

Text

Week 12

AEM 2850 / 5850 : R for Business Analytics
Cornell Dyson
Spring 2024

Acknowledgements: **Andrew Heiss**

Announcements

Group project due this Friday, April 19! ([link](#))

Office hours for the rest of this week:

- Tuesday: no office hours from 11:30-12:30
- Wednesday: TA office hours (see canvas)
- Thursday: Prof. Gerarden by appointment at [aem2850.youcanbook.me](#)
- Friday: Prof. Gerarden by appointment at [aem2850.youcanbook.me](#)

We will have a regular lab this week (due Monday 4/22)

- We will have regular TA office hours on Monday 4/22

Questions before we get started?

Plan for today

Prologue

Text mining with R

Time permitting: n-gram ratios

Prologue

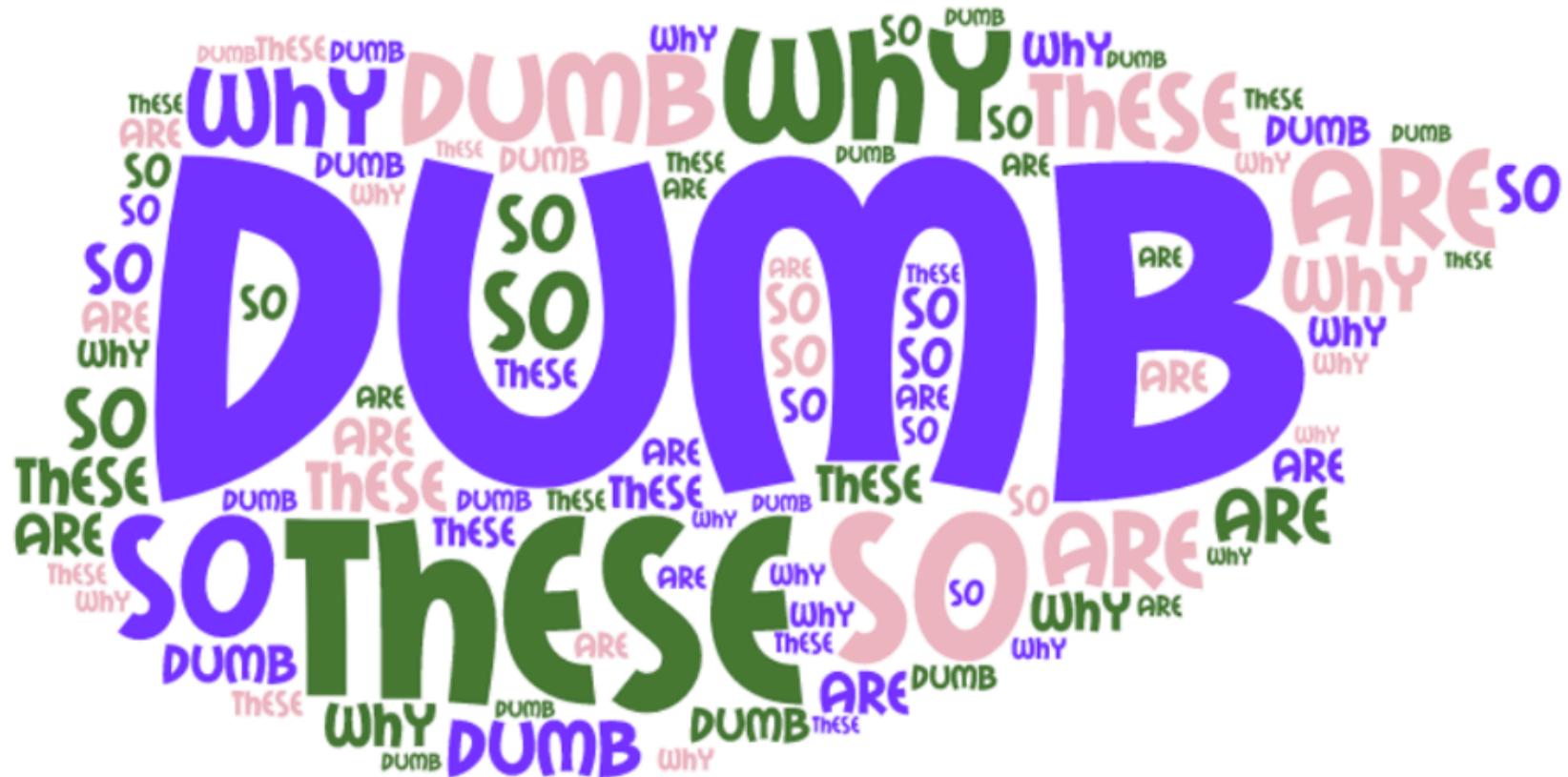
Text comes in many forms

For example: The `schrute` package contains transcripts of **The Office (US)**

```
library(schrute)
theoffice # this is an object from the schrute package

## # A tibble: 55,130 × 12
##   index season episode episode_name director   writer    character text  text_w_direction
##   <int>  <int>  <int> <chr>       <chr>      <chr>     <chr>    <chr> <chr>
## 1     1      1      1 Pilot        Ken Kwapis Ricky G... Michael All ... All right Jim. ...
## 2     2      2      1 Pilot        Ken Kwapis Ricky G... Jim      Oh, ... Oh, I told you...
## 3     3      3      1 Pilot        Ken Kwapis Ricky G... Michael So y... So you've come ...
## 4     4      4      1 Pilot        Ken Kwapis Ricky G... Jim      Actu... Actually, you c...
## 5     5      5      1 Pilot        Ken Kwapis Ricky G... Michael All ... All right. Well...
## 6     6      6      1 Pilot        Ken Kwapis Ricky G... Michael Yes,... [on the phone] ...
## 7     7      7      1 Pilot        Ken Kwapis Ricky G... Michael I've... I've, uh, I've ...
## 8     8      8      1 Pilot        Ken Kwapis Ricky G... Pam      Well... Well. I don't k...
## 9     9      9      1 Pilot        Ken Kwapis Ricky G... Michael If y... If you think sh...
## 10   10     10     1 Pilot        Ken Kwapis Ricky G... Pam      What? What?
## # i 55,120 more rows
## # i 3 more variables: imdb_rating <dbl>, total_votes <int>, air_date <chr>
```

Text can be analyzed in many ways



Take word clouds, the pie chart of text

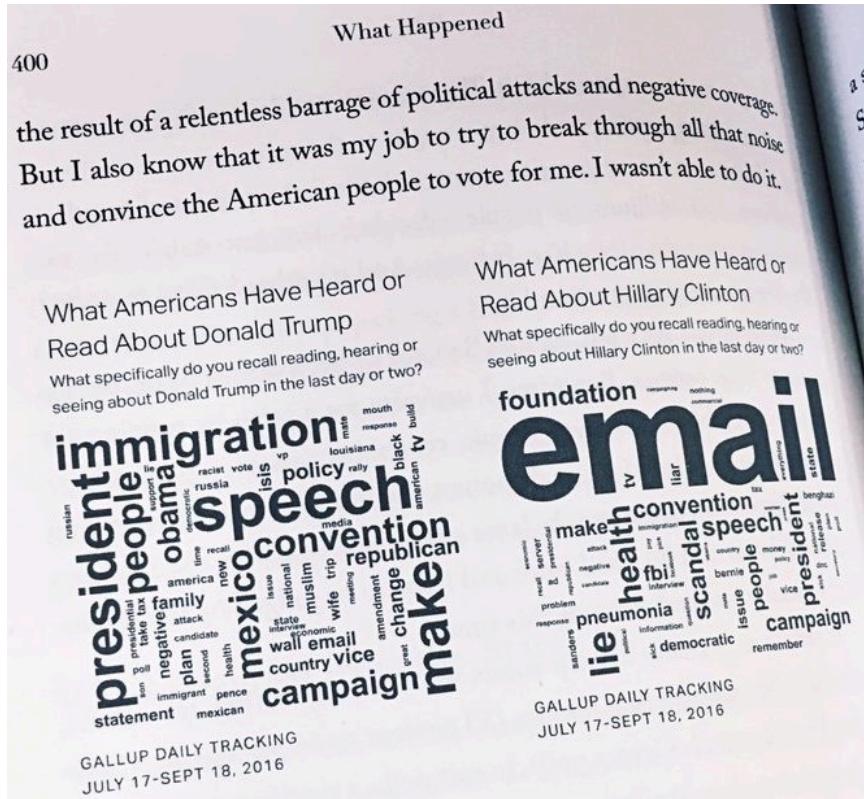
Why are word clouds bad?

- Poor grammar (of graphics)
 - Usually the only aesthetic is size
 - Color, position, etc. contain no content
- Raw word frequency is not always informative

Why are word clouds good?

- Can visualize one-word descriptions
- Can highlight a single dominant word or phrase
- Can make before/after comparisons

Some cases are okay

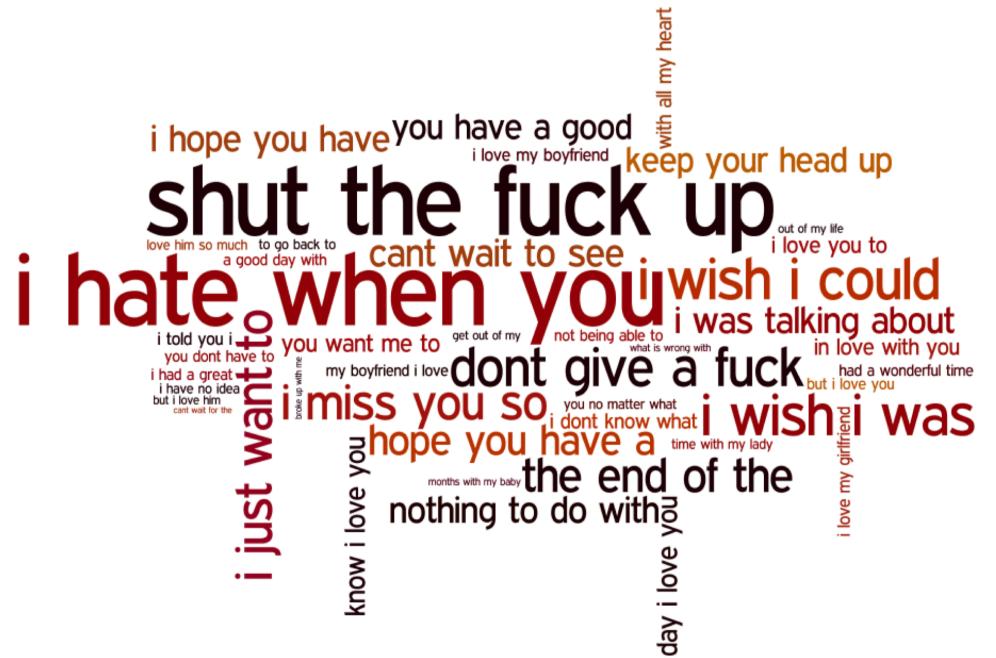


Trump word cloud is uninformative

Clinton word cloud is okay

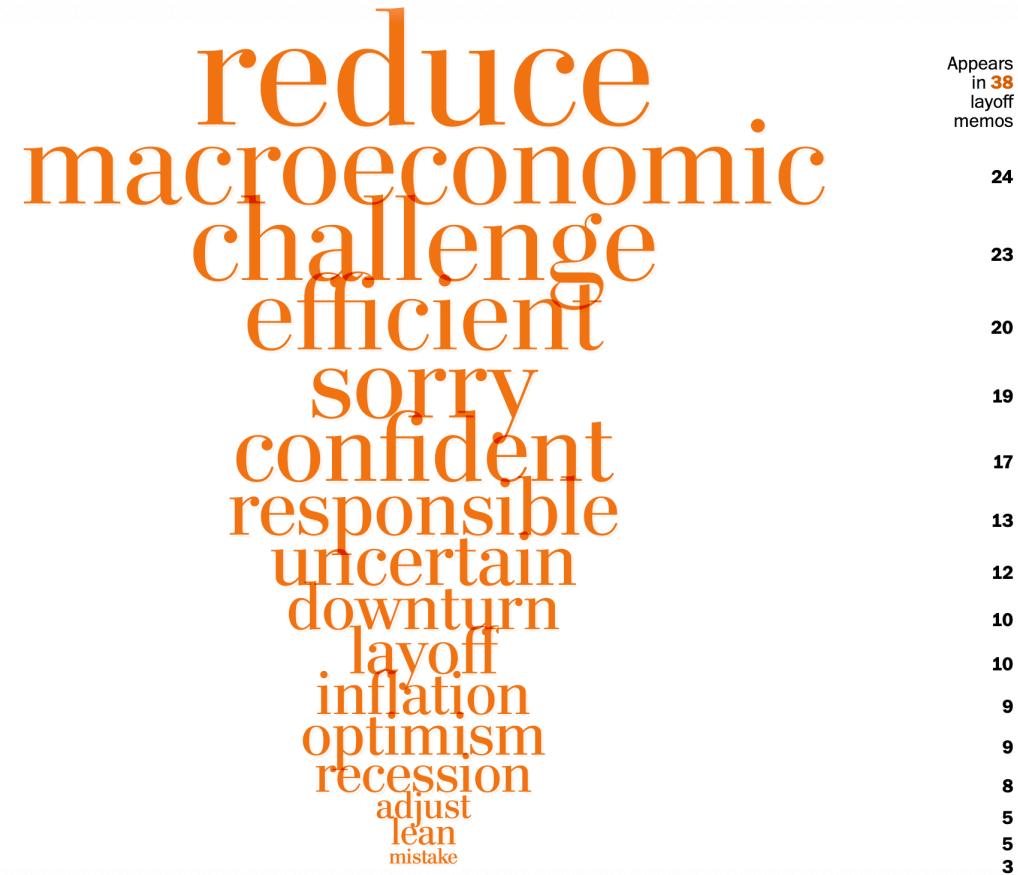
- Highlights **email** as the single dominant narrative about Hillary Clinton prior to the 2016 election

Twitter before and after breakups (4-grams)



What do you think of this word cloud?

Tech CEOs' memos re: layoffs in 2023



What makes this better or worse than a traditional word cloud?

Even better: other methods to visualize text

46 out of 48 layoff memos mentioned the economy



24 memos used the term **macroeconomic**



18 used the words **recession**, **downturn** or **slowdown**



12 reference economic or geopolitical **uncertainty**



10 mentioned **inflation** or **interest rates**

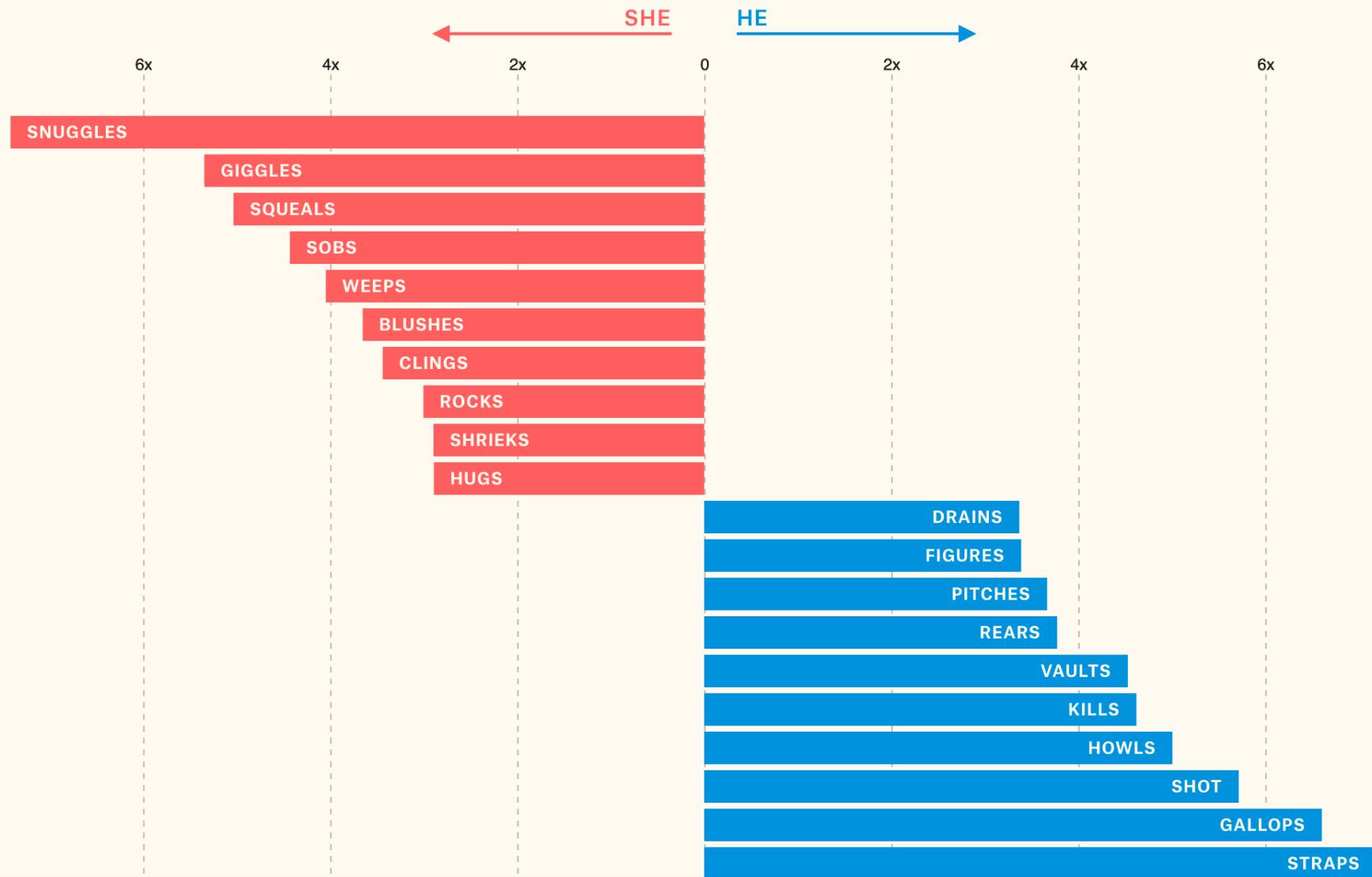


27 out of 48 layoff memos included apologies or taking responsibility for the layoffs

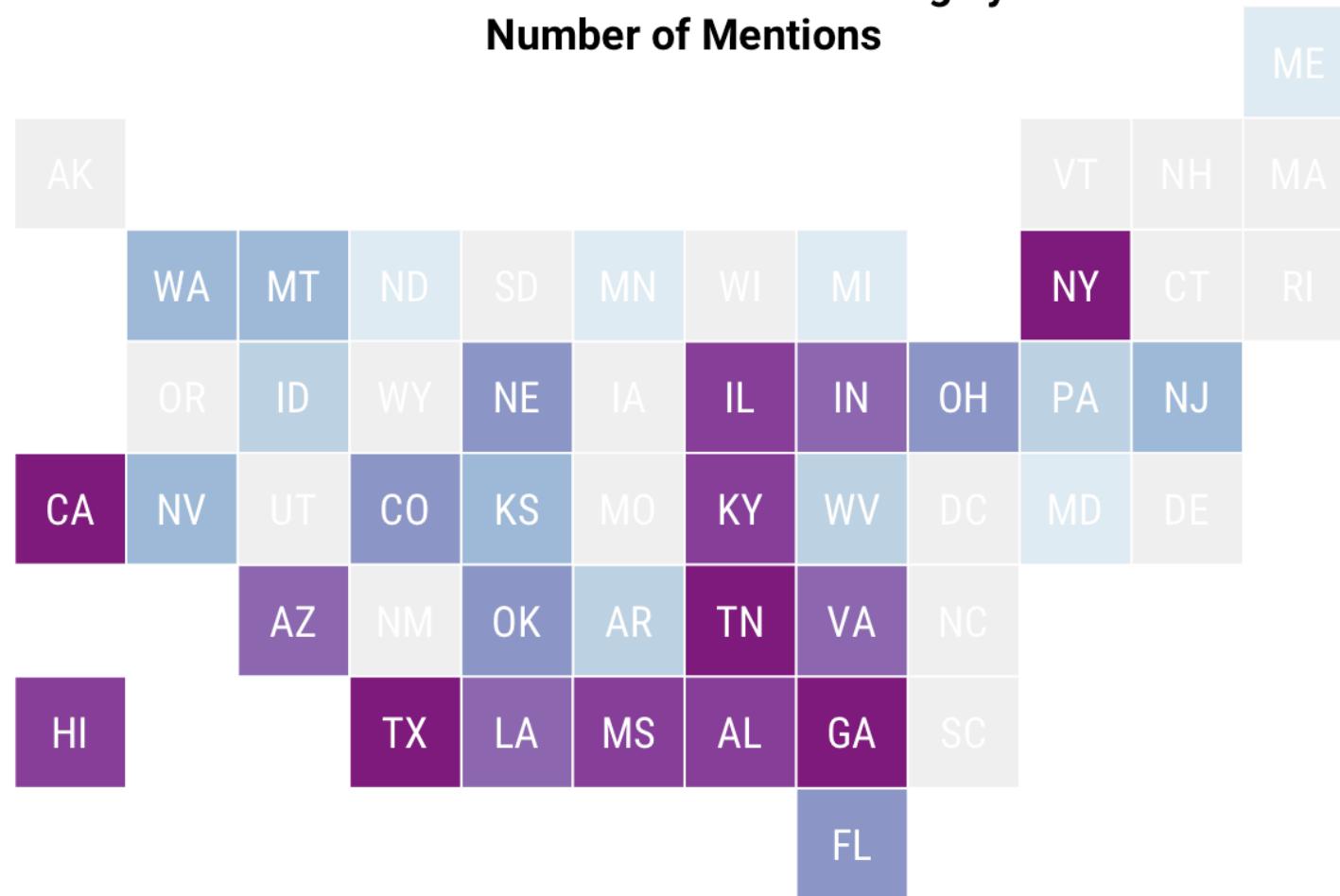


The most used words for women vs. men

Likelihood that certain words appear after “she” vs. “he” in screen direction.



What States Are Mentioned in Song Lyrics? Number of Mentions



Text mining with R

Text mining with R

We already learned about strings and regular expressions in week 5

Many of our data frames include `chr` variables that are useful for filtering and other operations (e.g., stock tickers used in the group project)

This week we will learn more about methods to glean insights from unstructured text (as well as revisit and reinforce what we learned in week 5)

Text mining refers to this process of converting unstructured text to higher-quality information

Core concepts and techniques

Tokens, lemmas, and parts of speech

Sentiment analysis

tf-idf

Topics and LDA

Core concepts and techniques

Tokens, lemmas, and parts of speech

Sentiment analysis

tf-idf

Topics and LDA

We will cover tokens, sentiment analysis, and tf-idf (time permitting)

The tidytext package

We will use the `tidytext` package

`tidytext` brings tidy data concepts
and `tidyverse` tools to text analysis



Let's start with some unstructured text

THE BOY WHO LIVED Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense. Mr. Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large mustache. Mrs. Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbors. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere. The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs. Potter was Mrs. Dursley's sister, but they hadn't met for several years; in fact, Mrs. Dursley pretended she didn't have a sister, because her sister and her good-for-nothing husband were as unDursleyish as it was possible to be. The Dursleys shuddered to think what the neighbors would say if the Potters arrived in the street. The Dursleys knew that the Potters had a small son, too, but they had never even seen him. This boy was another good r...

Data frames can hold unstructured text

As you know, text can be stored in data frames as character strings

Here, each row corresponds to a chapter

```
head(hp1_data)
```

```
## # A tibble: 6 × 2
##   chapter text
##   <int> <chr>
## 1 1 "THE BOY WHO LIVED Mr. and Mrs. Dursley, of number four, Privet Drive, were ...
## 2 2 "THE VANISHING GLASS Nearly ten years had passed since the Dursleys had woke...
## 3 3 "THE LETTERS FROM NO ONE The escape of the Brazilian boa constrictor earned ...
## 4 4 "THE KEEPER OF THE KEYS BOOM. They knocked again. Dudley jerked awake. \"Whe...
## 5 5 "DIAGON ALLEY Harry woke early the next morning. Although he could tell it w...
## 6 6 "THE JOURNEY FROM PLATFORM NINE AND THREE-QUARTERS Harry's last month with t...
```

Tidy text (data)

Tidy text format: a table with one **token** per row

What is a token?

- Any meaningful unit of text used for analysis
- Often words, but also letters, n-grams, sentences, paragraphs, chapters, etc.

The relevant token depends on the analysis you are doing

So the definition of tidy text depends on what you are doing!

Tokenization is the process of splitting text into tokens

Tokenization: words

`tidytext::unnest_tokens` tokenizes **words** by default

- Optionally: characters, ngrams, sentences, lines, paragraphs, etc.

```
hp1_data |>  
  unnest_tokens(    # convert data to tokens  
    input = text,   # split text column  
    output = "word",# make new word column  
  ) |>  
  relocate(word)  # move new column to front
```

Note: `unnest_tokens()` expects **output** before **input** if you don't name arguments

```
## # A tibble: 77,875 × 2  
##       word     chapter  
##       <chr>     <int>  
## 1 the         1  
## 2 boy         1  
## 3 who         1  
## 4 lived       1  
## 5 mr          1  
## 6 and         1  
## 7 mrs         1  
## 8 dursley     1  
## 9 of          1  
## 10 number      1  
## # i 77,865 more rows
```

We can treat tokens like any other data

For example, we can count them:

```
hp1_data |>  
  unnest_tokens(  
    input = text,  
    output = "word",  
  ) |>  
  count(word, sort = TRUE)
```

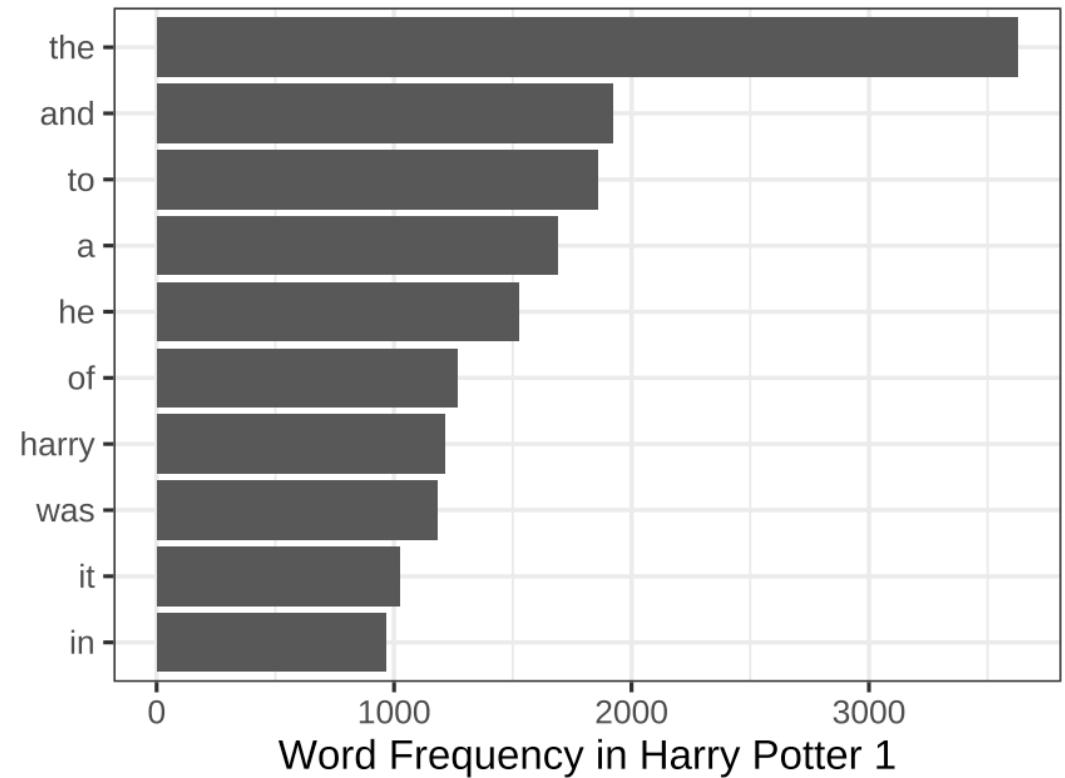
What do you think are the most common words?

```
## # A tibble: 5,978 × 2  
##       word     n  
##       <chr> <int>  
## 1 the     3629  
## 2 and     1923  
## 3 to      1861  
## 4 a       1691  
## 5 he      1528  
## 6 of      1267  
## 7 harry   1213  
## 8 was     1186  
## 9 it      1027  
## 10 in     968  
## # i 5,968 more rows
```

Let's visualize this...

Raw word frequency is not always informative

```
hp1_data |>  
  unnest_tokens("word", text) |>  
  count(word, sort = TRUE) |>  
  slice_max(  
    order_by = n, # order rows by count  
    n = 10        # slice the top 10 rows  
  ) |>  
  ggplot(aes(  
    x = n,  
    y = fct_reorder(word, n)  
  )) +  
  geom_col() +  
  labs(  
    x = "Word Frequency in Harry Potter 1",  
    y = NULL  
  ) +  
  theme_bw()
```



How can we make this better?

Stop words

We can filter out common **stop words** that we want to ignore

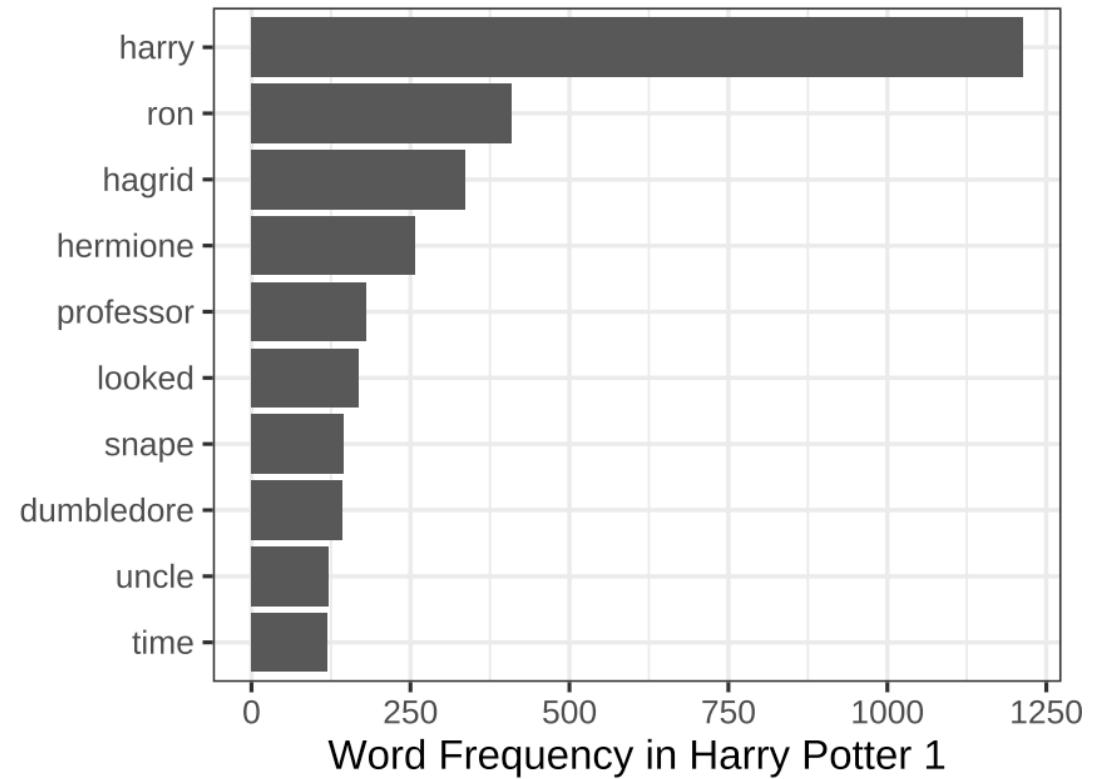
```
stop_words # this is an object from the tidytext package
```

```
## # A tibble: 1,149 × 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
## # i 1,139 more rows
```

How could we remove these stop words from our Harry Potter tokens?

Token frequency: words

```
hp1_data |>  
  unnest_tokens("word", text) |>  
  anti_join(stop_words, join_by(word)) |>  
  count(word, sort = TRUE) |>  
  slice_max(order_by = n, n = 10) |>  
  ggplot(aes(  
    x = n,  
    y = fct_reorder(word, n)  
) +  
  geom_col() +  
  labs(  
    x = "Word Frequency in Harry Potter 1",  
    y = NULL  
  ) +  
  theme_bw()
```



That's better!

Tokenization: n-grams

n contiguous tokens are an **n-gram**

Example: two consecutive words are a bigram

```
hp1_data |>
  unnest_tokens(
    input = text,
    output = "bigram", # call the new column bigram
    token = "ngrams", # we want to fill it with ngrams
    n = 2)           # more specifically, bigrams
  relocate(bigram)
```

```
## # A tibble: 77,858 × 2
##       bigram     chapter
##       <chr>      <int>
## 1 the boy      1
## 2 boy who      1
## 3 who lived    1
## 4 lived mr     1
## 5 mr and       1
## 6 and mrs      1
## 7 mrs dursley  1
## 8 dursley of    1
## 9 of number    1
## 10 number four   1
## # i 77,848 more rows
```

Tokenization: n-grams

How can we remove bigrams that include one or more stop words given what we have learned so far?

```
## # A tibble: 77,858 × 2
##   bigram     chapter
##   <chr>      <int>
## 1 the boy      1
## 2 boy who      1
## 3 who lived    1
## 4 lived mr     1
## 5 mr and       1
## 6 and mrs      1
## 7 mrs dursley  1
## 8 dursley of    1
## 9 of number     1
## 10 number four   1
## # i 77,848 more rows
```

Tokenization: n-grams

How can we remove bigrams that include one or more stop words given what we have learned so far?

```
hp1_data |>  
  unnest_tokens("bigram", text, token = "ngrams", n = 2) |>  
  separate_wider_delim(  
    bigram,                      # remember this?  
    delim = " ",                  # split bigrams  
    names = c("word1", "word2"),   # assign names  
    cols_remove = FALSE          # keep "bigram" col  
  ) |>  
  anti_join(stop_words,           # remove cases w/ stop  
    join_by(word1 == word)) |>  # words in first term  
  anti_join(stop_words,           # remove cases w/ stop  
    join_by(word2 == word)) |>  # words in second term  
  count(bigram, sort = TRUE)     # count
```

One word at a time:

1. get each word
2. filter out stop words

Tokenization: n-grams

What do you think are the most common bigrams in Harry Potter and the Philosopher's Stone, after removing bigrams that include stop words?

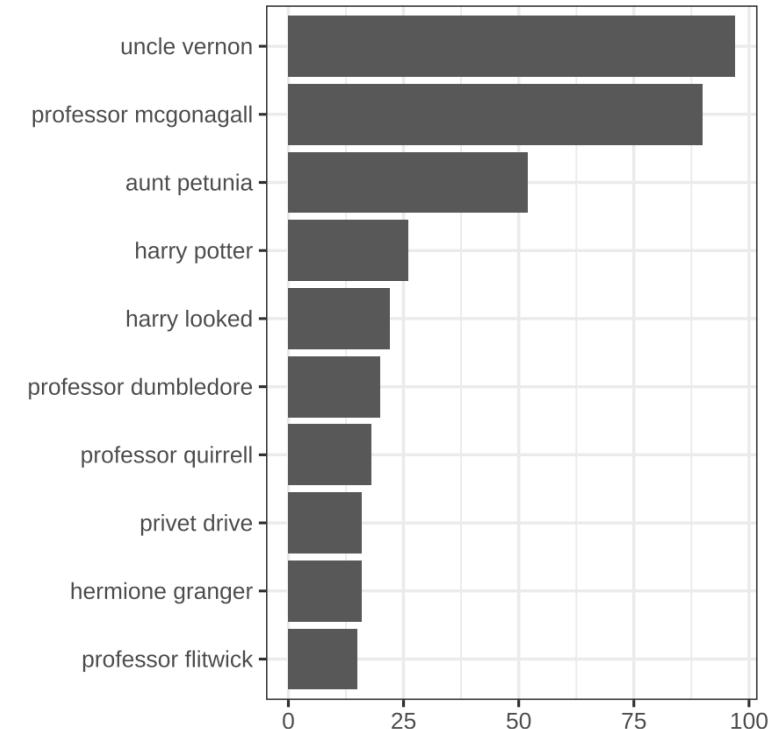
```
hp1_data |>  
  unnest_tokens("bigram", text, token = "ngrams", n = 2) |>  
  separate_wider_delim(  
    bigram,                      # remember this?  
    delim = " ",                  # split bigrams  
    names = c("word1", "word2"),   # assign names  
    cols_remove = FALSE          # keep "bigram" col  
  ) |>  
  anti_join(stop_words,          # remove cases w/ stop  
    join_by(word1 == word)) |>  # words in first term  
  anti_join(stop_words,          # remove cases w/ stop  
    join_by(word2 == word)) |>  # words in second term  
  count(bigram, sort = TRUE)    # count
```

```
## # A tibble: 7,695 × 2  
##       bigram      n  
##       <chr>     <int>  
## 1 uncle vernon      97  
## 2 professor mcgonagall  90  
## 3 aunt petunia      52  
## 4 harry potter      26  
## 5 harry looked      22  
## 6 professor dumbledore  20  
## 7 professor quirrell  18  
## 8 hermione granger    16  
## 9 privet drive       16  
## 10 professor flitwick 15  
## # i 7,685 more rows
```

Token frequency: n-grams

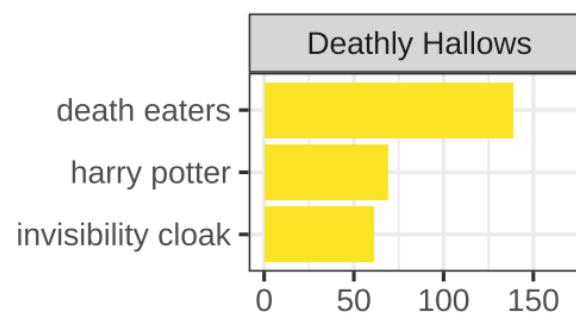
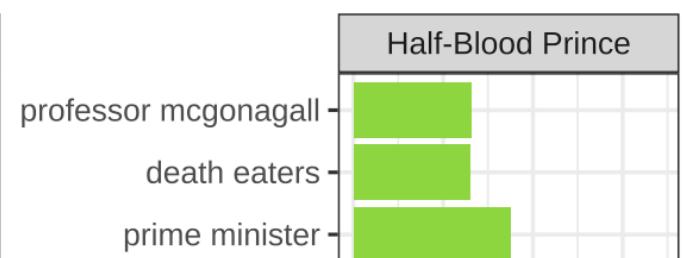
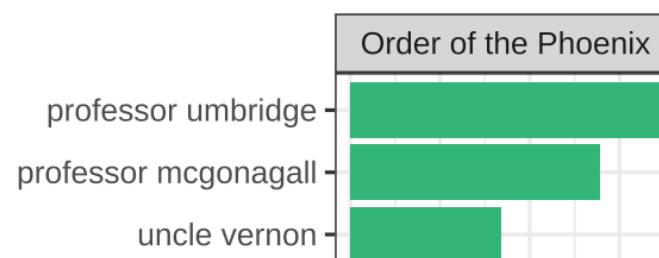
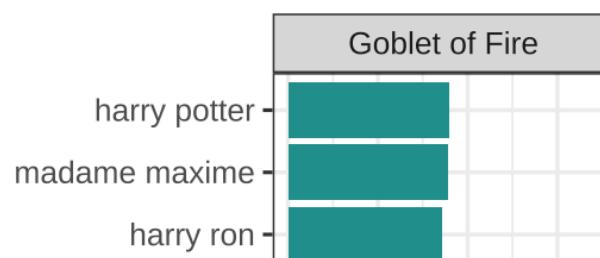
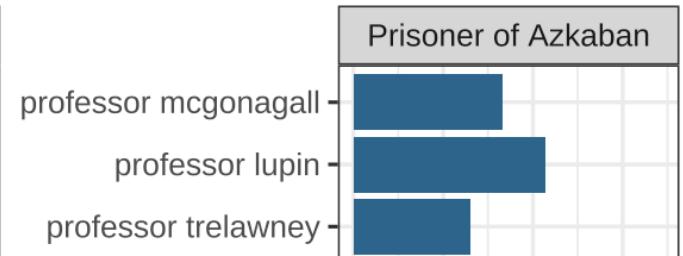
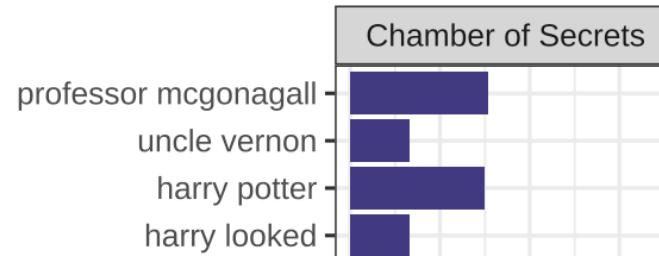
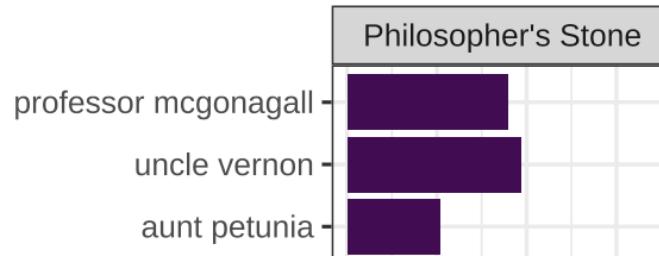
We can visualize the top bigrams in Harry Potter and the Philosopher's Stone:

```
hp1_data |>  
  unnest_tokens("bigram", text, token = "ngrams", n = 2) |>  
  separate_wider_delim(bigram, delim = " ",  
    names = c("word1", "word2"), cols_remove = FALSE) |>  
  anti_join(stop_words, join_by(word1 == word)) |>  
  anti_join(stop_words, join_by(word2 == word)) |>  
  count(bigram) |>  
  slice_max(order_by = n,  
            n = 10) |>  
  ggplot(aes(x = n,  
             y = fct_reorder(bigram, n))) +  
  geom_col() +  
  labs(x = NULL, y = NULL) +  
  theme_bw()
```



Token frequency: n-grams

As always, tools and tricks from the layered grammar of graphics apply:



Sentiment analysis

Simple approach: quantify sentiment of each word in a corpus, then sum them up

```
get_sentiments("bing")
```

```
# A tibble: 6,786 × 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
# i 6,776 more rows
```

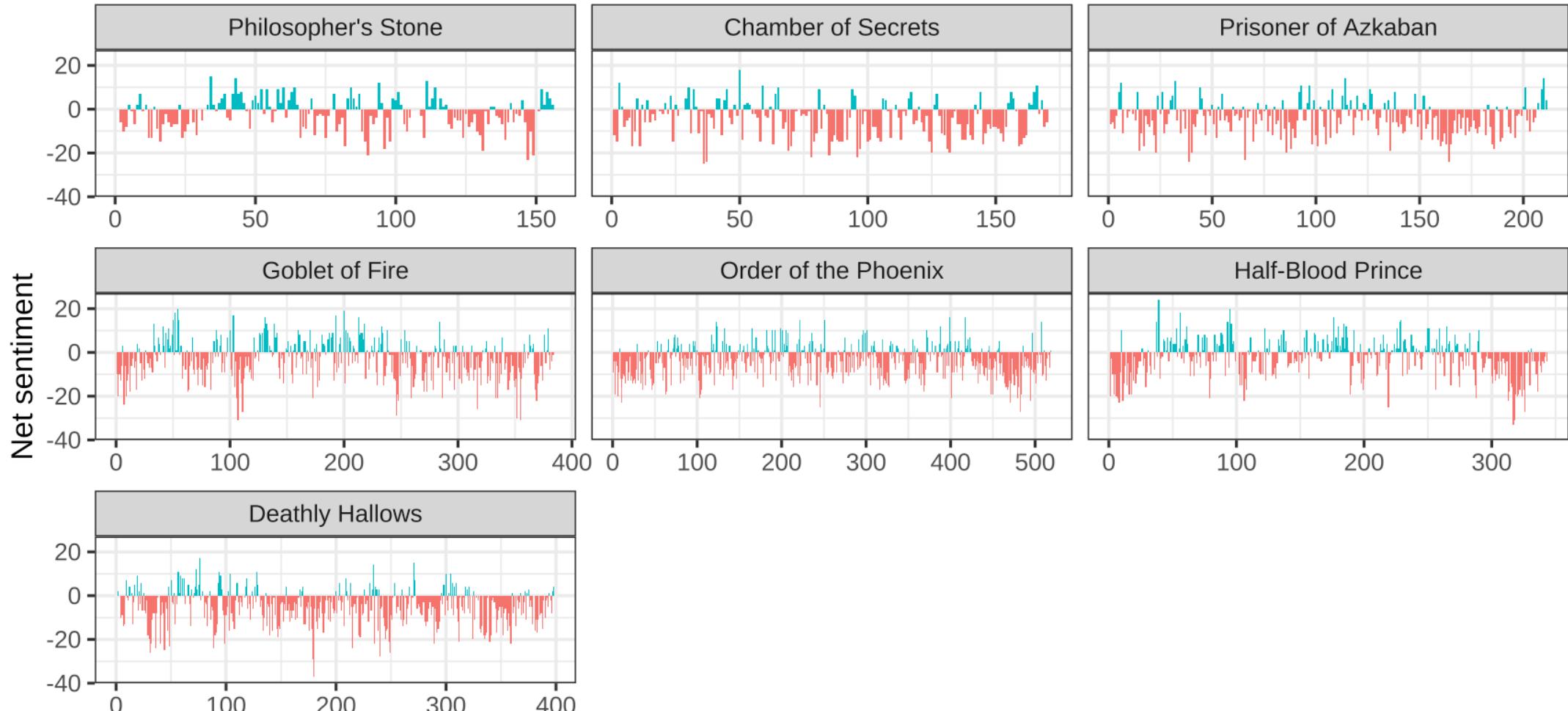
```
get_sentiments("afinn")
```

```
# A tibble: 2,477 × 2
  word      value
  <chr>     <dbl>
1 abandon    -2
2 abandoned  -2
3 abandons   -2
4 abducted   -2
5 abduction  -2
6 abductions -2
7 abhor      -3
8 abhorred   -3
9 abhorrent  -3
10 abhors    -3
# i 2,467 more rows
```

```
get_sentiments("nrc")
```

```
# A tibble: 13,872 × 2
  word      sentiment
  <chr>     <chr>
1 abacus    trust
2 abandon   fear
3 abandon   negative
4 abandon   sadness
5 abandoned anger
6 abandoned fear
7 abandoned negative
8 abandoned sadness
9 abandonment anger
10 abandonment fear
# i 13,862 more rows
```

Harry Potter sentiment analysis



tf-idf

Term frequency-inverse document frequency

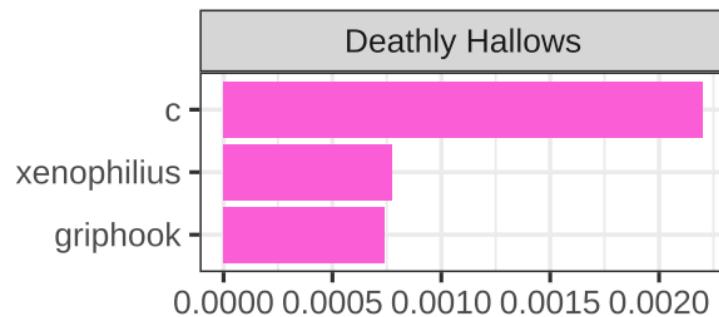
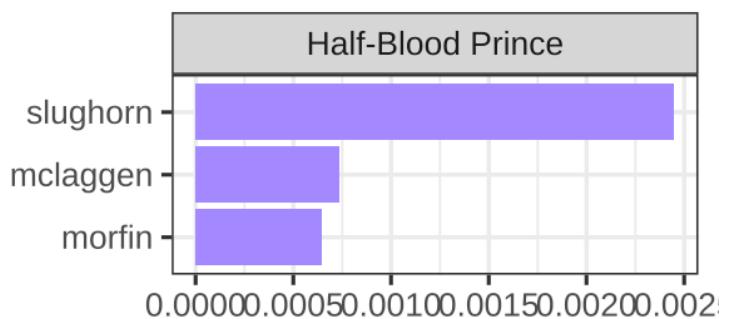
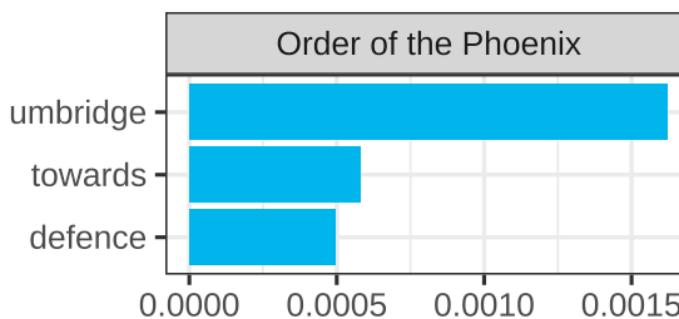
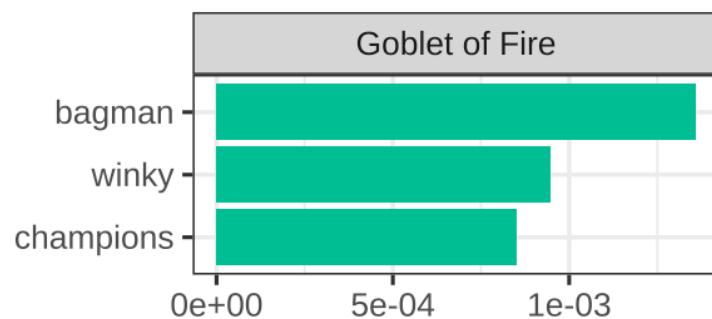
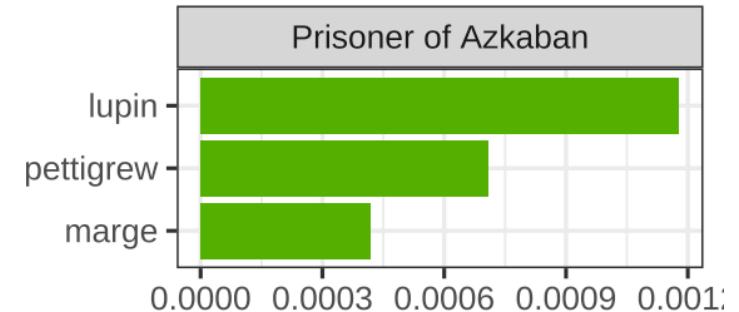
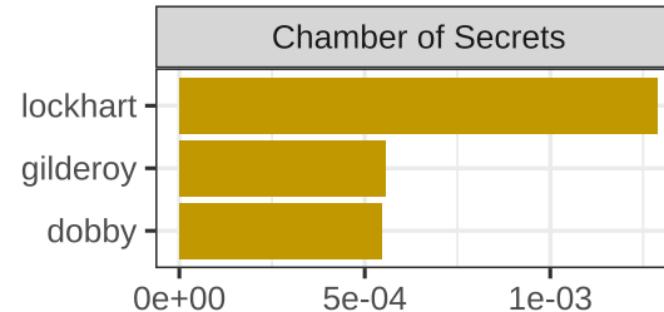
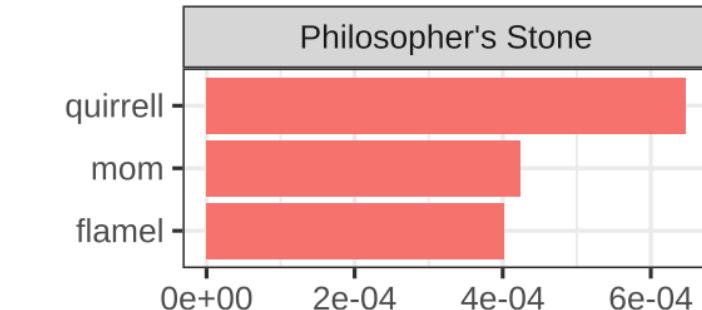
tf-idf is a measure of how important a term is to a document within a broader collection or corpus

$$tf(\text{term}) = \frac{n_{\text{term}}}{n_{\text{terms in document}}}$$

$$idf(\text{term}) = \ln \left(\frac{n_{\text{documents}}}{n_{\text{documents containing term}}} \right)$$

$$tf\text{-}idf(\text{term}) = tf(\text{term}) \times idf(\text{term})$$

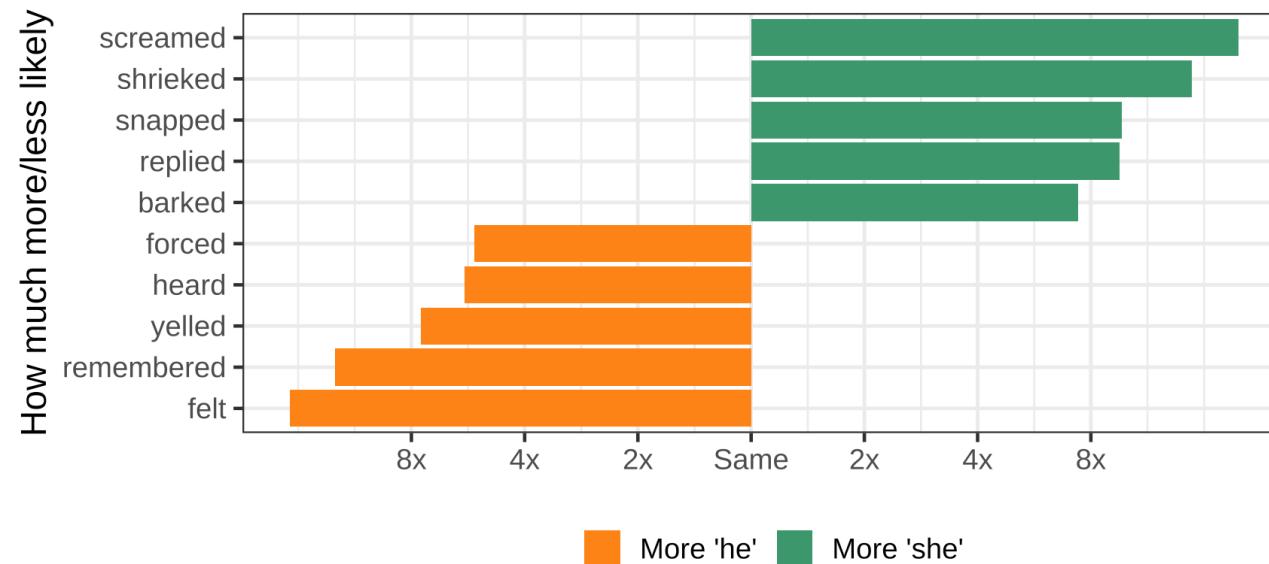
Harry Potter tf-idf



n-gram ratios (time permitting)

Token frequency: n-gram ratios

We can move beyond frequency plots to analyze relative frequencies by groups:



How would you make this?

This is not an endorsement of J. K. Rowling's views on sex and gender

Token frequency: n-gram ratios

1: Identify and separate bigrams

```
bigram_split <- hp |>  
  # tokenize into bigrams  
  unnest_tokens(  
    "bigram",  
    text,  
    token = "ngrams",  
    n = 2  
  ) |>  
  # separate the bigram column into two columns  
  separate_wider_delim(  
    bigram,  
    delim = " ",  
    names = c("word1", "word2"),  
  ) |>  
  select(word1, word2)  
  
bigram_split
```

```
## # A tibble: 1,089,186 × 2  
##       word1     word2  
##       <chr>     <chr>  
## 1 the      boy  
## 2 boy      who  
## 3 who      lived  
## 4 lived    mr  
## 5 mr       and  
## 6 and     mrs  
## 7 mrs     dursley  
## 8 dursley of  
## 9 of      number  
## 10 number four  
## # i 1,089,176 more rows
```

Token frequency: n-gram ratios

2: Filter and count bigrams that begin with "he", "she"

```
bigram_he_she_counts <- bigram_split |>
  # only choose rows where the first word is he or she
  filter(word1=="he" | word1=="she") |>
  # count unique pairs of words
  count(word1, word2, sort = TRUE)
```

```
bigram_he_she_counts
```

```
## # A tibble: 2,096 × 3
##       word1   word2     n
##       <chr>   <chr> <int>
## 1 he      was      2490
## 2 he      had      2138
## 3 he      said     1270
## 4 he      could    925
## 5 she     said     615
## 6 she     was      603
## 7 he      looked   428
## 8 he      did      386
## 9 she     had      385
## 10 he     would    349
## # ... i 2,086 more rows
```

Token frequency: n-gram ratios

3: Pivot the data to the word2 level (and clean up)

```
word_counts <- bigram_he_she_counts |>
  # keep rows where the second word appears 10+ times
  group_by(word2) |>
  filter(sum(n) > 10) |>
  ungroup() |>
  # pivot out the word1 column to columns "he" and "she"
  pivot_wider(names_from = word1, values_from = n) |>
  # remove cases where either he or she is missing
  # (can't divide by 0. note this discards info though!)
  filter(!is.na(he) & !is.na(she))

word_counts
```

```
## # A tibble: 262 × 3
##       word2     he     she
##       <chr>   <int>  <int>
## 1 was      2490    603
## 2 had      2138    385
## 3 said     1270    615
## 4 could    925     64
## 5 looked   428     144
## 6 did      386     58
## 7 would   349     49
## 8 knew     297     20
## 9 felt     285      4
## 10 saw     285     21
## # i 252 more rows
```

Token frequency: n-gram ratios

4: Compute pronoun-specific frequencies for the most common bigrams

```
word_frequencies <- word_counts |>  
  # normalize word2 counts to frequencies by he/she  
  mutate(he = he / sum(he),  
        she = she / sum(she))
```

```
word_frequencies
```

```
## # A tibble: 262 × 3  
##   word2      he     she  
##   <chr>    <dbl>   <dbl>  
## 1 was     0.142  0.146  
## 2 had     0.122  0.0930  
## 3 said    0.0725 0.149  
## 4 could   0.0528 0.0155  
## 5 looked   0.0244 0.0348  
## 6 did     0.0220 0.0140  
## 7 would   0.0199 0.0118  
## 8 knew    0.0169 0.00483  
## 9 felt    0.0163 0.000966  
## 10 saw    0.0163 0.00507  
## # i 252 more rows
```

Token frequency: n-gram ratios

5: Compute log ratios for the most common bigrams

```
log_ratios <- word_frequencies |>
  # compute logged ratio of the she counts to he counts
  # taking logs just helps to compress outliers
  mutate(logratio = log2(she / he)) |>
  # sort by that ratio
  arrange(desc(logratio)) |>
  select(word2, logratio)

log_ratios
```

```
## # A tibble: 262 × 2
##       word2    logratio
##       <chr>     <dbl>
## 1 screamed   4.30
## 2 shrieked  3.89
## 3 snapped    3.27
## 4 replied    3.25
## 5 barked    2.89
## 6 cried      2.43
## 7 sounded   2.08
## 8 say        1.89
## 9 checked    1.82
## 10 jerked   1.82
## # i 252 more rows
```

Token frequency: n-gram ratios

6: Filter the data for plotting

```
plot_word_ratios <- log_ratios |>  
  # take the top 5 logratios by pronoun  
  # (logratios are +/- for she/he)  
  group_by(logratio < 0) |>  
  slice_max(abs(logratio), n = 5) |>  
  ungroup()  
  
plot_word_ratios
```

```
## # A tibble: 10 × 3  
##   word2      logratio `logratio < 0'  
##   <chr>       <dbl> <lgl>  
## 1 screamed    4.30 FALSE  
## 2 shrieked   3.89 FALSE  
## 3 snapped     3.27 FALSE  
## 4 replied     3.25 FALSE  
## 5 barked     2.89 FALSE  
## 6 felt        -4.07 TRUE  
## 7 remembered -3.67 TRUE  
## 8 yelled      -2.92 TRUE  
## 9 heard       -2.53 TRUE  
## 10 forced     -2.44 TRUE
```

Token frequency: n-gram ratios

7: Plot the data

```
plot_word_ratios |>
  ggplot(aes(x = logratio,
              y = fct_reorder(word2, logratio),
              fill = logratio < 0)) +
  geom_col() +
  labs(title = "How much more/less likely",
       x = NULL, y = NULL) +
  scale_fill_manual(name = "",
                    labels = c("More 'she'", "More 'he'"),
                    values = c("#3D9970", "#FF851B"),
                    guide = guide_legend(reverse = TRUE)) +
  scale_x_continuous(breaks = seq(-3, 3),
                     labels = c("8x", "4x", "2x",
                               "Same", "2x", "4x", "8x")) +
  theme_bw(base_size = 20) +
  theme(legend.position = "bottom")
```

