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" \t "_blank)**'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" \t "_blank)** 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" \t "_blank)**.
* Turn a text source into a corpus using **[VCorpus()](https://www.rdocumentation.org/packages/tm/topics/VCorpus" \t "_blank)**.
* Remove unwanted characters from the corpus using cleaning functions like **[removePunctuation()](https://www.rdocumentation.org/packages/tm/topics/removePunctuation" \t "_blank)** and **[stripWhitespace()](https://www.rdocumentation.org/packages/tm/topics/stripWhitespace" \t "_blank)** from tm, and **[replace\_abbreviation()](https://www.rdocumentation.org/packages/qdap/topics/replace_abbreviation" \t "_blank)** 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" \t "_blank)** to tm\_define.
* Make tm\_corpus using **[VCorpus()](https://www.rdocumentation.org/packages/tm/topics/VCorpus" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)**. 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/" \t "_blank)**. 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/" \t "_blank)** is the **[mjs\_plot()](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_plot" \t "_blank)** 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/" \t "_blank)** 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" \t "_blank)**.
  + Pass in sb\_words with x = rank and y = freq.
  + Within **[mjs\_plot()](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_plot" \t "_blank)** set show\_rollover\_text to FALSE.
* Overwrite sb\_plot using **[mjs\_line()](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_line" \t "_blank)** and pass in sb\_plot.
* Add to sb\_plot with **[mjs\_add\_line()](https://www.rdocumentation.org/packages/metricsgraphics/topics/mjs_add_line" \t "_blank)**.  
  + 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" \t "_blank)**.  
  + 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" \t "_blank)** 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

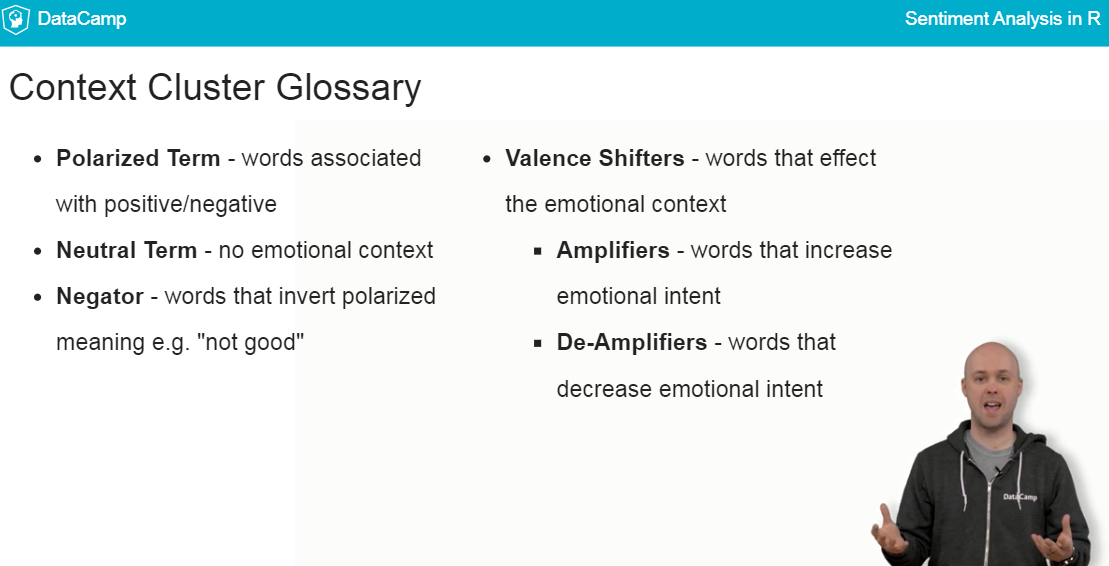
n\_good <- length(pos\_counts$pos.words[[1]])

# 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" \t "_blank)** 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" \t "_blank)** to see a portion of the subjectivity words and values.
* Examine the predefined **[negation.words](https://www.rdocumentation.org/packages/qdapDictionaries/topics/negation.words" \t "_blank)** to print all the negating terms.
* Now print the **[amplification.words](https://www.rdocumentation.org/packages/qdapDictionaries/topics/amplification.words" \t "_blank)** to see the words that add values to the lexicon.
* Check the **[deamplification.words](https://www.rdocumentation.org/packages/qdapDictionaries/topics/deamplification.words" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** lexicon you need to add it. The code below uses **[sentiment\_frame()](https://www.rdocumentation.org/packages/qdap/topics/sentiment_frame" \t "_blank)** 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" \t "_blank)**. Next, the original **[negative.words](https://www.rdocumentation.org/packages/qdapDictionaries/topics/negative.words" \t "_blank)** 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" \t "_blank)** and pass **[positive.words](https://www.rdocumentation.org/packages/qdapDictionaries/topics/positive.words" \t "_blank)** 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" \t "_blank)** 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

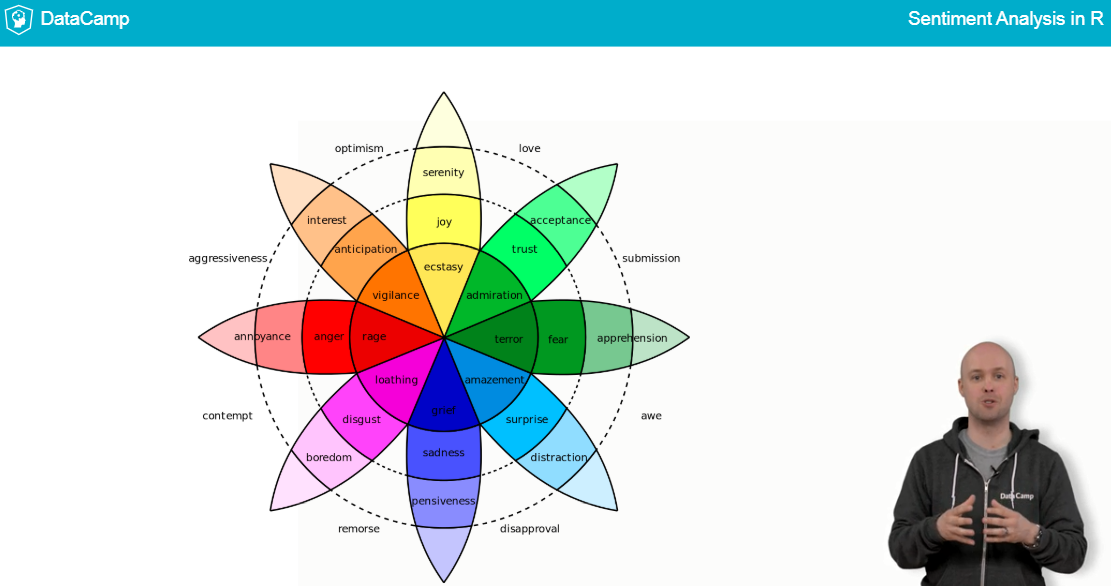
key.pol[grep("stress", x)]

# 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/" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** (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" \t "_blank)**'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/" \t "_blank)** 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" \t "_blank)** 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: Count & spread the white whale

In this exercise you will apply another **[inner\_join()](https://www.rdocumentation.org/packages/dplyr/topics/join" \t "_blank)** 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" \t "_blank)** and [**spread()**](https://www.rdocumentation.org/packages/tidyr/topics/spread) from **[tidyr](https://www.rdocumentation.org/packages/tidyr" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)** 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" \t "_blank)**.
  + Then specify the x and y within **[aes()](https://www.rdocumentation.org/packages/ggplot2/topics/aes" \t "_blank)** as line and total\_score without quotes.
  + Add a layer with **[geom\_smooth()](https://www.rdocumentation.org/packages/ggplot2/topics/geom_smooth" \t "_blank)**.

[**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" \t "_blank)** 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" \t "_blank)**, 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" \t "_blank)**.
  + 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" \t "_blank)** and passing sentiment and total\_count without quotes.
  + Add a column geom with **[geom\_col()](https://www.rdocumentation.org/packages/ggplot2/topics/geom_bar" \t "_blank)**. (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()