Visualizing Text Data

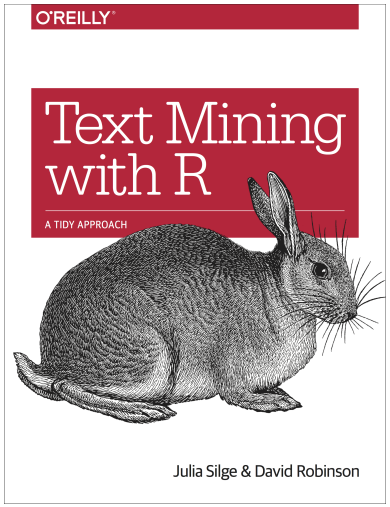
Rei Sanchez-Arias, Ph.D.

Visualizing Text Data: a Tidy Text Approach

Tidy text mining

Tidy data and tidy text format

"Text Mining with R: A Tidy Approach" by Julia Silge and David Robinson



Recall that our definition of **tidy data** is:

- Each variable is a column
- Each observation is a row
- Each type of observational unit is a table

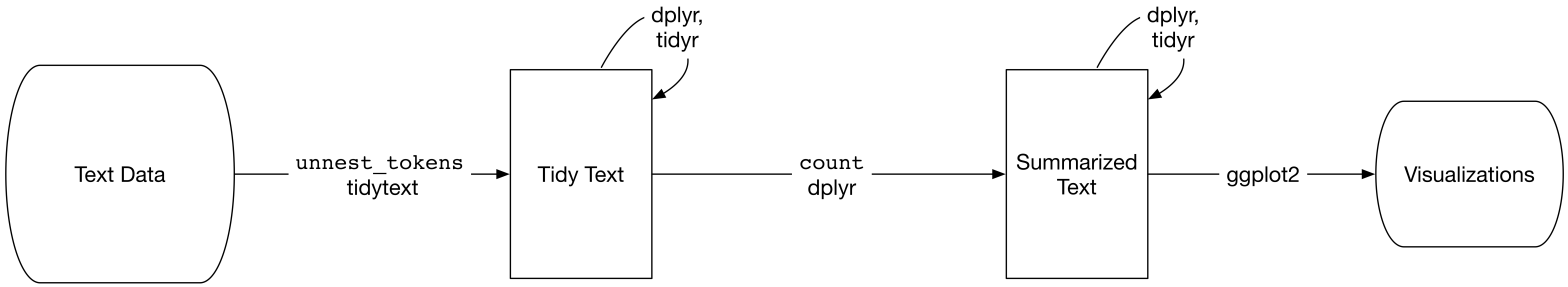
Tidy text format: a table with **one-token-per-row**.

Some core concepts and techniques for working with text data include: tokens, lemmas, parts of speech, sentiment analysis, TF-IDF, topic modeling, and fingerprinting

One-token-per-row

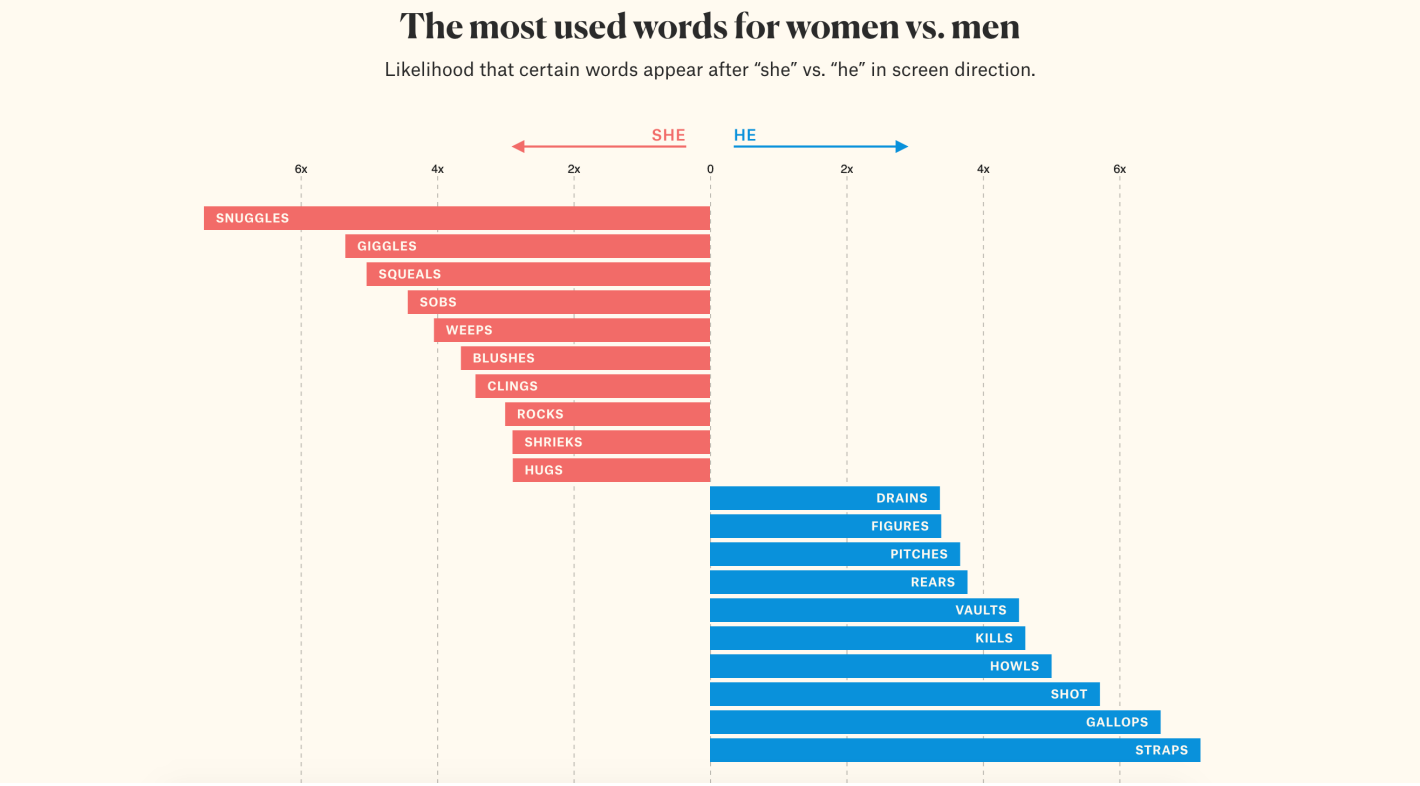
Tidy text format: a table with **one-token-per-row**. A **token** is a meaningful unit of text, such as a word, that we are interested in using for analysis, and **tokenization** is the process of splitting text into tokens. The token that is stored in each row is most often a single word, but can also be an n-gram, sentence, or paragraph.

The `tidytext` package, provides a functionality to tokenize by commonly used units of text and convert to a one-term-per-row format.



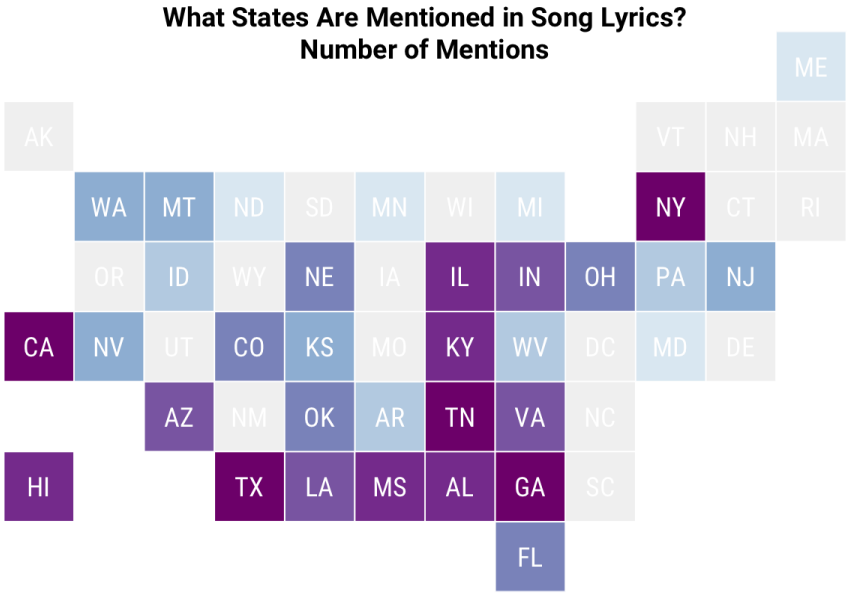
https://www.tidytextmining.com/tidytext.html

Example from pudding.cool



"She Giggles, He Gallops:" Analyzing gender tropes in film with screen direction from 2,000 scripts. By Julia Silge

Song lyrics across the United



"Song Lyrics Across the United States" by Julia Silge

Working with tidy text data

The `unnest_tokens()` function

```
queen <- c(
  "Buddy, you are a boy, make a big noise",
  "Playing in the street, gonna be a big man someday",
  "You got mud on your face, you big disgrace",
  "Kicking your can all over the place, singing"
)
```

This is a typical character vector that we might want to analyze. In order to turn it into a tidy text dataset, we first need to put it into a data frame.

```
library(tidyverse)
text_df <- tibble(line = 1:4, text = q
```

```
## # A tibble: 4 x 2
##    line text
##   <int> <chr>
## 1     1 Buddy, you are a boy, make a big noise
## 2     2 Playing in the street, gonna be a big ma
## 3     3 You got mud on your face, you big disgra
## 4     4 Kicking your can all over the place, sin
```

(a *tibble* has a convenient print method, will not convert strings to factors, and does not use row names)

The `unnest_tokens()` function

```
library(tidytext)
text_df %>%
  unnest_tokens(output = word,
                input = text)
```

```
## # A tibble: 36 x 2
##     line word
##    <int> <chr>
##  1     1 buddy
##  2     1 you
##  3     1 are
##  4     1 a
##  5     1 boy
##  6     1 make
##  7     1 a
##  8     1 big
##  9     1 noise
## 10     2 playing
## # … with 26 more rows
```

The two basic arguments to `unnest_tokens()` used here are column names. First we have the *output column* name that will be created as the text is unnested into it (`word`, in this case), and then the *input column* that the text comes from (`text`, in this case).

Remember that `text_df` above has a column called `text` that contains the data of interest.

Sample tidy text data

One row for each text element: it can be chapter, page, verse, etc.

```
hp <- read_csv("https://github.com/reisanar/datasets/raw/master/hp.csv")
hp <- hp %>% mutate(
  book = factor(book, levels = c("Philosopher's Stone",
                                 "Chamber of Secrets", "Prisoner of Azkaban", "Go
                                 "Order of the Phoenix", "Half-Blood Prince","Dea
head(hp, 6) # sample with text column per chapter
```

```
## # A tibble: 6 x 3
##   book              text                                                chapter
##   <fct>             <chr>                                                 <dbl>
## 1 Philosopher's … "THE BOY WHO LIVED   Mr. and Mrs. Dursley, of numb…       1
## 2 Philosopher's … "THE VANISHING GLASS   Nearly ten years had passed…       2
## 3 Philosopher's … "THE LETTERS FROM NO ONE   The escape of the Brazi…       3
## 4 Philosopher's … "THE KEEPER OF THE KEYS   BOOM. They knocked again…       4
## 5 Philosopher's … "DIAGON ALLEY   Harry woke early the next morning.…       5
## 6 Philosopher's … "THE JOURNEY FROM PLATFORM NINE AND THREE-QUARTER…       6
```

Dataset obtained from https://github.com/bradleyboehmke/harrypotter

Sample tidy text data (cont.)

One row for each text element: it can be chapter, page, verse, etc.

```
hp1_data <- read_csv("https://github.com/reisanar/datasets/raw/master/hp1_data.cs
head(hp1_data, 8) # sample with text column with
```

```
## # A tibble: 8 x 3
##   chapter book                        text
##     <dbl> <chr>                       <chr>
## 1       1 Harry Potter and the Ph… "THE BOY WHO LIVED   Mr. and Mrs. Dursley…
## 2       2 Harry Potter and the Ph… "THE VANISHING GLASS   Nearly ten years h…
## 3       3 Harry Potter and the Ph… "THE LETTERS FROM NO ONE   The escape of …
## 4       4 Harry Potter and the Ph… "THE KEEPER OF THE KEYS   BOOM. They knoc…
## 5       5 Harry Potter and the Ph… "DIAGON ALLEY   Harry woke early the next…
## 6       6 Harry Potter and the Ph… "THE JOURNEY FROM PLATFORM NINE AND THRE…
## 7       7 Harry Potter and the Ph… "THE SORTING HAT   The door swung open at…
## 8       8 Harry Potter and the Ph… "THE POTIONS MASTER   There, look.\"   \"W…
```

Tokens

Split the text into even smaller parts: paragraph, line, verse, sentence, n-gram, word, letter, etc.

```
hp1_words <- hp1_data %>%
  unnest_tokens(word, text) %>%
  mutate(
    book = str_trunc(book,
                     width = 15)) %>%
  select(word, chapter, book) %>%
  head()
```

```
# A tibble: 6 x 3
  word  chapter book
  <chr>   <dbl> <chr>
1 the         1 Harry Potter...
2 boy         1 Harry Potter...
3 who         1 Harry Potter...
4 lived       1 Harry Potter...
5 mr          1 Harry Potter...
6 and         1 Harry Potter...
```

Tokens (cont.)

Split the text into even smaller parts: paragraph, line, verse, sentence, n-gram, word, letter, etc.

```
hp1_bigrams <- hp1_data %>%
  unnest_tokens(bigram, text,
                token = "ngrams", n =
  mutate(
    book = str_trunc(book,
                     width = 15)) %>%
  select(bigram, chapter, book) %>%
  head()
```

```
## # A tibble: 6 x 3
##   bigram    chapter book
##   <chr>       <dbl> <chr>
## 1 the boy         1 Harry Potter..
## 2 boy who         1 Harry Potter..
## 3 who lived       1 Harry Potter..
## 4 lived mr        1 Harry Potter..
## 5 mr and          1 Harry Potter..
## 6 and mrs         1 Harry Potter..
```

Stop words

Common words that we can generally ignore

```
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
```

Use the `stopwords()` function to find a list of common stop-words in different languages

```
stopwords::stopwords(language = "engl
  sample(6)
```

```
## [1] "she's"  "that's" "few"    "why
```

```
# another sample
stopwords::stopwords(language = "span
  sample(6)
```

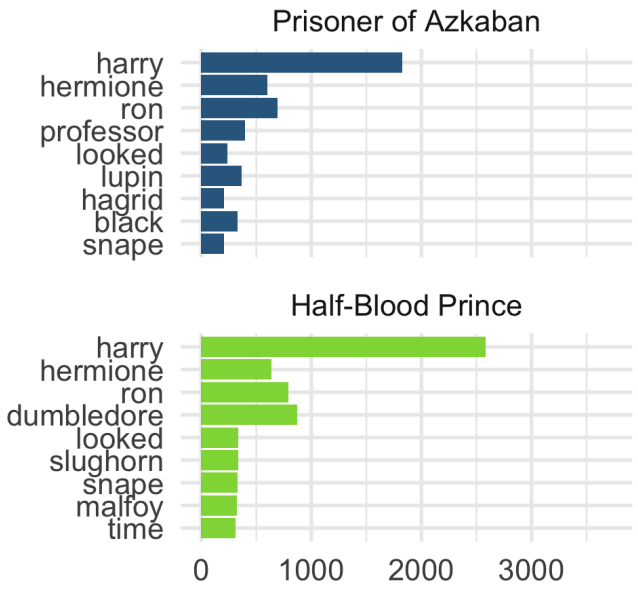
```
## [1] "ella"     "habríais" "esto"
```

Counting words

```
hp_tokens <- hp %>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words, by = "word") %>% # remove stopwords
  group_by(book) %>%
  count(word, sort = TRUE) %>%
  top_n(9, n) %>%
  ungroup() %>%
  mutate(word = fct_inorder(word))

# create a bar plot showing the token frequency
ggplot(hp_tokens, aes(x = n, y = fct_rev(word), fill = book)) +
  geom_col() +
  guides(fill = FALSE) +
  labs(x = NULL, y = NULL) +
  scale_fill_viridis_d() +
  facet_wrap(vars(book), scales = "free_y") +
  theme_minimal()
```

Counting words (cont.)

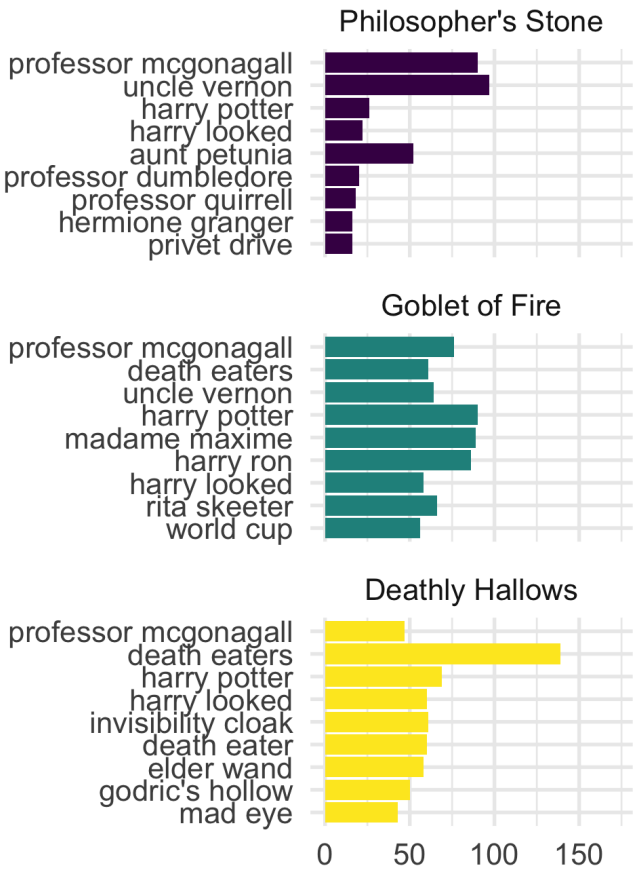


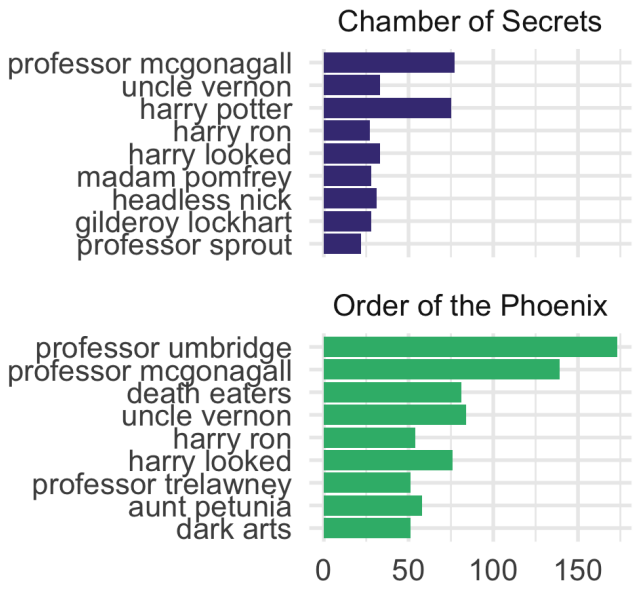
Token frequency: n-grams

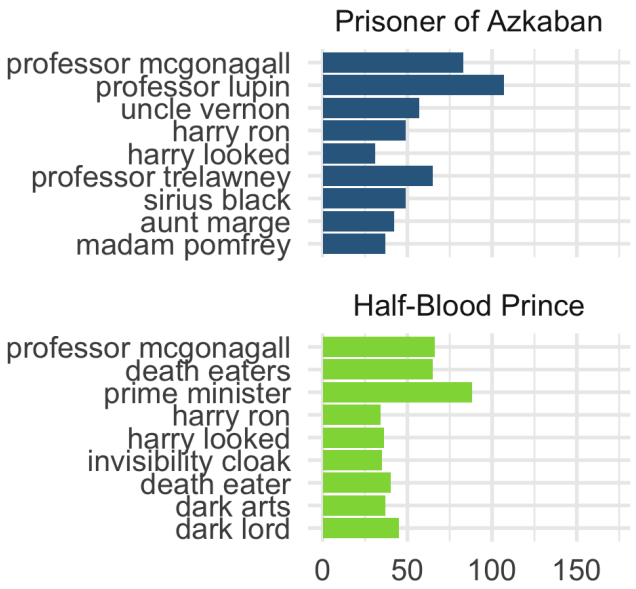
```
hp_bigrams <- hp %>%
  unnest_tokens(bigram, text, token = "ngrams", n = 2) %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%
  filter(!word1 %in% stop_words$word) %>% # remove stopwords
  filter(!word2 %in% stop_words$word) %>% # remove stopwords
  unite(bigram, word1, word2, sep = " ") %>%
  group_by(book) %>%
  count(bigram, sort = TRUE) %>%
  top_n(9, n) %>%
  ungroup() %>%
  mutate(bigram = fct_inorder(bigram))

# create a bar plot showing the token frequency (bigrams)
ggplot(hp_bigrams, aes(x = n, y = fct_rev(bigram), fill = book)) +
  geom_col() +
  guides(fill = FALSE) +
  labs(x = NULL, y = NULL) +
  scale_fill_viridis_d() +
  facet_wrap(vars(book), scales = "free_y") +
  theme_minimal()
```

Token frequency: n-grams (cont.)







Sentiment Analysis

Sentiment analysis

- **Sentiment analysis** can be thought of as the exercise of taking a sentence, paragraph, document, or any piece of natural language, and **determining whether that text's emotional tone is positive, negative or neutral**.

- One way to analyze the sentiment of a text is to consider the text as a *combination of its individual words* and the sentiment content of the whole text as the **sum of the sentiment content of the individual words**.

- When human readers approach a text, we use our understanding of the *emotional intent* of words to infer whether a section of text is positive or negative, or perhaps characterized by some other more nuanced emotion like surprise or disgust.

See also:
https://www.reisanar.com/slides/sentiment-analysis#1

https://www.reisanar.com/slides/n-grams#1

Lexicons

The `tidytext` package contains `sentiments` dataset. Three general-purpose lexicons are included:

- `AFINN` from Finn Arup Nielsen: scores a word with a number, which may range from -5 to +5.
- `bing` from Bing Liu and collaborators: scores a word as either positive or negative.
- `nrc` from Saif Mohammad and Peter Turney: categorizes a word under sentiment type categories such as positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust.

These sentiment lexicons, based on unigrams (i.e. single words) are derived from a single English word and are assigned different scores of positive/negative sentiments.

Lexicons (cont.)

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

```
# 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
```

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

```
# A tibble: 2,477 x 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
# … with 2,467 more rows
```

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

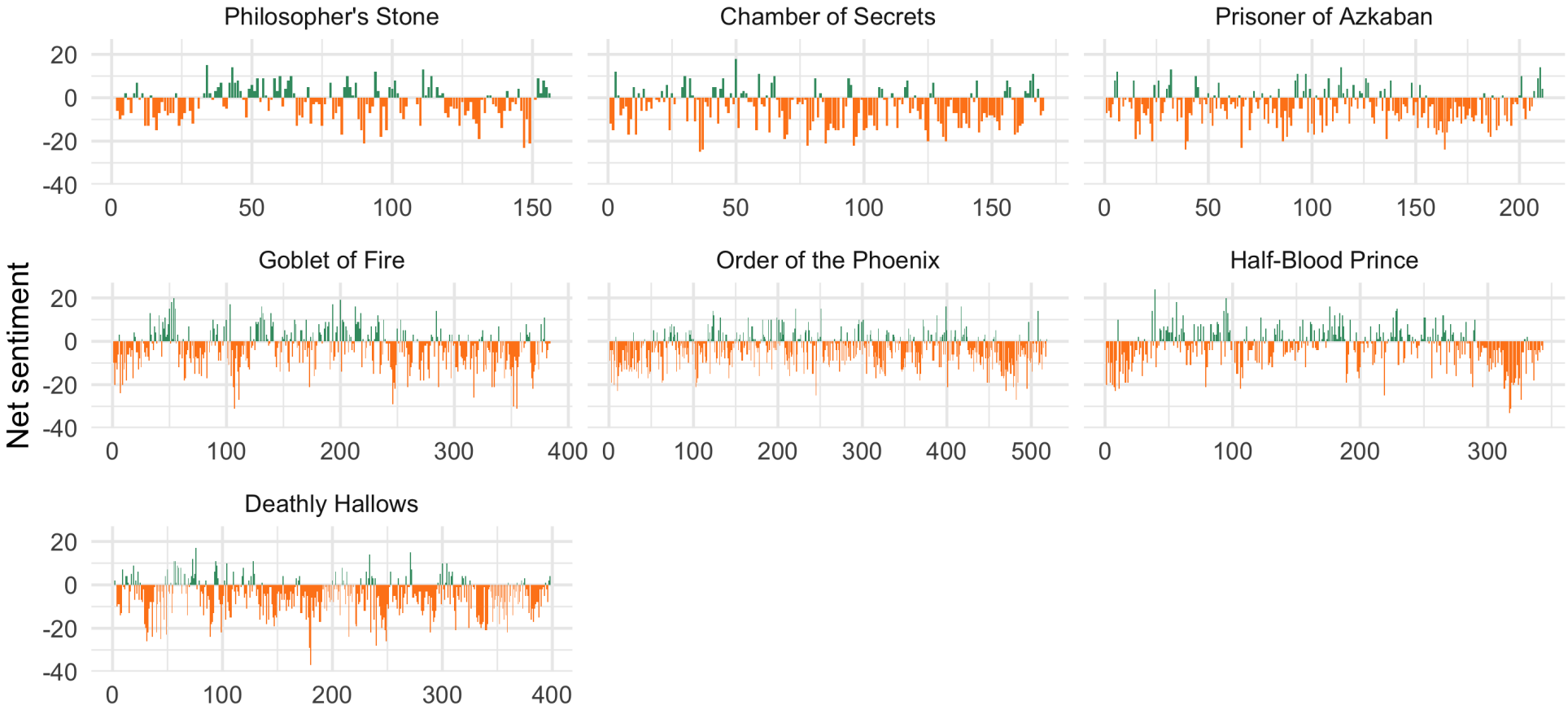
```
# A tibble: 13,901 x 2
   word        sentiment
   <chr>       <chr>
 1 abacus      trust
 2 abandon     fear
 3 abandon     negative
 4 abandon     sadness
 5 abandoned   anger
 6 abandoned   fear
 7 abandoned   negative
 8 abandoned   sadness
 9 abandonment anger
10 abandonment fear
# … with 13,891 more rows
```

Sentiment analysis from books

```
# See : https://rstudio-pubs-static.s3.amazonaws.com/300624_8260952d1f0346969e65
hp_sentiment <- hp %>%
  unnest_tokens(word, text) %>%
  group_by(book) %>%
  mutate(word_count = 1:n(),
         index = word_count %/% 500 + 1) %>%
  inner_join(get_sentiments("bing")) %>%
  count(book, index = index, sentiment) %>%
  pivot_wider(names_from = sentiment, values_from = n) %>%
  mutate(net_sentiment = positive - negative)

ggplot(hp_sentiment,
       aes(x = index, y = net_sentiment, fill = net_sentiment > 0)) +
  geom_col() +
  guides(fill = FALSE) +
  labs(x = NULL, y = "Net sentiment") +
  scale_fill_manual(name = "", labels = c("Positive", "Negative"),
                    values = c("#FF851B", "#3D9970")) +
  facet_wrap(vars(book), scales = "free_x") +
  theme_minimal()
```

Sentiment analysis from books



TF-IDF

For a given document d and term t, the **term frequency** is the number of times term t appears in document d :

$$TF(d,t) = \#\text{ times } t \text{ appears in document } d$$

To account for terms that appear frequently in the domain of interest, we compute the **Inverse Document Frequency** of term t, calculated over the entire corpus and defined as

$$IDF(t) = \ln\left(\frac{\text{total number of documents}}{\#\text{ documents containing term } t}\right)$$

TF-IDF is high where a rare term is present or frequent in a document TF-IDF is near zero where a term is absent from a document, or abundant across all documents

$$TF\text{-}IDF = TF \times IDF$$

TF-IDF (cont.)

$$TF - IDF(t, d) = TF(t, d) \times IDF(t) = TF(t, d) \times \ln\left(\frac{n_{\text{docs}}}{n_{\text{docs}} \text{ containing term } t}\right)$$

- If term t appears in few documents then $IDF(t, d)$ increases

TF-IDF measures how *important* is a word in a collection of documents

- TF-IDF is large when a rare terms is frequent in a document.
- TF-IDF is close to zero when term is absent from documents, or term is abundant across all documents.

(1) (2)

TF-IDF ≈ 0 → TF ~ 0 (1) or/and IDF ~ 0 (2)

TF-IDF example

```
# Get a list of words in all the books
hp_words <- hp %>%
  unnest_tokens(word, text) %>%
  count(book, word, sort = TRUE) %>%
  ungroup()

# Add the tf-idf for these words
hp_tf_idf <- hp_words %>%
  bind_tf_idf(word, book, n) %>%
  arrange(desc(tf_idf))

# Get the top 8 uniquest words
hp_tf_idf_plot <- hp_tf_idf %>%
  group_by(book) %>%
  top_n(8) %>%
  ungroup() %>%
  mutate(word = fct_inorder(word))

ggplot(hp_tf_idf_plot, aes(y = fct_rev(word), x = tf_idf, fill = book)) +
  geom_col() +
  guides(fill = FALSE) +
  labs(y = "tf-idf", x = NULL) +
  facet_wrap(~ book, scales = "free") +
  theme_minimal()
```

TF-IDF example (cont.)