Text Analysis with R

Claudia Engel, Scott Bailey

Last updated: May 04, 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.
- Libraries needed:
  - `tidyverse`
  - `tidytext`
  - `readtext`
  - `sotu`
  - `SnowballC`
  - `widyr`
  - `igraph`
  - `ggraph`
  - `tm`

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 libraries today. `sotu` will provide the metadata and text of State of the Union speeches ranging from George Washington to Barack Obama. `tidyverse` provides many of the standard "verbs" for working with our data. `tidytext` provides specific functions for a "tidy" approach to working with textual data. `readtext` provides a function well suited to reading textual data from a large number of formats into R.

```
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. We'll use the supplied metadata object, but we're going to use a utility function (`sotu_dir`) in the package to write the texts to disk so that we can practice reading text files from disk.

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

```
#>   president              year       years_active         party
#>  Length:236         Min.   :1790   Length:236         Length:236
#>  Class :character   1st Qu.:1848   Class :character   Class :character
#>  Mode  :character   Median :1906   Mode  :character   Mode  :character
#>                     Mean   :1905
#>                     3rd Qu.:1962
#>                     Max.   :2016
#>   sotu_type
#>  Length:236
#>  Class :character
```

```
#>  Mode  :character
#>
#>
#>
```

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

```
#> [1] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp3Sozde/file11f07a08734b/george-washington-
#> [2] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp3Sozde/file11f07a08734b/george-washington-
#> [3] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp3Sozde/file11f07a08734b/george-washington-
#> [4] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp3Sozde/file11f07a08734b/george-washington-
#> [5] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp3Sozde/file11f07a08734b/george-washington-
#> [6] "/var/folders/5y/9x92pjcx2xd2h7qxqx39vpmc0000gn/T//Rtmp3Sozde/file11f07a08734b/george-washington-
```

Now that we have the files on disk, and a list of filepaths stored in the `fp` variable, we can use `readtext` to read the texts into a new variable.

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

```
#> readtext object consisting of 6 documents and 0 docvars.
#> # Description: data.frame [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 \"..."
```

So that we can work with a single tabular dataset with a tidy approach, we'll convert the metadata and text tables to tibbles, and combine them into a single tibble. You can see that our texts are organized by alphabetical order, so first we'll need to sort our metadata to match.

```
sotu_meta_tib <- as_tibble(sotu_meta) %>%
  arrange(president)

head(sotu_meta_tib)
```

```
#> # A tibble: 6 x 5
#>   president        year years_active party      sotu_type
#>   <chr>           <int> <chr>        <chr>      <chr>
#> 1 Abraham Lincoln  1861 1861-1865    Republican written
#> 2 Abraham Lincoln  1862 1861-1865    Republican written
#> 3 Abraham Lincoln  1863 1861-1865    Republican written
#> 4 Abraham Lincoln  1864 1861-1865    Republican written
#> 5 Andrew Jackson   1829 1829-1833    Democratic written
#> 6 Andrew Jackson   1830 1829-1833    Democratic written
```

We can now combine the sotu metadata with the texts. We'll turn both pieces of data into tibbles, then combine.

```
sotu_texts <- as_tibble(texts)
sotu_whole <- bind_cols(sotu_meta_tib, sotu_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, but we'll look at some straightforward 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.

Perhaps one of the most comprehensive packages is `stringi`. However, we will here take a look at the `stringr` package, which is part of the `tidyverse`, wraps a lot of the stringi functions, and is easier to begin with.

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

- How many words in each speech?

```
str_count(sotu_whole$text, boundary("word"))
```

```
#>   [1]  6998  8410  6132  5975 10547 15109  7198  7887  7912 13472 10839
#>  [12] 12386  9258  7155 12032  9886  6092  7263  6909  7058  6851  7064
#>  [23]  6797  6078 13038 11559 16357 13718  6715  6978 10871 10333  8800
#>  [34]  8083 13382 10315  8414  8956  6997  6027  7295  1090  8339  4166
#>  [45]  4954  4995  5693  6260  2233  3536  3835  2743  4716  3785  3218
#>  [56]  3332  3523  4618  3842  3162  8202  9613 10152 11626 10512  4824
#>  [67]  3788  3964  5117  4380  3838  5390  5197  5071  5326  5571  5726
#>  [78]  1091  1404  2305  2100  1969  2922  1988  2879  4153  4999  4747
#>  [89] 19828 15196  5305 13275 12360 15972 14710 15568 27941  6090  5130
#> [100]  3418  5154  4000  5387  9737 11051  4560  5705  4232 13685 16396
#> [111] 12378 14085 16147 18251 16446 21366  1832  2448  2273  3248  3259
#> [122]  2115  3145  3367  4423  4378  4709  3447  5831  4733  6383  8416
#> [133]  4580 12155  3273 21588  3475 33537 33921  2062  2220  1507  1374
#> [144]  5300  6637  5429  9027  7745  6996  7329  8254  8436  8047  9331
#> [155]  3233  4453  5582  7221  4941  4137 11471 11520 13463  9007  8340
#> [166] 13273  9951  4482  4532  3997 17383  5199 22419  4469  5182  5582
#> [177]  4967  4235  3493  3814  4851 10755  7909 11670 13405 19708  9825
#> [188] 15017 17517 25147 23680 27519 19515  3228  2203  2270  2100  2930
#> [199]  2862  2387  2676  7720  8772  6480 10131 10058  9844 12238  6816
#> [210]  5621  5774 13947 27696 23838 25275  7029  7423  9203  6357  6780
#> [221]  7317  7513  9119 12157 20301 22907 19228  3566  4550  7735  2125
#> [232]  3931  5482  4765  2714  7637
```

- Measured by the average number of words per sentence for each speech - what is the length of the speech with the shortest/longest sentences?

```
range(str_count(sotu_whole$text, boundary("word"))/str_count(sotu_whole$text, boundary("sentence")))
```

```
#> [1]  9.143737 37.219008
```

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

```
str_count(sotu_whole$text, "[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
```

What are the names of the documents where the word "citizen" does **not** occur?

```
sotu_whole$doc_id[!str_detect(sotu_whole$text, "[C|c]itizen")]
```

```
#>  [1] "dwight-d-eisenhower-1958.txt" "gerald-r-ford-1975.txt"
#>  [3] "richard-m-nixon-1970.txt"     "richard-m-nixon-1971.txt"
#>  [5] "richard-m-nixon-1972a.txt"    "ronald-reagan-1988.txt"
#>  [7] "woodrow-wilson-1916.txt"      "woodrow-wilson-1917.txt"
#>  [9] "woodrow-wilson-1918.txt"      "woodrow-wilson-1919.txt"
#> [11] "woodrow-wilson-1920.txt"
```

- Get me the first 5 words for each speech.

```
word(sotu_whole$text, end = 5) %>%
  unique()
```

```
#>  [1] "\n\n Fellow-Citizens of the Senate"
#>  [2] "\n\n Fellow Citizens of the"
#>  [3] "Madam Speaker, Mr. Vice President,"
#>  [4] "Madam Speaker, Vice President Biden,"
#>  [5] "Mr. Speaker, Mr. Vice President,"
#>  [6] "Please, everybody, have a seat."
#>  [7] "The President. Mr. Speaker, Mr."
#>  [8] "Thank you. Mr. Speaker, Mr."
#>  [9] "\n\n To the Senate and"
#> [10] "\n\nSince the close of the"
#> [11] "\n\nTo the Congress of the"
#> [12] "Members of the Congress: \n\nIn"
#> [13] "\n\nMembers of the Congress: \n\nIn"
#> [14] "\n\n Members of the Congress:"
#> [15] "\n\n To the Congress of"
#> [16] "Mr. President, Mr. Speaker, Members"
#> [17] "\n\n[Recorded on film and tape"
#> [18] "\n\n[Read before a joint session"
#> [19] "To the Congress of the"
```

```
#> [20] "\n\n[Delivered in person before a"
#> [21] "Mr. President, Mr. Speaker, Senators"
#> [22] "Mr. Vice President, Mr. Speaker,"
#> [23] "IN FULFILLING my duty to"
#> [24] "To the Congress: \n\nThis Nation"
#> [25] "\n\nToday, in pursuance of my"
#> [26] "\n\nTo the Congress:\n\nIn considering the"
#> [27] "\n\nMr. Speaker, Mr. President, and"
#> [28] "\n\nMr. President, Mr. Speaker, Members"
#> [29] "\n\nMr. President and Mr. Speaker"
#> [30] "\n\nMr. Speaker and Mr. President,"
#> [31] "Thank you very much. Mr."
#> [32] "Mr. Speaker, Vice President Cheney,"
#> [33] "Thank you all. Mr. Speaker,"
#> [34] "Thank you very much. And"
#> [35] "Thank you all. Madam Speaker,"
#> [36] "Fellow-Citizens of the Senate and"
#> [37] "\n\nFellow-Citizens of the Senate and"
#> [38] "\n\nMr. Speaker, Mr. Vice President,"
#> [39] "[Released January 21, 1946. Dated"
#> [40] "Mr. President, Mr. Speaker, and"
#> [41] "To the Senate and House"
#> [42] "\n\nTo the Senate and House"
#> [43] "\n\n Gentlemen of the Senate"
#> [44] "\n\n[ As delivered in person"
#> [45] "\n\n[As delivered in person before"
#> [46] "Mr. Speaker, Mr. President, Members"
#> [47] "\n\n[ Delivered in person before"
#> [48] "\n\nMr. Speaker, Mr. President, my"
#> [49] "Mr. Speaker, Mr. President, my"
#> [50] "Mr. Speaker, Mr. President, distinguished"
#> [51] "Mr. Speaker, Mr. President, and"
#> [52] "\n\n To The Senate and"
#> [53] "\n\n The Senate and House"
#> [54] "\n\nMR. SPEAKER AND MEMBERS OF"
#> [55] "\n\nMEMBERS OF THE CONGRESS: \n\nSo"
#> [56] "\n\nThe relations of the United"
#> [57] "\n\n Jump to Part II"
#> [58] "\n\nGentlemen of the Congress:\n\nIn pursuance"
#> [59] "\n\nGENTLEMEN OF THE CONGRESS: \n\nThe"
#> [60] "GENTLEMEN OF THE CONGRESS: \n\nSince"
#> [61] "\n\nGENTLEMEN OF THE CONGRESS: \n\nIn"
#> [62] "Gentlemen of the Congress:\n\nEight months"
#> [63] "\n\nTO THE SENATE AND HOUSE"
#> [64] "\n\nGENTLEMEN OF THE CONGRESS:\n\nWhen I"
```

- Now remove the newline character (`\n`) and get rid of any leading white space:

```
word(sotu_whole$text, end = 5) %>%
  unique() %>%
  str_replace_all("\\n", " ") %>%
  str_trim()
```

```
#>  [1] "Fellow-Citizens of the Senate"
#>  [2] "Fellow Citizens of the"
```

```
#>  [3] "Madam Speaker, Mr. Vice President,"
#>  [4] "Madam Speaker, Vice President Biden,"
#>  [5] "Mr. Speaker, Mr. Vice President,"
#>  [6] "Please, everybody, have a seat."
#>  [7] "The President. Mr. Speaker, Mr."
#>  [8] "Thank you. Mr. Speaker, Mr."
#>  [9] "To the Senate and"
#> [10] "Since the close of the"
#> [11] "To the Congress of the"
#> [12] "Members of the Congress:   In"
#> [13] "Members of the Congress:   In"
#> [14] "Members of the Congress:"
#> [15] "To the Congress of"
#> [16] "Mr. President, Mr. Speaker, Members"
#> [17] "[Recorded on film and tape"
#> [18] "[Read before a joint session"
#> [19] "To the Congress of the"
#> [20] "[Delivered in person before a"
#> [21] "Mr. President, Mr. Speaker, Senators"
#> [22] "Mr. Vice President, Mr. Speaker,"
#> [23] "IN FULFILLING my duty to"
#> [24] "To the Congress:   This Nation"
#> [25] "Today, in pursuance of my"
#> [26] "To the Congress:  In considering the"
#> [27] "Mr. Speaker, Mr. President, and"
#> [28] "Mr. President, Mr. Speaker, Members"
#> [29] "Mr. President and Mr. Speaker"
#> [30] "Mr. Speaker and Mr. President,"
#> [31] "Thank you very much. Mr."
#> [32] "Mr. Speaker, Vice President Cheney,"
#> [33] "Thank you all. Mr. Speaker,"
#> [34] "Thank you very much. And"
#> [35] "Thank you all. Madam Speaker,"
#> [36] "Fellow-Citizens of the Senate and"
#> [37] "Fellow-Citizens of the Senate and"
#> [38] "Mr. Speaker, Mr. Vice President,"
#> [39] "[Released January 21, 1946. Dated"
#> [40] "Mr. President, Mr. Speaker, and"
#> [41] "To the Senate and House"
#> [42] "To the Senate and House"
#> [43] "Gentlemen of the Senate"
#> [44] "[ As delivered in person"
#> [45] "[As delivered in person before"
#> [46] "Mr. Speaker, Mr. President, Members"
#> [47] "[ Delivered in person before"
#> [48] "Mr. Speaker, Mr. President, my"
#> [49] "Mr. Speaker, Mr. President, my"
#> [50] "Mr. Speaker, Mr. President, distinguished"
#> [51] "Mr. Speaker, Mr. President, and"
#> [52] "To The Senate and"
#> [53] "The Senate and House"
#> [54] "MR. SPEAKER AND MEMBERS OF"
#> [55] "MEMBERS OF THE CONGRESS:   So"
#> [56] "The relations of the United"
```

```
#> [57] "Jump to Part II"
#> [58] "Gentlemen of the Congress:  In pursuance"
#> [59] "GENTLEMEN OF THE CONGRESS:   The"
#> [60] "GENTLEMEN OF THE CONGRESS:   Since"
#> [61] "GENTLEMEN OF THE CONGRESS:   In"
#> [62] "Gentlemen of the Congress:  Eight months"
#> [63] "TO THE SENATE AND HOUSE"
#> [64] "GENTLEMEN OF THE CONGRESS:  When I"
```

(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 data cleaning involves tokenization. While our data is already "tidy" insofar as each row is a single observation, a single text with metdata, the tidytext approach goes a step further to make each word it's own observation with metadata. We could write our own function to do this using a tokenizer, but `tidytext` provides a handy utility function just for this purpose.

```
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
```

Before we move on, we should note that the `unnest_tokens` function didn't just tokenize our texts at the word level. It also lowercased each word, and it could do quite a bit more. For instance, we could tokenize the text at the level of ngrams or sentences, if those are the best units of analysis for our work. We could also leave punctuation, which has been removed by default. Depending on what you need to do for analysis, you could do these operations during this step, or write custom functions and do it before you unnest tokens.

```
# Word tokenization with punctuation
tidy_sotu_w_punct <- sotu_whole %>%
  unnest_tokens(word, text, strip_punct = FALSE)

tidy_sotu_w_punct
```

```
#> # 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
```

```r
# Sentence tokenization
tidy_sotu_sentences <- sotu_whole %>%
  unnest_tokens(sentence, text, token = "sentences", to_lower = FALSE)

tidy_sotu_sentences
```

```
#> # A tibble: 69,158 x 7
#>    president   year years_active party  sotu_type doc_id   sentence
#>    <chr>      <int> <chr>        <chr>  <chr>     <chr>    <chr>
#>  1 Abraham L~  1861 1861-1865    Repub~ written   abraham~ Fellow-Citizens~
#>  2 Abraham L~  1861 1861-1865    Repub~ written   abraham~ You will not be~
#>  3 Abraham L~  1861 1861-1865    Repub~ written   abraham~ A disloyal port~
#>  4 Abraham L~  1861 1861-1865    Repub~ written   abraham~ A nation which ~
#>  5 Abraham L~  1861 1861-1865    Repub~ written   abraham~ Nations thus te~
#>  6 Abraham L~  1861 1861-1865    Repub~ written   abraham~ The disloyal ci~
#>  7 Abraham L~  1861 1861-1865    Repub~ written   abraham~ If it were just~
#>  8 Abraham L~  1861 1861-1865    Repub~ written   abraham~ If we could dar~
#>  9 Abraham L~  1861 1861-1865    Repub~ written   abraham~ The principal l~
#> 10 Abraham L~  1861 1861-1865    Repub~ written   abraham~ Those nations, ~
#> # ... with 69,148 more rows
```

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

tidy_sotu_trigram
```

```
#> # A tibble: 1,964,740 x 7
#>    president    year years_active party   sotu_type doc_id      trigram
#>    <chr>       <int> <chr>        <chr>   <chr>     <chr>       <chr>
#>  1 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ fellow cit~
#>  2 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ citizens o~
#>  3 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ of the sen~
#>  4 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ the senate~
#>  5 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ senate and~
#>  6 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ and house ~
#>  7 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ house of r~
#>  8 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ of represe~
#>  9 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ representa~
#> 10 Abraham Li~  1861 1861-1865    Republ~ written   abraham-li~ in the mid~
#> # ... with 1,964,730 more rows
```

1.4 Stopwords

Another common type of cleaning in text analysis is to remove stopwords, or common words that theoretically provide less information about the content of a text. Depending on the type of analysis you're doing, you might leave these words in or use a highly curated list of stopwords. For now, as we move toward looking at words in documents based on frequency, we will remove some standard stopwords using a tidytext approach.

First, let's look at the stopwords that tidytext gives us to get a sense of what they are.

```
data(stop_words)
head(stop_words, n = 60)
```

```
#> # A tibble: 60 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 50 more rows
```

You can see that we now have one word per row with associated metadata. We can now remove stopwords using an `anti-join`.

```
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
```

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 take a closer look at the stopwords and determine if we should use a different stopword list or otherwise create our own.

1.5 Word Stemming

Another thing you may want to do is to stem your words, that is, to reduce them to their word stem or root form, like 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. (`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)) %>% head()
```

```
#> # A tibble: 6 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
```

For lemmatization, 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.

Now that we've read in our text and metadata, reshaped it a bit into the tidytext format, and cleaned it a bit while doing so, let's move on to some basic analysis.

Chapter 2

Analyzing Texts

Learning Objectives

- perform basic text analysis operations in R
- determine differnd kinds of frequency counts
- 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

---

First, we'll load the libraries we 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

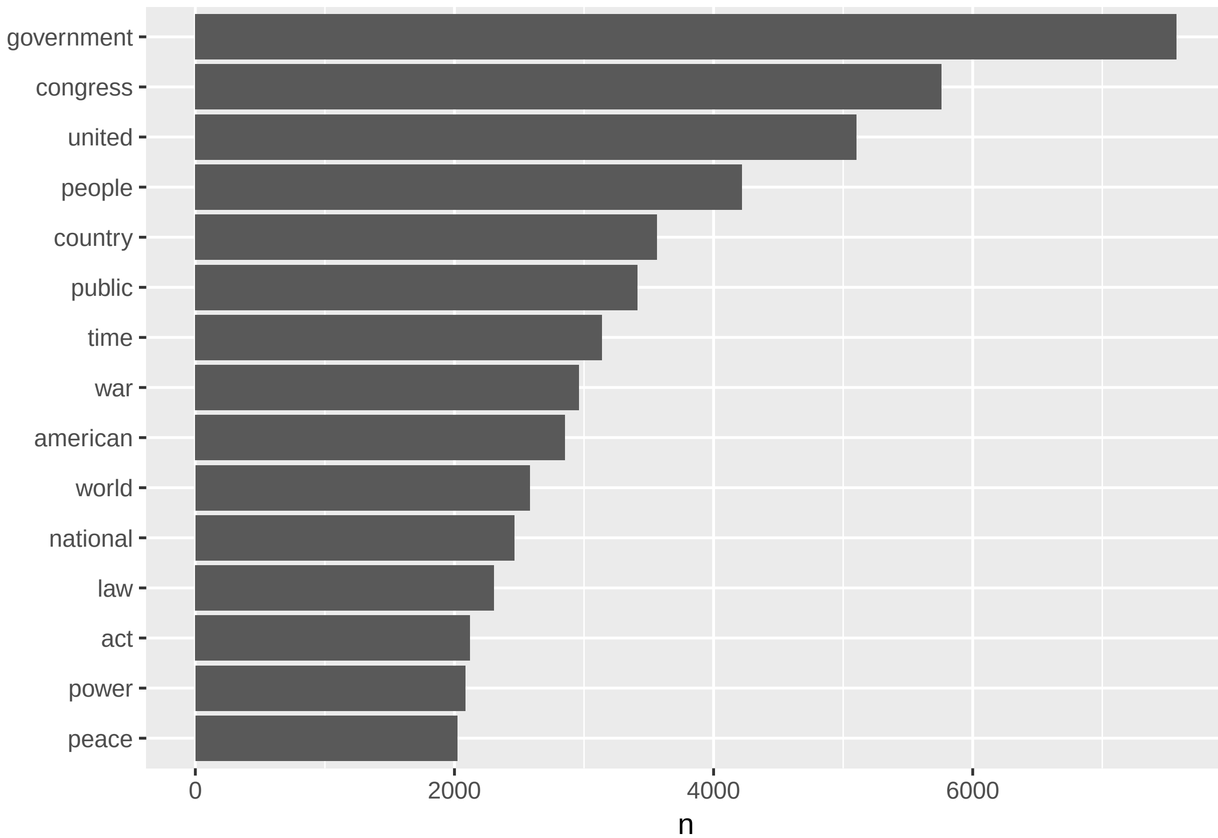
Since our unit of analysis at this point is a word, let's do some straightforward counting to figure out which words occur most frequently in the corpus as a whole.

```r
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 could start adding in a bit of visualization here. Let's show the most frequent words that occur more than 2000 times.

```r
tidy_sotu_words %>%
  count(word, sort = TRUE) %>%
  filter(n > 2000) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col() +
  xlab(NULL) +
  coord_flip()
```



What if we're interested in most used words per speech?

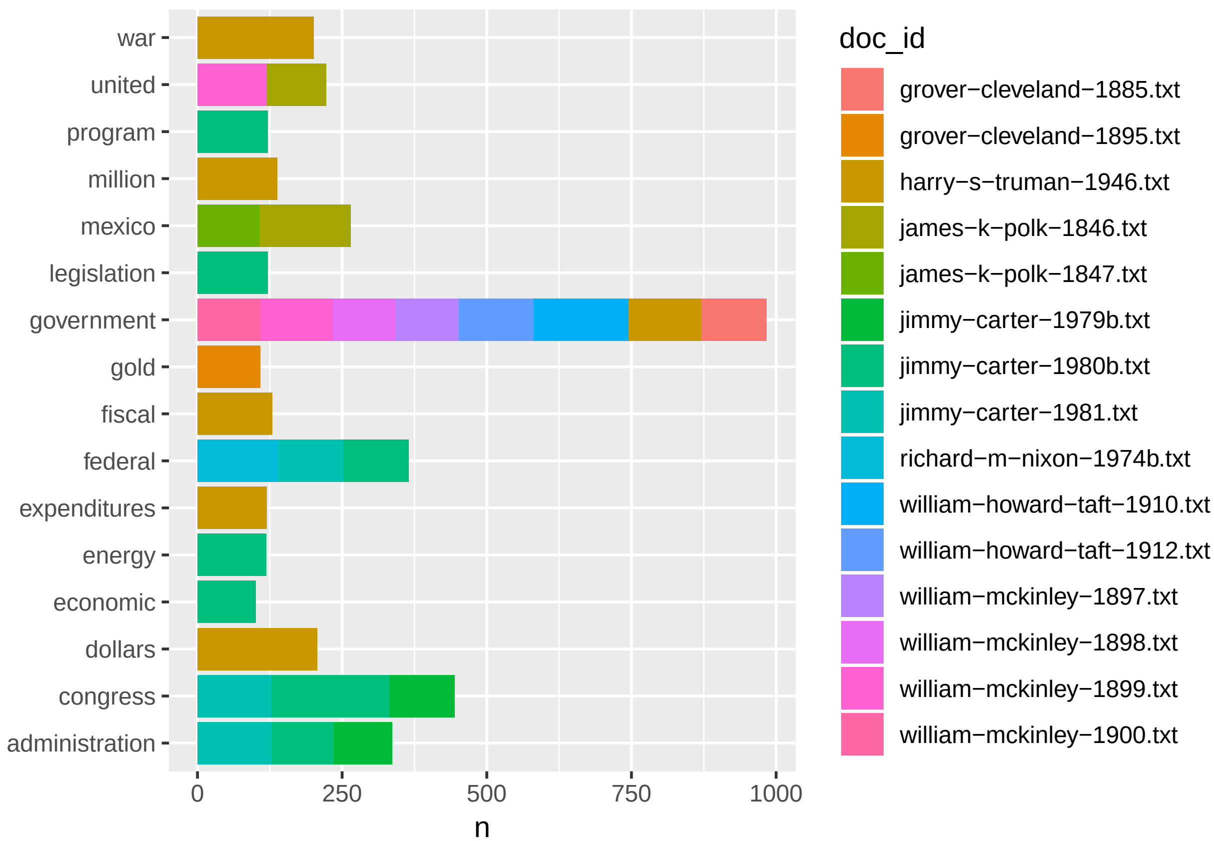
```
# Count words by book
doc_words <- tidy_sotu_words %>%
  count(doc_id, word, sort = TRUE)

# Calculate the total number of words by book and save them to a tibble
total_words <- doc_words %>%
  group_by(doc_id) %>%
  summarize(total = sum(n))

# Join the total column with the rest of the data so we can calculate frequency
doc_words <- left_join(doc_words, total_words)

doc_words
```

```
#> # A tibble: 352,846 x 4
#>    doc_id                         word               n total
#>    <chr>                          <chr>          <int> <int>
#>  1 harry-s-truman-1946.txt       dollars          207 12614
#>  2 jimmy-carter-1980b.txt        congress         204 16128
#>  3 harry-s-truman-1946.txt       war              201 12614
#>  4 william-howard-taft-1910.txt government       164 11178
#>  5 james-k-polk-1846.txt         mexico           158  7023
#>  6 richard-m-nixon-1974b.txt     federal          141  9996
#>  7 harry-s-truman-1946.txt       million          138 12614
#>  8 harry-s-truman-1946.txt       fiscal           129 12614
#>  9 jimmy-carter-1981.txt         administration   129 16595
```

```
#> 10 william-howard-taft-1912.txt government       129 10215
#> # ... with 352,836 more rows
```

Let's graph the top words per book.

```
doc_words %>%
  filter(n > 100) %>%
  ggplot(aes(word, n, fill = doc_id)) +
  geom_col() +
  xlab(NULL) +
  coord_flip()
```



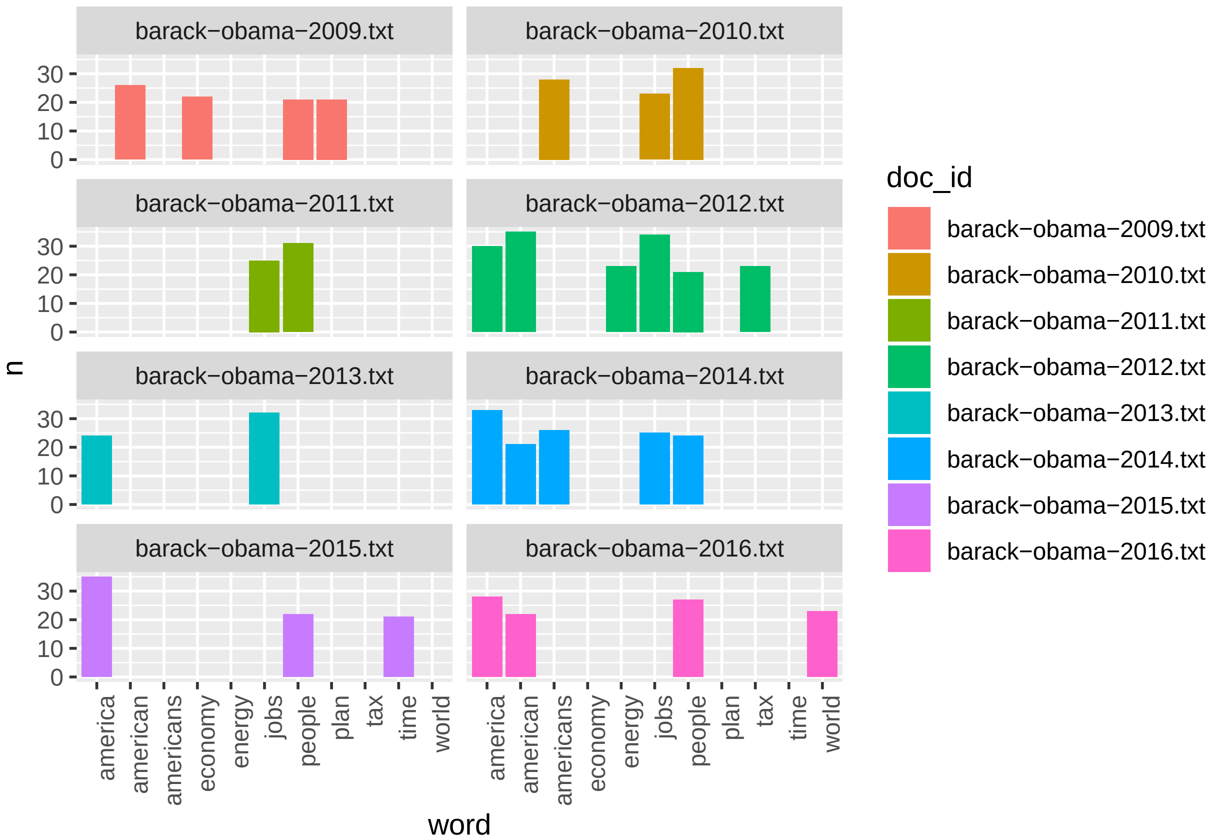
That's cool looking, but let's split it into facets so we can see by speech.

```
doc_words %>%
  filter(n > 100) %>%
  ggplot(aes(word, n, fill = doc_id)) +
  geom_col(show.legend = FALSE) +
  xlab(NULL) +
  facet_wrap(~doc_id, ncol = 2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

We could keep cleaning this figure up by setting some minimum sizing, determining the spacing between y-axis labels better, and so forth, but for now we'll accept it as showing some sense of variation across speeches where certain words are used most.

What if we want to check the most common words per speech for a single president? We could filter this `doc_words` dataset based on the president's name being in the doc_id, but I think it's easier to filter from the initial tidy data and recount.

```
tidy_sotu_words %>%
  filter(president == "Barack Obama") %>%
  count(doc_id, word, sort = TRUE) %>%
  filter(n > 20) %>%
  ggplot(aes(word, n, fill=doc_id)) +
  geom_col() +
  facet_wrap(~doc_id, ncol = 2) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



2.2 Term frequency

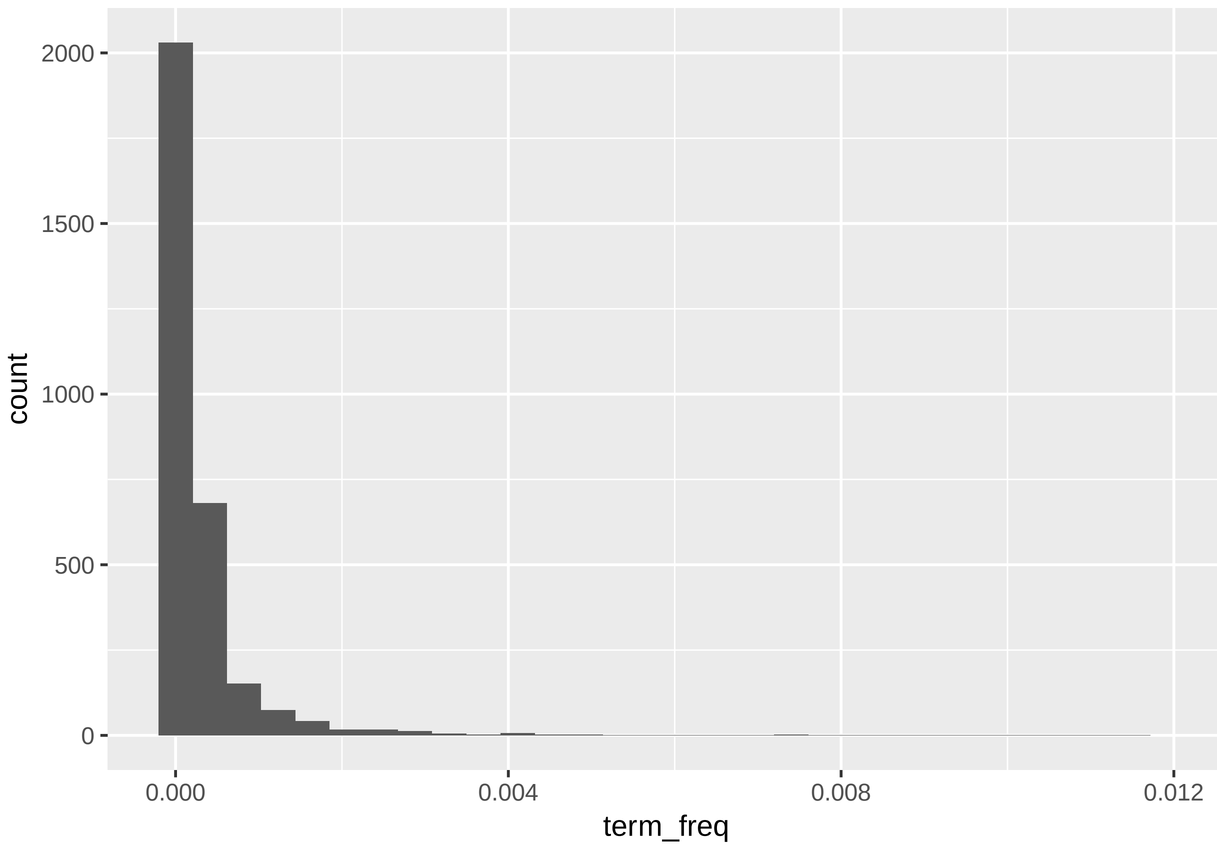
Sometimes, a raw count of a word is less important than understanding how often that word appears in respect to the total number of words in a text. This ratio would be the **term frequency**.

```
doc_words <- doc_words %>%
  mutate(term_freq = n / total)

doc_words
```

```
#> # A tibble: 352,846 x 5
#>    doc_id                        word               n total 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.

```
doc_words %>%
  filter(doc_id == "harry-s-truman-1946.txt") %>%
  ggplot(aes(term_freq)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA, .012)
```



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. We could also subset based on decade, and get a sense of what was important in each decade. We're going to take a slightly different approach though. 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?

2.3 Tf-idf

For this, we can use term-frequency according to inverse document frequency (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).

We could write our own function for tf-idf, but in this case we'll take advantage of tidytext's implementation.

```
doc_words <- doc_words %>%
  bind_tf_idf(word, doc_id, n)
```

```
doc_words
```

```
#> # A tibble: 352,846 x 8
#>    doc_id            word          n total term_freq      tf     idf  tf_idf
#>    <chr>             <chr>     <int> <int>     <dbl>   <dbl>   <dbl>   <dbl>
#>  1 harry-s-truman--~ dollars     207 12614   0.0164  0.0164  0.612   1.00e-2
#>  2 jimmy-carter-19~  congress    204 16128   0.0126  0.0126  0.00425 5.37e-5
#>  3 harry-s-truman--~ war         201 12614   0.0159  0.0159  0.0345  5.50e-4
#>  4 william-howard-~  governme~   164 11178   0.0147  0.0147  0.00425 6.23e-5
#>  5 james-k-polk-18~  mexico      158  7023   0.0225  0.0225  0.810   1.82e-2
#>  6 richard-m-nixon~  federal     141  9996   0.0141  0.0141  0.293   4.14e-3
#>  7 harry-s-truman--~ million     138 12614   0.0109  0.0109  0.728   7.96e-3
#>  8 harry-s-truman--~ fiscal      129 12614   0.0102  0.0102  0.494   5.05e-3
#>  9 jimmy-carter-19~  administ~   129 16595   0.00777 0.00777 0.282   2.19e-3
#> 10 william-howard-~  governme~   129 10215   0.0126  0.0126  0.00425 5.36e-5
#> # ... with 352,836 more rows
```

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.

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.

```
doc_words %>%
  select(-total) %>%
  arrange(desc(tf_idf))
```

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

These results seem appropriate given our history. To understand the occurrence of the years 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. It might be that even if they don't need to be removed from the text overall, they still need to be filtered out within the context of this analysis.

In the same way that we narrowed our analysis to Obama speeches earlier, we could subset the corpus before we calculate the tf-idf score to understand which words are most important for a single president within their sotu speeches. Let's do that for Obama.

```
obama_tf_idf <- tidy_sotu_words %>%
  filter(president == "Barack Obama") %>%
```

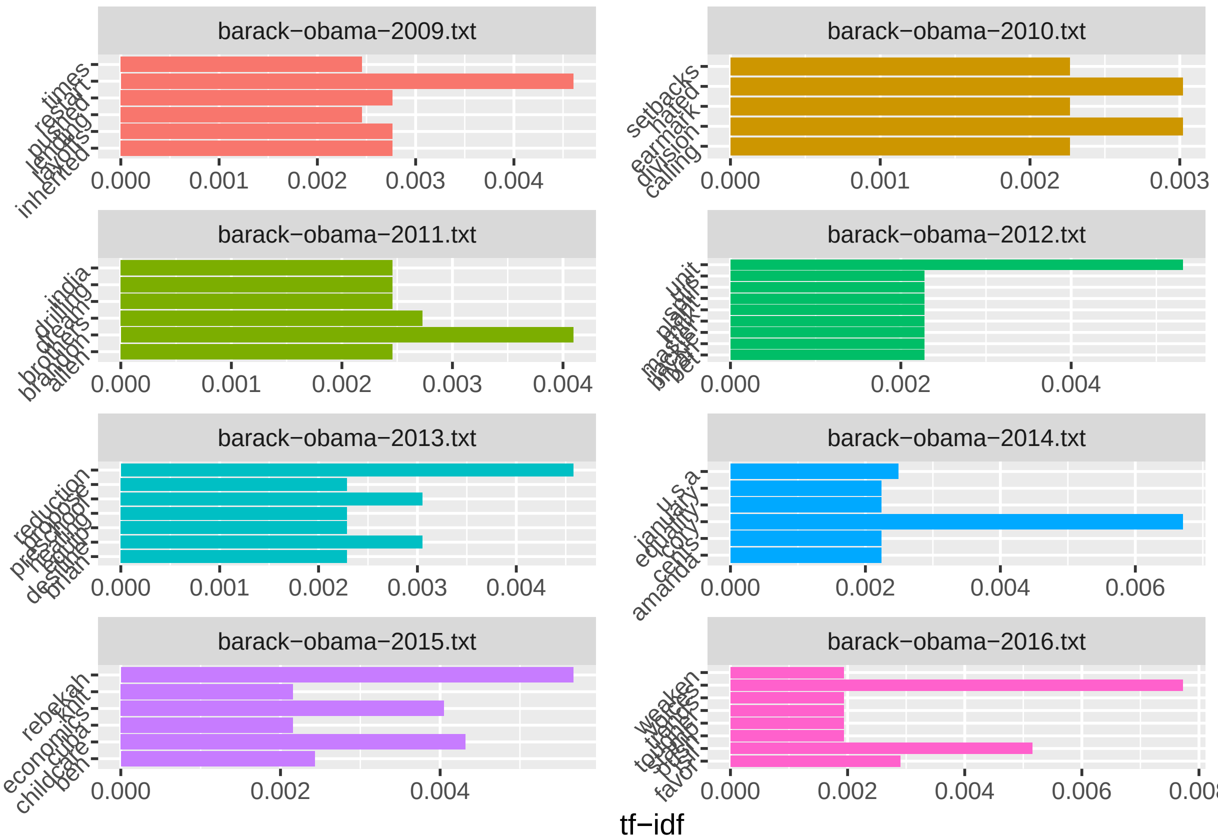
```r
  count(doc_id, word, sort = TRUE) %>%
  bind_tf_idf(word, doc_id, n) %>%
  arrange(desc(tf_idf))

obama_tf_idf
```

```
#> # A tibble: 10,656 x 6
#>    doc_id                word          n      tf   idf  tf_idf
#>    <chr>                 <chr>     <int>   <dbl> <dbl>   <dbl>
#>  1 barack-obama-2016.txt voices        8 0.00372  2.08 0.00773
#>  2 barack-obama-2014.txt cory          9 0.00322  2.08 0.00671
#>  3 barack-obama-2015.txt rebekah       7 0.00273  2.08 0.00567
#>  4 barack-obama-2012.txt unit          7 0.00255  2.08 0.00531
#>  5 barack-obama-2016.txt isil          8 0.00372  1.39 0.00515
#>  6 barack-obama-2009.txt restart       5 0.00221  2.08 0.00460
#>  7 barack-obama-2013.txt reduction     6 0.00220  2.08 0.00458
#>  8 barack-obama-2015.txt childcare     8 0.00312  1.39 0.00432
#>  9 barack-obama-2011.txt brandon       5 0.00197  2.08 0.00409
#> 10 barack-obama-2015.txt economics     5 0.00195  2.08 0.00405
#> # ... with 10,646 more rows
```

Based on what you know of the Obama years and sotu speeches generally, how would you interpret these results?

Let's try graphing these results, showing the top tf-idf terms per speech for Obama's speeches.

```r
obama_tf_idf %>%
  group_by(doc_id) %>%
  mutate(word = factor(word, levels = rev(unique(word)))) %>%
  group_by(doc_id) %>%
  top_n(5) %>%
  ungroup() %>%
  ggplot(aes(word, tf_idf, fill = doc_id)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~doc_id, ncol = 2, scales = "free") +
  coord_flip() +
  theme(axis.text.y = element_text(angle = 45))
```



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 this is what we get back:

```
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 sord 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. This time let's use dplyr's `filter` function for this. And before that we will `separate` the two words into two columns.

```
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)

sotu_bigrams %>%
  count(word1, word2, sort = TRUE)
```

```
#> # A tibble: 129,622 x 3
#>    word1    word2          n
#>    <chr>    <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.

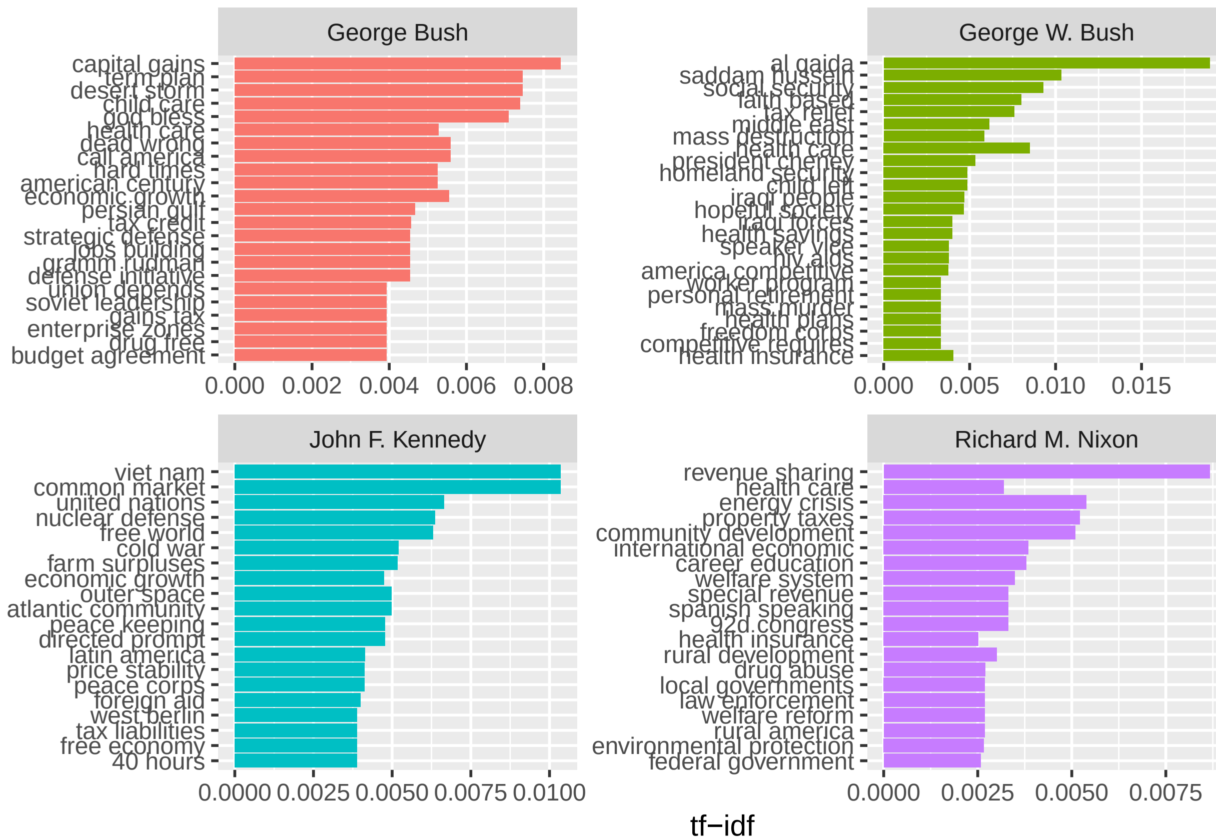
First we will re-unite the two word columns again, and then generate the tf-idf count as above.

```
bigram_tf_idf <- sotu_bigrams %>%
  unite(bigram, word1, word2, sep = " ") %>% # combine columns
  count(president, bigram) %>%
  bind_tf_idf(bigram, president, n) %>%
  arrange(desc(tf_idf))
```

What makes the speeches of different presidents unique?

Let's pick a few presidents and plot their highest scoring tf-idf values here.

```
potus <- c("John F. Kennedy", "Richard M. Nixon", "George Bush", "George W. Bush")

bigram_tf_idf %>%
  filter(president %in% potus) %>%
  group_by(president) %>%
  top_n(20) %>%
  ggplot(aes(reorder(bigram, tf_idf), tf_idf, fill = president)) +
  geom_col(show.legend = FALSE) +
  labs(x = NULL, y = "tf-idf") +
  facet_wrap(~president, scales = "free", nrow = 2) +
  coord_flip()
```



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. It allows us to turn 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 how many times they co-occur. Then we will turn the matrix back into a tidy form, where each row contains the word pairs and the count of their co-occurrence. This lets us count common pairs of words co-appearing within the same speech.

The function which helps us do this is the `pairwise_count()` function.

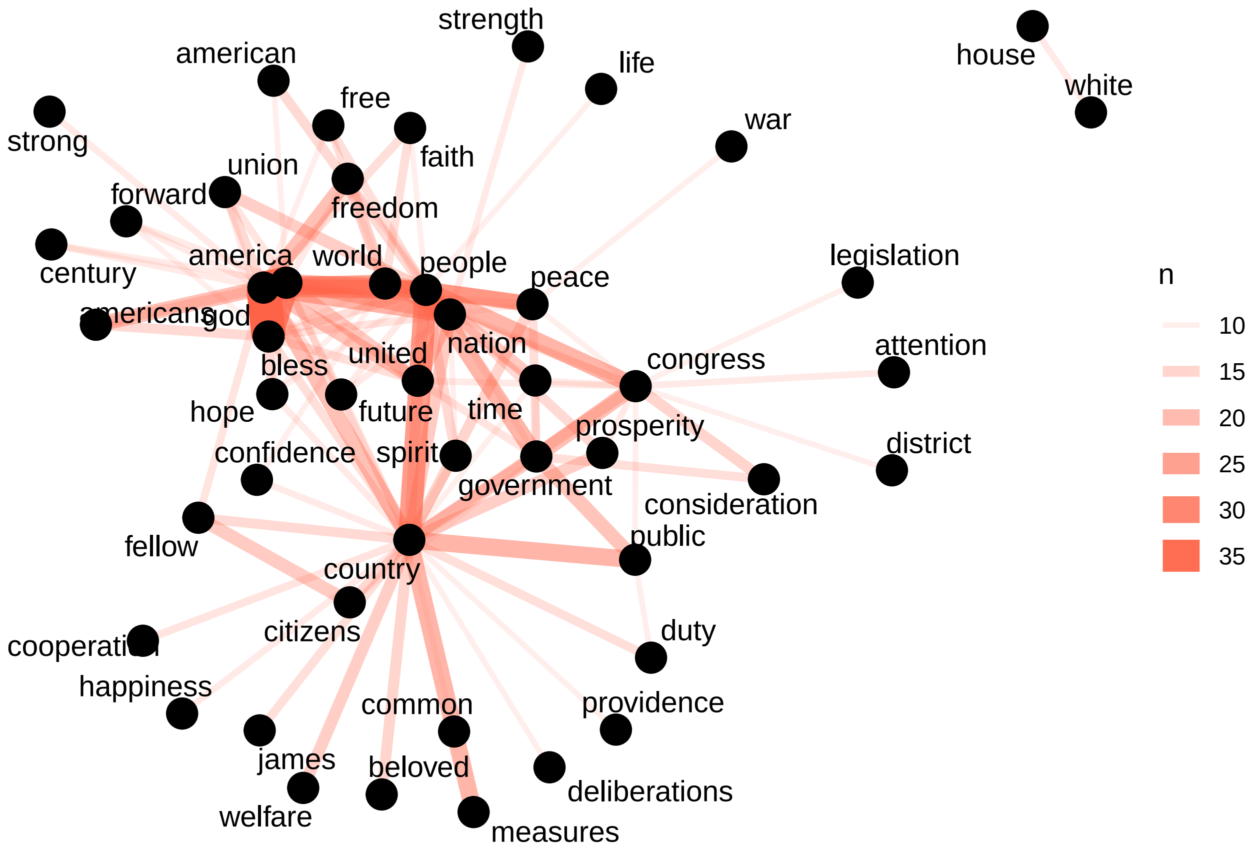
Since processing the entire corpus would take too long here, we will only look at the last 20 words of each speech.

```
library(widyr)

# extract last 100 words from text
sotu_whole$speech_end <- word(sotu_whole$text, -100, end = -1)

sotu_word_pairs <- sotu_whole %>%
  unnest_tokens(word, speech_end) %>%
  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 plot the co-occurrence network, we use the **igraph** library to convert our table into a network graph and **ggraph** which adds functionality to ggplot and makes it easier to create a network plot.

```
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 Re
  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: 27,314 x 4
#>    word        sentiment lexicon score
#>    <chr>       <chr>     <chr>   <int>
#>  1 abacus      trust     nrc        NA
```

```
#>  2 abandon     fear      nrc         NA
#>  3 abandon     negative  nrc         NA
#>  4 abandon     sadness   nrc         NA
#>  5 abandoned   anger     nrc         NA
#>  6 abandoned   fear      nrc         NA
#>  7 abandoned   negative  nrc         NA
#>  8 abandoned   sadness   nrc         NA
#>  9 abandonment anger     nrc         NA
#> 10 abandonment fear      nrc         NA
#> # ... with 27,304 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: 6,788 x 2
#>    word        sentiment
#>    <chr>       <chr>
#>  1 2-faced     negative
#>  2 2-faces     negative
#>  3 a+          positive
#>  4 abnormal    negative
#>  5 abolish     negative
#>  6 abominable  negative
#>  7 abominably  negative
#>  8 abominate   negative
#>  9 abomination negative
#> 10 abort       negative
#> # ... with 6,778 more rows
```

Since this is a regular tibble, we can use these sentiments and join them to the words of our speeches. We will use `inner_join` from `dplyr`. Since our columns to join on have the same name (`word`) we don't need to explicitly name it.

```
tidy_sotu_words %>%
  inner_join(bing_lex) %>% # join
  count(year, sentiment) # group by year and sentiment
```

```
#> # A tibble: 450 x 3
#>     year sentiment     n
#>    <int> <chr>     <int>
#>  1  1790 negative     39
#>  2  1790 positive    125
#>  3  1791 negative     52
#>  4  1791 positive    103
#>  5  1792 negative     57
#>  6  1792 positive     78
#>  7  1793 negative     58
#>  8  1793 positive     72
#>  9  1794 negative    110
#> 10  1794 positive    106
#> # ... with 440 more rows
```

Finally we can visualize it like this:

```
tidy_sotu_words %>%
  inner_join(bing_lex) %>% # join
  count(year, sentiment) %>% # group by year and sentiment
  ggplot(aes(year, n, color = sentiment)) +
    geom_line() +
    scale_x_continuous(breaks = seq(1790, 2016, by = 10)) +
    theme(axis.text.x = element_text(angle = 45, hjust = 1))
```