Introduction to TidyText

Kaylee Alexander, Duke University

2019-18-04

RStudio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics. Working in RStudio, you can easy install R packages, such as the Tidyverse packages for data science. These packages (i.e. dplyr, tidyr, ggplot2, etc.) facilitate data cleaning, analysis and visualization. R can also be used to analyze non-quantitative data, such as texts. In this workshop we will use the **tidytext** package (as well as others) to analyze and visualize written works, such as novels and song lyrics.

1. Getting Started

To begin, we need to open RStudio and create a new script.

- Go to **File** → **New File** → **R Script**

For this workshop, you will be using the following packages:

1. dplyr
2. tidytext
3. gutenbergr
4. ggplot2
5. ggthemes
6. stringr
7. tidyr

If you have not worked with these packages in RStudio before, you will need to install them. To install these packages in RStudio:

- click select the **Packages** tab in the bottom right window
- click **Install**
- type in the names of the packages above (separated by a space or a comma)
- click **Install**

2. Manually Loading Text & Creating a Data Frame

Text can be entered into RStudio manually and transformed into a data frame for analysis. You can separate multiple lines of text by separating each line with a comma (,). Go to your script window and type in the following code chunk:

```
text <- c("The time has come,' the Walrus said,",
          "To talk of many things:",
          "Of shoes — and ships — and sealing-wax —",
          "Of cabbages — and kings —",
          "And why the sea is boiling hot —",
          "And whether pigs have wings.'")

text
```

Next, you will need to transform the value **text** into a **tibble** (data frame), which will allow you to perform text analysis. For this, you will use the **dplyr** package that you installed earlier. To use an installed package, you need to first load it into your R session. To load **dplyr**, run the following:

```
library(dplyr)
```

Now you can create a data frame from the value you previously created. Run the following:

```
text_df <- data_frame(line = 1:6, text = text)

text_df
```

3. Text Tokenization

In order to perform more analysis on this text, we want to convert the data frame so that we have one token (in this case, a word) per document per row, a process called **tokenization**. Tokenizing text will retain the line number, remove punctuation, and default all words to lowercase characters.[1] For this, we will be using the **tidytext** package.

Load the **tidytext** package into the session, as we did with **dplyr**.

```
library(tidytext)
```

Then, run the following to tokenize:

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

You can replace the data in your environment with this new tibble by running the following:

[1] If you would like to retain the original case of the text, you can include an additional argument (to_lower = FALSE).

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

text_df
```

This is the format in which we can begin to analyze texts using R. In the next section, we will work with a larger text, and begin to perform some basic analyses and visualizations.

4. Analyzing Mary Shelley's *Frankenstein* (1831)

Project Gutenberg is a free database of eBooks, predominately texts for which the U.S. copyright has expired. These ebooks are available in a variety of formats, including Plain Text UTF-8, which we can easily use for text analysis in R. Each e-book has its own unique e-book number, which we can use with the package **gutenbergr** to automatically load the text into an RStudio environment. In this section, we will load and analyze the text of Mary Shelley's *Frankenstein* (e-book # 42324).

Begin by loading the **gutenbergr** package into the session, as we did with **dplyr** and **tidytext**.

```
library(gutenbergr)
```

Then, run the following:

```
frankenstein <- gutenberg_download(c(42324))
```

If you look at the first few rows, you see that this also includes the novel's front matter (i.e. title page, publishing info, introduction, etc.). If you were to examine the last rows, you would also see it includes additional publishing information and the transcriber's notes. Since we're only interested in analyzing Shelley's novel, it would be useful to remove these rows.

Run the following to remove the front matter (rows 1-237) and back matter (rows 7620-7631):

```
frankenstein <- frankenstein[-c(1:237, 7620:7631), ]
```

Now we're ready to tokenize. During this process, we can add an additional argument to order the words according to the word count. Run the following to create a new tibble with the tokenized text, ordered according to word count:

```
tidy_frankenstein <- frankenstein %>%
  unnest_tokens(word, text) %>%
  count(word, sort = TRUE)

tidy_frankenstein
```

**Question 1: Which two (2) words appear most frequently in *Frankenstein*? How many times do they each appear?

You may have noticed that these top used words don’t tell you too much about the text of Frankenstein. These types of words are called **stop words** (i.e. ‘the’, ‘of’, ‘to’, etc.). We can remove these types of words from our analysis by using the **anti_join** function. Luckily, you won’t have to come up with a whole list of these words yourself; RStudio includes a preloaded package with a data frame, **stop_words**, which contains English stop words compiled from three different lexicons.[2]

To remove the stop words from the **tidy_frankenstein tibble**, you must first load the **stop_words** data frame:

```
data("stop_words")
```

Now, you can remove the stop words from your data frame:

```
tidy_frankenstein <- tidy_frankenstein %>%
  anti_join(stop_words)
```

**Question 2: Now, which two (2) words appear most frequently? How many times do they each appear?

5. Visualizing Mary Shelley’s *Frankenstein* (1831)

We are now ready to plot the text of Frankenstein. To do this we will use the packages **ggplot2** and **ggthemes**. These packages are used to create data graphics, according to the Grammar of Graphics. This allows you to easily manipulate the design of your graphs, and to create the most visually appealing data visualizations in R.

First, we are going to plot the most commonly used words in Frankenstein. To do this, first load ggplot2 and ggthemes into your session.

```
library(ggplot2)
library(ggthemes)
```

Next, run the following to plot the words in *Frankenstein* that appear more than 50 times:

```
tidy_frankenstein %>%
  filter(n > 50) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  geom_col(fill = "darkred") +
  theme_fivethirtyeight() +
  xlab(NULL) +
  ylab("Word Count") +
```

[2] Datasets of stop words for other languages are also available online and can be loaded into R as needed.

```
  coord_flip() +
  ggtitle("Word Usage in Frankenstein")
```

This code chunk first calls the dataset that we would like to plot, then filters the dataset to only words that appear more than 50 times in the novel and orders the data according to the number of mentions. Then it calls **ggplot**, and plots **word** on the **x-axis** and **n** (word count) on the **y-axis**. **Geom_col** tells R to use a bar chart with the value of the *y* variable (here, *n*), while the function **fill = "darkred"** colors in the bars.[3] **theme_fivethirtyeight** is a pre-set design from the package ggthemes, which we then manipulate with the subsequent arguments to adjust the x-axis labels and y-axis labels, change the graph into a horizontal format, and add an appropriate title.

****Question 3: How many words appear more than 50 times in *Frankenstein*?**

6. Sentiment Lexicons

One type of textual analysis that can be conducted using R is sentiment analysis in which the emotions of a given text or set of texts is examined. To conduct sentiment analysis, we can use sentiment lexicons. There are three lexicons that can be used for general sentiment analysis on English-language texts: **AFINN** (Finn Årup Nielsen), **bing** (Bing Liu et. Al) and **nrc** (Saif Mohammad and Peter Turney). These lexicons assign scores for positive and negative sentiment, and also emotions such as joy, anger, fear, etc.

- **nrc** categorizes words as positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise and trust.
- **bing** categorizes words as positive or negative.
- **AFINN** assigns words a numeric score between -5 and 5, where the negative scores indicate negative sentiment and positive scores indicate positive sentiment.

All three of these lexicons are included in the **sentiments** dataset, and **tidytext** provides the function **get_sentiments( )**, which can be used to call specific lexicons. To call the **bing** lexicon, for example, run the following:

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

Now, call the **AFINN** sentiment lexicon.

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

****Question 4: What sentiment schore is assigned to "abhorrent"?**

Finally, call the **nrc** lexicon.

[3] Additional codes for colors can be found here.

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

****Question 5: What sentiments are associated with "abandon"?**

These sentiment lexicons can be joined with your tokenized text(s) to conduct sentiment analysis by using the **inner_join** function from the **dplyr** package.

Run the following code chunk to create a new data frame connecting the **bing** sentiment lexicon to the **tidy_frankenstein** tibble you created earlier.

```r
frankenstein_bing <- tidy_frankenstein %>%
  inner_join(get_sentiments("bing"))

frankenstein_bing
```

****Question 6: Which three (3) words from the bing lexicon appear most frequently in *Frankenstein*? How many times do they each appear and what sentiment is associated with each one?**

Now, let's use **ggplot2** to produce a horizontal bar chart showing positive and negative word usage in *Frankenstein* usiong the Bing et al. sentiment lexicon.

```r
frankenstein_bing %>%
  filter(n > 25) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill=sentiment)) +
  theme_fivethirtyeight() +
  geom_col() +
  xlab(NULL) +
  coord_flip() +
  ylab("Word Count") +
  ggtitle("Word Usage in Frankenstein", subtitle = "Sentiment Analysis Using
Bing et al.")
```

We can also use the **nrc** sentiment lexicon to get a better insight into how specific emotions play a role in a given text by filtering the data and joining it to your tokenized text. To see how fear appears in Frankenstein, run the following code chunks:

```r
nrc_fear <- get_sentiments("nrc") %>%
  filter(sentiment == "fear")


nrc_fear

tidy_frankenstein %>%
  inner_join(nrc_fear)
```

****Question 7: How many words associated with fear appear in *Frankenstein*? Which fear word appears most frequently?**

Repeat this process looking at the words associated with **disgust**.

```r
nrc_disgust <- get_sentiments("nrc") %>%
  filter(sentiment == "disgust")

nrc_disgust

tidy_frankenstein %>%
  inner_join(nrc_disgust)
```

****Question 8: How many words associated with disgust appear in *Frankenstein*? Which disgust word appears most frequently?**

7. Comparing Sentiment Lexicons

In performing sentiment analysis, it might be useful to compare how the use of a given lexicon impacts the analysis. In this section we will explore how the **AFINN**, **bing**, and **nrc** lexicons compare when we examine the narrative arc of *Frankenstein.* First, we will divide the text of *Frankenstein* into its narrative sections[4] using the package **stringr**. We will then use the **tidyr** package to bind the sections to the different sentiment lexicons while manipulating the nrc and bing sentiment lexicons to match the numeric scoring used by the AFINN lexicon.

First, load the **stringr** and **tidyr** packages into your session.

```r
library(stringr)
library(tidyr)
```

Then, run the following code chunk to section off the text of *Frankenstein* according to each letter or chapter.

```r
frankenstein_sections <- frankenstein %>%
  mutate(section =
           cumsum(str_detect(text,regex("^chapter|letter [\\divxlc]",
                                         ignore_case = TRUE)))) %>%
  ungroup()%>%
  unnest_tokens(word, text) %>%
  anti_join(stop_words)

frankenstein_sections
```

Now, run the following to create a tibble summarizing the total sentiment score (according to the AFINN lexicon) per section of *Frankenstein*.

```r
frankenstein_afinn <- frankenstein_sections %>%
  inner_join(get_sentiments("afinn")) %>%
```

[4] If you've not read Frankenstein, the first parts of the book are organized as a series of written letters, while the remainder of the book is organized into chapters.

```
  group_by(index = section) %>%
  summarize(sentiment = sum(score)) %>%
  mutate(method = "AFINN")

frankenstein_afinn
```

Next, create a tibble joining the **bing** and **nrc** lexicons to **frankenstein_sections**, and converting the binary values “positive” and “negative” to a numeric score comparable to the **AFINN** lexicon.

```
frankenstein_bingnrc <- bind_rows(frankenstein_sections %>%
                                    inner_join(get_sentiments("bing")) %>%
                                    mutate(method = "Bing et al."),
                                  frankenstein_sections %>%
                                    inner_join(get_sentiments("nrc") %>%
                                                 filter(sentiment %in%
                                                          c("positive",
                                                            "negative"))) %>%
                                    mutate(method = "NRC")) %>%
  count(method, index = section, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)

frankenstein_bingnrc
```

Now we can plot the **frankenstein_afinn** and **frankenstein_bingnrc** tibbles together using **ggplot2**.

```
bind_rows(frankenstein_afinn,
          frankenstein_bingnrc) %>%
  ggplot(aes(index, sentiment, fill = method)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~method, ncol = 1, scales = "free_y") +
  geom_smooth() +
  theme_fivethirtyeight() +
  xlab("Index") +
  ylab("Sentiment Score") +
  ggtitle("Sentiment in Frankenstein per Chapter", subtitle = "Comparing Lexi
cons")
```

****Question 9: Which lexicon gives the final chapter of *Frankenstein* the most negative score?**

****Question 10: Which lexicon is shows the most positive reading of *Frankenstein*?**

Discrepancies among sentiment scores can be due to the ways in which sentiment is categorized in or scored, as well as which words appear or do not appear in each lexicon. To understand these lexicons further, you can count the number of positive and negative words included in each. To look further into **nrc**, run the following:

```
get_sentiments("nrc") %>%
  filter(sentiment %in% c("positive",
```

```r
                          "negative")) %>%
  count(sentiment)
```

Now, examine **bing**.

```r
get_sentiments("bing") %>%
  count(sentiment)
```

**Question 11: Which lexicon includes more negative words? How many negative words does it include

It is also useful to know how much each word contributes to the overall sentiment of the book. To do this, we can create and plot a tibble joining a given lexicon to the text and creating a variable *n* to indicate the word count.

To create the tibble, run the following code chunk:

```r
frank_bingcounts <- frankenstein_sections %>%
  inner_join(get_sentiments("bing")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()

frank_bingcounts
```

To plot:

```r
frank_bingcounts %>%
  group_by(sentiment) %>%
  top_n(20) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  coord_flip() +
  theme_fivethirtyeight() +
  ggtitle("Words' Contribution to Sentiment in Frankenstein", subtitle = "Usi
ng the Bing et. al Lexicon")
```

Now, let’s plot the contrubution to sentiment that positive and negative words in *Frankenstein* have using the nrc lexicon. (Note: you will have to include a filter to only include the categories “positive” and “negative.”)

Create the tibble:

```r
frank_nrccounts <- frankenstein_sections %>%
  inner_join(get_sentiments("nrc")) %>%
  filter(sentiment %in% c("positive",
                          "negative")) %>%
  count(word, sentiment, sort = TRUE) %>%
  ungroup()
```

```
frank_nrccounts
```

Plot:

```
frank_nrccounts %>%
  group_by(sentiment) %>%
  top_n(20) %>%
  ungroup() %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n, fill = sentiment)) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~sentiment, scales = "free_y") +
  coord_flip() +
  theme_fivethirtyeight() +
  ggtitle("Words' Contribution to Sentiment in Frankenstein", subtitle = "Using the NRC Lexicon")
```

8. Analyzing Queen Lyrics

In this section we will experiment with using sentiment analysis on song lyrics. To begin, we will import the Queen.csv file containing all of the lyrics to songs by the band Queen.

First, load the **Queen.csv** file into your environment.

```
queen <- read.csv("https://raw.githubusercontent.com/kayleealexander/TidyText/master/Queen.csv", stringsAsFactors = FALSE)

queen
```

Next, we will need to tokenize the lyrics, and group them per song.

```
queen <- queen %>%
  group_by(song) %>%
  ungroup()%>%
  unnest_tokens(word, text)

queen
```

Now let’s count the most common words in Queen lyrics. Create a tibble for words in Queen lyrics, sorted by word count and with stop words removed.

```
tidy_queen <- queen %>%
  count(word, sort = TRUE) %>%
  anti_join(stop_words)

tidy_queen
```

****Question 12: Which two (2) words appear most commonly in Queen lyrics? How often are they used?**

Song lyrics often include words such as “yeah” and “hey,” which you might want to exclude from your analysis.[5] You can create a custom stop words list by running the following:

```
lyric_stop_words <- bind_rows(data_frame(word =
                                           c("yeah","baby","hey", "la","oooh"
,
                                             "ah","ooh"),
                                         lexicon = c("custom")), stop_words)

lyric_stop_words
```

Use the **anti_join()** function to remove your custom stop words from **tidy_queen**.

```
tidy_queen <- tidy_queen %>%
  anti_join(lyric_stop_words)

tidy_queen
```

Now, use **ggplot2** to plot the most common words in Queen lyrics that are used more than 60 times.

```
tidy_queen %>%
  filter(n > 60) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, n)) +
  theme_fivethirtyeight() +
  geom_col() +
  xlab(NULL) +
  coord_flip() +
  ggtitle("Word Frequency in Queen Lyrics")
```

9. Analyzing Sentiment in Queen Lyrics

Now that we have an overview of the data, we can begin to apply some sentiment analysis. Use the **inner_join()** function to **create a new tibble** connecting the **queen** dataframe to both the **AFINN** lexicon *and* your existing **tidy_queen** dataframe.

```
queen_afinn <- queen %>%
  inner_join(get_sentiments("afinn")) %>%
  inner_join(tidy_queen)
```

[5] Sometimes it is useful to first visualize the most frequently occurring words in your dataset to identify these types of words that you might want to exclude. This is just a selection of some that appeared in this dataset.

```
queen_afinn
```

Now, plot this new tibble using **ggplot2** to create a horizontal bar chart showing all words in Queen lyrics occurring more than 30 times, ordered on word count and filled according to their AFINN sentiment score.

```
queen_afinn %>%
  filter(n > 30) %>%
  mutate(word = reorder(word, n)) %>%
  ggplot(aes(word, fill=score)) +
  geom_bar() +
  xlab(NULL) +
  coord_flip() +
  theme_fivethirtyeight() +
  ggtitle("Word Frequency and Sentiment in Queen Songs", subtitle = "Using th
e AFINN Lexicon")
```

Now, let’s create a data frame summarizing the AFINN sentiment score per song.

```
queen_songs <- queen_afinn %>%
  group_by(index = song) %>%
  summarize(sentiment = sum(score))
```

Then, plot this new data frame using **ggplot2**.

```
queen_songs %>%
  mutate(index = reorder(index, sentiment)) %>%
  ggplot(aes(index, sentiment, fill = sentiment)) +
  geom_col() +
  theme_fivethirtyeight() +
  xlab(NULL) +
  ylab("Total Sentiment Score") +
  ylim(-100,200) +
  theme(axis.text.x=element_blank(),
        axis.line.y = element_line(),
        panel.grid = element_blank()) +
  ggtitle("Songs by Queen",
          subtitle = "Sentiment Analysis Using the AFINN Lexicon")
```

We now see each song represented by a bar along the *x-axis*, with the *y-axis* value representing the total AFINN score per Queen song (sum of the scores given for each word in the song). It is difficult, however, to see which bars represent which songs. Let’s explore the data to get some insight:

- Click on **queen_songs** in the list of **Data** in your **Global Environment**.
- This pulls up a window showing your data frame.
- Sort the songs according to their sentiment score by clicking the arrows in the upper right corner of the cell with the column name sentiment.

You can now see the songs ordered according to their sum sentiment score. We can use this view to get a better insight into which songs in our visualization are represented in which areas of the graph. We can then add annotations to the graph to show the highest and lowest scoring songs, as well as the more neutral scoring songs to help the viewer understand better what the graph might represent.

Order the songs according to their sentiment scores by clicking on the arrows in the upper right corner of the column name.

****Question 13: Which song received a sentiment score of twelve (12)?**

Annotations need to be added to the graph by specifying the coordinates at which you would like them to appear. You can adjust the coordinates easily by playing with the values in order to get the text situated at the precise point you'd like it to. To add annotations (text and arrows) for the highest, lowest, and most neutral Queen songs to your visualization, run the following (see next page):

```
queen_songs %>%
  mutate(index = reorder(index, sentiment)) %>%
  ggplot(aes(index, sentiment, fill = sentiment)) +
  geom_col() +
  theme_fivethirtyeight() +
  xlab(NULL) +
  ylab("Total Sentiment Score") +
  ylim(-100,200) +
  theme(axis.text.x=element_blank(),
        axis.line.y = element_line(),
        panel.grid = element_blank()) +
  ggtitle("Songs by Queen",
          subtitle = "Sentiment Analysis Using the AFINN Lexicon") +
  annotate("text", label = "''How Funny Love Is''",
           size = 3, color = "black", fontface = 2,
           x = 160, y = 150) +
  annotate("segment", x = 178, xend = 190, y = 150, yend = 150,
           colour = "red",
           size=.5, arrow=arrow(length = unit(.1, "inches"))) +
  annotate("text", label = "''Liar''",
           size = 3, color = "black", fontface = 2,
           x = 18, y = -78) +
    annotate("segment", x = 13, xend = 1, y = -78, yend = -78,
           colour = "red",
           size=.5, arrow=arrow(length = unit(.1, "inches"))) +
  annotate("text", label = "''Some Day One Day''",
           size = 3, color = "black", fontface = 2,
           x = 82, y = 53) +
      annotate("segment", x = 83, xend = 83, y = 45, yend = 1,
           colour = "red",
           size=.5, arrow=arrow(length = unit(.1, "inches"))) +
  annotate("text", label = "''A Kind of Magic''",
           size = 3, color = "black", fontface = 2,
           x = 77, y = -50) +
      annotate("segment", x = 77, xend = 77, y = -45, yend = -1,
           colour = "red",
           size=.5, arrow=arrow(length = unit(.1, "inches")))
```

The visualization has now been reproduced with 8 annotations (4 arrows, 4 texts). The size, style, and position of each annotation can be adjusted by changing the values within the parenthesis for **size**, **color**, **fontface**, **x**, **xend**, **y**, and **yend**. Feel free to adjust these settings, or add additional annotations to the chart based on the data frame.