# 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 pairwaise_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 code 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!