per document. What if we want to know about words that seem more important based on the contents of the entire corpus?

For this, we can use **term-frequency according to inverse document frequency**, also callled **tf-idf**. Tf-idf measures how important a word is within a corpus by scaling term frequency per document according to the inverse of the term's document frequency (number of documents within the corpus in which the term appears divided by the number of documents).

The tf-idf value will be:

- lower for words that appear in many documents in the corpus, and lowest when the word occurs in virtually all documents.
- high for words that appear many times in few documents in the corpus, this lending high discriminatory power to those documents.

The `tidytext` package includes a function `bind_tf_idf`. It takes a table that contains one-row-per-term-per-document, the name of the column that contains the words (terms), the name of the column which contains the doc-id, and the name of the column that contains the document-term counts.

So below we aggregate our tibble with the word tokens to create the one-row-per-term-per-document table and then pipe it into the `bind_tf_idf` function.

```
tidy_sotu_words %>%
  count(doc_id, word, sort = TRUE)  %>%  # aggregate to count n for each word
  bind_tf_idf(word, doc_id, n)
```

```
#> # A tibble: 352,846 x 6
#>    doc_id                       word                n      tf     idf   tf_idf
#>    <chr>                        <chr>           <int>   <dbl>   <dbl>    <dbl>
#>  1 harry-s-truman-1946.txt      dollars           207 0.0164  0.612    1.00e-2
#>  2 jimmy-carter-1980b.txt       congress          204 0.0126  0.00425  5.37e-5
#>  3 harry-s-truman-1946.txt      war               201 0.0159  0.0345   5.50e-4
#>  4 william-howard-taft-1910.~ government        164 0.0147  0.00425  6.23e-5
#>  5 james-k-polk-1846.txt        mexico            158 0.0225  0.810    1.82e-2
#>  6 richard-m-nixon-1974b.txt  federal           141 0.0141  0.293    4.14e-3
#>  7 harry-s-truman-1946.txt      million           138 0.0109  0.728    7.96e-3
#>  8 harry-s-truman-1946.txt      fiscal            129 0.0102  0.494    5.05e-3
#>  9 jimmy-carter-1981.txt        administration    129 0.00777 0.282    2.19e-3
#> 10 william-howard-taft-1912.~ government        129 0.0126  0.00425  5.36e-5
#> # ... with 352,836 more rows
```

We can see in the output that our function added three columns to our aggregated table: which contain term frequency, inverse document frequency and Tf-idf.

Let's look at some of the words in the corpus that have the highest tf-idf scores, which means words that are particularly distinctive for their documents.

```
tidy_sotu_words %>%
  count(doc_id, word, sort = TRUE)  %>%
  bind_tf_idf(word, doc_id, n) %>%
  arrange(desc(tf_idf))
```

```
#> # A tibble: 352,846 x 6
#>    doc_id                          word          n      tf   idf tf_idf
#>    <chr>                           <chr>     <int>   <dbl> <dbl>  <dbl>
#>  1 lyndon-b-johnson-1966.txt       vietnam      32 0.0152   2.42 0.0367
#>  2 jimmy-carter-1980a.txt          soviet       31 0.0218   1.47 0.0321
#>  3 george-w-bush-2003.txt          hussein      19 0.00811  3.85 0.0313
#>  4 george-w-bush-2003.txt          saddam       19 0.00811  3.67 0.0298
#>  5 franklin-d-roosevelt-1943.txt 1942         13 0.00758  3.85 0.0292
#>  6 dwight-d-eisenhower-1961.txt  1953         23 0.00747  3.85 0.0288
#>  7 john-adams-1800.txt             gentlemen     8 0.0153   1.80 0.0275
#>  8 benjamin-harrison-1892.txt      1892         40 0.00741  3.52 0.0261
#>  9 franklin-d-roosevelt-1942.txt hitler        7 0.00527  4.77 0.0251
#> 10 herbert-hoover-1930.txt         1928         14 0.00711  3.52 0.0250
#> # ... with 352,836 more rows
```

To understand the occurrence of the years as being particularly distinctive we might need to look more closely at the speeches themselves, and determine whether the years are significant or whether they need to be removed from the text either permanently in the clean up or temporarily using `filter`.

> CHALLENGE: Pick the same president you chose above. For each of his speeches, which is the term with highest tf-idf? Create a table as output. (Hint: Remember to group by doc_id before you use top_n)

2.4 N-Grams

We mentioned n-grams in the intro, but let's revisit them here and take a look at the most common bigrams in the speeches. Remember we use the `unnest_token` function on our texts and explicitly tell it to generate bigrams:

```
sotu_whole %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) # create bigram
```

```
#> # A tibble: 1,964,976 x 7
#>    president     year years_active party  sotu_type doc_id        bigram
#>    <chr>        <int> <chr>        <chr>  <chr>     <chr>         <chr>
#>  1 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ fellow ci~
#>  2 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ citizens ~
#>  3 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ of the
#>  4 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ the senate
#>  5 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ senate and
#>  6 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ and house
#>  7 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ house of
#>  8 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ of repres~
#>  9 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ represent~
#> 10 Abraham Lin~  1861 1861-1865    Repub~ written   abraham-lin~ in the
#> # ... with 1,964,966 more rows
```

Let's see the most common bigrams:

```
sotu_whole %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  count(bigram, sort = TRUE) # count ocurrences and sort descending
```

```
#> # A tibble: 469,092 x 2
#>    bigram            n
#>    <chr>         <int>
#>  1 of the        33610
#>  2 in the        12499
#>  3 to the        11643
#>  4 for the        6892
#>  5 and the        6224
#>  6 by the         5606
#>  7 of our         5172
#>  8 the united     4767
#>  9 united states  4760
#> 10 it is          4756
#> # ... with 469,082 more rows
```

Ok, so we again need to remove the stopwords. First let us separate the two words into two columns "word1" and "word2" with `separate` from the `tidyr` package:

```
sotu_whole %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  separate(bigram, c("word1", "word2"), sep = " ")
```

```
#> # A tibble: 1,964,976 x 8
#>    president    year years_active party  sotu_type doc_id     word1  word2
```

```
#>    <chr>       <int> <chr>        <chr>  <chr>     <chr>      <chr>  <chr>
#>  1 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ fellow citiz~
#>  2 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ citiz~ of
#>  3 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ of     the
#>  4 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ the    senate
#>  5 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ senate and
#>  6 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ and    house
#>  7 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ house  of
#>  8 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ of     repre~
#>  9 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ repre~ in
#> 10 Abraham Li~  1861 1861-1865    Repub~ written   abraham-l~ in     the
#> # ... with 1,964,966 more rows
```

Now we use dplyr's `filter` function to select only the words in each column that are not in the stopwords.

```
sotu_whole %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>% # separate into cols
  filter(!word1 %in% stop_words$word) %>% # remove stopwords
  filter(!word2 %in% stop_words$word)
```

```
#> # A tibble: 215,992 x 8
#>    president    year years_active party   sotu_type doc_id    word1  word2
#>    <chr>       <int> <chr>        <chr>   <chr>     <chr>     <chr>  <chr>
#>  1 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ fellow citiz~
#>  2 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ unpre~ polit~
#>  3 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ polit~ troub~
#>  4 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ abund~ harve~
#>  5 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ pecul~ exige~
#>  6 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ forei~ natio~
#>  7 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ profo~ solic~
#>  8 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ solic~ chief~
#>  9 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ domes~ affai~
#> 10 Abraham Li~  1861 1861-1865    Republ~ written   abraham-~ dislo~ porti~
#> # ... with 215,982 more rows
```

Lastly, we re-unite the two word columns into back into our bigrams and save it into a new table `sotu_bigrams`.

```
sotu_bigrams <- sotu_whole %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>% # separate into cols
  filter(!word1 %in% stop_words$word) %>% # remove stopwords
  filter(!word2 %in% stop_words$word) %>%
  unite(bigram, word1, word2, sep = " ")  # combine columns


sotu_bigrams %>%
  count(bigram, sort = TRUE)
```

```
#> # A tibble: 129,622 x 2
#>    bigram                 n
#>    <chr>              <int>
#>  1 federal government   479
#>  2 american people      428
#>  3 june 30              325
#>  4 fellow citizens      296
#>  5 public debt          283
#>  6 public lands         256
#>  7 health care          240
```

```
#>  8 social security      232
#>  9 post office          202
#> 10 annual message       200
#> # ... with 129,612 more rows
```

(Bonus question: What happened on that June 30th?)

A bigram can also be treated as a term in a document in the same way that we treated individual words. That means we can look at tf-idf values in the same way. For example, we can find out the most distinct bigrams that the presidents uttered in all their respective speeches taken together.

We group by president and bigram and then bind the tf-idf value with the `bind_tf_idf` function.

```
sotu_bigrams %>%
  count(president, bigram) %>%
  bind_tf_idf(bigram, president, n) %>%
  arrange(desc(tf_idf))
```

```
#> # A tibble: 169,227 x 6
#>    president          bigram                      n      tf   idf tf_idf
#>    <chr>              <chr>                   <int>   <dbl> <dbl>  <dbl>
#>  1 George W. Bush     al qaida                   35 0.00628  3.02 0.0190
#>  2 John Adams         john adams                  3 0.00510  3.71 0.0189
#>  3 John Adams         amity commerce              5 0.00850  2.10 0.0179
#>  4 John Adams         st croix                    5 0.00850  2.10 0.0179
#>  5 John Adams         6th article                 4 0.00680  2.33 0.0158
#>  6 John Adams         7th article                 4 0.00680  2.33 0.0158
#>  7 John Adams         commissioners acting        3 0.00510  3.02 0.0154
#>  8 John Adams         damages sustained           3 0.00510  3.02 0.0154
#>  9 William J. Clinton 21st century               59 0.00830  1.77 0.0147
#> 10 John Adams         commissioners appointed     7 0.0119   1.23 0.0146
#> # ... with 169,217 more rows
```

CHALLENGE: Again, pick the same president you chose above. For each of his speeches, which is the bigram with highest tf-idf? Create a table as output.

2.5 Co-occurrence

Co-occurrences give us a sense of words that appear in the same text, but not necessarily next to each other.

For this section we will make use of the `widyr` package. The function which helps us do this is the `pairwise_count()` function. It lets us count common pairs of words co-appearing within the same speech.

Behind the scenes, this function first turns our table into a wide matrix. In our case that matrix will be made up of the individual words and the cell values will be the counts of in how many speeches they co-occur, like this:

```
#>      we thus have
#> we   NA    4    5
#> thus  4   NA    2
#> have  5    2   NA
```

It then will turn the matrix back into a tidy form, where each row contains the word pairs and the count of their co-occurrence. Since we don't care about the order of the words, we will not count the upper triangle of the wide matrix, which leaves us with:

```
#>
#>    we thus 4
#>    we have 5
#>  thus have 2
```

we | thus | 4 we | have | 5 thus | have | 2

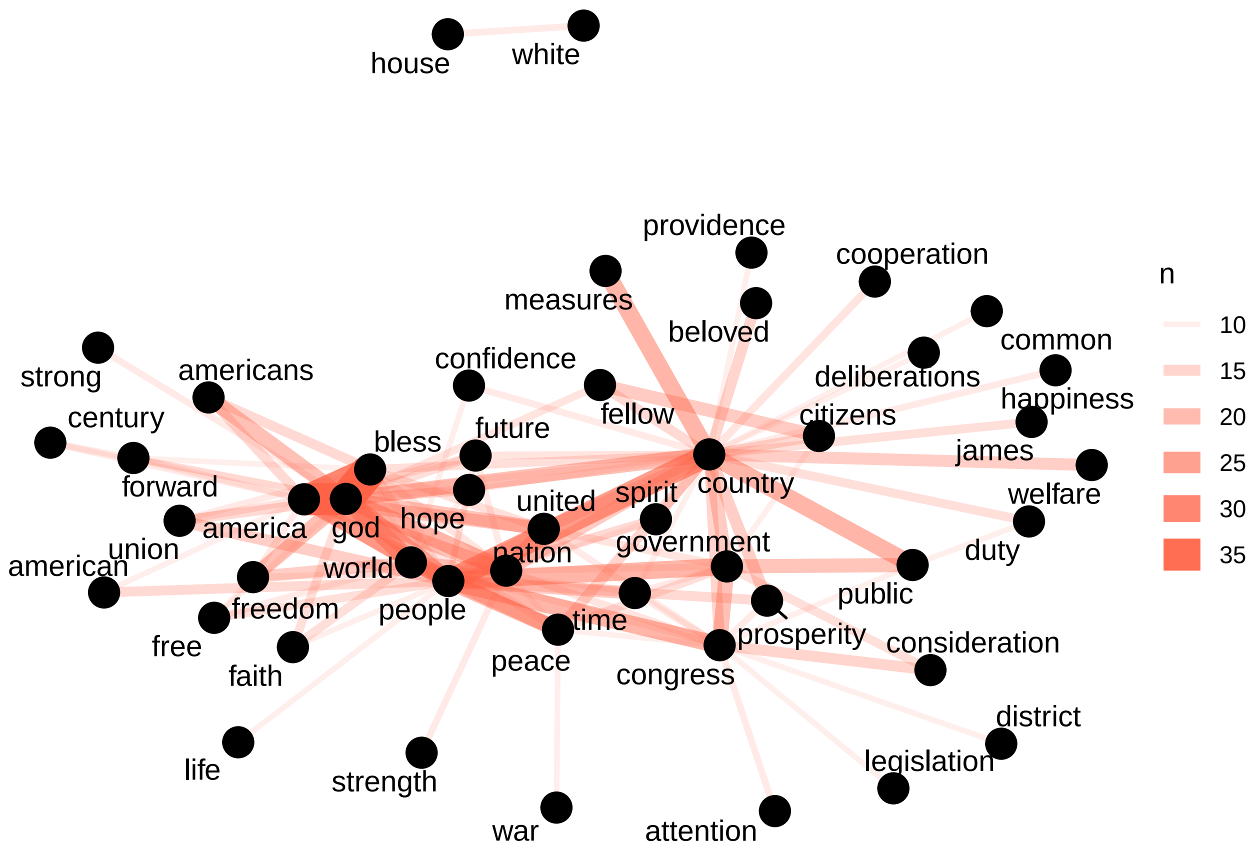
Since processing the entire corpus would take too long here, we will only look at the last 100 words of each speech: which words occurr most commonly together at the end of the speeches?

```
library(widyr)

sotu_word_pairs <- sotu_whole %>%
  mutate(speech_end = word(text, -100, end = -1)) %>%  # extract last 100 words
  unnest_tokens(word, speech_end) %>%   # tokenize
  filter(!word %in% stop_words$word) %>%  # remove stopwords
  pairwise_count(word, doc_id, sort = TRUE, upper = FALSE) # don't include upper triangle of matrix

sotu_word_pairs
```

```
#> # A tibble: 125,576 x 3
#>    item1      item2       n
#>    <chr>      <chr>   <dbl>
#>  1 god        bless      37
#>  2 god        america    35
#>  3 bless      america    30
#>  4 people     country    26
#>  5 world      god        22
#>  6 god        people     22
#>  7 government people     21
#>  8 congress   people     21
#>  9 public     country    21
#> 10 god        nation     21
#> # ... with 125,566 more rows
```

To visualize the co-occurrence network of words that occur together at the end of 10 or more speeches, we use the `igraph` package to convert our table into a network graph and the `ggraph` package which adds functionality to ggplot to make it easier to plot a network.

```
library(igraph)
library(ggraph)

sotu_word_pairs %>%
  filter(n >= 10) %>%  # only word pairs that occur 10 or more times
  graph_from_data_frame() %>% #convert to graph
  ggraph(layout = "fr") + # place nodes according to the force-directed algorithm of Fruchterman and Reingold
  geom_edge_link(aes(edge_alpha = n, edge_width = n), edge_colour = "tomato") +
  geom_node_point(size = 5) +
  geom_node_text(aes(label = name), repel = TRUE,
                 point.padding = unit(0.2, "lines")) +
  theme_void()
```



There are alternative approaches for this as well. See for example the `findAssocs` function in the `tm` package.

2.6 Document-Term Matrix

A document-term matrix (DTM) is a format which is frequently used in text analysis. It is a matrix where we can see the counts of each term per document. In a DTM each row represents a document, each column represents a term, and the cell values are the counts of the occurrences of the term for the particular document.

`tidytext` provides functionality to convert to and from DTMs, if for example, your analyis requires specific functions that require you to use a different R package which only works with DTM objects.

The `cast_dtm` function can be used to create a DTM object from a tidy table.

Let's assume that for some reason we want to use the `findAssoc` function from the `tm` package.

First we use dplyr to create a table with the document name, the term, and the count.

```
# make a table with document, term, count
tidy_sotu_words %>%
  count(doc_id, word)
```

```
#> # A tibble: 352,846 x 3
#>    doc_id                    word                n
#>    <chr>                     <chr>           <int>
#>  1 abraham-lincoln-1861.txt 1,470,018           1
#>  2 abraham-lincoln-1861.txt 1,500               1
#>  3 abraham-lincoln-1861.txt 100,000             1
#>  4 abraham-lincoln-1861.txt 102,532,509.27      1
#>  5 abraham-lincoln-1861.txt 12,528,000          1
#>  6 abraham-lincoln-1861.txt 13,606,759.11       1
#>  7 abraham-lincoln-1861.txt 1830                1
```

```
#>  8 abraham-lincoln-1861.txt 1859              1
#>  9 abraham-lincoln-1861.txt 1860              2
#> 10 abraham-lincoln-1861.txt 1861              6
#> # ... with 352,836 more rows
```

Now we cast it as a DTM.

```
sotu_dtm <- tidy_sotu_words %>%
  count(doc_id, word) %>%
  cast_dtm(doc_id, word, n)

class(sotu_dtm)
```

```
#> [1] "DocumentTermMatrix"    "simple_triplet_matrix"
```

Finally, let's use it in the `tm` package.

```
library(tm)

# look at the terms with tm function
Terms(sotu_dtm) %>% tail()
```

```
#> [1] "queretaro"    "refreshments" "schleswig"    "sedulous"
#> [5] "subagents"    "transcript"
```

```
# most frequent terms
findFreqTerms(sotu_dtm, lowfreq = 5000)
```

```
#> [1] "congress"   "government" "united"
```

```
# find terms associated with ...
findAssocs(sotu_dtm, "citizen", corlimit = 0.5)
```

```
#> $citizen
#>        laws citizenship  protection   contained    entitled  government
#>        0.62        0.59        0.56        0.55        0.53        0.53
#>    citizens  postmaster     careful    question      report       suits
#>        0.52        0.52        0.51        0.51        0.51        0.51
```

Conversely, `tidytext` implements the `tidy` function (originally from the `broom` package) to import DocumentTermMatrix objects. Note that it only takes the cells from the DTM that are not 0, so there will be no rows with 0 counts.

2.7 Sentiment analysis

`tidytext` comes with a dataset `sentiments` which contains several sentiment lexicons, where each word is attributed a certain sentiment, like this:

```
sentiments
```

```
#> # A tibble: 6,786 x 2
#>    word        sentiment
#>    <chr>       <chr>
#>  1 2-faces     negative
#>  2 abnormal    negative
#>  3 abolish     negative
#>  4 abominable  negative
#>  5 abominably  negative
#>  6 abominate   negative
#>  7 abomination negative
#>  8 abort       negative
```

```
#>  9 aborted     negative
#> 10 aborts      negative
#> # ... with 6,776 more rows
```

Here we will take a look at how the sentiment of the speeches change over time. We will use the lexicon from Bing Liu and collaborators, which assigns positive/negative labels for each word:

```
bing_lex <- get_sentiments("bing")
bing_lex
```

```
#> # A tibble: 6,786 x 2
#>    word        sentiment
#>    <chr>       <chr>
#>  1 2-faces     negative
#>  2 abnormal    negative
#>  3 abolish     negative
#>  4 abominable  negative
#>  5 abominably  negative
#>  6 abominate   negative
#>  7 abomination negative
#>  8 abort       negative
#>  9 aborted     negative
#> 10 aborts      negative
#> # ... with 6,776 more rows
```

We can use these sentiments attached to each word and join them to the words of our speeches. We will use `inner_join` from `dplyr`. It will take all rows with words from `tidy_sotu_words` that match words in `bing_lex`, eliminating rows where the word cannot be found in the lexicon. Since our columns to join on have the same name (`word`) we don't need to explicitly name it.

```
sotu_sentiments <- tidy_sotu_words %>%
  inner_join(bing_lex)  # join to add semtinemt column

sotu_sentiments
```

```
#> # A tibble: 105,206 x 8
#>    president   year years_active party  sotu_type doc_id    word  sentiment
#>    <chr>      <int> <chr>        <chr>  <chr>     <chr>     <chr> <chr>
#>  1 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ trou~ negative
#>  2 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ grat~ positive
#>  3 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ unus~ negative
#>  4 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ abun~ positive
#>  5 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ pecu~ negative
#>  6 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ prof~ positive
#>  7 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ soli~ negative
#>  8 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ disl~ negative
#>  9 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ dest~ negative
#> 10 Abraham L~  1861 1861-1865    Repub~ written   abraham-~ disr~ negative
#> # ... with 105,196 more rows
```

Finally we can visualize the proportion of positive sentiment (out of the total of positive and negative) in US State of the Union Addresses over time like this:

```
sotu_sentiments %>%
  count(year, sentiment) %>% # group by year and sentiment
  spread(sentiment, n) %>% # spread by seniment
  mutate(pct_positive_sentiment = positive/(negative + positive) * 100) %>% # calculate %
  ggplot(aes(year, pct_positive_sentiment)) +
    geom_line(color="gray") +
    geom_smooth(span = 0.3, se = FALSE) + # for easier viewing
```

```
    geom_hline(yintercept = 50, linetype="dotted") + # 50% as reference
    scale_x_continuous(breaks = seq(1790, 2016, by = 10)) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```