Sentiment Analysis in R

Ritu Sanjay

**Jump right in! Visualize polarity**

Sentiment analysis helps you extract an author's feelings towards a subject. This exercise will give you a taste of what's to come!

We created text\_df representing a conversation with person and text columns.

Use [**qdap**](https://www.rdocumentation.org/packages/qdap/topics/qdap)'s [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) function to score text\_df. polarity() will accept a single character object or data frame with a grouping variable to calculate a positive or negative score.

In this example you will use the [**magrittr**](https://www.rdocumentation.org/packages/magrittr/topics/magrittr) package's dollar pipe operator [**%$%**](https://www.rdocumentation.org/packages/magrittr/topics/%25%24%25?). The dollar sign forwards the data frame into polarity() and you declare a text column name or the text column *and* a grouping variable without quotes.

text\_data\_frame %$% polarity(text\_column\_name)

To create an object with the dollar sign operator:

polarity\_object <- text\_data\_frame %$%

polarity(text\_column\_name, grouping\_column\_name)

More specifically, to make a quantitative judgement about the sentiment of some text, you need to give it a score. A simple method is a positive or negative value related to a sentence, passage or a collection of documents called a corpus. Scoring with positive or negative values only is called "polarity." A useful function for extracting polarity scores is [**counts()**](https://www.rdocumentation.org/packages/qdap/topics/counts) applied to the polarity object. For a quick visual call [**plot()**](https://www.rdocumentation.org/packages/graphics/topics/plot) on the polarity() outcome.

**Instructions**

**100 XP**

* Examine the text\_df conversation data frame.
* Using [**%$%**](https://www.rdocumentation.org/packages/magrittr/topics/%25%24%25?) pass text\_df to [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) along with the column name text **without** quotes. This will print the polarity for all text.
* Create a new object datacamp\_conversation by forwarding text\_df with %$% to polarity(). Pass in text followed by the grouping person column. This will calculate polarity according to each individual person. Since it is all within parentheses the result will be printed too.
* Apply counts() to datacamp\_conversation to print the specific emotional words that were found.
* [**plot()**](https://www.rdocumentation.org/packages/graphics/topics/plot) the datacamp\_conversation.

[**Take Hint (-30 XP)**](javascript:void(0))

# Examine the text data

text\_df

# Calc overall polarity score

text\_df %$% polarity(text)

# Calc polarity score by person

(datacamp\_conversation <- text\_df %$% polarity(text, person))

# Counts table from datacamp\_conversation

counts(datacamp\_conversation)

# Plot the conversation polarity

plot(datacamp\_conversation)

**TM refresher (I)**

In the [**Text Mining: Bag of Words**](https://www.datacamp.com/courses/intro-to-text-mining-bag-of-words) course you learned that a corpus is a set of texts, and you studied some functions for preprocessing the text. To recap, one way to create a corpus is with the functions below. Even though this is a different course, sentiment analysis is part of text mining so a refresher can be helpful.

* Turn a character vector into a text source using [**VectorSource()**](https://www.rdocumentation.org/packages/tm/topics/VectorSource).
* Turn a text source into a corpus using [**VCorpus()**](https://www.rdocumentation.org/packages/tm/topics/VCorpus).
* Remove unwanted characters from the corpus using cleaning functions like [**removePunctuation()**](https://www.rdocumentation.org/packages/tm/topics/removePunctuation) and [**stripWhitespace()**](https://www.rdocumentation.org/packages/tm/topics/stripWhitespace) from tm, and [**replace\_abbreviation()**](https://www.rdocumentation.org/packages/qdap/topics/replace_abbreviation) from qdap.

In this exercise a custom clean\_corpus() function has been created using standard preprocessing functions for easier application.

clean\_corpus() accepts the output of VCorpus() and applies cleaning functions. For example:

processed\_corpus <- clean\_corpus(my\_corpus)

**Instructions**

**100 XP**

Your R session has a text vector, tm\_define, containing two small documents and the function clean\_corpus().

* Create an object called tm\_vector by applying [**VectorSource()**](https://www.rdocumentation.org/packages/tm/topics/VectorSource) to tm\_define.
* Make tm\_corpus using [**VCorpus()**](https://www.rdocumentation.org/packages/tm/topics/VCorpus) on tm\_vector.
* Use [**content()**](https://www.rdocumentation.org/packages/NLP/topics/content) to examine the contents of the first document in tm\_corpus.
  + Documents in the corpus are accessed using list syntax, so use double square brackets, e.g. [[1]].
* Clean the corpus text using the custom function clean\_corpus() on tm\_corpus. Call this new object tm\_clean.
* Examine the first document of the new tm\_clean object again to see how the text changed after clean\_corpus() was applied.

[**Take Hint (-30 XP)**](javascript:void(0))

# clean\_corpus(), tm\_define are pre-defined

clean\_corpus

tm\_define

# Create a VectorSource

tm\_vector <- VectorSource(tm\_define)

# Apply VCorpus

tm\_corpus <- VCorpus(tm\_vector)

# Examine the first document's contents

content(tm\_corpus[[1]])

# Clean the text

tm\_clean <- clean\_corpus(tm\_corpus)

# Reexamine the contents of the first doc

content(tm\_clean[[1]])

# TM refresher (II)

Now let's create a Document Term Matrix (DTM). In a DTM:

* Each row of the matrix represents a document.
* Each column is a unique word token.
* Values of the matrix correspond to an individual document's word usage.

The DTM is the basis for many bag of words analyses. Later in the course, you will also use the related Term Document Matrix (TDM). This is the transpose; that is, columns represent documents and rows represent unique word tokens.

You should construct a DTM after cleaning the corpus (using clean\_corpus()). To do so, call [**DocumentTermMatrix()**](https://www.rdocumentation.org/packages/tm/topics/DocumentTermMatrix) on the corpus object.

tm\_dtm <- DocumentTermMatrix(tm\_clean)

If you need a more in-depth refresher check out the [**Text Mining: Bag of Words**](https://www.datacamp.com/courses/intro-to-text-mining-bag-of-words) course. Hopefully these two exercises have prepared you well enough to embark on your sentiment analysis journey!

##### Instructions

**100 XP**

We've created a [**VCorpus()**](https://www.rdocumentation.org/packages/tm/topics/VCorpus) object called clean\_text containing 1000 tweets mentioning coffee. The tweets have been cleaned with the previously mentioned preprocessing steps and your goal is to create a DTM from it.

* Apply [**DocumentTermMatrix()**](https://www.rdocumentation.org/packages/tm/topics/DocumentTermMatrix) to the clean\_text corpus to create a term frequency weighted DTM called tf\_dtm .
* Change the DocumentTermMatrix() object into a simple matrix with [**as.matrix()**](https://www.rdocumentation.org/packages/dtwclust/versions/3.1.0/topics/as.matrix). Call the new object tf\_dtm\_m.
* Check the dimensions of the matrix using [**dim()**](https://www.rdocumentation.org/packages/base/topics/dim).
* Use square bracket indexing to see a subset of the matrix.
* Select rows 16 to 20, and columns 2975 to 2985
* Note the frequency value of the word "working."

[**Take Hint (-30 XP)**](javascript:void(0))

# clean\_text is pre-defined

clean\_text

# Create tf\_dtm

tf\_dtm <- DocumentTermMatrix(clean\_text)

# Create tf\_dtm\_m

tf\_dtm\_m <- as.matrix(tf\_dtm)

# Dimensions of DTM matrix

dim(tf\_dtm\_m)

# Subset part of tf\_dtm\_m for comparison

tf\_dtm\_m[16:20,2975:2985]

# Where can you observe Zipf's law?

Although Zipf observed a steep and predictable decline in word usage you may not buy into Zipf's law. You may be thinking "I know plenty of words, and have a distinctive vocabulary". That may be the case, but the same can't be said for most people! To prove it, let's construct a visual from 3 million tweets mentioning "#sb". Keep in mind that the visual doesn't follow Zipf's law perfectly, the tweets all mentioned the same hashtag so it is a bit skewed. That said, the visual you will make follows a steep decline showing a small lexical diversity among the millions of tweets. So there is some science behind using lexicons for natural language analysis!

In this exercise, you will use the package [**metricsgraphics**](https://www.rdocumentation.org/packages/metricsgraphics/). Although the author suggests using the pipe [**%>%**](https://www.rdocumentation.org/packages/magrittr/topics/%25%3E%25) operator, you will construct the graphic step-by-step to learn about the various aspects of the plot. The main function of the package [**metricsgraphics**](https://www.rdocumentation.org/packages/metricsgraphics/) is the [**mjs\_plot()**](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_plot) function which is the first step in creating a JavaScript plot. Once you have that, you can add other layers on top of the plot.

An example [**metricsgraphics**](https://www.rdocumentation.org/packages/metricsgraphics/) workflow without using the [**%>%**](https://www.rdocumentation.org/packages/magrittr/topics/%25%3E%25) operator is below:

metro\_plot <- mjs\_plot(data, x = x\_axis\_name, y = y\_axis\_name, show\_rollover\_text = FALSE)

metro\_plot <- mjs\_line(metro\_plot)

metro\_plot <- mjs\_add\_line(metro\_plot, line\_one\_values)

metro\_plot <- mjs\_add\_legend(metro\_plot, legend = c('names', 'more\_names'))

metro\_plot

##### Instructions

**100 XP**

* Use [**head()**](https://www.rdocumentation.org/packages/utils/topics/head) on sb\_words to review top words.
* Create a new column expectations by dividing the largest word frequency, freq[1], by the rank column.
* Start sb\_plot using [**mjs\_plot()**](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_plot).
  + Pass in sb\_words with x = rank and y = freq.
  + Within [**mjs\_plot()**](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_plot) set show\_rollover\_text to FALSE.
* Overwrite sb\_plot using [**mjs\_line()**](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_line) and pass in sb\_plot.
* Add to sb\_plot with [**mjs\_add\_line()**](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_add_line).  
  + Pass in the previous sb\_plot object and the vector, expectations.
* Place a legend on a new sb\_plot object using [**mjs\_add\_legend()**](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_add_legend).  
  + Pass in the previous sb\_plot object
  + The legend labels should consist of "Frequency" and "Expectation".
* Call sb\_plot to display the plot. Mouseover a point to simultaneously highlight a freq and Expectation point. The magic of JavaScript!

[**Take Hint (-30 XP)**](javascript:void(0))

# Examine sb\_words

head(sb\_words)

# Create expectations

sb\_words$expectations <- sb\_words %$%

{freq[1] / rank}

# Create metrics plot

sb\_plot <- mjs\_plot(sb\_words, x = rank, y = freq, show\_rollover\_text = FALSE)

# Add 1st line

sb\_plot <- mjs\_line(sb\_plot)

# Add 2nd line

sb\_plot <- mjs\_add\_line(sb\_plot, expectations)

# Add legend

sb\_plot <- mjs\_add\_legend(sb\_plot, legend = c("Frequency", "Expectation"))

# Display plot

sb\_plot

# Polarity on actual text

So far you have learned the basic components needed for assessing positive or negative intent in text. Remember the following points so you can feel confident in your results.

* The **subjectivity lexicon** is a predefined list of words associated with emotions or positive/negative feelings.
* You don't have to list every word in a subjectivity lexicon because **Zipf's law** describes human expression.

A quick way to get started is to use the [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) function which has a built-in subjectivity lexicon.

The function scans the text to identify words in the lexicon. It then creates a word group around the identified positive or negative subjectivity word. Within the group **valence shifters** adjust the score. Valence shifters are words that amplify or negate the emotional intent of the subjectivity word. For example, "well known" is positive while "not well known" is negative. Here "not" is a negating term and reverses the emotional intent of "well known." In contrast, "very well known" employs an amplifier increasing the positive intent.

The polarity() function then calculates a score using subjectivity terms, valence shifters and the total number of words in the passage. This exercise demonstrates a simple polarity calculation. In the next video we look under the hood of polarity() for more detail.

##### Instructions 1/2

**25 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))

Calculate the [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) of positive in a new object called pos\_score. Encase the entire call in parentheses so the output is also printed.

[**Take Hint (-7 XP)**](javascript:void(0))

##### Instructions 2/2

**75 XP**

* [2](javascript:void(0))

Manually perfrom the same polarity calculation.

* Get a word count object by calling [**counts()**](https://www.rdocumentation.org/packages/qdap/topics/counts) on the polarity object.
* All the identified subjectivity words are part of count object's list. Specifically, positive words are in $pos.words element vector. Find the number of positive words in n\_good by calling length() on the first part of the $pos.words element.
* Capture the total number of words and assign it to n\_words. This value is stored in pos\_count as the wc element.
* Deconstruct the polarity() calculation by dividing n\_good by [**sqrt()**](https://www.rdocumentation.org/packages/base/topics/MathFun) of n\_words. Compare the result to pos\_pol to the equation's result.

[**Take Hint (-22 XP)**](javascript:void(0))

# From previous step

positive <- "DataCamp courses are good for learning"

pos\_score <- polarity(positive)

# Get counts

(pos\_counts <- counts(pos\_score))

# Number of positive words

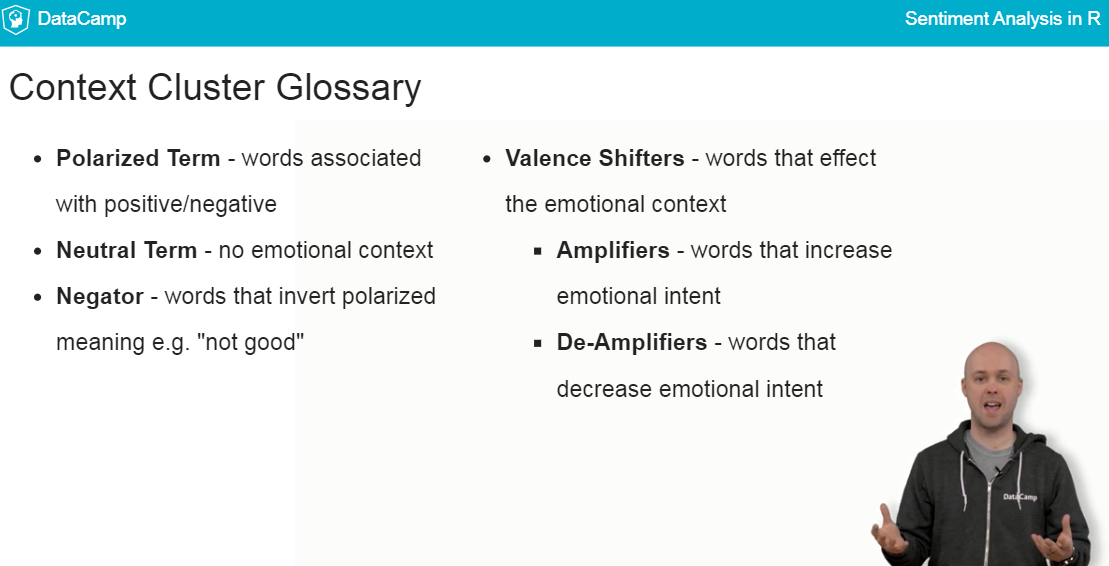
n\_good <- length(pos\_counts$pos.words[[1]])

# Total number of words

n\_words <- pos\_counts$wc

# Verify polarity score

n\_good / sqrt(n\_words)



# Happy songs!

Of course just positive and negative words aren't enough. In this exercise you will learn about valence shifters which tell you about the author's emotional intent. Previously you applied [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) to text without valence shifters. In this example you will see amplifers and negating words in action.

Recall that an **amplifying** word adds 0.8 to a positive word in polarity() so the positive score becomes 1.8. For negative words 0.8 is subtracted so the total becomes -1.8. Then the score is divided by the square root of the total number of words.

Consider the following example from Frank Sinatra:

* **"It was a very good year"**

"Good" equals 1 and "very" adds another 0.8. So, 1.8/sqrt(6) results in 0.73 polarity.

A **negating** word such as "not" will inverse the subjectivity score. Consider the following example from Bobby McFerrin:

* **"Don't worry Be Happy"**

"worry is now 1 due to the negation "don't." Adding the "happy", +1, equals 2. With 4 total words, 2 / sqrt(4) equals a polarity score of 1.

##### Instructions

**100 XP**

* Examine the conversation data frame,conversation. Note the valence shifters like "never" in the text column.
* Apply polarity() to the text column of conversation to calculate polarity for the entire conversation.
* Calculate the polarity scores by student, assigning the result to student\_pol.
  + Call polarity() again, this time passing two columns of conversation.
  + The text variable is text and the grouping variable is student.
* To see the student level results, use scores() on student\_pol.
* The counts() function applied to student\_pol will print the sentence level polarity for the entire data frame along with lexicon words identified.
* The polarity object, student\_pol, can be plotted with plot().

[**Take Hint (-30 XP)**](javascript:void(0))

# Examine conversation

conversation

# Polarity - All

polarity(conversation$text)

# Polarity - Grouped

student\_pol <- conversation %$%

polarity(conversation$text,conversation$student)

# Student results

scores(student\_pol)

# Sentence by sentence

counts(student\_pol)

# qdap plot

plot(student\_pol)

# LOL, this song is wicked good

Even with Zipf's law in action, you will still need to adjust lexicons to fit the text source (for example twitter versus legal documents) or the author's demographics (teenage girl versus middle aged man). This exercise demonstrates the explicit components of [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) so you can change it if needed.

In Trey Songz "Lol :)" song there is a lyric "LOL smiley face, LOL smiley face." In the basic polarity() function, "LOL" is not defined as positive. However, "LOL" stands for "Laugh Out Loud" and should be positive. As a result, you should adjust the lexicon to fit the text's context which includes pop-culture slang. If your analysis contains text from a specific channel (Twitter's "LOL"), location (Boston's "Wicked Good"), or age group (teenagers "sick") you will likely have to adjust the lexicon.

In this exercise you are not adjusting the subjectivity lexicon or [**qdap**](https://www.rdocumentation.org/packages/qdap/topics/qdap) dictionaries containing valence shifters. Instead you are examining the existing word data frame objects so you can change them in the following exercise.

We've created text containing two excerpts from Beyoncé's "Crazy in Love" lyrics for the exercise.

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Print [**key.pol**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/key.pol) to see a portion of the subjectivity words and values.
* Examine the predefined [**negation.words**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/negation.words) to print all the negating terms.
* Now print the [**amplification.words**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/amplification.words) to see the words that add values to the lexicon.
* Check the [**deamplification.words**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/deamplification.words) to print the words that reduce the lexicon values.
* Call text to see conversation.

[**Take Hint (-15 XP)**](javascript:void(0))

##### Instructions 2/2

**50 XP**

* [2](javascript:void(0))
* Calculate [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) as follows.
  + Set text.var to text$words.
  + Set grouping.var to text$speaker.
  + Set polarity.frame to key.pol.
  + Set negators to negation.words.
  + Set amplifiers to amplification.words.
  + Set deamplifiers to deamplification.words.

[**Take Hint (-15 XP)**](javascript:void(0))

# Examine the key.pol

key.pol

# Negators

negation.words

# Amplifiers

amplification.words

# De-amplifiers

deamplification.words

# Examine

text

Complete the polarity parameters

polarity(

text.var = text$words,

grouping.var = text$speaker,

polarity.frame = key.pol,

negators = negation.words,

amplifiers = amplification.words,

deamplifiers = deamplification.words

)

# Stressed Out!

Here you will adjust the negative words to account for the specific text. You will then compare the basic and custom polarity() scores.

A popular song from Twenty One Pilots is called "Stressed Out". If you scan the lyrics of this song, you will observe the song is about youthful nostalgia. Overall, most people would say the polarity is negative. Repeatedly the lyrics mention stress, fears and pretending.

Let's compare the song lyrics using the default subjectivity lexicon and also a custom one.

To start, you need to verify the [**key.pol**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/key.pol) subjectivity lexicon does not already have the term you want to add. One way to check is with [**grep()**](https://www.rdocumentation.org/packages/base/topics/grep). The grep() function returns the row containing characters that match a search pattern. Here is an example used while indexing.

data\_frame[grep("search\_pattern", data\_frame$column), ]

After verifying the slang or new word is not already in the [**key.pol**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/key.pol) lexicon you need to add it. The code below uses [**sentiment\_frame()**](https://www.rdocumentation.org/packages/qdap/topics/sentiment_frame) to construct the new lexicon. Within the code sentiment\_frame() accepts the original positive word vector, [**positive.words**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/positive.words). Next, the original [**negative.words**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/negative.words) are concatenated to "smh" and "kappa", both considered negative slang. Although you can declare the positive and negative weights, the default is 1 and -1 so they are not included below.

custom\_pol <- sentiment\_frame(positive.words, c(negative.words, "hate", "pain"))

Now you are ready to apply polarity and it will reference the custom subjectivity lexicon!

##### Instructions

**100 XP**

We've created stressed\_out which contains the lyrics to the song "Stressed Out", by Twenty One Pilots.

* Use [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) on stressed\_out to see the default score.
* Check [**key.pol**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/key.pol) for any words containing "stress". Use [**grep()**](https://www.rdocumentation.org/packages/base/topics/grep) to index the data frame by searching in the x column.
* Create custom\_pol as a new sentiment data frame.  
  + Call [**sentiment\_frame()**](https://www.rdocumentation.org/packages/qdap/topics/sentiment_frame) and pass [**positive.words**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/positive.words) as the first argument without concatenating any new terms.
  + Next, use [**c()**](https://www.rdocumentation.org/packages/base/topics/c) to combine [**negative.words**](https://www.rdocumentation.org/packages/qdapDictionaries/topics/negative.words) with new terms **"stressed"** and **"turn back"**.
* Reapply polarity() to stressed\_out with the additional parameter polarity.frame = custom\_pol to compare how the new words change the score to a more accurate representation of the song.

[**Take Hint (-30 XP)**](javascript:void(0))

# stressed\_out has been pre-defined

head(stressed\_out)

# Basic lexicon score

polarity(stressed\_out)

# Check the subjectivity lexicon

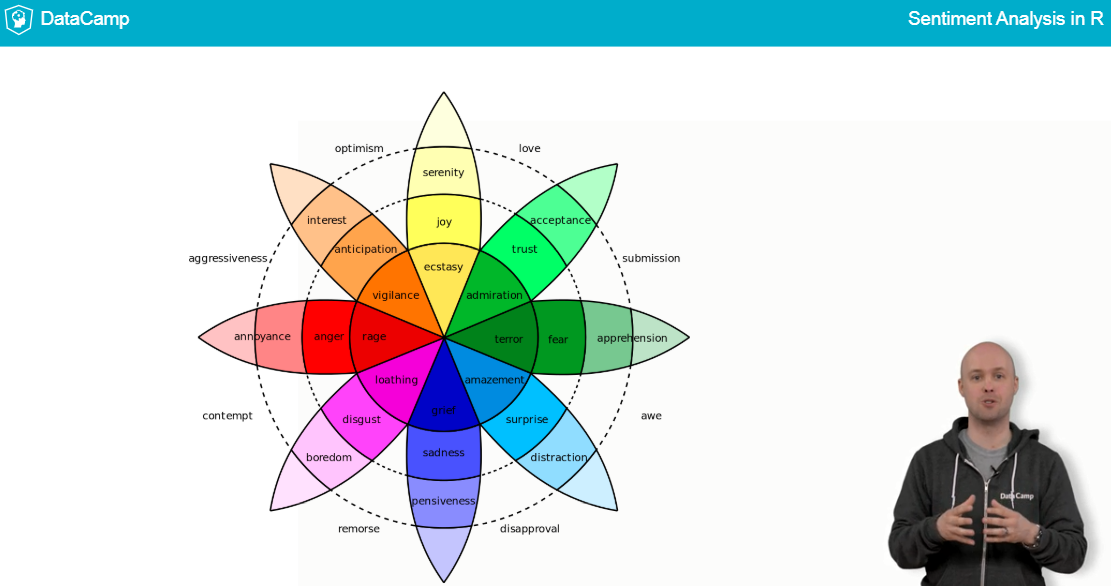
key.pol[grep("stress", x)]

# New lexicon

custom\_pol <- sentiment\_frame(positive.words, c(negative.words, "stressed", "turn back"))

# Compare new score

polarity(stressed\_out, polarity.frame = custom\_pol)



# DTM vs. tidytext matrix

The [**tidyverse**](http://tidyverse.org/) is a collection of R packages that share common philosophies and are designed to work together. This chapter covers some tidy functions to manipulate data. In this exercise you will compare a DTM to a tidy text data frame called a tibble.

Within the tidyverse, each observation is a single row in a data frame. That makes working in different packages much easier since the fundamental data structure is the same. Parts of this course borrow heavily from the [**tidytext**](https://www.rdocumentation.org/packages/tidytext) package which uses this data organization.

For example, you may already be familiar with the %>% operator from the [**magrittr**](https://www.rdocumentation.org/packages/magrittr/topics/magrittr) package. This forwards an object on its left-hand side as the first argument of the function on its right-hand side.

In the example below, you are forwarding the data object to function1(). Notice how the parentheses are empty. This in turn is forwarded to function2(). In the last function you don't have to add the data object because it was forwarded from the output of function1(). However, you do add a fictitious parameter, some\_parameter as TRUE. These pipe forwards ultimately create the object.

object <- data %>%

function1() %>%

function2(some\_parameter = TRUE)

To use the %>% operator, you don't necessarily need to load the magrittr package, since it is also available in the dplyr package. [**dplyr**](https://www.rdocumentation.org/packages/dplyr) also contains the functions [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) (which you'll learn more about later) and [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/tally) for tallying data. The last function you'll need is [**mutate()**](https://www.rdocumentation.org/packages/dplyr/topics/mutate) to create new variables or modify existing ones.

object <- data %>%

mutate(new\_Var\_name = Var1 - Var2)

or to modify a variable

object <- data %>%

mutate(Var1 = as.factor(Var1))

You will also use [**tidyr**](https://www.rdocumentation.org/packages/tidyr)'s [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) function to organize the data with each row being a line from the book and the positive and negative values as columns.

| **index** | **negative** | **positive** |
| --- | --- | --- |
| 42 | 2 | 0 |
| 43 | 0 | 1 |
| 44 | 1 | 0 |

To change a DTM to a tidy format use [**tidy()**](https://www.rdocumentation.org/packages/broom/topics/tidy) from the [**broom**](https://www.rdocumentation.org/packages/broom/) package.

tidy\_format <- tidy(Document\_Term\_Matrix)

This exercise uses text from the Greek tragedy, Agamemnon. Agamemnon is a story about marital infidelity and murder. You can download a copy [**here**](http://www.gutenberg.org/cache/epub/14417/pg14417.txt).

##### Instructions

**100 XP**

We've already created a clean DTM called ag\_dtm for this exercise.

* Create ag\_dtm\_m by applying as.matrix() to ag\_dtm.
* Using brackets, [ and ], index ag\_dtm\_m to row 2206.
* Apply [**tidy()**](https://www.rdocumentation.org/packages/broom/topics/tidy) to ag\_dtm. Call the new object ag\_tidy.
* Examine ag\_tidy at rows [831:835, ] to compare the tidy format. You will see a common word from the examined part of ag\_dtm\_m in step 2.

[**Take Hint (-30 XP)**](javascript:void(0))

# As matrix

ag\_dtm\_m <- as.matrix(ag\_dtm)

# Examine line 2206 and columns 245:250

ag\_dtm\_m[2206, 245:250]

# Tidy up the DTM

ag\_tidy <- tidy(ag\_dtm)

# Examine tidy with a word you saw

ag\_tidy[831:835, ]

# Examine the sentiments data frame

So far you have used a single lexicon. Now we will transition to using three, each measuring sentiment in different ways.

The [**tidytext**](https://www.rdocumentation.org/packages/tidytext/) package contains a data frame called [**sentiments**](https://www.rdocumentation.org/packages/tidytext/topics/sentiments). The data frame contains over 23000 terms from three different subjectivity lexicons with corresponding information. Here are some example rows from the sentiments data frame.

| **Word** | **Sentiment** | **Lexicon** | **Score** |
| --- | --- | --- | --- |
| abhorrent | NA | AFINN | -3 |
| cool | NA | AFINN | 1 |
| congenial | positive | Bing | NA |
| enemy | negative | Bing | NA |
| ungrateful | anger | NRC | NA |
| sectarian | anger | NRC | NA |

Notice the tidy format. Each word is a row and NAs fill in columns that are not applicable. The "AFINN" lexicon scores words from 5 to -5. The "Bing" lexicon is the same lexicon used in qdap's [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) function. "Bing" words are only labeled as positive or negative. The "NRC" lexicon has distinct emotional classes covering Plutchik's Wheel and positive and negative.

Let's explore the sentiments data frame in more detail!

##### Instructions 1/3

**30 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Use [**get\_sentiments()**](https://www.rdocumentation.org/packages/tidytext/topics/get_sentiments) to obtain the "afinn" lexicon, assigning to afinn\_lex.
* Review the overall [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) of score in afinn\_lex.

[**Take Hint (-9 XP)**](javascript:void(0))

##### Instructions 2/3

**30 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Do the same again, this time with the "nrc" lexicon. That is,
  + get the sentiments, assigning to nrc\_lex, then
  + count the sentiment column, assigning to nrc\_counts.

[**Take Hint (-9 XP)**](javascript:void(0))

##### Instructions 3/3

**40 XP**

* [3](javascript:void(0))
* Create a ggplot of n vs. sentiment.
* Add a col layer using geom\_col(). (This is like geom\_bar(), but used when you've already summarized with count().)

[**Take Hint (-12 XP)**](javascript:void(0))

Subset to AFINN

afinn\_lex <- get\_sentiments("afinn")

# Count AFINN scores

afinn\_lex %>%

count(score)

# Subset to nrc

nrc\_lex <- get\_sentiments("nrc")

# Make the nrc counts object

nrc\_counts <- nrc\_lex %>%

count(sentiment)

# From previous step

nrc\_counts <- get\_sentiments("nrc") %>%

count(sentiment)

# Plot n vs. sentiment

ggplot(nrc\_counts, aes(x = sentiment, y = n)) +

# Add a col layer

geom\_col() +

theme\_gdocs()

# Bing tidy polarity: Simple example

The Bing lexicon labels words as positive or negative. The next three exercises let you interact with this specific lexicon. Instead of using [**filter()**](https://www.rdocumentation.org/packages/dplyr/topics/filter) to extract a lexicon this exercise uses [**get\_sentiments()**](https://www.rdocumentation.org/packages/tidytext/topics/get_sentiments) which accepts a string such as "afinn", "bing", "nrc", or "loughran".

Now that you understand the basics of an inner join, let's apply this to the "Bing" lexicon. Keep in mind the [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) function comes from [**dplyr**](https://www.rdocumentation.org/packages/dplyr/) and the [**sentiments**](https://www.rdocumentation.org/packages/tidytext/topics/sentiments) object is from [**tidytext**](https://www.rdocumentation.org/packages/tidytext/).

The inner join workflow:

* Obtain the correct lexicon using either filter() or get\_sentiments().
* Pass the lexicon and the tidy text data to inner\_join().
* In order for inner\_join() to work there must be a shared column name. If there are no shared column names, declare them with an additional parameter, by equal to [**c**](https://www.rdocumentation.org/packages/base/topics/c) with column names like below.

object <- x %>%

inner\_join(y, by = c("column\_from\_x" = "column\_from\_y")

* Perform some aggregation and analysis on the table intersection.

##### Instructions

**100 XP**

We've loaded ag\_txt containing the first 100 lines from Agamemnon and ag\_tidy which is the tidy version.

* For comparison, use [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) on ag\_txt.
* Get the "bing" lexicon by passing that string to [**get\_sentiments()**](https://www.rdocumentation.org/packages/tidytext/topics/get_sentiments).
* Perform an [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) with ag\_tidy and bing.  
  + The word columns are called "term" in ag\_tidy and "word" in the lexicon, so you need to specify the by argument.
  + Call the new object ag\_bing\_words.
* Print ag\_bing\_words, and look at some of the words that are in the result.
* Pass ag\_bing\_words to [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) of sentiment using the pipe operator, %>%. Compare the [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) score to sentiment count ratio.

[**Take Hint (-30 XP)**](javascript:void(0))

# Qdap polarity

polarity(ag\_txt)

# Get Bing lexicon

bing <- get\_sentiments("bing")

# Join text to lexicon

ag\_bing\_words <- inner\_join(ag\_tidy, bing, by = c("term" = "word"))

# Examine

ag\_bing\_words

# Get counts by sentiment

ag\_bing\_words %>%

count(sentiment)

# Bing tidy polarity: Count & spread the white whale

In this exercise you will apply another [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) using the "bing" lexicon.

Then you will manipulate the results with both [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) from [**dplyr**](https://www.rdocumentation.org/packages/dplyr) and [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) from [**tidyr**](https://www.rdocumentation.org/packages/tidyr) to learn about the text.

The spread() function spreads a key-value pair across multiple columns. In this case key is the sentiment and the values are the frequency of positive or negative terms for each line. Using spread() changes the data so that each row now has positive and negative values, even if it is 0.

##### Instructions

**100 XP**

In this exercise, your R session has m\_dick\_tidy which contains the book Moby Dick and bing, containing the lexicon similar to the previous exercise.

* Perform an [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) on m\_dick\_tidy and bing.  
  + As before, join the "term" column in m\_dick\_tidy to the "word" column in the lexicon.
  + Call the new object moby\_lex\_words.
* Create a column index, equal to [**as.numeric()**](https://www.rdocumentation.org/packages/base/topics/numeric) applied to document. This occurs within [**mutate()**](https://www.rdocumentation.org/packages/dplyr/topics/mutate) in the tidyverse.
* Create moby\_count by forwarding moby\_lex\_words to [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count), passing in sentiment, index.
* Generate moby\_spread by piping moby\_count to spread() which contains sentiment, n, and fill = 0.

[**Take Hint (-30 XP)**](javascript:void(0))

# Inner join

moby\_lex\_words <- inner\_join(m\_dick\_tidy, bing, by = c("term" = "word"))

moby\_lex\_words <- moby\_lex\_words %>%

# Set index to numeric document

mutate(index = as.numeric(document))

moby\_count <- moby\_lex\_words %>%

# Count by sentiment, index

count(sentiment, index)

# Examine the counts

moby\_count

moby\_spread <- moby\_count %>%

# Spread sentiments

spread(sentiment, n, fill = 0)

# Review the spread data

moby\_spread

# Bing tidy polarity: Call me Ishmael (with ggplot2)!

The last Bing lexicon exercise! In this exercise you will use the pipe operator (%>%) to create a timeline of the sentiment in Moby Dick. In the end you will also create a simple visual following the code structure below. The next chapter goes into more depth for visuals.

ggplot(spread\_data, aes(index\_column, polarity\_column)) +

geom\_smooth()

##### Instructions 1/2

**75 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Inner join moby to the bing lexicon.
  + Call [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) to join the tibbles.
  + Join by the term column in the text and the word column in the lexicon.
* Count by sentiment and index.
* Reshape so that each sentiment has its own column.
  + Call [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread).
  + The key column (to split into multiple columns) is sentiment.
  + The value column (containing the counts) is n.
  + Also specify fill = 0 to fill out missing values with a zero.
* Use mutate() to add the polarity column. Define it as the difference between the positive and negative columns.

[**Take Hint (-22 XP)**](javascript:void(0))

##### Instructions 2/2

**25 XP**

* [2](javascript:void(0))
* Using moby\_polarity, plot polarity vs. index.
* Add a smooth trend layer by calling geom\_smooth() with no arguments.

[**Take Hint (-7 XP)**](javascript:void(0))

moby\_polarity <- moby %>%

# Inner join to lexicon

inner\_join(bing, by = c("term" = "word")) %>%

# Count the sentiment scores

count(sentiment, index) %>%

# Spread the sentiment into positive and negative columns

spread(sentiment, n, fill = 0) %>%

# Add polarity column

mutate(polarity = positive - negative)

# From previous step

moby\_polarity <- moby %>%

inner\_join(bing, by = c("term" = "word")) %>%

count(sentiment, index) %>%

spread(sentiment, n, fill = 0) %>%

mutate(polarity = positive - negative)

# Plot polarity vs. index

ggplot(moby\_polarity, aes(index, polarity)) +

# Add a smooth trend curve

geom\_smooth()

# AFINN: I'm your Huckleberry

Now we transition to the AFINN lexicon. The AFINN lexicon has numeric values from 5 to -5, not just positive or negative. Unlike the Bing lexicon's sentiment, the AFINN lexicon's sentiment score column is called score.

As before, you apply [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) then [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count). Next, to sum the scores of each line, we use dplyr's [**group\_by()**](https://www.rdocumentation.org/packages/dplyr/topics/group_by) and [**summarize()**](https://www.rdocumentation.org/packages/dplyr/topics/summarise) functions. The group\_by() function takes an existing data frame and converts it into a grouped data frame where operations are performed "by group". Then, the summarize() function lets you calculate a value for each group in your data frame using a function that aggregates data, like sum() or mean(). So, in our case we can do something like

data\_frame %>%

group\_by(book\_line) %>%

summarize(total\_score = sum(book\_line))

In the tidy version of Huckleberry Finn, line 9703 contains words "best", "ever", "fun", "life" and "spirit". "best" and "fun" have AFINN scores of 3 and 4 respectively. After aggregating, line 9703 will have a total score of 7.

In the tidyverse, [**filter()**](https://www.rdocumentation.org/packages/dplyr/topics/filter) is preferred to [**subset()**](https://www.rdocumentation.org/packages/base/topics/subset) because it combines the functionality of subset() with simpler syntax. Here is an example that filter()s data\_frame where some value in column1 is equal to 24. Notice the column name is not in quotes.

filter(data\_frame, column1 == 24)

The afinn object contains the AFINN lexicon. The huck object is a tidy version of Mark Twain's Adventures of Huckleberry Finn for analysis.

Line 5400 is All the loafers looked glad; I reckoned they was used to having fun out of Boggs. Stopwords and punctuation have already been removed in the dataset.

##### Instructions 1/3

**30 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Run the code to look at line 5400, and see the sentiment scores of some words.
* [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) huck to the afinn lexicon.
  + Remember huck is already piped into the function so just add the lexicon.
  + Join by the term column in the text and the word column in the lexicon.
* Use [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/tally) with score and line to tally/count observations by group.  
  + Assign the result to huck\_afinn.

[**Take Hint (-9 XP)**](javascript:void(0))

huck %>% filter(line == 5400)

# What are the scores of the sentiment words?

afinn %>% filter(word %in% c("fun", "glad"))

huck\_afinn <- huck %>%

# Inner Join to AFINN lexicon

inner\_join(afinn, by = c("term" = "word")) %>%

# Count by score and line

count(score, line)

##### Instructions 2/3

**40 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Get the total sentiment score by line forwarding huck\_afinn to [**group\_by()**](https://www.rdocumentation.org/packages/dplyr/topics/group_by) and passing line without quotes.
  + Create huck\_afinn\_agg using [**summarize()**](https://www.rdocumentation.org/packages/dplyr/topics/summarise), setting total\_score equal to the [**sum()**](https://www.rdocumentation.org/packages/base/topics/sum) of score \* n.
* Use [**filter()**](https://www.rdocumentation.org/packages/dplyr/topics/filter) on huck\_afinn\_agg and line == 5400 to review a single line.

[**Take Hint (-12 XP)**](javascript:void(0))

# From previous step

huck\_afinn <- huck %>%

inner\_join(afinn, by = c("term" = "word")) %>%

count(score, line)

huck\_afinn\_agg <- huck\_afinn %>%

# Group by line

group\_by(line) %>%

# Sum scores times n (by line)

summarize(total\_score = sum(score \* n))

huck\_afinn\_agg %>%

# Filter for line 5400

filter(line == 5400)

##### Instructions 3/3

**30 XP**

* [3](javascript:void(0))
* Create a sentiment timeline. Pass huck\_afinn\_agg to the data argument of [**ggplot()**](https://www.rdocumentation.org/packages/ggplot2/topics/ggplot).
  + Then specify the x and y within [**aes()**](https://www.rdocumentation.org/packages/ggplot2/topics/aes) as line and total\_score without quotes.
  + Add a layer with [**geom\_smooth()**](https://www.rdocumentation.org/packages/ggplot2/topics/geom_smooth).

[**Take Hint (-9 XP)**](javascript:void(0))

From previous steps

huck\_afinn\_agg <- huck %>%

inner\_join(afinn, by = c("term" = "word")) %>%

count(score, line) %>%

group\_by(line) %>%

summarize(total\_score = sum(score \* n))

# Plot total\_score vs. line

ggplot(huck\_afinn\_agg, aes(line, total\_score)) +

# Add a smooth trend curve

geom\_smooth()

# The wonderful wizard of NRC

Last but not least, you get to work with the NRC lexicon which labels words across multiple emotional states. Remember Plutchik's wheel of emotion? The NRC lexicon tags words according to Plutchik's 8 emotions plus positive/negative.

In this exercise there is a new operator, [**%in%**](https://www.rdocumentation.org/packages/base/topics/match), which matches a vector to another. In the code below %in% will return FALSE, FALSE, TRUE. This is because within some\_vec, 1 and 2 are not found within some\_other\_vector but 3 is found and returns TRUE. The %in% is useful to find matches.

some\_vec <- c(1, 2, 3)

some\_other\_vector <- c(3, "a", "b")

some\_vec %in% some\_other\_vector

Another new operator is !. For logical conditions, adding ! will inverse the result. In the above example, the FALSE, FALSE, TRUE will become TRUE, TRUE, FALSE. Using it in concert with %in% will inverse the response and is good for removing items that are matched.

!some\_vec %in% some\_other\_vector

We've created oz which is the tidy version of The Wizard of Oz along with nrc containing the "NRC" lexicon with renamed columns.

##### Instructions 1/2

**70 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Inner join oz to the nrc lexicon.
  + Call [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) to join the tibbles.
  + Join by the term column in the text and the word column in the lexicon.
* Filter to only Pluchik's emotions and drop the positive or negative words in the lexicon.  
  + Use [**filter()**](https://www.rdocumentation.org/packages/dplyr/topics/filter) to keep rows where the sentiment is not "positive" or "negative".
* Group by sentiment.
  + Call [**group\_by()**](https://www.rdocumentation.org/packages/dplyr/topics/group_by), passing sentiment without quotes.
* Get the total count of each sentiment.
  + Call [**summarize()**](https://www.rdocumentation.org/packages/dplyr/topics/summarise), setting total\_count equal to the [**sum()**](https://www.rdocumentation.org/packages/base/topics/sum) of count.
  + Assign the result to oz\_plutchik.

[**Take Hint (-21 XP)**](javascript:void(0))

oz\_plutchik <- oz %>%

# Join to nrc lexicon by term = word

inner\_join(nrc, by = c("term" = "word")) %>%

# Only consider Plutchik sentiments

filter(!sentiment %in% c("positive", "negative")) %>%

# Group by sentiment

group\_by(sentiment) %>%

# Get total count by sentiment

summarize(total\_count = sum(count))

##### Instructions 2/2

**30 XP**

* [2](javascript:void(0))
* Create a bar plot with [**ggplot()**](https://www.rdocumentation.org/packages/ggplot2/topics/ggplot).
  + Pass in oz\_plutchik to the data argument.
  + Then specify the x and y aesthetics, calling [**aes()**](https://www.rdocumentation.org/packages/ggplot2/topics/aes) and passing sentiment and total\_count without quotes.
  + Add a column geom with [**geom\_col()**](https://www.rdocumentation.org/packages/ggplot2/topics/geom_bar). (This is the same as geom\_bar(), but doesn't summarize the data, since you've done that already.)

[**Take Hint (-9 XP)**](javascript:void(0))

# From previous step

oz\_plutchik <- oz %>%

inner\_join(nrc, by = c("term" = "word")) %>%

filter(!sentiment %in% c("positive", "negative")) %>%

group\_by(sentiment) %>%

summarize(total\_count = sum(count))

# Plot total\_count vs. sentiment

ggplot(oz\_plutchik, aes(x = sentiment, y = total\_count)) +

# Add a column geom

geom\_col()

# Unhappy ending? Chronological polarity

Sometimes you want to track sentiment over time. For example, during an ad campaign you could track brand sentiment to see the campaign's effect. You saw a few examples of this at the end of the last chapter.

In this exercise you'll recap the workflow for exploring sentiment over time using the novel Moby Dick. One should expect that happy moments in the book would have more positive words than negative. Conversely dark moments and sad endings should use more negative language. You'll also see some tricks to make your sentiment time series more visually appealling.

Recall that the workflow is:

1. Inner join the text to the lexicon by word.
2. Count the sentiments by line.
3. Reshape the data so each sentiment has its own column.
4. (Depending upon the lexicon) Calculate the polarity as positive score minus negative score.
5. Draw the polarity time series.

This exercise should look familiar: it extends Bing tidy polarity: Call me Ishmael (with ggplot2)!.

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))

##### Instructions 1/2

**50 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) the pre-loaded tidy version of Moby Dick, moby, to the bing lexicon.
  + Join by the "term" column in the text and the "word" column in the lexicon.
* Count by sentiment and index.
* Reshape so that each sentiment has its own column using [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) with the column sentiment and the counts column called n.
  + Also specify fill = 0 to fill out missing values with a zero.
* Using mutate() add two columns: polarity and line\_number.
  + Set polarity equal to the positive score minus the negative score.
  + Set line\_number equal to the row number using the [**row\_number()**](https://www.rdocumentation.org/packages/dplyr/topics/ranking) function.

[**Take Hint (-15 XP)**](javascript:void(0))

##### Instructions 2/2

**50 XP**

* [2](javascript:void(0))
* Create a sentiment time series with [**ggplot()**](https://www.rdocumentation.org/packages/ggplot2/topics/ggplot).
  + Pass in moby\_polarity to the data argument.
  + Call [**aes()**](https://www.rdocumentation.org/packages/ggplot2/topics/aes) and pass in line\_number and polarity without quotes.
  + Add a smoothed curve with [**geom\_smooth()**](https://www.rdocumentation.org/packages/ggplot2/topics/geom_smooth).
  + Add a red horizontal line at zero by calling [**geom\_hline()**](https://www.rdocumentation.org/packages/ggplot2/topics/geom_abline), with parameters 0 and "red".
  + Add a title with [**ggtitle()**](https://www.rdocumentation.org/packages/ggplot2/topics/labs) set to "Moby Dick Chronological Polarity".

[**Take Hint (-15 XP)**](javascript:void(0))

# From previous step

moby\_polarity <- moby %>%

inner\_join(bing, by = c("term" = "word")) %>%

count(sentiment, index) %>%

spread(sentiment, n, fill = 0) %>%

mutate(

polarity = positive - negative,

line\_number = row\_number()

)

# Plot polarity vs. line\_number

ggplot(moby\_polarity, aes(line\_number, polarity)) +

# Add a smooth trend curve

geom\_smooth() +

# Add a horizontal line at y = 0

geom\_hline(yintercept = 0, color = "red") +

# Add a plot title

ggtitle("Moby Dick Chronological Polarity") +

theme\_gdocs()

# Word impact, frequency analysis

One of the easiest ways to explore data is with a frequency analysis. Although not difficult, in sentiment analysis this simple method can be surprisingly illuminating. Specifically, you will build a barplot. In this exercise you are once again working with moby and bing to construct your visual.

To get the bars ordered from lowest to highest, you will use a trick with factors. [**reorder()**](https://www.rdocumentation.org/packages/stats/topics/reorder.default) lets you change the order of factor levels based upon another scoring variable. In this case, you will reorder the factor variable term by the scoring variable polarity.

##### Instructions 1/3

**40 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Create moby\_tidy\_sentiment.
  + Use [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) with term, sentiment, and wt = count.
  + Pipe to [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) with sentiment, n, and fill = 0.
  + Pipe to [**mutate()**](https://www.rdocumentation.org/packages/dplyr/topics/mutate). Call the new variable polarity; calculated as positive minus negative.
* Call moby\_tidy\_sentiment to review and compare it to the previous exercise.

[**Take Hint (-12 XP)**](javascript:void(0))

##### Instructions 2/3

**30 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Use [**filter()**](https://www.rdocumentation.org/packages/dplyr/topics/filter) on moby\_tidy\_sentiment to keep rows where the absolute polarity is greater than or equal to 50. [**abs()**](https://www.rdocumentation.org/packages/base/topics/MathFun) gives you absolute values.
* [**mutate()**](https://www.rdocumentation.org/packages/dplyr/topics/mutate) a new vector pos\_or\_neg with an [**ifelse()**](https://www.rdocumentation.org/packages/base/topics/ifelse) function checking if polarity > 0 then declare the document "positive" else declare it "negative".

##### [Take Hint (-9 XP)](javascript:void(0)) Instructions 3/3

**30 XP**

* [3](javascript:void(0))
* Using moby\_tidy\_pol, plot polarity vs. term, reordered by polarity (reorder(term, polarity)), filled by pos\_or\_neg.
* Inside element\_text(), rotate the x-axis text 90 degrees by setting angle = 90 and shifting the vertical justification with vjust = -0.1.

[**Take Hint (-9 XP)**](javascript:void(0))

# From previous steps

moby\_tidy\_pol <- moby %>%

inner\_join(bing, by = c("term" = "word")) %>%

count(term, sentiment, wt = count) %>%

spread(sentiment, n, fill = 0) %>%

mutate(polarity = positive - negative) %>%

filter(abs(polarity) >= 50) %>%

mutate(

pos\_or\_neg = ifelse(polarity > 0, "positive", "negative")

)

# Plot polarity vs. (term reordered by polarity), filled by pos\_or\_neg

ggplot(moby\_tidy\_pol, aes(reorder(term,polarity),polarity, fill =pos\_or\_neg)) +

geom\_col() +

ggtitle("Moby Dick: Sentiment Word Frequency") +

theme\_gdocs() +

# Rotate text and vertically justify

theme(axis.text.x = element\_text(angle = 90, vjust = -0.1))

# Divide & conquer: Using polarity for a comparison cloud

Now that you have seen how polarity can be used to divide a corpus, let's do it! This code will walk you through dividing a corpus based on sentiment so you can peer into the informaton in subsets instead of holistically.

Your R session has oz\_pol which was created by applying [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) to "The Wonderful Wizard of Oz."

For simplicity's sake, we created a simple custom function called pol\_subsections() which will divide the corpus by polarity score. First, the function accepts a data frame with each row being a sentence or document of the corpus. The data frame is subset anywhere the polarity values are greater than or less than 0. Finally, the positive and negative sentences, non-zero polarities, are pasted with parameter collapse so that the terms are grouped into a single corpus. Lastly, the two documents are concatenated into a single vector of two distinct documents.

pol\_subsections <- function(df) {

x.pos <- subset(df$text, df$polarity > 0)

x.neg <- subset(df$text, df$polarity < 0)

x.pos <- paste(x.pos, collapse = " ")

x.neg <- paste(x.neg, collapse = " ")

all.terms <- c(x.pos, x.neg)

return(all.terms)

}

At this point you have omitted the neutral sentences and want to focus on organizing the remaining text. In this exercise we use the %>% operator again to forward objects to functions. After some simple cleaning use [**comparison.cloud()**](https://www.rdocumentation.org/packages/wordcloud/topics/comparison.cloud) to make the visual.

##### Instructions 1/3

**35 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Extract the bits you need from oz\_pol.
  + Call [**select()**](https://www.rdocumentation.org/packages/dplyr/topics/select), declaring the first column text as text.var which is the raw text. The second column polarity should refer to the polarity scores polarity.
* Now apply pol\_subsections() to oz\_df. Call the new object all\_terms.
* To create all\_corpus apply [**VectorSource()**](https://www.rdocumentation.org/packages/tm/topics/VectorSource) to all\_terms and then %>% to [**VCorpus()**](https://www.rdocumentation.org/packages/tm/topics/VCorpus).

[**Take Hint (-10 XP)**](javascript:void(0))

oz\_df <- oz\_pol$all %>%

# Select text.var as text and polarity

select(text = text.var, polarity = polarity)

# Apply custom function pol\_subsections()

all\_terms <- pol\_subsections(oz\_df)

all\_corpus <- all\_terms %>%

# Source from a vector

VectorSource() %>%

# Make a volatile corpus

VCorpus()

##### Instructions 2/3

**35 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Create a term-document matrix, all\_tdm, using [**TermDocumentMatrix()**](https://www.rdocumentation.org/packages/tm/topics/TermDocumentMatrix) on all\_corpus.  
  + Add in the parameters control = list(removePunctuation = TRUE, stopwords = stopwords(kind = "en"))).
  + Then %>% to [**as.matrix()**](https://www.rdocumentation.org/packages/base/topics/matrix) and %>% again to set\_colnames(c("positive", "negative")).

[**Take Hint (-10 XP)**](javascript:void(0))

# From previous step

all\_corpus <- oz\_pol$all %>%

select(text = text.var, polarity = polarity) %>%

pol\_subsections() %>%

VectorSource() %>%

VCorpus()

all\_tdm <- TermDocumentMatrix(all\_corpus

# Create TDM from corpus

,

control = list(

# Yes, remove the punctuation

removePunctuation = TRUE,

# Use English stopwords

stopwords = stopwords(kind = "en")

)

) %>%

# Convert to matrix

as.matrix() %>%

# Set column names

set\_colnames(c("positive", "negative"))

##### Instructions 3/3

**30 XP**

* [3](javascript:void(0))

Apply [**comparison.cloud()**](https://www.rdocumentation.org/packages/wordcloud/topics/comparison.cloud) to all\_tdm with parameters max.words = 50, and colors = c("darkgreen","darkred").

[**Take Hint (-9 XP)**](javascript:void(0))

comparison.cloud(

# Create plot from the all\_tdm matrix

all\_tdm,

# Limit to 50 words

max.words = 50,

# Use darkgreen and darkred colors

colors = c("darkgreen", "darkred"))

# Emotional introspection

In this exercise you go beyond subsetting on positive and negative language. Instead you will subset text by each of the 8 emotions in Plutchik's emotional wheel to construct a visual. With this approach you will get more clarity in word usage by mapping to a specific emotion instead of just positive or negative.

Using the [**tidytext**](https://www.rdocumentation.org/packages/tidytext/) subjectivity lexicon, "nrc", you perform an [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) with your text. The "nrc" lexicon has the 8 emotions plus positive and negative term classes. So you will have to drop positive and negative words after performing your [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join). One way to do so is with the negation, !, and [**grepl()**](https://www.rdocumentation.org/packages/base/topics/grep).

The "Global Regular Expression Print Logical" function, [**grepl()**](https://www.rdocumentation.org/packages/base/topics/grep), will return a True or False if a string pattern is identified in each row. In this exercise you will search for positive OR negative using the | operator, representing "or" as shown below. Often this straight line is above the enter key on a keyboard. Since the ! negation precedes [**grepl()**](https://www.rdocumentation.org/packages/base/topics/grep), the T or F is switched so the "positive|negative" is dropped instead of kept.

Object <- tibble %>%

filter(!grepl("positive|negative", column\_name))

Next you apply [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) on the identified words along with [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) to get the data frame organized.

[**comparison.cloud()**](https://www.rdocumentation.org/packages/wordcloud/topics/comparison.cloud) requires its input to have row names, so you'll have to convert it to a base-R data.frame, calling data.frame() with the row.names argument.

##### Instructions 1/2

**75 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) moby to nrc.
* Using [**filter()**](https://www.rdocumentation.org/packages/dplyr/topics/filter) with a negation (!) and [**grepl()**](https://www.rdocumentation.org/packages/topics/grep) search for "positive|negative". The column to search is called sentiment.
* Use [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) to count by sentiment and term.
* Reshape the data frame with [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread), passing in sentiment, n, and fill = 0.
* Convert to plain data frame with [**data.frame()**](https://www.rdocumentation.org/packages/base/topics/as.data.frame), making the term column into rownames.
* Examine moby\_tidy using [**head()**](https://www.rdocumentation.org/packages/utils/topics/head).

[**Take Hint (-22 XP)**](javascript:void(0))

##### Instructions 2/2

**25 XP**

* [2](javascript:void(0))
* Using moby\_tidy, draw a [**comparison.cloud()**](https://www.rdocumentation.org/packages/wordcloud/topics/comparison.cloud).
  + Limit to 50 words.
  + Increase the title size to 1.5.

[**Take Hint (-7 XP)**](javascript:void(0))

# From previous step

moby\_tidy <- moby %>%

inner\_join(nrc, by = c("term" = "word")) %>%

filter(!grepl("positive|negative", sentiment)) %>%

count(sentiment, term) %>%

spread(sentiment, n, fill = 0) %>%

data.frame(row.names = "term")

# Plot comparison cloud

comparison.cloud(moby\_tidy, max.words = 50, title.size = 1.5)

# Compare & contrast stacked bar chart

Another way to slice your text is to understand how much of the document(s) are made of positive or negative words. For example a restaurant review may have some positive aspects such as "the food was good" but then continue to add "the restaurant was dirty, the staff was rude and parking was awful." As a result, you may want to understand how much of a document is dedicated to positive vs negative language. In this example it would have a higher negative percentage compared to positive.

One method for doing so is to [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) the positive and negative words then divide by the number of subjectivity words identified. In the restaurant review example, "good" would count as 1 positive and "dirty," "rude," and "awful" count as 3 negative terms. A simple calculation would lead you to believe the restaurant review is 25% positive and 75% negative since there were 4 subjectivity terms.

Start by performing the [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) on a unified tidy data frame containing 4 books, Agamemnon, Oz, Huck Finn, and Moby Dick. Just like the previous exercise you will use [**filter()**](https://www.rdocumentation.org/packages/dplyr/topics/filter) and [**grepl()**](https://www.rdocumentation.org/packages/base/topics/grep).

To perform the [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) you have to group the data by book and then sentiment. For example all the positive words for Agamemnon have to be grouped then tallied so that positive words from all books are not mixed. Luckily, you can pass multiple variables into [**count()**](https://www.rdocumentation.org/packages/dplyr/topics/count) directly.

##### Instructions 1/3

**40 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Inner join all\_books to the lexicon, nrc.
* Filter to keep rows where sentiment contains "positive" or "negative". That is, use [**grepl()**](https://www.rdocumentation.org/packages/base/topics/grep) on the sentiment column, checking without the negation so that "positive|negative" are kept.
* Count by book and sentiment.

[**Take Hint (-12 XP)**](javascript:void(0))

# Review tail of all\_books

tail(all\_books)

# Count by book & sentiment

books\_sent\_count <- all\_books %>%

# Inner join to nrc lexicon

inner\_join(nrc, by = c("term" = "word")) %>%

# Keep only positive or negative

filter(grepl("positive|negative", sentiment)) %>%

# Count by book and by sentiment

count(book, sentiment)

# Review entire object

books\_sent\_count

##### Instructions 2/3

**30 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Group books\_sent\_count by line.
* Mutate to add a column named percent\_positive. This should e calculated as 100 times n divided by the sum of n.

[**Take Hint (-9 XP)**](javascript:void(0))

# From previous step

books\_sent\_count <- all\_books %>%

inner\_join(nrc, by = c("term" = "word")) %>%

filter(grepl("positive|negative", sentiment)) %>%

count(book, sentiment)

book\_pos <- books\_sent\_count %>%

# Group by book

group\_by(book) %>%

# Mutate to add % positive column

mutate(percent\_positive = 100 \* n / sum(n))

##### Instructions 3/3

**30 XP**

* [3](javascript:void(0))
* Using book\_pos, plot percent\_positive vs. book, using sentiment as the fill color.
* Add a column layer with geom\_col().

[**Take Hint (-9 XP)**](javascript:void(0))

# From previous steps

book\_pos <- all\_books %>%

inner\_join(nrc, by = c("term" = "word")) %>%

filter(grepl("positive|negative", sentiment)) %>%

count(book, sentiment) %>%

group\_by(book) %>%

mutate(percent\_positive = 100 \* n / sum(n))

# Plot percent\_positive vs. book, filled by sentiment

ggplot(book\_pos, aes(book, percent\_positive, fill = sentiment)) +

# Add a col layer

geom\_col()

# Kernel density plot

Now that you learned about a kernel density plot you can create one! Remember it's like a smoothed histogram but isn't affected by binwidth. This exercise will help you construct a kernel density plot from sentiment values.

In this exercise you will plot 2 kernel densities. One for Agamemnon and another for The Wizard of Oz. For both you will perform an [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) with the "afinn" lexicon. Recall the "afinn" lexicon has terms scored from -5 to 5. Once in a tidy format, both books will retain words and corresponding scores for the lexicon.

After that, you need to row bind the results into a larger data frame using [**bind\_rows()**](https://www.rdocumentation.org/packages/dplyr/topics/bind_rows) and create a plot with [**ggplot2**](https://www.rdocumentation.org/packages/ggplot2).

From the visual you will be able to understand which book uses more positive versus negative language. There is clearly overlap as negative things happen to Dorothy but you could infer the kernel density is demonstrating a greater probability of positive language in the Wizard of Oz compared to Agamemnon.

We've loaded ag and oz as tidy versions of Agamemnon and The Wizard of Oz respectively, and created afinn as a subset of the [**tidytext**](https://www.rdocumentation.org/packages/tidytext/) "afinn" lexicon.

##### Instructions 1/2

**70 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Inner join ag to the lexicon, afinn, assigning to ag\_afinn.
* Do the same for The Wizard of Oz. This is the same code, but starting with the oz dataset and assigning to oz\_afinn.
* Use bind\_rows() to combine ag\_afinn to oz\_afinn. Set the .id argument to "book" to create a new column with the name of each book.

[**Take Hint (-21 XP)**](javascript:void(0))

ag\_afinn <- ag %>%

# Inner join to afinn lexicon

inner\_join(afinn, by = c("term" = "word"))

oz\_afinn <- oz %>%

# Inner join to afinn lexicon

inner\_join(afinn, by = c("term" = "word"))

# Combine

all\_df <- bind\_rows(agamemnon = ag\_afinn, oz = oz\_afinn, .id = "book")

##### Instructions 2/2

**30 XP**

* [2](javascript:void(0))
* Using all\_df, plot score, using book as the fill color.
* Set the alpha transparency to 0.3.

[**Take Hint (-9 XP)**](javascript:void(0))

# From previous step

all\_df <- bind\_rows(

agamemnon = ag %>% inner\_join(afinn, by = c("term" = "word")),

oz = oz %>% inner\_join(afinn, by = c("term" = "word")),

.id = "book"

)

# Plot score, filled by book

ggplot(all\_df, aes(x = score, fill = book)) +

# Set transparency to 0.3

geom\_density(alpha = 0.3) +

theme\_gdocs() +

ggtitle("AFINN Score Densities")

# Box plot

An easy way to compare multiple distributions is with a box plot. This code will help you construct multiple box plots to make a compact visual.

In this exercise the all\_book\_polarity object is already loaded. The data frame contains two columns, book and polarity. It comprises all books with [**qdap**](https://www.rdocumentation.org/packages/qdap)'s [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) function applied. Here are the first 3 rows of the large object.

|  | **book** | **polarity** |
| --- | --- | --- |
| 14 | huck | 0.2773501 |
| 22 | huck | 0.2581989 |
| 26 | huck | -0.5773503 |

This exercise introduces [**tapply()**](https://www.rdocumentation.org/packages/base/topics/tapply) which allows you to apply functions over a ragged array. You input a vector of values and then a vector of factors. For each factor, value combination the third parameter, a function like [**min()**](https://www.rdocumentation.org/packages/base/topics/Extremes), is applied. For example here's some code with [**tapply()**](https://www.rdocumentation.org/packages/base/topics/tapply) used on two vectors.

f1 <- as.factor(c("Group1", "Group2", "Group1", "Group2"))

stat1 <- c(1, 2, 1, 2)

tapply(stat1, f1, sum)

The result is an array where Group1 has a value of 2 (1+1) and Group2 has a value of 4 (2+2).

##### Instructions

**100 XP**

* Since it's already loaded, examine the all\_book\_polarity with [**str()**](https://www.rdocumentation.org/packages/utils/topics/str).
* Using [**tapply()**](https://www.rdocumentation.org/packages/base/topics/tapply), pass in all\_book\_polarity$polarity, all\_book\_polarity$book and the [**summary()**](https://www.rdocumentation.org/packages/base/topics/summary) function. This will print the summary statistics for the 4 books in terms of their [**polarity()**](https://www.rdocumentation.org/packages/qdap/topics/polarity) scores. You would expect to see Oz and Huck Finn to have higher averages than Agamemnon or Moby Dick. Pay close attention to the median.
* Create a box plot with [**ggplot()**](https://www.rdocumentation.org/packages/ggplot2/topics/ggplot) by passing in all\_book\_polarity.
  + Aesthetics should be aes(x = book, y = polarity).
  + Using a + add the [**geom\_boxplot()**](https://www.rdocumentation.org/packages/ggplot2/topics/geom_boxplot) with col = "darkred". Pay close attention to the dark line in each box representing median.
  + Next add another layer called [**geom\_jitter()**](https://www.rdocumentation.org/packages/ggplot2/topics/geom_jitter) to add points for each of the words.

[**Take Hint (-30 XP)**](javascript:void(0))

# Examine

str(all\_book\_polarity)

# Summary by document

tapply(all\_book\_polarity$polarity, all\_book\_polarity$book, summary)

# Box plot

ggplot(all\_book\_polarity, aes(x = book, y = polarity)) +

geom\_boxplot(fill = c("#bada55", "#F00B42", "#F001ED", "#BA6E15"), col = "darkred") +

geom\_jitter(position = position\_jitter(width = 0.1, height = 0), alpha = 0.02) +

theme\_gdocs() +

ggtitle("Book Polarity")

# Radar chart

Remember [**Plutchik's wheel of emotion**](https://en.wikipedia.org/wiki/Robert_Plutchik#/media/File:Plutchik-wheel.svg)? The NRC lexicon has the 8 emotions corresponding to the first ring of the wheel. Previously you created a [**comparison.cloud()**](https://www.rdocumentation.org/packages/wordcloud/topics/comparison.cloud) according to the 8 primary emotions. Now you will create a radar chart similar to the wheel in this exercise.

A [**radarchart**](https://www.rdocumentation.org/packages/radarchart) is a two-dimensional representation of multidimensional data (at least 3). In this case the tally of the different emotions for a book are represented in the chart. Using a radar chart, you can review all 8 emotions simultaneously.

As before we've loaded the "nrc" lexicon as nrc and moby\_huck which is a combined tidy version of both Moby Dick and Huck Finn.

In this exercise you once again use a negated [**grepl()**](https://www.rdocumentation.org/packages/topics/grep) to remove "positive|negative" emotional classes from the chart. As a refresher here is an example:

object <- tibble %>%

filter(!grepl("positive|negative", column\_name))

This exercise reintroduces [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) which rearranges the tallied emotional words. As a refresher consider this raw data called datacamp.

| **people** | **food** | **like** |
| --- | --- | --- |
| Nicole | bread | 78 |
| Nicole | salad | 66 |
| Ted | bread | 99 |
| Ted | salad | 21 |

If you applied [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) as in spread(datacamp, people, like) the data looks like this.

| **food** | **Nicole** | **Ted** |
| --- | --- | --- |
| bread | 78 | 99 |
| salad | 66 | 21 |

##### Instructions 1/2

**75 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* Review moby\_huck with [**tail()**](https://www.rdocumentation.org/packages/utils/topics/head).
* [**inner\_join()**](https://www.rdocumentation.org/packages/dplyr/topics/join) moby\_huck and nrc.
* Next, filter() negating "positive|negative" in the sentiment column. Assign the result to books\_pos\_neg.
* After books\_pos\_neg is forwarded to [**group\_by()**](https://www.rdocumentation.org/packages/dplyr/topics/group_by) with book and sentiment. Then [**tally()**](https://www.rdocumentation.org/packages/dplyr/topics/tally) the object with an empty function.
* Then [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) the books\_tally by the book and n column.
* Review the scores data.

[**Take Hint (-22 XP)**](javascript:void(0))

# Review tail of moby\_huck

tail(moby\_huck)

scores <- moby\_huck %>%

# Inner join to lexicon

inner\_join(nrc, by = c("term" = "word")) %>%

# Drop positive or negative

filter(!grepl("positive|negative", sentiment)) %>%

# Count by book and sentiment

count(book, sentiment) %>%

# Spread book, using n as values

spread(book, n)

# Review scores

scores

##### Instructions 2/2

**25 XP**

* [2](javascript:void(0))

Call [**chartJSRadar()**](https://www.rdocumentation.org/packages/radarchart/topics/chartJSRadar) on scores which is an [**htmlwidget**](http://www.htmlwidgets.org/) from the [**radarchart**](https://www.rdocumentation.org/packages/radarchart) package.

[**Take Hint (-7 XP)**](javascript:void(0))

# From previous step

scores <- moby\_huck %>%

inner\_join(nrc, by = c("term" = "word")) %>%

filter(!grepl("positive|negative", sentiment)) %>%

count(book, sentiment) %>%

spread(book, n)

# JavaScript radar chart

chartJSRadar(scores)

# Treemaps for groups of documents

Often you will find yourself working with documents in groups, such as author, product or by company. This exercise lets you learn about the text while retaining the groups in a compact visual. For example, with customer reviews grouped by product you may want to explore multiple dimensions of the customer reviews at the same time. First you could calculate the polarity() of the reviews. Another dimension may be length. Document length can demonstrate the emotional intensity. If a customer leaves a short "great shoes!" one could infer they are actually less enthusiastic compared to a lengthier positive review. You may also want to group reviews by product type such as women's, men's and children's shoes. A treemap lets you examine all of these dimensions.

For text analysis, within a treemap each individual box represents a document such as a tweet. Documents are grouped in some manner such as author. The size of each box is determined by a numeric value such as number of words or letters. The individual colors are determined by a sentiment score.

After you organize the tibble, you use the [**treemap**](https://www.rdocumentation.org/packages/treemap) library containing the function [**treemap()**](https://www.rdocumentation.org/packages/treemap/treemap) to make the visual. The code example below declares the data, grouping variables, size, color and other aesthetics.

treemap(

data\_frame,

index = c("group", "individual\_document"),

vSize = "doc\_length",

vColor = "avg\_score",

type = "value",

title = "Sentiment Scores by Doc",

palette = c("red", "white", "green")

)

The pre-loaded all\_books object contains a combined tidy format corpus with 4 Shakespeare, 3 Melville and 4 Twain books. Based on the treemap you should be able to tell who writes longer books, and the polarity of the author as a whole and for individual books.

##### Instructions 1/3

**20 XP**

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))

Calculate each book's length in a new object called book\_length using count() with the book column.

[**Take Hint (-6 XP)**](javascript:void(0))

book\_length <- all\_books %>%

# Count number of words per book

count(book)

# Examine the results

book\_length

##### Instructions 2/3

**40 XP**

* [2](javascript:void(0))
* [3](javascript:void(0))
* Inner join all\_books to the lexicon, afinn.
* Group by author and book.
* Use summarize() to calculate the mean\_score as the mean() of score.
* Inner join again, this time to book\_length. Join by the book column.

[**Take Hint (-12 XP)**](javascript:void(0))

# From previous step

book\_length <- all\_books %>%

count(book)

book\_tree <- all\_books %>%

# Inner join to afinn lexicon

inner\_join(afinn, by = c("term" = "word")) %>%

# Group by author, book

group\_by(author, book) %>%

# Calculate mean book score

summarize(mean\_score = mean(score)) %>%

# Inner join by book

inner\_join(book\_length, by = "book")

# Examine the results

book\_tree

##### Instructions 3/3

**40 XP**

* [3](javascript:void(0))
* Draw a treemap, setting the following arguments.
  + Use the book\_tree from the previous step.
  + Specify the aggregation index columns as "author" and "book".
  + Specify the vertex size column, vSize, as "n".
  + Specify the vertex color column, vColor, as "mean\_score".
  + Specify a direct mapping from vColor to the palette by setting type = "value".

[**Take Hint (-12 XP)**](javascript:void(0))

# From previous steps

book\_length <- all\_books %>%

count(book)

book\_tree <- all\_books %>%

inner\_join(afinn, by = c("term" = "word")) %>%

group\_by(author, book) %>%

summarize(mean\_score = mean(score)) %>%

inner\_join(book\_length, by = "book")

treemap(

# Use the book tree

book\_tree,

# Index by author and book

index = c("author", "book"),

# Use n as vertex size

vSize = "n",

# Color vertices by mean\_score

vColor = "mean\_score",

# Draw a value type

type = "value",

title = "Book Sentiment Scores",

palette = c("red", "white", "green")

)