Text Mining

HS

TEXT MINING

- Extracting, Processing, Organizing, Summarizing, and producing visualizations for text data.

What is Text mining?

- It also known as text analytics, is a process of analyzing large amounts of unstructured text data to extract meaningful insights and patterns.
- It involves using techniques from natural language processing, machine learning, and statistics to automatically identify and extract key information from text, such as sentiment, topics, entities, and relationships between them.

Install and Load Packakegs

```
if (!require("pacman")) install.packages("pacman")

# Load contributed packages with pacman
pacman::p_load(tidyverse, dplyr, textdata,
               tidytext, stringr, janeaustenr, tm)
```

Intorduction

- Organizing text data using tidy principles can simplify text mining tasks and make them more effective. Many existing tools, like `dplyr`, `broom`, `tidyr`, and `ggplot2`, are already compatible with this approach. The {`tidytext`} package provides functions and datasets to easily convert text into tidy formats and integrate seamlessly with other text mining tools. Learn more about text mining with tidy data principles in Text Mining With R!

- Hadley Wickham (2014) describes tidy data as data organized in a specific way:
  - Each variable is a column.
  - Each observation is a row.
  - Each observational unit is a table.

In tidy text mining, the "tidy text format" organizes data with **one token per row**.

- A token represents a meaningful unit of text, such as a word, used for analysis.
- The process of splitting text into tokens is called tokenization. While text is often stored as strings or document-term matrices, tidy text mining uses the one-token-per-row format for better analysis.

Token and Tokenization

What is token?

- Tokens can be individual words, phrases or even whole sentences.

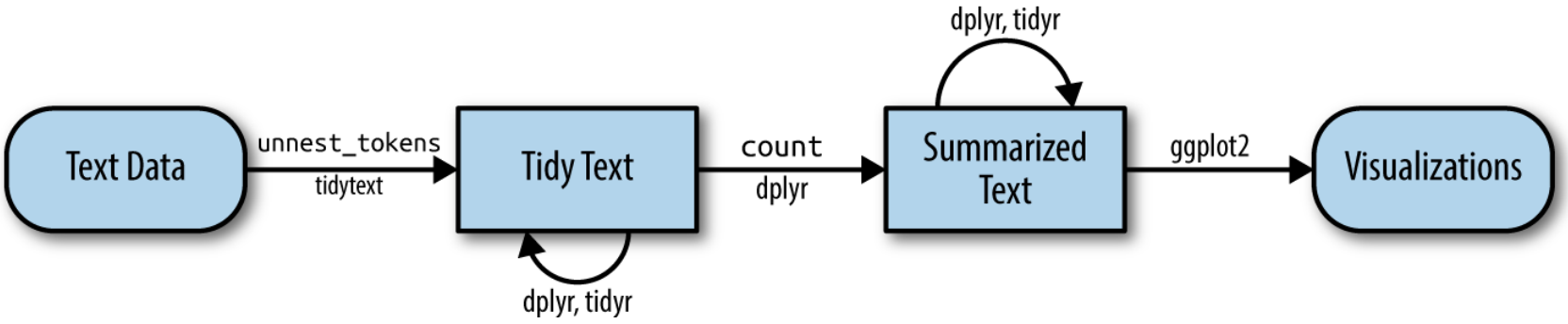
What is tokenization?

- The process of breaking up a given text into units called tokens.

Four Steps Process

1. **Create a Data Frame**: Organize your text data into a structured format (e.g., rows and columns). This helps in managing and processing the data efficiently.
2. **Tokenization**: Break the text into smaller units, such as words or phrases (tokens). This simplifies analysis by isolating meaningful elements.
3. **Find Word Frequencies**: Count how often each word appears in the text. This identifies the most common terms and their importance.
4. **Visualize**: Use charts like bar graphs, word clouds, or histograms to represent the frequencies. Visualization makes patterns and insights more understandable.

This process is simple, effective, and widely applicable to text analysis tasks.


Text Data
unnest_tokens
tidytext
Tidy Text
dplyr, tidyr
count
dplyr
Summarized Text
dplyr, tidyr
ggplot2
Visualizations

Basic Example

Exampl 1

- Let's save the details about `corpus` and `document-term matrix` objects for later and focus on the basics of converting text into a tidy format.

Using `unnest_tokens`

Emily Dickinson wrote some beautiful poems, so let's start with her work.

```r
text_1 <- c("Because I could not stop for Death -",
            "He kindly stopped for me -",
            "The Carriage held but just Ourselves -",
            "and Immortality")

text_1 # 4 lines
```

```
[1] "Because I could not stop for Death -"
[2] "He kindly stopped for me -"
[3] "The Carriage held but just Ourselves -"
[4] "and Immortality"
```

Step 1: Create a Data Frame

- 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.

```
text_df_1 <- tibble(line = c(1:4), text = text_1)

text_df_1  # There are four lines
```

```
# A tibble: 4 x 2
   line text
  <int> <chr>
1     1 Because I could not stop for Death -
2     2 He kindly stopped for me -
3     3 The Carriage held but just Ourselves -
4     4 and Immortality
```

- This data frame contains text but isn't ready for tidy text analysis yet. Each row has multiple words combined, so we can't filter individual words or count their frequency. To fix this, we need to transform it into a format where each row contains just one word (or "token").

Step 2: Tokenization

- To work with text in a tidy format, we need to break it into smaller parts (called tokens) and organize it as a tidy dataset. We can do this using the `unnest_tokens()` function from the **tidytext** package.

```
unnest_tokens(data, output, input)
```

```
text_df_1 %>%
  unnest_tokens(word, text) ->
  tibble_1
tibble_1 %>%
  slice(c(1, 7, 15, 20))
```

```
# A tibble: 4 x 2
   line word
  <int> <chr>
1     1 because
2     1 death
3     3 held
4     4 immortality
```

- The `unnest_tokens` function works with two key arguments:
  - the name of the new column (for the tokens, like `word`) and
  - the name of the existing column that contains the text to be split (like `text`). In this case, the data frame `text_df_1` has a column called `text` with the text data we want to break into tokens.

- When we use `unnest_tokens`, each row of the original text is split so that each word becomes its own row in the new data frame. By default:
- Words are split (tokenized) individually.
- Other columns, like line numbers, stay in the data.
- Punctuation is removed.
- Tokens are converted to lowercase for easier comparison (this can be turned off with `to_lower = FALSE`).

Step 3: Find Word Frequencies

- Find frequencies for each word

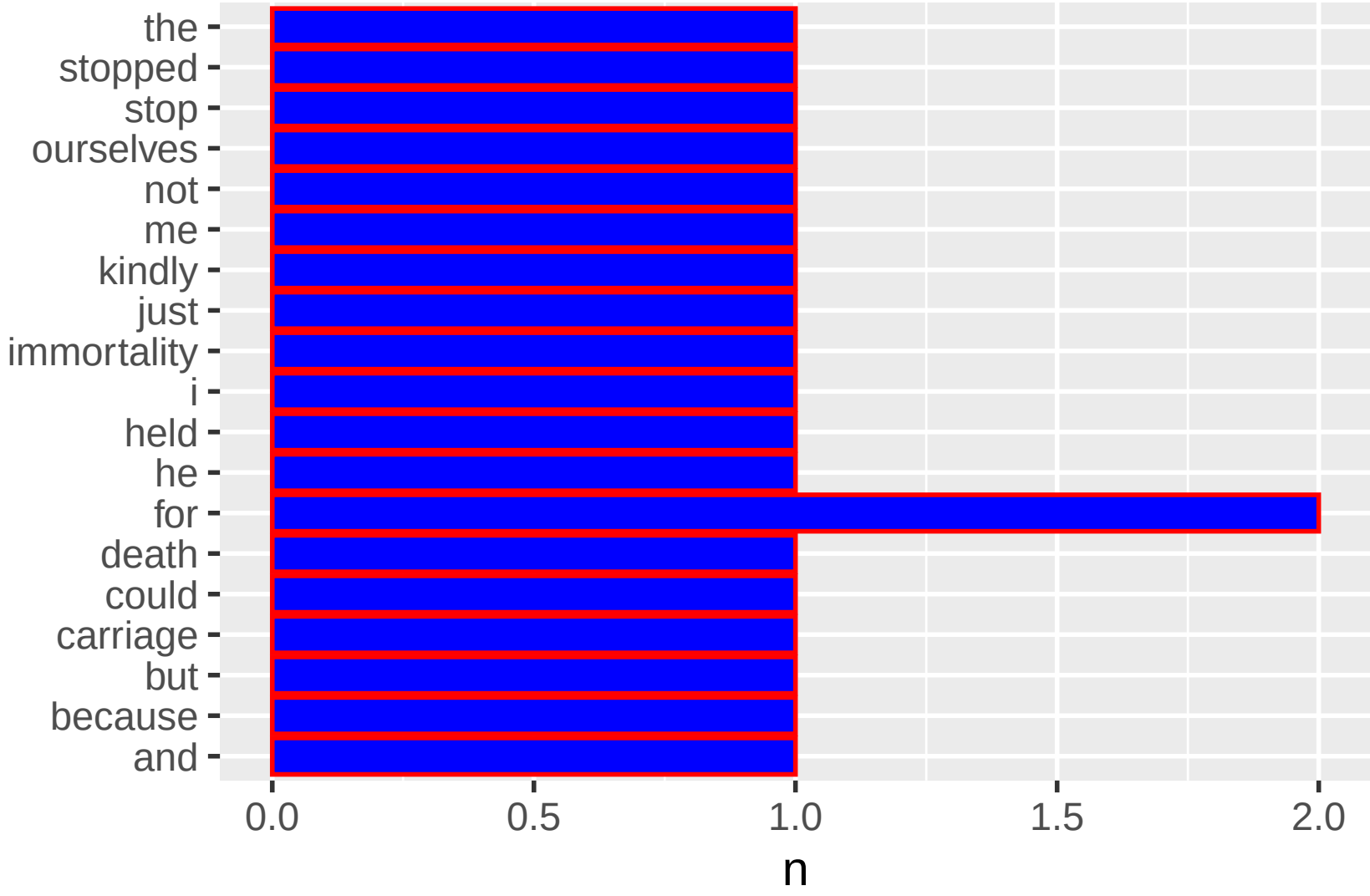
```
tibble_1%>%
  count(word, sort =TRUE) %>%
  filter(n >= 1)
```

```
# A tibble: 19 x 2
   word              n
   <chr>         <int>
 1 for               2
 2 and               1
 3 because           1
 4 but               1
 5 carriage          1
 6 could             1
 7 death             1
 8 he                1
 9 held              1
10 i                 1
```

Step 4: Visualize

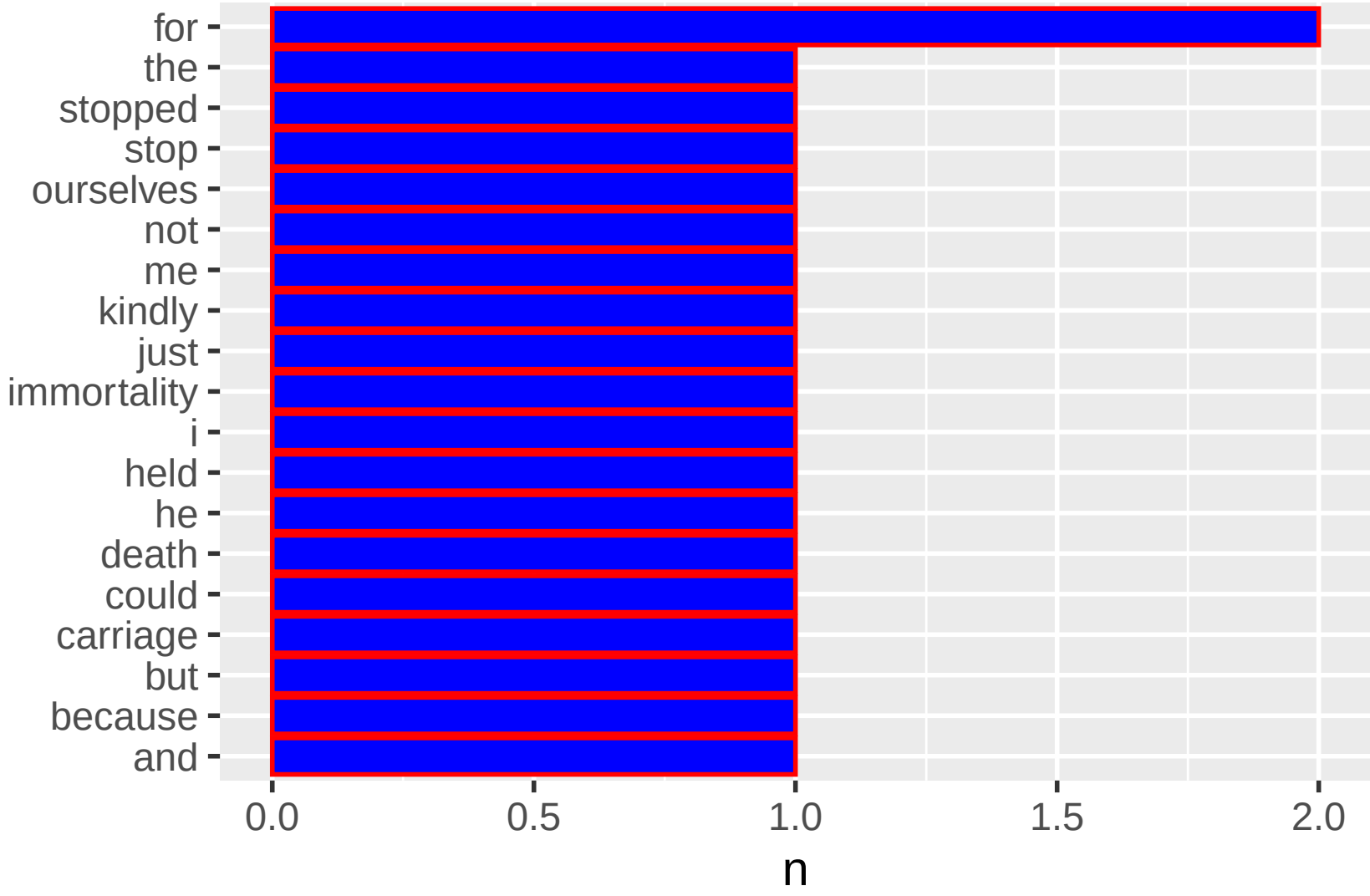
- Create a Data visual (Bar Graph) showing and comparing word frequencies

```
tibble_1%>%
  count(word, sort =TRUE) %>%
  filter(n >= 1) %>%
  ggplot(aes(n, word)) +
  geom_col(fill = "blue", color = "red") +
  labs(y = NULL)->
  g1
```


the
stopped
stop
ourselves
not
me
kindly
just
immortality
i
held
he
for
death
could
carriage
but
because
and
0.0
0.5
1.0
1.5
2.0
n

- We want reorder the levels of the `word` column based on the counts `n`.
- We use `reorder()` function from {`base`}

```
tibble_1%>%
  count(word, sort =TRUE) %>%
  filter(n >= 1) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col(fill = "blue", color = "red") +
  labs(y = NULL) ->
  g2
```


for
the
stopped
stop
ourselves
not
me
kindly
just
immortality
i
held
he
death
could
carriage
but
because
and
0.0
0.5
1.0
1.5
2.0
n

Example 2:

- Famous Poam

```
text_2 <- c("No man is an island",
            "Entire of itself",
            "Every man is a piece of the continent",
            "A part of the main.",
            "If a clod be washed away by the sea",
            "Europe is the less.",
            "As well as if a promontory were.",
            "As well as if a manor of thy friend's",
            "Or of thine own were:",
            "Any man's death diminishes me",
            "Because I am involved in mankind",
            "And therefore never send to know for whom the be
            "It tolls for thee.")
```

```
 [1] "No man is an island"
 [2] "Entire of itself"
 [3] "Every man is a piece of the continent"
 [4] "A part of the main."
 [5] "If a clod be washed away by the sea"
 [6] "Europe is the less."
 [7] "As well as if a promontory were."
 [8] "As well as if a manor of thy friend's"
 [9] "Or of thine own were:"
[10] "Any man's death diminishes me"
[11] "Because I am involved in mankind"
[12] "And therefore never send to know for whom the bell to
[13] "It tolls for thee."
```

Step 1: Create a DATA FRAME

```
text_tibble_2 <- tibble(line = 1:13, text = text_2)
text_tibble_2 %>%
  head()
```

```
# A tibble: 6 x 2
   line text
  <int> <chr>
1     1 No man is an island
2     2 Entire of itself
3     3 Every man is a piece of the continent
4     4 A part of the main.
5     5 If a clod be washed away by the sea
6     6 Europe is the less.
```

Step 2: Tokenization

- Find line locations for each word in the text

```
text_tibble_2 %>%
  unnest_tokens(word, text) -> tibble_2

tibble_2
```

```
# A tibble: 81 x 2
    line word
   <int> <chr>
 1     1 no
 2     1 man
 3     1 is
 4     1 an
 5     1 island
 6     2 entire
 7     2 of
 8     2 itself
 9     3 every
```

Step 3: Find Word Frequencies

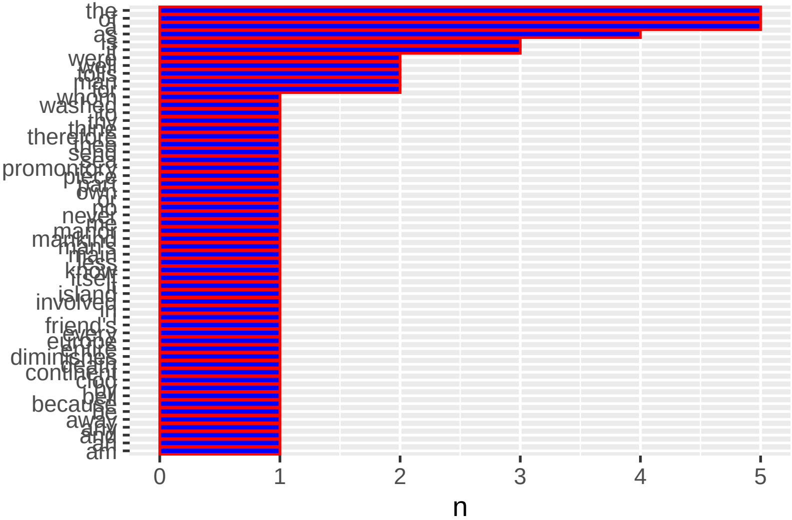
```
tibble_2 %>%
  count(word, sort =TRUE) %>%
  filter(n > 0) ->
  tibble_freq_2

 tibble_freq_2
```

```
# A tibble: 57 x 2
   word      n
   <chr> <int>
 1 a         5
 2 of        5
 3 the       5
 4 as        4
 5 if        3
 6 is        3
 7 for       2
 8 man       2
```

Step 4: Visualize

```
tibble_freq_2 %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(n, word)) +
  geom_col(fill = "blue", color = "red") +
  labs(y = NULL) ->
  g3
```


the
as
were
washed
therefore
promontory
never
mankind
island
involved
friend's
diminishes
continent
because
away
am
0
1
2
3
4
5
n

Examp 3

- Novels of Jane Austen

Jane Austen's novels are wonderfully structured! We'll work with the text of her six completed, published novels using the `janeaustenr` package. This package organizes the text in a one-row-per-line format. Let's convert it into a tidy format for analysis.

Step 1: Create a DATA FRAME

First part:

- We group the data based

```
austen_books() %>%
  group_by(book)
```

```
# A tibble: 73,422 x 2
# Groups:   book [6]
   text                     book
   <chr>                    <fct>
 1 "SENSE AND SENSIBILITY"  Sense & Sensibility
 2 ""                       Sense & Sensibility
 3 "by Jane Austen"         Sense & Sensibility
 4 ""                       Sense & Sensibility
 5 "(1811)"                 Sense & Sensibility
 6 ""                       Sense & Sensibility
 7 ""                       Sense & Sensibility
 8 ""                       Sense & Sensibility
```

First part:

- We group the data based on the book and give each row a unique number. So, we need to create a column for `linenumber`

```
austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number())
```

```
# A tibble: 73,422 x 3
# Groups:   book [6]
   text                    book                linenumber
   <chr>                   <fct>                    <int>
 1 "SENSE AND SENSIBILITY" Sense & Sensibility          1
 2 ""                      Sense & Sensibility          2
 3 "by Jane Austen"        Sense & Sensibility          3
 4 ""                      Sense & Sensibility          4
 5 "(1811)"                Sense & Sensibility          5
 6 ""                      Sense & Sensibility          6
 7 ""                      Sense & Sensibility          7
```

Second part:

- Now we want to find out each line of text is included in what chapter of the book!. So, we need to creat a column for Chapter

```
austen_books() %>%
  group_by(book) %>%
  mutate(linenumber = row_number(),
         chapter = cumsum(str_detect(text,
                                     regex("^chapter [\\di
                                           ignore_case = TR
```

```
# A tibble: 73,422 x 4
# Groups:   book [6]
   text                    book                linenumber c
   <chr>                   <fct>                    <int>
 1 "SENSE AND SENSIBILITY" Sense & Sensibility          1
 2 ""                      Sense & Sensibility          2
 3 "by Jane Austen"        Sense & Sensibility          3
 4 ""                      Sense & Sensibility          4
```

Third part:

- In this step we ungroup the data and pit it into a new object

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

What does the following mean?

- regex("^chapter [\divxlc]"

Answer:

- This regular expression pattern would match any chapter heading in Jane Austen's novels that starts with the word "chapter" followed by a space and then one of the Roman numerals I, V, X, L, or C, or the lowercase letter 'd'.
- \d is a metacharacter that matches any digit from 0 to 9

```
# A tibble: 73,422 x 4
   text                    book                linenumber c
   <chr>                   <fct>                    <int>
 1 "SENSE AND SENSIBILITY" Sense & Sensibility          1
 2 ""                      Sense & Sensibility          2
 3 "by Jane Austen"        Sense & Sensibility          3
 4 ""                      Sense & Sensibility          4
 5 "(1811)"                Sense & Sensibility          5
 6 ""                      Sense & Sensibility          6
 7 ""                      Sense & Sensibility          7
 8 ""                      Sense & Sensibility          8
 9 ""                      Sense & Sensibility          9
10 "CHAPTER 1"             Sense & Sensibility         10
# i 73,412 more rows
```

Step 2: Tokenization and remove `stop_words`

- There are a few words that they do not have special meaning, so we want to remove them. These words are listed in a file called `stop_words`

Part 1: Tokenization.

- Find line locations for each word in the text

```
tidy_books <- original_books %>%
  unnest_tokens(word, text)

tidy_books   # there are 725,055 rows
```

```
# A tibble: 725,055 x 4
   book                linenumber chapter word
   <fct>                    <int>   <int> <chr>
 1 Sense & Sensibility          1       0 sense
 2 Sense & Sensibility          1       0 and
 3 Sense & Sensibility          1       0 sensibility
 4 Sense & Sensibility          3       0 by
 5 Sense & Sensibility          3       0 jane
 6 Sense & Sensibility          3       0 austen
 7 Sense & Sensibility          5       0 1811
 8 Sense & Sensibility         10       1 chapter
 9 Sense & Sensibility         10       1 1
```

- We separated each line of text in the original data frame into tokens. Now that the data is in **one-word-per-row** format, we can manipulate it with `tidy` tools like `dplyr`. Often in text analysis, we will want to remove `stop words`.

Part 2: Stop Words

- `stop words` are words that are not useful for an analysis, typically extremely common words such as “the”, “of”, “to”, and so forth in English.
- A data frame with 1149 rows and 2 variables:

```
stop_words is a document from tidytext package and
 has 1149  rows and  2  columns.
```

```
head(stop_words)
```

```
# A tibble: 6 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
```

Remove Stop Words

- In this part we remove the stop_words from our data by `anti_join()` from {dplyr}

```
tidy_stop <- tidy_books %>%
  anti_join(stop_words)

tidy_stop # now it has 217,609 rows
```

```
# A tibble: 217,609 x 4
   book                linenumber chapter word
   <fct>                    <int>   <int> <chr>
 1 Sense & Sensibility          1       0 sense
 2 Sense & Sensibility          1       0 sensibility
 3 Sense & Sensibility          3       0 jane
 4 Sense & Sensibility          3       0 austen
 5 Sense & Sensibility          5       0 1811
 6 Sense & Sensibility         10       1 chapter
 7 Sense & Sensibility         10       1 1
```

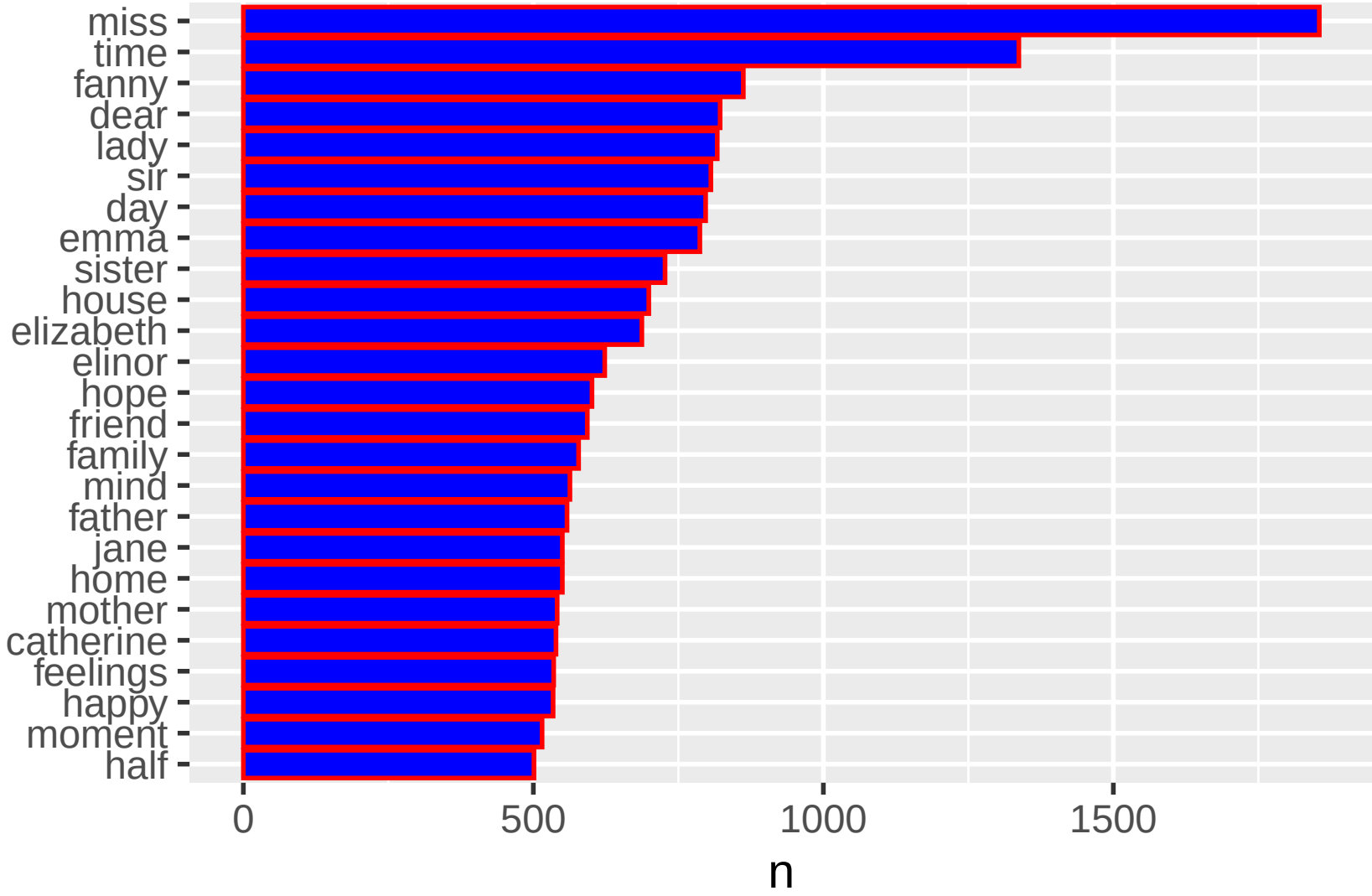
```
The book had 725055  rows and  4  columns.
 The book now has 217609  rows and  4  columns.
```

Step 3: Find Word Frequencies

- We can also use dplyr's `count()` to find the most common words in all the books as a whole.

```
tidy_stop %>%
  count(word, sort = TRUE) |>
  filter(n > 500)
```

```
# A tibble: 25 x 2
   word        n
   <chr>   <int>
 1 miss     1855
 2 time     1337
 3 fanny     862
 4 dear      822
 5 lady      817
 6 sir       806
 7 day       797
 8 emma      787
 9 sister    727
```


miss
time
fanny
dear
lady
sir
day
emma
sister
house
elizabeth
elinor
hope
friend
family
mind
father
jane
home
mother
catherine
feelings
happy
moment
half
0
500
1000
1500
n

Sentiment Analysis

- Here we address the topic of opinion mining or sentiment analysis. 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

Why we use tidy data?

- Tidy data is a data format in which each variable is a column and each observation is a row, and there is only one type of data per table.
- Tidy data is ideal for text mining and sentiment analysis because it allows for easy manipulation and visualization of the data.

Sentiment Analysis

- One important application of text mining is sentiment analysis, which involves determining the sentiment or emotion expressed in a piece of text.
- One popular approach to sentiment analysis with tidy data is the "tidytext" package in R, which provides a framework for organizing and analyzing textual data.
- The package includes functions for tokenizing text, calculating word frequencies, and scoring the sentiment of each word using various lexicons.

What is Lexicon

- In the context of text mining, a **lexicon** refers to a collection or list of words that are associated with specific meanings or categories, typically used for tasks like sentiment analysis, emotion detection, or text classification. Each word in the lexicon is often assigned a "score" that reflects its sentiment (positive or negative) or other attributes.
- For example, in sentiment analysis, a lexicon might contain words like "happy" with a positive score, "sad" with a negative score, or even neutral words like "book" with no sentiment score.
- So, when the package uses lexicons to score sentiment, it means it looks up words in these predefined lists and assigns scores based on their associated meanings. This helps analyze the overall sentiment (positive, negative, neutral) in a text.

tidytext Package and Lexicons

- The tidytext package provides access to several sentiment lexicons.
- There are three lexicons:
- All three of these lexicons are based on unigrams, i.e., single words.
- These lexicons 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.

Three general-purpose lexicons are

1. AFINN from Finn Årup Nielsen

- The AFINN lexicon assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment.

```
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
```

2. bing from Bing Liu and collaborators

- The bing lexicon categorizes words in a binary fashion into positive and negative categories.

```
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
# i 6,776 more rows
```

3. nrc from Saif Mohammad and Peter Turney

- Categorizes words in a binary fashion ("yes"/"no") into categories of positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust.
- get_sentiments() allows us to get specific sentiment lexicons

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

```
# A tibble: 13,872 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
```

Examples

Example 1

- Sentiment analysis with `inner join`
- Let's look at the words with a `joy score` from the NRC lexicon.
- What are the most common joy words in Emma?

Step 1:

- We need to take the text of the novels and convert the text to the tidy format using `unnest_tokens()`,

Get the books

```
tidy_books <- austen_books()
```

Group them by the book

```
tidy_books <- tidy_books %>%
  group_by(book)
```

Create Row Numbers

- Now create a column that keeps the row numbers. We want to detect the Chapter numbers

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

- Line 1 to 9 has the chapter 0 and Chapter 1 starts in line 10.

Ungroup

- Now we need to ungroup it

```
tidy_books <- tidy_books %>%
  ungroup()
```

```
# A tibble: 73,422 x 4
   text                    book                linenumber c
   <chr>                   <fct>                    <int>
 1 "SENSE AND SENSIBILITY" Sense & Sensibility          1
 2 ""                      Sense & Sensibility          2
 3 "by Jane Austen"        Sense & Sensibility          3
 4 ""                      Sense & Sensibility          4
 5 "(1811)"                Sense & Sensibility          5
 6 ""                      Sense & Sensibility          6
 7 ""                      Sense & Sensibility          7
 8 ""                      Sense & Sensibility          8
 9 ""                      Sense & Sensibility          9
10 "CHAPTER 1"             Sense & Sensibility         10
# i 73,412 more rows
```

Step 2:

- Now this is the time to tokenization

```
tidy_books <- tidy_books %>%
  unnest_tokens(word, text)
```

```
# A tibble: 725,055 x 4
   book                linenumber chapter word
   <fct>                    <int>   <int> <chr>
 1 Sense & Sensibility          1       0 sense
 2 Sense & Sensibility          1       0 and
 3 Sense & Sensibility          1       0 sensibility
 4 Sense & Sensibility          3       0 by
 5 Sense & Sensibility          3       0 jane
 6 Sense & Sensibility          3       0 austen
 7 Sense & Sensibility          5       0 1811
 8 Sense & Sensibility         10       1 chapter
 9 Sense & Sensibility         10       1 1
10 Sense & Sensibility         13       1 the
# i 725,045 more rows
```

A Nice Question

- Why we use the name `word` for output column from unnest_tokens()?
- Answer: This is a convenient choice because the sentiment lexicons and stop word datasets have columns named word; performing inner joins and anti-joins is thus easier.

- The text now is in a tidy format with one word per row.
- we are ready to do the sentiment analysis

- Let's use the NRC lexicon and `filter()` for the joy words.

```
# A tibble: 687 x 2
   word          sentiment
   <chr>         <chr>
 1 absolution    joy
 2 abundance     joy
 3 abundant      joy
 4 accolade      joy
 5 accompaniment joy
 6 accomplish    joy
 7 accomplished  joy
 8 achieve       joy
 9 achievement   joy
10 acrobat       joy
# i 677 more rows
```

Step 3:

- let's filter() the data frame with the text from the books for the words from the book Emma

```
tidy_books %>%
  filter(book == "Emma") ->
  Emma_joy
```

```
# A tibble: 160,996 x 4
   book  linenumber chapter word
   <fct>      <int>   <int> <chr>
 1 Emma           1       0 emma
 2 Emma           3       0 by
 3 Emma           3       0 jane
 4 Emma           3       0 austen
 5 Emma           8       0 volume
 6 Emma           8       0 i
 7 Emma          12       1 chapter
 8 Emma          12       1 i
 9 Emma          15       1 emma
10 Emma          15       1 woodhouse
# i 160,986 more rows
```

Perform Sentiment

Use `inner_join()` to perform the sentiment analysis.

```
Emma_joy %>%
  inner_join(nrc_joy) ->
  n_E_joy
```

Question:

- What does the message `Joining with`by = join_by(word)‘
- The message “Joining with `by = join_by(word)`” indicates that the function is performing a join operation on two datasets, where the datasets are being merged based on the common column “word.”

```
# A tibble: 4,232 x 5
   book  linenumber chapter word      sentiment
   <fct>      <int>   <int> <chr>     <chr>
 1 Emma          16       1 happy     joy
 2 Emma          16       1 blessings joy
 3 Emma          21       1 marriage  joy
 4 Emma          22       1 mother    joy
 5 Emma          24       1 excellent joy
 6 Emma          25       1 mother    joy
 7 Emma          25       1 affection joy
 8 Emma          28       1 friend    joy
 9 Emma          33       1 friend    joy
10 Emma          33       1 friend    joy
# i 4,222 more rows
```

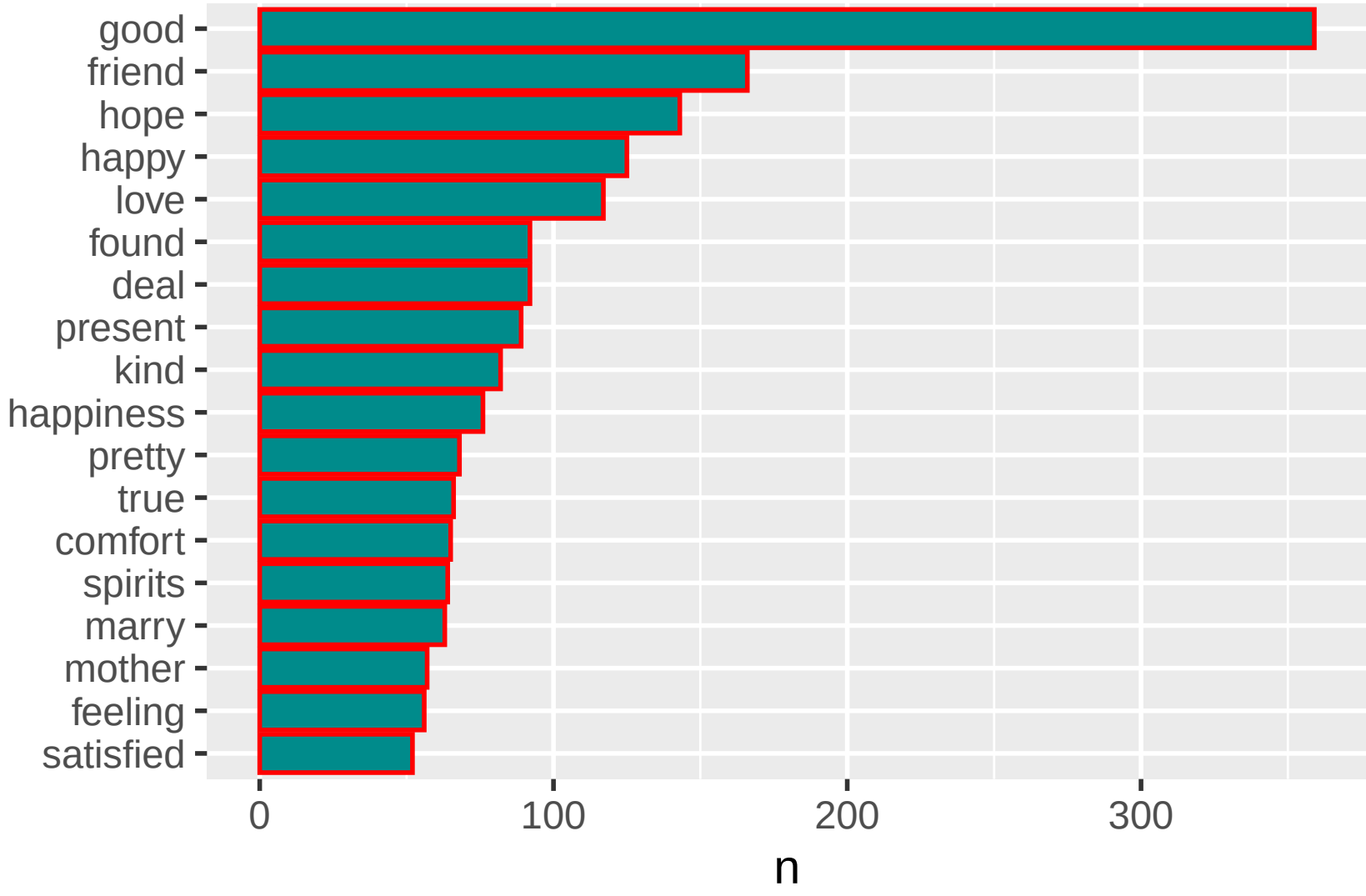
Step 3:

- We see mostly positive, happy words about hope, friendship, and love here.

```
n_E_joy %>%
  count(word, sort = TRUE)
```

```
# A tibble: 301 x 2
   word           n
   <chr>      <int>
 1 good         359
 2 friend       166
 3 hope         143
 4 happy        125
 5 love         117
 6 deal          92
 7 found         92
 8 present       89
 9 kind          82
10 happiness     76
```

Step 4:


good
friend
hope
happy
love
found
deal
present
kind
happiness
pretty
true
comfort
spirits
marry
mother
feeling
satisfied
0
100
200
300
n

Example 2

- Sentiment analysis with `inner join`
- Let's look at the words with a `joy score` from the AFFIN lexicon.
- What are the most common joy words in Emma?

Step 1:

- We need to take the text of the novels and convert the text to the tidy format using `unnest_tokens()`,

Get the books

```
tidy_books <- austen_books()
```

Group them by the book

```
tidy_books <- tidy_books %>%
  group_by(book)
```

Create Row Numbers

- Now create a column that keeps the row numbers. We want to detect the Chapter numbers

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

- Line 1 to 9 has the chapter 0 and Chapter 1 starts in line 10.

Ungroup

- Now we need to ungroup it

```
tidy_books <- tidy_books %>%
  ungroup()
```

```
# A tibble: 725,055 x 4
   book                linenumber chapter word
   <fct>                    <int>   <int> <chr>
 1 Sense & Sensibility          1       0 sense
 2 Sense & Sensibility          1       0 and
 3 Sense & Sensibility          1       0 sensibility
 4 Sense & Sensibility          3       0 by
 5 Sense & Sensibility          3       0 jane
 6 Sense & Sensibility          3       0 austen
 7 Sense & Sensibility          5       0 1811
 8 Sense & Sensibility         10       1 chapter
 9 Sense & Sensibility         10       1 1
10 Sense & Sensibility         13       1 the
# i 725,045 more rows
```

```r
affin_joy <- get_sentiments("afinn")%>%
  filter(value >= 1)
```

```
# A tibble: 878 x 2
   word       value
   <chr>      <dbl>
 1 abilities      2
 2 ability        2
 3 aboard         1
 4 absolve        2
 5 absolved       2
 6 absolves       2
 7 absolving      2
 8 absorbed       1
 9 accept         1
10 accepted       1
# i 868 more rows
```

Step 3:

- let's filter() the data frame with the text from the books for the words from the book Emma

```
tidy_books %>%
  filter(book == "Emma") ->
  Emma_joy
```

```
# A tibble: 160,996 x 4
   book  linenumber chapter word
   <fct>      <int>   <int> <chr>
 1 Emma           1       0 emma
 2 Emma           3       0 by
 3 Emma           3       0 jane
 4 Emma           3       0 austen
 5 Emma           8       0 volume
 6 Emma           8       0 i
 7 Emma          12       1 chapter
 8 Emma          12       1 i
 9 Emma          15       1 emma
10 Emma          15       1 woodhouse
# i 160,986 more rows
```

Perform Sentiment

Use `inner_join()` to perform the sentiment analysis.

```
Emma_joy %>%
  inner_join(affin_joy) ->
  affin_joy
```

Question:

- What does the message `Joining with`by = join_by(word)‘
- The message “Joining with `by = join_by(word)`” indicates that the function is performing a join operation on two datasets, where the datasets are being merged based on the common column “word.”

```
# A tibble: 6,472 x 5
   book  linenumber chapter word          value
   <fct>      <int>   <int> <chr>         <dbl>
 1 Emma          15       1 clever            2
 2 Emma          15       1 rich              2
 3 Emma          15       1 comfortable       2
 4 Emma          16       1 happy             3
 5 Emma          16       1 best              3
 6 Emma          20       1 affectionate      3
 7 Emma          24       1 excellent         3
 8 Emma          25       1 affection         3
 9 Emma          28       1 fond              2
10 Emma          32       1 authority         1
# i 6,462 more rows
```

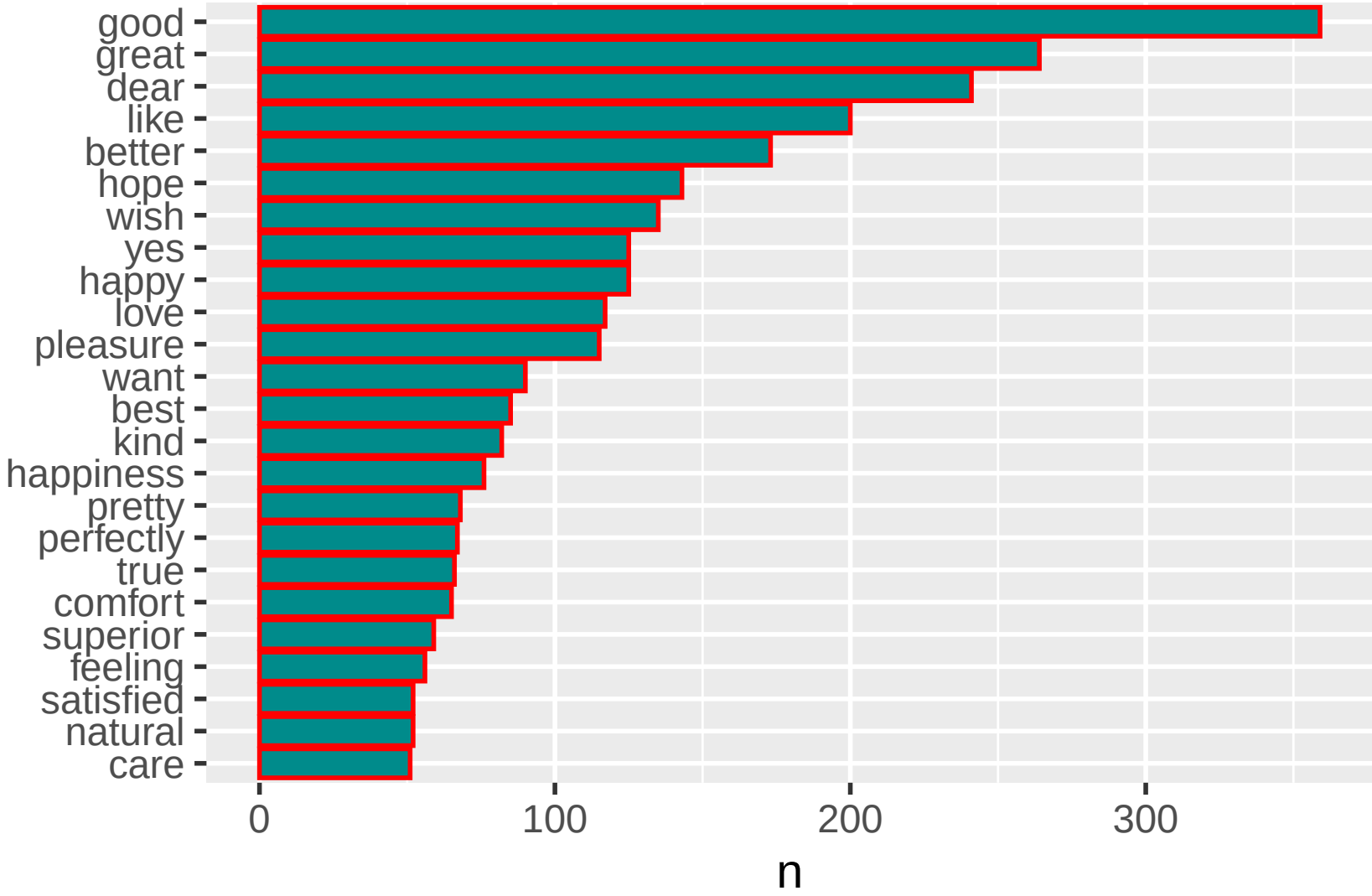
Step 3:

- We see mostly positive, happy words about hope, friendship, and love here.

```
affin_joy %>%
  count(word, sort = TRUE)
```

```
# A tibble: 415 x 2
   word        n
   <chr>   <int>
 1 good      359
 2 great     264
 3 dear      241
 4 like      200
 5 better    173
 6 hope      143
 7 wish      135
 8 happy     125
 9 yes       125
10 love      117
```

Step 4:


good
great
dear
like
better
hope
wish
yes
happy
love
pleasure
want
best
kind
happiness
pretty
perfectly
true
comfort
superior
feeling
satisfied
natural
care
0
100
200
300
n