Text mining with Tidy Text

First things first: Instructions

- You can find the exercises – with code to complete - in the RMarkdown file called tidytext_exercises
- The PPT slide titles corresponding to the exercises are marked with # in **BLUE**
- You can find the coding solutions to the exercises in the RMarkdown file called tidytext_codingsolutions
- There are also solution slides, titled tidytext_solutions. Those include code, but also the output of your analyses, like tibbles and visualizations. Interpretive questions encountered on the slides are also answered here.

Part 1: The tidy text format

- Text mining based on tidy data principles

- The specific structure of tidy data:
  - Each variable is a column
  - Each observation is a row
  - Each type of observational unit is a table

The tidy text format = a table with one-token-per row

What is a token when we text mine?

- A meaningful unit of text, like…
  - A single word
  - An n-gram
  - A sentence
  - A paragraph

A token is a meaningful unit of text, most often a word, that we are interested in using for further analysis, and tokenization is the process of splitting text into tokens.

Tidy format a text corpus I

Exercise 7

Let’s try and turn this poem by Emily Dickinson into a tidy text dataset!

First, install the Tidyverse packages you need during the afternoon session; they are listed in your RStudio environment in the second code block (this might take a while).

7a. Now let’s start tidy-texting Dickinson’s poem. Run the following lines of code:

```
text <- c("Because I could not stop for Death -",
          "He kindly stopped for me -",
          "The Carriage held but just Ourselves -",
          "and Immortality")
```

← The output you will get 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.

Tidy format a text corpus II

7b. Let’s call on a package from the Tidyverse that will give us the right data frame: dplyr. You can call on this package by running the following code:

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

← The output you will get is a data frame we call a ‘tibble’, which is great for use with tidy tools.

Tidy format a text corpus III

We now need to ensure that we can filter out words or count which occur most frequently, since each row is made up of multiple combined words. We need to convert this so that it has **one-token-per-document-per-row**.

A token is a meaningful unit of text, most often a word, that we are interested in using for further analysis, and tokenization is the process of splitting text into tokens.

Tidy format a text corpus IV

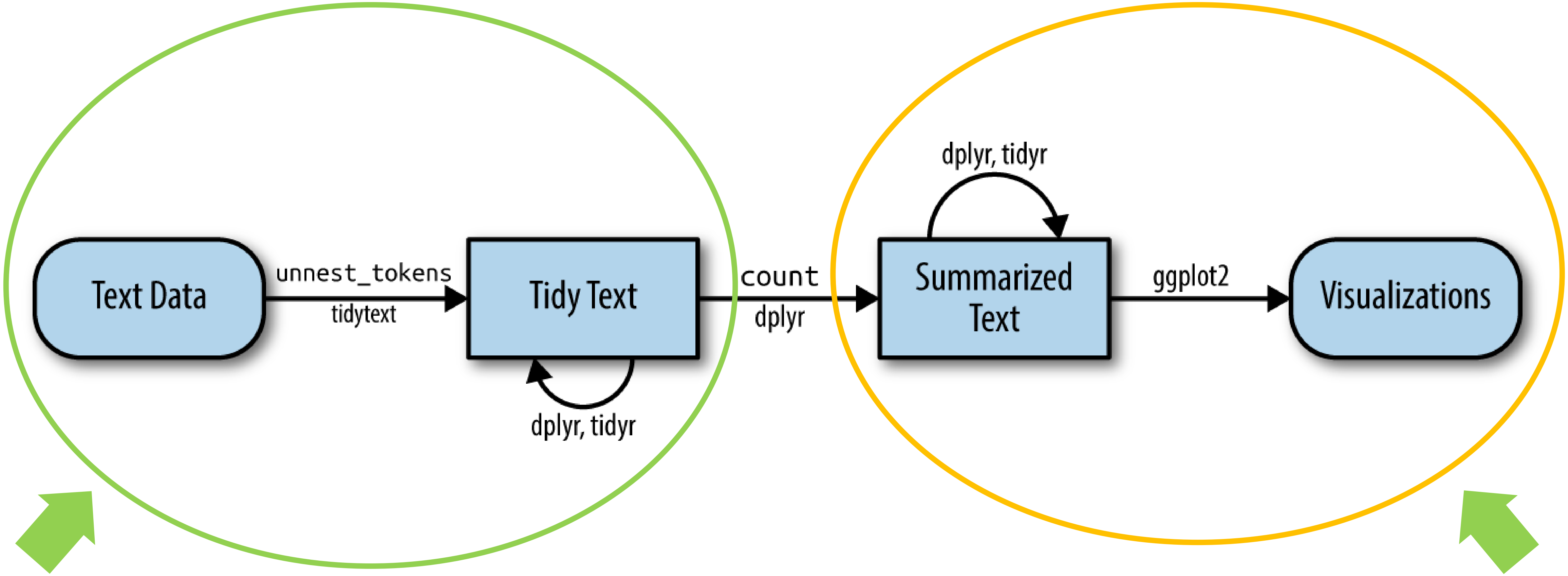
7c. We will now break the text into individual tokens (tokenization) *and* transform it to a tidy data structure. To do this, call on the unnest_tokens() function:

```
library(tidytext)

text_df %>%
  unnest_tokens(word, text)
```

Now check the output! Do you see a simple table with line numbers (left) and words (right)? Then you have tokenized and structured your data for text mining within the Tidyverse 👍

What can we do with Tidy text data?



Been there, done that! …But can you do it again? We will see in a minute!

There is still more fun and useful stuff to learn, like counting words and visualizing your findings. We will get there soon…

Tidying Jane Austen’s novels I

Exercise 8

Let’s move from one short poem to six novels and transform them into a tidy format. We will use the janeaustenr package, which provides all of these texts in a one-row-per-line format, where a line in this context is analogous to a literal printed line in a physical book.

8a. Based on the previous exercises with Dickenson’s poem, are you now able to call on the janeaustenr package, as well as the dplyr and stringr packages needed for your analysis?

Tidying Jane Austen’s novels II

We will now go on to use mutate() to annotate a linenumber quantity to keep track of lines in the original format and a chapter (using a regex) to find where al the chapters are.

8b. Run the following code… and then challenge yourself with exercise 8c on the next slide!

```
original_books <- austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text,
                                     regex("^chapter [\\divxlc]",
                                           ignore_case = TRUE)))) %>%
  ungroup()

original_books
```

Tidying Jane Austen’s novels III

8c. You are now ready to tokenize your dataset again! Can you fill in the blanks in this next piece of code so that we will get a **one-token-per-row** format for Austen’s novels?

```
library(???)
tidy_books <- original_books %>%
  ???_???(word, text)

tidy_books
```

Tidying Jane Austen’s novels IV

Our data is now neatly structured in a one-word-per-row format. Often in text analysis, we will want to remove stop words that are not useful for analysis, typically extremely common words such as “the”, “of”, “to”, and so forth in English. We can remove stop words with an anti_join(), kept in the tidytext dataset stop_words.

8d. This is the first time you call on a dataset from a package. Can you call on the anti_join() function by completing the code below? Note: The code block does not stop at #, so make sure you run the lines of code following “# [sentence]” as well!

```
data(stop_words)

tidy_books <- tidy_books %>%
  ???_???(???_???)

# We can also use dplyr's count() to find the most common words in all the books as a whole.
tidy_books %>%
  count(word, sort = TRUE)
```

Tidying Jane Austen’s novels V

Now we have removed the stop words, we are ready to go find the most common words in all the books as a whole. We can use dplyr’s count() to find those Most Common Words or MCW (also know as Most Frequent Words or MFW).

Because we’ve been using tidy tools, our word counts are stored in a tidy data frame. This allows us to pipe this directly to the ggplot2 package, for example to create a visualization of the most common words.

8e. Let’s build ourselves a pipeline! Run this code and see what happens…

```
library(ggplot2)

tidy_books %>%
  count(word, sort = TRUE) %>%
  filter(n > 600) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col() +
  labs(y = NULL)
```

← If all went well, this pipeline just made a beautiful plot appear, showing you the MCW in Austen’s novels. Victory!

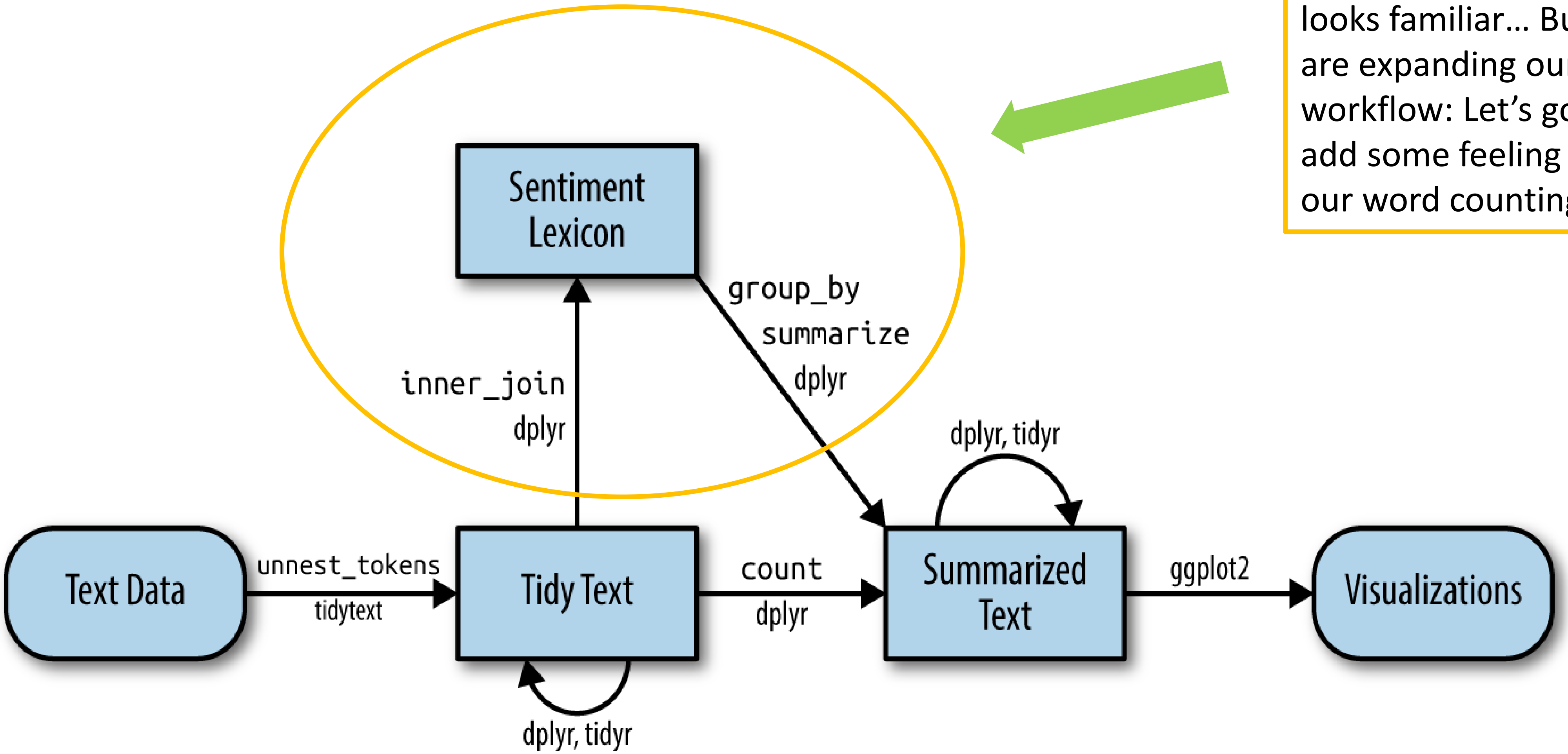
However, what does all of this counting and plotting tell us? Let’s find out during exercise 9!

Part 2: Sentiment analysis with tidy text data I

- Your tidy text mining skills so far include…
  - Tidying a text corpus
  - Remove stop words from your data set
  - Find the Most Common Words in a text corpus
  - Compare MCW's across texts
- Ready for the next step? Sure! Let's dive into sentiment analysis, since…

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. We can use the tools of text mining to approach the emotional content of text programmatically.

Sentiment analysis with tidy text data II



Well, this flowchart looks familiar… But we are expanding our workflow: Let’s go and add some feeling to our word counting!

Sentiment analysis with tidy text data III

- To analyze the sentiment of a text we view the text as a combination of its individual words
- We define the sentiment content of the whole text as the sum of the sentiment content of the individual words
- The tidytext package provides access to several sentiment lexicons (sentiments datasets) . These lexicons are based on unigrams (single words) and contain many English words and the words are assigned scores for positive/negative sentiment, and also possibly emotions like joy, anger, sadness, and so forth.

Sentiment analysis with tidy text data IV

- Three general-purpose lexicons (sentiments datasets) are
  - AFINN from Finn Årup Nielsen
  - bing from Bing Liu and collaborators
  - nrc from Saif Mohammad and Peter Turney

Exercise 9

9a. The function get_sentiments allows us to get specific sentiment lexicons with the appropriate measures for each one.

Can you call on the tidytext package and then use the function mentioned above to get tibbles of the three lexicons mentioned on this slide? This only requires calling on the package and using the function for an individual sentiments dataset. Have a go at it!

Have a look at the different outputs these lexicons provide. Can you characterize the various ways they score sentiment?

Sentiment analysis of *Emma* I

9b. Let’s ask ourselves: What are the most common joy words in Austen’s novel *Emma*? Run this code in order to make your data tidy first and do some real code work in the next exercise!

```
library(janeaustenr)
library(dplyr)
library(stringr)

tidy_books <- austen_books() %>%
  group_by(book) %>%
  mutate(
    linenumber = row_number(),
    chapter = cumsum(str_detect(text,
                                regex("^chapter [\\divxlc]",
                                      ignore_case = TRUE)))) %>%
  ungroup() %>%
  unnest_tokens(word, text)
```

Sentiment analysis of *Emma* II

9c. The text is now in a tidy format with one word per row: We are ready to do sentiment analysis! **We want to know what the most common joy words in *Emma* are.** Can you complete the code and run the script based on these pointers?

i) Use the nrc lexicon and filter() for the joy words

ii) filter() the data frame with the text from the books for the words from *Emma*

iii) Use inner_join() to perform the sentiment analysis

iv) And last but not least, let's count() the most common joy words in *Emma*

Don't worry, you can click for the next slide and some fun code to complete!

Sentiment analysis of *Emma* III

```
nrc_joy <- ???_???(“???") %>%
  filter(sentiment == “???")


tidy_books %>%
  ???(book == “???") %>%
  ???_???(nrc_joy) %>%
  ???(word, sort = TRUE)
```

← Did you get a tibble with words and a word count? Congratulations, you have found “joy” in *Emma*! But there is more to explore…

Sentiment analysis of Austen's novels I

9d. We can also examine how sentiment changes throughout each of Austen's novels. We can do this with just a handful of lines that are mostly dplyr functions. Can you complete the code and run the script based on these pointers?

i) Use the bing lexicon to find a sentiment score for each word

ii) Define an index of 80 lines, so we count up how many positive and negative words there are in defined sections of each book

iii) Use pivot_wider() so that we have negative and positive sentiment in separate columns

iv) Press 'Run' and calculate a net sentiment (positive - negative)

In for completing some code? Go the next slide!

Sentiment analysis of Austen’s novels II

```
library(tidyr)


jane_austen_sentiment <- tidy_books %>%
  inner_join(???_???(“???")) %>%
  count(book, index = linenumber %/% ??, sentiment) %>%
  ???_???(names_from = sentiment, values_from = n, values_fill = 0)
%>%
  mutate(sentiment = positive - negative)
```

…and then →

Sentiment analysis of Austen’s novels III

9e. It’s time for some more visualization to (literally) bring the results of your sentiment analysis into view. We can plot the sentiment scores you just calculated across the plot trajectory of each novel. Notice that we are plotting against the index (the 80 line increment) on the x-axis that keeps track of narrative time in sections of text. They only thing you need to do is call on the ggplot2 package (do you remember how to call on a package? You have done so numerous times before!) and then run the following code:

```
ggplot(jane_austen_sentiment, aes(index, sentiment, fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free_x")
```

← If you were successful, you see 6 different plots, corresponding to the titles of Austen’s 6 novels. Can you discern any trends or differences in the novels’ sentiment structures?

Part 3: Analyzing word and document frequency: tf-idf

- In the previous exercises we looked at *term frequency* (tf), meaning how frequently a word occurs in a document in a specific way.
- We removed stop words from our analysis, since they are highly frequent. But we can do better than that! We can actually keep the stop words, since they might be important as well.
    - We can then look at the *inverse document frequency* (idf), which decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents.
- This can be combined with term frequency to calculate a term's *tf-idf* (the two quantities multiplied together), *the frequency of a term adjusted for how rarely it is used*.

Term frequency in Austen’s novels I

- So, let’s try and find the important words Austen’s novels. We will use the statistic tf-idf to measure this. The inverse document frequency for any given term is defined as

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

- No worries, we have the skills to put this into code!

Term frequency in Austen’s novels II

Exercise 10

10a. Let’s start by looking at the novels of Austen and examine first term frequency, then tf-idf. We can start just by using dplyr verbs such as group_by() and join(). Can you fill in the blanks in the code below based on what you have learned so far and determine the most commonly used words in the novels? (Let’s also calculate the total words in each novel here, for later use)

```
library(dplyr)
library(janeaustenr)
library(tidytext)

book_words <- austen_books() %>%
  unnest_tokens(???,???) %>%
  count(book, word, sort = TRUE)

total_words <- book_words %>%
  group_by(???) %>%
  summarize(total = sum(n))

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

← Now have a look at your output. What strikes you in the data your tibble presents?

Term frequency in Austen's novels III

10b. Now let's plot the distribution of n/total = the number of times a word is used in a book/the total words in that book. Do you remember what package to call on to plot this distribution?

```
library(???)

ggplot(book_words, aes(n/total, fill = book)) +
  geom_histogram(show.legend = FALSE) +
  xlim(NA, 0.0009) +
  facet_wrap(~book, ncol = 2, scales = "free_y")
```

← Does the distribution of term frequency differ greatly per novel or is it quite similar? How would you interpret the plots?

Inverse document frequency in Austen’s novels I

10c. Let’s move on from term frequency to calculating tf-idf and attempt to find the words with high tf-idf (so high relative frequency). The bind_tf_idf() function in the tidytext package takes a tidy text dataset as input with one row per token (term), per document.

- The **word** column contains the terms/tokens
- The **book** column contains the documents
- The **n** column contains how many times each document contains each term

Based on those column headers, can you fill in the code below and calculate tf-idf?

```
book_tf_idf <- book_words %>%
  bind_tf_idf(???, ???, ???)

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

← What type of words have a high tf-idf score?

Inverse document frequency in Austen’s novels II

10d. Since everything in life is better with graphics, let’s visualize our tf-ifd findings! Run the code below to plot the highest tf-idf words in each of Austen’s novels. Can you make it so that you plot the scores per novel? And can you make sure that we see tf-idf for the tokens/terms we have been analyzing? Have a go…

```
library(forcats)


book_tf_idf %>%
  group_by(???) %>%
  slice_max(tf_idf, n = 15) %>%
  ungroup() %>%
  ggplot(aes(tf_idf, fct_reorder(???, tf_idf), fill = book)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~book, ncol = 2, scales = "free") +
  labs(x = "tf-idf", y = NULL)
```

← You have calculated and visualized what type of words distinguish Austen’s novels from each other. Another slam dunk! Time for one more? We have consistently looked at unigrams (single words) in our analyses so far, but now it’s time for a change…

Part 4: Relationships between words: n-grams and correlations

- So far we have considered words as individual units, and considered their relationships to sentiments or to documents
- We are also interested in the relationships between words. We can, for example, examine which words tend to follow others immediately, or which words tend to co-occur within the same documents
- In the final exercise, we'll explore some of the methods tidytext offers for calculating and visualizing relationships between words in your text dataset. This includes the token = "ngrams" argument, which tokenizes by pairs of adjacent words rather than by individual ones
- You will also learn to work with the widyr package to calculate pairwise correlations and distances

Tokenizing Austen’s novels by n-gram I

Exercise 11

You already know how to tokenize by word, using the unnest_tokens function. We can also use the function to tokenize into consecutive sequences of words, called **n-grams**. By seeing how often word X is followed by word Y, we can then build a model of the relationships between them.

We do this by adding the token = “ngrams” option to unnest_tokens(), and setting n to the number of words we wish to capture in each n-gram.

→

Tokenizing Austen’s novels by n-gram II

11a. Can you set the number of words in each n-gram to 2 in the code below? This allows us to examine pairs of two consecutive words, called ‘bigrams’ in Austen’s novels.

```
library(dplyr)
library(tidytext)
library(janeaustenr)

austen_bigrams <- austen_books() %>%
  unnest_tokens(bigram, text, ??? = "ngrams", n = ?)

austen_bigrams
```

← What do you notice about the bigrams in your tibble?

Tokenizing Austen’s novels by n-gram III

11b. When we count our bigrams using dlpyr’s count(), we see that a lot of the most common bigrams are pairs of common words, like stop words. Run this code and you’ll see…

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

We are of course not only interested in the stop word bigrams. So let’s filter our n-grams with tidyr’s separate() and remove cases where either word is a stop word.

Run it! →

Tokenizing Austen’s novels by n-gram IV

```
library(tidyr)

bigrams_separated <- austen_bigrams %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)

bigram_counts
```

<- Your tibble should look somewhat familiar at this point…

Tokenizing Austen’s novels by n-gram V

11c. We will now use tidyr’s unite() function to recombine the columns into one. Using the “separate/filter/count/unite” combination lets us find the most common bigrams *not containing stop-words*. Run the code below.

```
bigrams_united <- bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")

bigrams_united
```

← The tibble is now where we want it to be in order to perform the final step in our analysis… →

Tokenizing Austen’s novels by n-gram VI

11d. Now it’s your time to finalize this piece of code and get the results we were working towards!

A bigram can also be treated as a term in a document in the same way that we treated individual words. For example, we can look at the tf-idf of bigrams across Austen’s novels. Can you complete the code below and produce a tibble of your results?

```
bigram_tf_idf <- bigrams_united %>%
  count(book, bigram) %>%
  bind_tf_idf(???, book, n) %>%
  arrange(desc(tf_idf))

bigram_tf_idf
```

← You have found the bigrams with the highest tf-idf from each Austen novel! The only thing left for us to do now is look at correlations…

Counting and correlating pairs of words

- We have analyzed pairs of words in the previous exercise. However, we are also interested in words that tend to co-occur within particular documents or particular chapters, even if they don’t occur next to each other. In other words, we need to look at *correlations*.
- The widyr package allows us to compute correlations. We will focus on a set of functions that make pairwise comparisons between groups of observations (for example, between documents, or sections of text).

Counting and correlating among sections: *Pride and Prejudice* I

Exercise 12

12a. We want to find out what words tend to appear within each 10-line section of Austen’s *Pride and Prejudice*. Let’s first find the Most Common Words, filtering out stop words. Can you complete the code?

```
austen_section_words <- austen_books() %>%
  filter(book == "Pride & Prejudice") %>%
  mutate(section = row_number() %/% ??) %>%
  filter(section > 0) %>%
  unnest_tokens(???, text) %>%
  filter(!word %in% stop_words$word)

austen_section_words
```

← This is the basis for finding common pairs of words co-appearing within the same section. That’s our next challenge!

Counting and correlating among sections: *Pride and Prejudice* II

12b. Let’s use the pairwise_count() function from the widyr package to let us count common pairs of words co-appearing within the same section. It will result in one row for each pair of words in the word variable. Can you complete the count by using the function mentioned above and providing it with the information on what to count?

```
library(widyr)

# count words co-occuring within sections
word_pairs <- austen_section_words %>%
  ???_???(section, sort = TRUE)

word_pairs
```

Counting and correlating among sections: *Pride and Prejudice* III

- Pairs like “Elizabeth” and “Darcy” are the most common co-occurring words, but that’s not particularly meaningful since *they’re also the most common individual words.* We may instead want to examine **correlation** among words, which indicates how often they appear together relative to how often they appear separately.
- We will now use the pairwise_corr() function in widyr to help us find the *phi coefficient* between words based on how often they appear in the same section.

*The focus of the phi coefficient is how much more likely it is that either **both** word X and Y appear, or **neither** do, than that one appears without the other.*

Counting and correlating among sections: *Pride and Prejudice* IV

12c. The syntax of the pairwise_corr() function is similar to that of pairwise_count(). Just run it!

```
# we need to filter for at least relatively common words first
word_cors <- austen_section_words %>%
  group_by(word) %>%
  filter(n() >= 20) %>%
  pairwise_cor(word, section, sort = TRUE)

word_cors
```

← The output format is helpful for exploration. For example, we could find the words most correlated with specific words…

Counting and correlating among sections: *Pride and Prejudice* V

12d. Let’s pick some interesting words and find the other words most associated with them! You can pick your own and add them to the code below. And do you remember the function we have used a few times to plot your results? Fill it in as well!

```
word_cors %>%
  filter(item1 %in% c(“???", “???", “???", “???")) %>%
  group_by(item1) %>%
  slice_max(correlation, n = 6) %>%
  ungroup() %>%
  mutate(item2 = reorder(item2, correlation)) %>%
  ???(aes(item2, correlation)) +
  geom_bar(stat = "identity") +
  facet_wrap(~ item1, scales = "free") +
  coord_flip()
```

← You can play around with this code, changing the words you want to filter by. In this way, you can perform exploratory analyses by looking at your bar graphs: What word correlations stand out? What is worth looking into in more depth?

Congratulations, you now know the basics of R and Text Mining with R within the Tidyverse!