Sentiment analysis is a method of natural language processing that involves classifying words in a document based on whether a word is positive or negative, or whether it is related to a set of basic human emotions; the exact results differ based on the sentiment analysis method selected. The tidytext R package has 4 different sentiment analysis methods:

* “AFINN” for Finn Årup Nielsen – which classifies words from -5 to +5 in terms of negative or positive valence
* “bing” for Bing Liu and colleagues – which classifies words as either positive or negative
* “loughran” for Loughran-McDonald – mostly for financial and nonfiction works, which classifies as positive or negative, as well as topics of uncertainty, litigious, modal, and constraining
* “nrc” for the NRC lexicon – which classifies words into eight basic emotions (anger, fear, anticipation, trust, surprise, sadness, joy, and disgust) as well as positive or negative sentiment

Sentiment analysis works on unigrams – single words – but you can aggregate across multiple words to look at sentiment across a text.

To demonstrate sentiment analysis, I’ll use one of my favorite songs: “Hotel California” by the Eagles.

I know, I know.

[](https://i1.wp.com/3.bp.blogspot.com/-E8RqtLBZX1U/Wv46lGHnMBI/AAAAAAAALkQ/Ewq0bsYAUSw5ea-o4sFKKw0Nd9aGJQKmACLcBGAs/s1600/eagles.gif?ssl=1)

Using similar code as last week, let’s pull in the lyrics of the song.

library(geniusR)  
library(tidyverse)

hotel\_calif <- genius\_lyrics(artist = "Eagles", song = "Hotel California") %>%  
 mutate(line = row\_number())

First, we’ll chop up these 43 lines into individual words, using the tidytext package and unnest\_tokens function.

library(tidytext)  
tidy\_hc <- hotel\_calif %>%  
 unnest\_tokens(word,lyric)

This is also probably the point I would remove stop words with anti\_join. But these common words are very unlikely to have a sentiment attached to them, so I’ll leave them in, knowing they’ll be filtered out anyway by this analysis. We have 4 lexicons to choose from. Loughran is more financial and textual, but we’ll still see how well it can classify the words anyway. First, let’s create a data frame of our 4 sentiment lexicons.

new\_sentiments <- sentiments %>%  
 mutate( sentiment = ifelse(lexicon == "AFINN" & score >= 0, "positive",  
 ifelse(lexicon == "AFINN" & score < 0,  
 "negative", sentiment))) %>%  
 group\_by(lexicon) %>%  
 mutate(words\_in\_lexicon = n\_distinct(word)) %>%  
 ungroup()

Now, we’ll see how well the 4 lexicons match up with the words in the lyrics.:

my\_kable\_styling <- function(dat, caption) {  
 kable(dat, "html", escape = FALSE, caption = caption) %>%  
 kable\_styling(bootstrap\_options = c("striped", "condensed", "bordered"),  
 full\_width = FALSE)  
}  
  
  
library(kableExtra)  
library(formattable)  
library(yarrr)

tidy\_hc %>%  
 mutate(words\_in\_lyrics = n\_distinct(word)) %>%  
 inner\_join(new\_sentiments) %>%  
 group\_by(lexicon, words\_in\_lyrics, words\_in\_lexicon) %>%  
 summarise(lex\_match\_words = n\_distinct(word)) %>%  
 ungroup() %>%  
 mutate(total\_match\_words = sum(lex\_match\_words),  
 match\_ratio = lex\_match\_words/words\_in\_lyrics) %>%  
 select(lexicon, lex\_match\_words, words\_in\_lyrics, match\_ratio) %>%  
 mutate(lex\_match\_words = color\_bar("lightblue")(lex\_match\_words),  
 lexicon = color\_tile("lightgreen","lightgreen")(lexicon)) %>%  
 my\_kable\_styling(caption = "Lyrics Found In Lexicons")

## Joining, by = "word"

| Lyrics Found In Lexicons | | | |
| --- | --- | --- | --- |
| **lexicon** | **lex\_match\_words** | **words\_in\_lyrics** | **match\_ratio** |
| AFINN | 18 | 175 | 0.1028571 |
| bing | 18 | 175 | 0.1028571 |
| loughran | 1 | 175 | 0.0057143 |
| nrc | 23 | 175 | 0.1314286 |

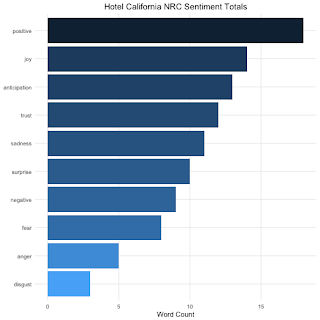
NRC offers the best match, classifying about 13% of the words in the lyrics. (It’s not unusual to have such a low percentage. Not all words have a sentiment.)

hcsentiment <- tidy\_hc %>%  
 inner\_join(get\_sentiments("nrc"), by = "word")  
  
hcsentiment

## # A tibble: 103 x 4  
## track\_title line word sentiment  
##   
## 1 Hotel California 1 dark sadness   
## 2 Hotel California 1 desert anger   
## 3 Hotel California 1 desert disgust   
## 4 Hotel California 1 desert fear   
## 5 Hotel California 1 desert negative   
## 6 Hotel California 1 desert sadness   
## 7 Hotel California 1 cool positive   
## 8 Hotel California 2 smell anger   
## 9 Hotel California 2 smell disgust   
## 10 Hotel California 2 smell negative   
## # ... with 93 more rows

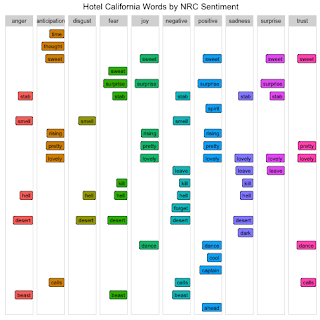
Let’s visualize the counts of different emotions and sentiments in the NRC lexicon.

theme\_lyrics <- function(aticks = element\_blank(),  
 pgminor = element\_blank(),  
 lt = element\_blank(),  
 lp = "none")  
{  
 theme(plot.title = element\_text(hjust = 0.5), #Center the title  
 axis.ticks = aticks, #Set axis ticks to on or off  
 panel.grid.minor = pgminor, #Turn the minor grid lines on or off  
 legend.title = lt, #Turn the legend title on or off  
 legend.position = lp) #Turn the legend on or off  
}  
  
hcsentiment %>%  
 group\_by(sentiment) %>%  
 summarise(word\_count = n()) %>%  
 ungroup() %>%  
 mutate(sentiment = reorder(sentiment, word\_count)) %>%  
 ggplot(aes(sentiment, word\_count, fill = -word\_count)) +  
 geom\_col() +  
 guides(fill = FALSE) +  
 theme\_minimal() + theme\_lyrics() +  
 labs(x = NULL, y = "Word Count") +  
 ggtitle("Hotel California NRC Sentiment Totals") +  
 coord\_flip()

[](https://i1.wp.com/1.bp.blogspot.com/-S79qwh3n2_U/Wv46zQIK0rI/AAAAAAAALkU/td8fTe2DtfcYeSeF4GGfyTQhXMhrp7ljACLcBGAs/s1600/unnamed-chunk-6-1.png?ssl=1)

Most of the words appear to be positively-valenced. How do the individual words match up?

library(ggrepel)  
  
plot\_words <- hcsentiment %>%  
 group\_by(sentiment) %>%  
 count(word, sort = TRUE) %>%  
 arrange(desc(n)) %>%  
 ungroup()  
  
plot\_words %>%  
 ggplot(aes(word, 1, label = word, fill = sentiment)) +  
 geom\_point(color = "white") +  
 geom\_label\_repel(force = 1, nudge\_y = 0.5,  
 direction = "y",  
 box.padding = 0.04,  
 segment.color = "white",  
 size = 3) +  
 facet\_grid(~sentiment) +  
 theme\_lyrics() +  
 theme(axis.text.y = element\_blank(), axis.line.x = element\_blank(),  
 axis.title.x = element\_blank(), axis.text.x = element\_blank(),  
 axis.ticks.x = element\_blank(),  
 panel.grid = element\_blank(), panel.background = element\_blank(),  
 panel.border = element\_rect("lightgray", fill = NA),  
 strip.text.x = element\_text(size = 9)) +  
 xlab(NULL) + ylab(NULL) +  
 ggtitle("Hotel California Words by NRC Sentiment") +  
 coord\_flip()

[](https://i1.wp.com/1.bp.blogspot.com/-EtATflHXVSs/Wv464JMmxQI/AAAAAAAALkc/bCLO-HkdCr8Oau2phcTVN9F89riDTLubQCLcBGAs/s1600/unnamed-chunk-7-1.png?ssl=1)

It looks like some words are being misclassified. For instance, “smell” as in “warm smell of colitas” is being classified as anger, disgust, and negative. But that doesn’t explain the overall positive bent being applied to the song. If you listen to the song, you know it’s not really a happy song. It starts off somewhat negative – or at least, ambiguous – as the narrator is driving on a dark desert highway. He’s tired and having trouble seeing, and notices the Hotel California, a shimmering oasis on the horizon. He stops in and is greated by a “lovely face” in a “lovely place.” At the hotel, everyone seems happy: they dance and drink, they have fancy cars, they have pretty “friends.”

But the song is in a minor key. Though not always a sign that a song is sad, it is, at the very least, a hint of something ominous, lurking below the surface. Soon, things turn bad for the narrator. The lovely-faced woman tells him they are “just prisoners here of our own device.” He tries to run away, but the night man tells him, “You can check out anytime you like, but you can never leave.”

The song seems to be a metaphor for something, perhaps fame and excess, which was also the subject of another song on the same album, “Life in the Fast Lane.” To someone seeking fame, life is dreary, dark, and deserted. Fame is like an oasis – beautiful and shimmering, an escape. But it isn’t all it appears to be. You may be surrounded by beautiful people, but you can only call them “friends.” You trust no one. And once you join that lifestyle, you might be able to check out, perhaps through farewell tour(s), but you can never leave that life – people know who you are (or were) and there’s no disappearing. And it could be about something even darker that it’s hard to escape from, like substance abuse. Whatever meaning you ascribe to the song, the overall message seems to be that things are not as wonderful as they appear on the surface.

So if we follow our own understanding of the song’s trajectory, we’d say it starts off somewhat negatively, becomes positive in the middle, then dips back into the negative at the end, when the narrator tries to escape and finds he cannot.

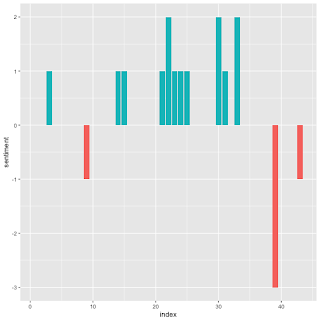
We can chart this, using the line number, which coincides with the location of the word in the song. We’ll stick with NRC since it offered the best match, but for simplicity, we’ll only pay attention to the positive and negative sentiment codes.

hcsentiment\_index <- tidy\_hc %>%  
 inner\_join(get\_sentiments("nrc")%>%  
 filter(sentiment %in% c("positive",  
 "negative"))) %>%  
 count(index = line, sentiment) %>%  
 spread(sentiment, n, fill = 0) %>%  
 mutate(sentiment = positive - negative)

## Joining, by = "word"

This gives us a data frame that aggregates sentiment by line. If a line contains more positive than negative words, its overall sentiment is positive, and vice versa. Because not every word in the lyrics has a sentiment, not every line has an associated aggregate sentiment. But it gives us a sort of trajectory over the course of the song. We can visualize this trajectory like this:

hcsentiment\_index %>%  
 ggplot(aes(index, sentiment, fill = sentiment > 0)) +  
 geom\_col(show.legend = FALSE)

[](https://i2.wp.com/4.bp.blogspot.com/-GDjSB5zx3eY/Wv468cfK7kI/AAAAAAAALkg/foyTNo0VmCEk8C5EM69vFDL6YQ5dwlVWQCLcBGAs/s1600/unnamed-chunk-9-1.png?ssl=1)

As the chart shows, the song starts somewhat positive, with a dip soon after into the negative. The middle of the song is positive, as the narrator describes the decadence of the Hotel California. But it turns dark at the end, and stays that way as the guitar solo soars in.

Sentiment Analysis in R

### Sentiment Analysis Overview

* Methods: Sentiment analysis is a type of text mining which aims to determine the opinion and subjectivity of its content. When applied to lyrics, the results can be representative of not only the artist's attitudes, but can also reveal pervasive, cultural influences. There are different methods used for sentiment analysis, including training a known dataset, creating your own classifiers with rules, and using predefined lexical dictionaries (lexicons). In this tutorial, you will use the lexicon-based approach, but I would encourage you to investigate the other methods as well as their associated trade-offs.
* Levels: Just as there are different methods used for sentiment analysis, there are also different levels of analysis based on the text. These levels are typically identified as **document**, **sentence**, and **word**. In lyrics, the document could be defined as sentiment per decade, year, chart-level, or song. The sentence level is not usually an option with lyrics as punctuation can detract from rhymes and patterns. Word level analysis exposes detailed information and can be used as foundational knowledge for more advanced practices in topic modeling.

### Prepare Your Questions!

Every data science project needs to have a set of questions to explore. Here are a few to keep in mind as you work through this tutorial: is it possible to write a program to determine the mood expressed in lyrics? Are predefined lexicons sufficient? How much data preparation is necessary? Can you link your results to real-life events? Does sentiment change over time? Are hit songs more positive or negative than uncharted songs? What words stand out in the lyrics during the year Prince was said to predict 9/11? Did he predict his own death?

**Tip**: lyrical analysis is very different and typically more complex than speeches or books, making it difficult to capture insights, so remember to be cautious of all assumptions! I will propose more questions than answers in this tutorial, so be prepared to think outside of the quadrilateral parallelogram!

## Prep Work

### Libraries and Functions

To get started analyzing Prince's lyrics, load the libraries below. These may seem daunting at first, but most of them are simply for graphs and charts. Given the frequent use of visuals, it's preferable to define a standard color scheme for consistency. I've created a list using specific ANSI color codes. The theme() function from ggplot2 allows customization of individual graphs, so you will also create your own function, theme\_lyrics(), that will modify the default settings. The knitr package is an engine for dynamic report generation with R. Use it along with kableExtra and formattable to create attractive text tables with color. Again create your own function, my\_kable\_styling() to standardize the resulting output of these libraries. I will mention the other packages as they are utilized.

library(dplyr) #Data manipulation (also included in the tidyverse package)

library(tidytext) #Text mining

library(tidyr) #Spread, separate, unite, text mining (also included in the tidyverse package)

library(widyr) #Use for pairwise correlation

#Visualizations!

library(ggplot2) #Visualizations (also included in the tidyverse package)

library(ggrepel) #`geom\_label\_repel`

library(gridExtra) #`grid.arrange()` for multi-graphs

library(knitr) #Create nicely formatted output tables

library(kableExtra) #Create nicely formatted output tables

library(formattable) #For the color\_tile function

library(circlize) #Visualizations - chord diagram

library(memery) #Memes - images with plots

library(magick) #Memes - images with plots (image\_read)

library(yarrr) #Pirate plot

library(radarchart) #Visualizations

library(igraph) #ngram network diagrams

library(ggraph) #ngram network diagrams

#Define some colors to use throughout

my\_colors <- c("#E69F00", "#56B4E9", "#009E73", "#CC79A7", "#D55E00", "#D65E00")

#Customize ggplot2's default theme settings

#This tutorial doesn't actually pass any parameters, but you may use it again in future tutorials so it's nice to have the options

theme\_lyrics <- function(aticks = element\_blank(),

pgminor = element\_blank(),

lt = element\_blank(),

lp = "none")

{

theme(plot.title = element\_text(hjust = 0.5), #Center the title

axis.ticks = aticks, #Set axis ticks to on or off

panel.grid.minor = pgminor, #Turn the minor grid lines on or off

legend.title = lt, #Turn the legend title on or off

legend.position = lp) #Turn the legend on or off

}

#Customize the text tables for consistency using HTML formatting

my\_kable\_styling <- function(dat, caption) {

kable(dat, "html", escape = FALSE, caption = caption) %>%

kable\_styling(bootstrap\_options = c("striped", "condensed", "bordered"),

full\_width = FALSE)

}

### Sushi Data

As always, you'll start by reading the raw data into a data frame. In , you performed some data conditioning on the original dataset, such as expanding contractions, removing escape sequences, and converting text to lower case. You saved that clean dataset to a CSV file for use in this tutorial. Use read.csv() to create prince\_data and keep in mind that you want your lyrics to be character strings, so make sure to set stringsAsFactors = FALSE. Since you don't need the rows numbered, set row.names = 1.

**Note** that you could avoid these parameters by reading this into a modern data frame called a tibble, using read\_csv(), but for consistency with , read.csv() is used instead.

prince\_data <- read.csv('prince\_new.csv', stringsAsFactors = FALSE, row.names = 1)

Take a quick peek at the data:

glimpse(prince\_data) #Transposed version of `print()`

## Observations: 824

## Variables: 10

## $ lyrics <chr> "all 7 and we will watch them fall they stand in t...

## $ song <chr> "7", "319", "1999", "2020", "3121", "7779311", "u"...

## $ year <int> 1992, NA, 1982, NA, 2006, NA, NA, NA, NA, NA, NA, ...

## $ album <chr> "Symbol", NA, "1999", "Other Songs", "3121", NA, N...

## $ peak <int> 3, NA, 2, NA, 1, NA, NA, NA, NA, NA, NA, NA, NA, N...

## $ us\_pop <chr> "7", NA, "12", NA, "1", NA, NA, NA, NA, NA, NA, NA...

## $ us\_rnb <chr> "61", NA, "4", NA, "1", NA, NA, NA, NA, NA, NA, NA...

## $ decade <chr> "1990s", NA, "1980s", NA, "2000s", NA, NA, NA, NA,...

## $ chart\_level <chr> "Top 10", "Uncharted", "Top 10", "Uncharted", "Top...

## $ charted <chr> "Charted", "Uncharted", "Charted", "Uncharted", "C...

You can see that prince\_data is a data frame of 824 songs and 10 columns. This means that a record is a song, literally!

### Good Clean Fun: prince\_tidy

Here are a few remaining data wrangling steps:

* Remove undesirable words (manual list of unnecessary words)
* Remove stop words (overly common words such as "and", "the", "a", "of", etc.)
* Remove words with fewer than three characters (often used for phonetic effect in music)
* Split the lyrics into individual words

In order to turn your raw data into a tidy format, use unnest\_tokens() from tidytext to create prince\_tidy which breaks out the lyrics into individual words with one word per row. Then, you can use anti\_join() and filter() from dplyr for the remaining cleaning steps.

#Created in the first tutorial

undesirable\_words <- c("prince", "chorus", "repeat", "lyrics",

"theres", "bridge", "fe0f", "yeah", "baby",

"alright", "wanna", "gonna", "chorus", "verse",

"whoa", "gotta", "make", "miscellaneous", "2",

"4", "ooh", "uurh", "pheromone", "poompoom", "3121",

"matic", " ai ", " ca ", " la ", "hey", " na ",

" da ", " uh ", " tin ", " ll", "transcription",

"repeats", "la", "da", "uh", "ah")

#Create tidy text format: Unnested, Unsummarized, -Undesirables, Stop and Short words

prince\_tidy <- prince\_data %>%

unnest\_tokens(word, lyrics) %>% #Break the lyrics into individual words

filter(!word %in% undesirable\_words) %>% #Remove undesirables

filter(!nchar(word) < 3) %>% #Words like "ah" or "oo" used in music

anti\_join(stop\_words) #Data provided by the tidytext package

glimpse(prince\_tidy) #From `dplyr`, better than `str()`.

## Observations: 76,116

## Variables: 10

## $ song <chr> "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", ...

## $ year <int> 1992, 1992, 1992, 1992, 1992, 1992, 1992, 1992, 19...

## $ album <chr> "Symbol", "Symbol", "Symbol", "Symbol", "Symbol", ...

## $ peak <int> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,...

## $ us\_pop <chr> "7", "7", "7", "7", "7", "7", "7", "7", "7", "7", ...

## $ us\_rnb <chr> "61", "61", "61", "61", "61", "61", "61", "61", "6...

## $ decade <chr> "1990s", "1990s", "1990s", "1990s", "1990s", "1990...

## $ chart\_level <chr> "Top 10", "Top 10", "Top 10", "Top 10", "Top 10", ...

## $ charted <chr> "Charted", "Charted", "Charted", "Charted", "Chart...

## $ word <chr> "watch", "fall", "stand", "love", "smoke", "intell...

Your new dataset, prince\_tidy, is now in a tokenized format with one word per row along with the song from which it came. You now have a data frame of 76116 words and 10 columns.

## Descriptive Statistics

you may need to take a quick look at a few summary graphs of the full dataset. You'll do this using creative graphs from the ggplot2, circlize, and yarrr packages.

### Shipshape: Word Count Per Song

A pirate would say shipshape when everything is in good order, tidy and clean. So here is an interesting view of the clean and tidy data showing the lexical diversity, or, in other words, vocabulary, of the lyrics over time. A pirate plot is an advanced method of plotting a continuous dependent variable, such as the word count, as a function of a categorical independent variable, like decade. This combines raw data points, descriptive and inferential statistics into a single effective plot. Check out this great blog for more details about pirateplot() from the yarrr package.

Create the word\_summary data frame that calculates the distinct word count per song. The more diverse the lyrics, the larger the vocabulary. Thinking about the data in this way gets you ready for word level analysis. Reset the decade field to contain the value "NONE" for songs without a release date and relabel those fields with cleaner labels using select().

word\_summary <- prince\_tidy %>%

mutate(decade = ifelse(is.na(decade),"NONE", decade)) %>%

group\_by(decade, song) %>%

mutate(word\_count = n\_distinct(word)) %>%

select(song, Released = decade, Charted = charted, word\_count) %>%

distinct() %>% #To obtain one record per song

ungroup()

pirateplot(formula = word\_count ~ Released + Charted, #Formula

data = word\_summary, #Data frame

xlab = NULL, ylab = "Song Distinct Word Count", #Axis labels

main = "Lexical Diversity Per Decade", #Plot title

pal = "google", #Color scheme

point.o = .2, #Points

avg.line.o = 1, #Turn on the Average/Mean line

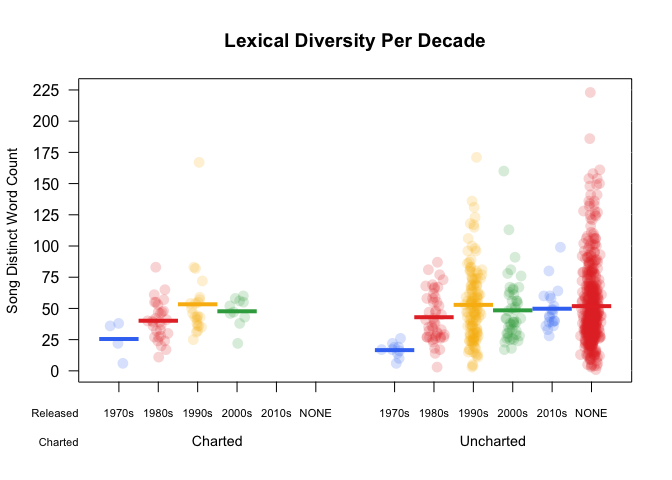
theme = 0, #Theme

point.pch = 16, #Point `pch` type

point.cex = 1.5, #Point size

jitter.val = .1, #Turn on jitter to see the songs better

cex.lab = .9, cex.names = .7) #Axis label size



Every colored circle in this pirate plot represents a song. The dense red area with the "NONE" value shows that a large number of songs in the dataset do not have a release date. There is a slight upward trend in the unique number of words per song in the early decades: the solid horizontal line shows the mean word count for that decade. This is important to know when you begin to analyze the sentiment over time. The words become more illuminating throughout Prince's career.

I would challenge you to explore pirate plots in more detail as you've only touched the surface with this one!

### All Year Round: Song Count Per Year

Circular graphs are a unique way to visualize complicated (or simple!) relationships among several categories. (Plus, albums used to be round!). The graph below is simply a circular bar chart using coord\_polar() from ggplot2 that shows the relative number of songs per year. It's a ton of code to produce such a simple graph, but it's totally worth it. A similar example can be found in more detail in the R Graph Gallery here. As with pirate plots, this is only an introduction to what you can do with circular plots.

**Song Count Per Year**

songs\_year <- prince\_data %>%

select(song, year) %>%

group\_by(year) %>%

summarise(song\_count = n())

id <- seq\_len(nrow(songs\_year))

songs\_year <- cbind(songs\_year, id)

label\_data = songs\_year

number\_of\_bar = nrow(label\_data) #Calculate the ANGLE of the labels

angle = 90 - 360 \* (label\_data$id - 0.5) / number\_of\_bar #Center things

label\_data$hjust <- ifelse(angle < -90, 1, 0) #Align label

label\_data$angle <- ifelse(angle < -90, angle + 180, angle) #Flip angle

ggplot(songs\_year, aes(x = as.factor(id), y = song\_count)) +

geom\_bar(stat = "identity", fill = alpha("purple", 0.7)) +

geom\_text(data = label\_data, aes(x = id, y = song\_count + 10, label = year, hjust = hjust), color = "black", alpha = 0.6, size = 3, angle = label\_data$angle, inherit.aes = FALSE ) +

coord\_polar(start = 0) +

ylim(-20, 150) + #Size of the circle

theme\_minimal() +

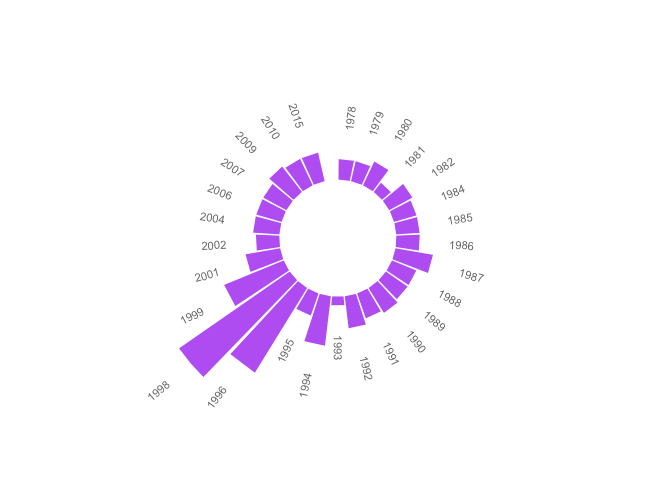
theme(axis.text = element\_blank(),

axis.title = element\_blank(),

panel.grid = element\_blank(),

plot.margin = unit(rep(-4,4), "in"),

plot.title = element\_text(margin = margin(t = 10, b = -10)))



See the gap at the very top? This is an indicator of the hundreds of songs without release dates you identified in the pirate plot. With such a large number of unreleased songs, the chart would not be useful if they were included. The missing years indicate those where no songs were released. The most prolific years were 1996 and 1998.

Do you want to know why? Keep reading!

### Chords: Charted Songs By Decade

The following graph shows the relationship between the decade a song was released and whether or not it hit the Billboard charts. Using a chordDiagram() for musical analysis just seemed appropriate! This graphical tool is from the beautiful circlize package by Zuguang Gu. The graph is split into two categories: charted (top), and decade (bottom). The two categories are separated by wide gaps, with smaller gaps between the values. The high-level comments in the code below can be supplemented with more detail here.

decade\_chart <- prince\_data %>%

filter(decade != "NA") %>% #Remove songs without release dates

count(decade, charted) #Get SONG count per chart level per decade. Order determines top or bottom.

circos.clear() #Very important - Reset the circular layout parameters!

grid.col = c("1970s" = my\_colors[1], "1980s" = my\_colors[2], "1990s" = my\_colors[3], "2000s" = my\_colors[4], "2010s" = my\_colors[5], "Charted" = "grey", "Uncharted" = "grey") #assign chord colors

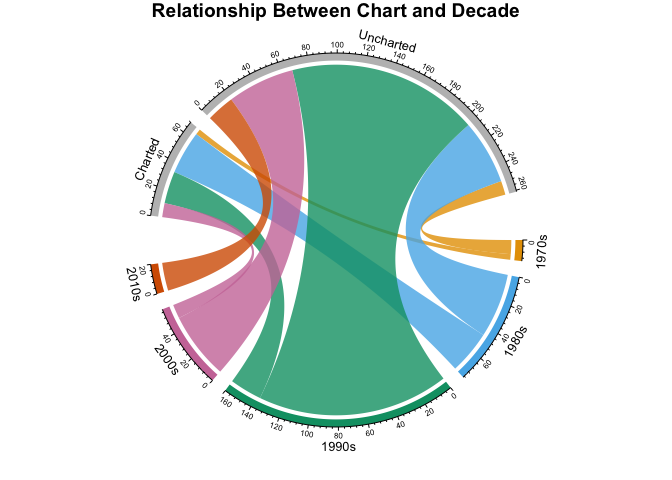
# Set the global parameters for the circular layout. Specifically the gap size

circos.par(gap.after = c(rep(5, length(unique(decade\_chart[[1]])) - 1), 15,

rep(5, length(unique(decade\_chart[[2]])) - 1), 15))

chordDiagram(decade\_chart, grid.col = grid.col, transparency = .2)

title("Relationship Between Chart and Decade")



The above circle graph may seem complex at first glance, but it nicely illustrates the counts of songs per decade, per chart level. You can see that Prince began his career in the 1970s with only a few releases, some of which charted. If you compare the 1980s to the 1990s, you'll find that more songs were released in the 1990s, but more songs charted in the 1980s. There were only a few commercially successful songs in the 2000s and in the 2010s there were no hit songs.

## Lexicons and Lyrics

In this section you will:

* Explore the sentiment lexicons and their word counts
* Identify how many words from the lyrics exist in the lexicons
* Look at specific words and word forms
* Consider additional data conditioning

### Explore Sentiment Lexicons

The tidytext package includes a dataset called sentiments which provides several distinct lexicons. These lexicons are dictionaries of words with an assigned sentiment category or value. tidytext provides three general purpose lexicons:

* **AFINN:** assigns words with a score that runs between -5 and 5, with negative scores indicating negative sentiment and positive scores indicating positive sentiment
* **Bing:** assigns words into positive and negative categories
* **NRC:** assigns words into one or more of the following ten categories: positive, negative, anger, anticipation, disgust, fear, joy, sadness, surprise, and trust

In order to examine the lexicons, create a data frame called new\_sentiments. Filter out a financial lexicon, create a binary (also described as polar) sentiment field for the AFINN lexicon by converting the numerical score to positive or negative, and add a field that holds the distinct word count for each lexicon.

new\_sentiments has one column with the different sentiment categories, so for a better view of the word counts per lexicon, per category, use spread() from tidyr to pivot those categories into separate fields. Take advantage of the knitr and kableExtra packages in the my\_kable\_styling() function you created earlier. Add color to your chart using color\_tile() and color\_bar() from formattable to create a nicely formatted table. Print your table and examine the differences between each lexicon.

new\_sentiments <- sentiments %>% #From the tidytext package

filter(lexicon != "loughran") %>% #Remove the finance lexicon

mutate( sentiment = ifelse(lexicon == "AFINN" & score >= 0, "positive",

ifelse(lexicon == "AFINN" & score < 0,

"negative", sentiment))) %>%

group\_by(lexicon) %>%

mutate(words\_in\_lexicon = n\_distinct(word)) %>%

ungroup()

new\_sentiments %>%

group\_by(lexicon, sentiment, words\_in\_lexicon) %>%

summarise(distinct\_words = n\_distinct(word)) %>%

ungroup() %>%

spread(sentiment, distinct\_words) %>%

mutate(lexicon = color\_tile("lightblue", "lightblue")(lexicon),

words\_in\_lexicon = color\_bar("lightpink")(words\_in\_lexicon)) %>%

my\_kable\_styling(caption = "Word Counts Per Lexicon")

| Word Counts Per Lexicon | | | | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **lexicon** | **words\_in\_lexicon** | **anger** | **anticipation** | **disgust** | **fear** | **joy** | **negative** | **positive** | **sadness** | **surprise** | **trust** |
| AFINN | 2476 | NA | NA | NA | NA | NA | 1597 | 879 | NA | NA | NA |
| bing | 6785 | NA | NA | NA | NA | NA | 4782 | 2006 | NA | NA | NA |
| nrc | 6468 | 1247 | 839 | 1058 | 1476 | 689 | 3324 | 2312 | 1191 | 534 | 1231 |

The table above gives you an idea of the size and structure of each lexicon.

### Match Dot Common

In order to determine which lexicon is more applicable to the lyrics, you'll want to look at the match ratio of words that are common to both the lexicon and the lyrics. As a reminder, there are 76116 total words and 7851 distinct words in prince\_tidy.

So how many of those words are actually in the lexicons?

Use an inner\_join() between prince\_tidy and new\_sentiments and then group by lexicon. The NRC lexicon has 10 different categories, and a word may appear in more than one category: that is, words can be negative and sad. That means that you'll want to use n\_distinct() in summarise() to get the distinct word count per lexicon.

prince\_tidy %>%

mutate(words\_in\_lyrics = n\_distinct(word)) %>%

inner\_join(new\_sentiments) %>%

group\_by(lexicon, words\_in\_lyrics, words\_in\_lexicon) %>%

summarise(lex\_match\_words = n\_distinct(word)) %>%

ungroup() %>%

mutate(total\_match\_words = sum(lex\_match\_words), #Not used but good to have

match\_ratio = lex\_match\_words / words\_in\_lyrics) %>%

select(lexicon, lex\_match\_words, words\_in\_lyrics, match\_ratio) %>%

mutate(lex\_match\_words = color\_bar("lightpink")(lex\_match\_words),

lexicon = color\_tile("lightgreen", "lightgreen")(lexicon)) %>%

my\_kable\_styling(caption = "Lyrics Found In Lexicons")

| Lyrics Found In Lexicons | | | |
| --- | --- | --- | --- |
| **lexicon** | **lex\_match\_words** | **words\_in\_lyrics** | **match\_ratio** |
| AFINN | 770 | 7851 | 0.0980767 |
| bing | 1185 | 7851 | 0.1509362 |
| nrc | 1678 | 7851 | 0.2137307 |

The NRC lexicon has more of the distinct words from the lyrics than AFINN or Bing. Notice the sum of the match ratios is low. No lexicon could have all words, nor should they. Many words are considered neutral and would not have an associated sentiment. For example, 2000 is typically a neutral word, and therefore does not exist in the lexicons. However, if you remember, people predicted planes would fall out of the sky and computers would just stop working during that year. So there is an associated fear that exists in the song but is not captured in sentiment analysis using typical lexicons.

Here are a few reasons that a word may not appear in a lexicon:

* Not every word has a sentiment.
* The lexicons were created for other types of text, so not for lyrics.
* The actual form of the word may not appear. For example, **strong** may appear, but **strongly** may not. There could be more cleaning needed on the data! (This is touched on in a later section.)

### Don't Take My Word For It

Take a look at some specific words from Prince's lyrics which seem like they would have an impact on sentiment. Are they in all lexicons?

new\_sentiments %>%

filter(word %in% c("dark", "controversy", "gangster",

"discouraged", "race")) %>%

arrange(word) %>% #sort

select(-score) %>% #remove this field

mutate(word = color\_tile("lightblue", "lightblue")(word),

words\_in\_lexicon = color\_bar("lightpink")(words\_in\_lexicon),

lexicon = color\_tile("lightgreen", "lightgreen")(lexicon)) %>%

my\_kable\_styling(caption = "Specific Words")

| Specific Words | | | |
| --- | --- | --- | --- |
| **word** | **sentiment** | **lexicon** | **words\_in\_lexicon** |
| controversy | negative | nrc | 6468 |
| controversy | negative | bing | 6785 |
| dark | sadness | nrc | 6468 |
| dark | negative | bing | 6785 |
| discouraged | negative | AFINN | 2476 |
| gangster | negative | bing | 6785 |

Controversy and dark appear in NRC and Bing, but gangster only appears in Bing. Race doesn't appear at all and is a critical topic in Prince's lyrics. But is it easily associated with a sentiment? Note that AFINN is much smaller and only has one of these words.

### Word Forms

Now look at a more complicated example. Sexuality is a common theme in Prince's music. How will sentiment analysis based on predefined lexicons be affected by different forms of a word? For example, here are all the references to the root word sex in the lyrics. Compare these to Bing and NRC and see where there are matches.

my\_word\_list <- prince\_data %>%

unnest\_tokens(word, lyrics) %>%

filter(grepl("sex", word)) %>% #Use `grepl()` to find the substring `"sex"`

count(word) %>%

select(myword = word, n) %>% #Rename word

arrange(desc(n))

new\_sentiments %>%

#Right join gets all words in `my\_word\_list` to show nulls

right\_join(my\_word\_list, by = c("word" = "myword")) %>%

filter(word %in% my\_word\_list$myword) %>%

mutate(word = color\_tile("lightblue", "lightblue")(word),

instances = color\_tile("lightpink", "lightpink")(n),

lexicon = color\_tile("lightgreen", "lightgreen")(lexicon)) %>%

select(-score, -n) %>% #Remove these fields

my\_kable\_styling(caption = "Dependency on Word Form")

| Dependency on Word Form | | | | |
| --- | --- | --- | --- | --- |
| **word** | **sentiment** | **lexicon** | **words\_in\_lexicon** | **instances** |
| sexy | positive | bing | 6785 | 220 |
| sexy | positive | AFINN | 2476 | 220 |
| sex | anticipation | nrc | 6468 | 185 |
| sex | joy | nrc | 6468 | 185 |
| sex | positive | nrc | 6468 | 185 |
| sex | trust | nrc | 6468 | 185 |
| superfunkycalifragisexy | NA | NA | NA | 19 |
| lovesexy | NA | NA | NA | 16 |
| sexual | NA | NA | NA | 11 |
| sexuality | NA | NA | NA | 11 |
| sexiness | NA | NA | NA | 2 |
| sexually | NA | NA | NA | 2 |
| sexe | NA | NA | NA | 1 |
| sexed | NA | NA | NA | 1 |
| sexier | NA | NA | NA | 1 |
| superfunkycalifraagisexy | NA | NA | NA | 1 |
| superfunkycalifragisexi | NA | NA | NA | 1 |

Notice that Prince uses sexy frequently, but it doesn't exist in this form in NRC. The word sex is found in NRC but not in Bing. What if you looked at the **stems** or **roots** of words, would that help? What a conundrum! Your text could contain a past tense, a plural, or an adverb of a root word, but it may not exist in any lexicon. How do you deal with this?

### More Data Preparation?

It may be the case that you need a few more data preparation steps. Here are three techniques to consider before performing sentiment analysis:

* Stemming: generally refers to removing suffixes from words to get the common origin
* Lemmatization: reducing inflected (or sometimes derived) words to their word stem, base or root form
* Word replacement: replace words with more frequently used synonyms

An advanced concept in sentiment analysis is that of synonym (semantically similar peer) and hypernym (a common parent) replacement. These are words that are more frequently used than the related word in the lyric, and actually do appear in a lexicon, thus giving a higher match percentage. There is not enough space in this tutorial to address additional data preparation, but it's definitely something to consider!

**Challenge**: do a little research on lexicons and how they are created. Is there already one that exists that is better suited to musical lyrics? If you're really interested, maybe consider what it would take to build your own lexicon. What is the difference between classifier-based sentiment analysis and lexicon-based sentiment analysis?

## Detailed Analysis

Now that you have a foundational understanding of the dataset and the lexicons, you can apply that knowledge by joining them together for analysis. Here are the high-level steps you'll take:

* Create lexicon-specific datasets
* Look at polar sentiment across all songs
* Examine sentiment change over time
* Validate your results against specific events in Prince's life
* Study song level sentiment
* Review how pairs of words affect sentiment

### Create Sentiment Datasets

Start off by creating Prince sentiment datasets for each of the lexicons by performing an inner\_join() on the get\_sentiments() function. Pass the name of the lexicon for each call. For this exercise, use Bing for binary and NRC for categorical sentiments. Since words can appear in multiple categories in NRC, such as Negative/Fear or Positive/Joy, you'll also create a subset without the positive and negative categories to use later on.

prince\_bing <- prince\_tidy %>%

inner\_join(get\_sentiments("bing"))

prince\_nrc <- prince\_tidy %>%

inner\_join(get\_sentiments("nrc"))

prince\_nrc\_sub <- prince\_tidy %>%

inner\_join(get\_sentiments("nrc")) %>%

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

### In The Mood: Overall Sentiment

In the detailed analysis of the lyrics, you'll want to examine the different levels of text, such as all songs, chart level, decade level and word level. Start by graphing the NRC sentiment analysis of the entire dataset.

(Just for fun, I used the memery and magick packages to add images (memes) to the graphs.)

nrc\_plot <- prince\_nrc %>%

group\_by(sentiment) %>%

summarise(word\_count = n()) %>%

ungroup() %>%

mutate(sentiment = reorder(sentiment, word\_count)) %>%

#Use `fill = -word\_count` to make the larger bars darker

ggplot(aes(sentiment, word\_count, fill = -word\_count)) +

geom\_col() +

guides(fill = FALSE) + #Turn off the legend

theme\_lyrics() +

labs(x = NULL, y = "Word Count") +

scale\_y\_continuous(limits = c(0, 15000)) + #Hard code the axis limit

ggtitle("Prince NRC Sentiment") +

coord\_flip()

img <- "prince\_background2.jpg" #Load the background image

lab <- "" #Turn off the label

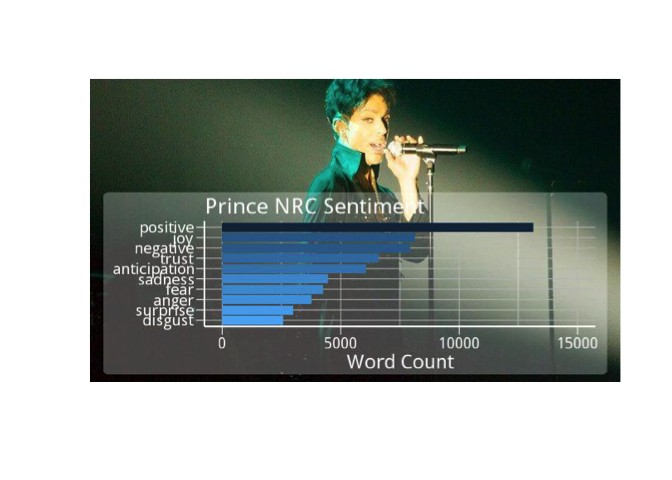
#Overlay the plot on the image and create the meme file

meme(img, lab, "meme\_nrc.jpg", inset = nrc\_plot)

#Read the file back in and display it!

nrc\_meme <- image\_read("meme\_nrc.jpg")

plot(nrc\_meme)



It appears that for Prince's lyrics, NRC strongly favors the positive. But are all words with a sentiment of disgust or anger also in the negative category as well? It may be worth checking out.

Now take a look at Bing overall sentiment. Of the 1185 distinct words from Prince's lyrics that appear in the Bing lexicon, how many are positive and how many are negative?

bing\_plot <- prince\_bing %>%

group\_by(sentiment) %>%

summarise(word\_count = n()) %>%

ungroup() %>%

mutate(sentiment = reorder(sentiment, word\_count)) %>%

ggplot(aes(sentiment, word\_count, fill = sentiment)) +

geom\_col() +

guides(fill = FALSE) +

theme\_lyrics() +

labs(x = NULL, y = "Word Count") +

scale\_y\_continuous(limits = c(0, 8000)) +

ggtitle("Prince Bing Sentiment") +

coord\_flip()

img1 <- "prince\_background1.jpg"

lab1 <- ""

meme(img1, lab1, "meme\_bing.jpg", inset = bing\_plot)

x <- image\_read("meme\_bing.jpg")

plot(x)



Could it be the case that for Bing, there appears to be equal positive and negative sentiment in Prince's music? Does overall sentiment cancel itself out when looking at a dataset that is too large? It's hard to know, but try looking at it in chunks and see what happens.

In acoustics, there is something called phase cancellation where the frequency of two instances of the same wave are exactly out of phase and cancel each other out, for example, when you're recording a drum with two mics and it takes the sound longer to get to one mic than the other. This results in total silence at that frequency! How may this apply to sentiment analysis of large datasets? Give it some thought.

*Linguistic Professor: "In English, a double negative forms a positive. In Russian, a double negative is still a negative. However, there is no language wherein a double positive can form a negative." Disagreeing student: "Yeah, right."*

### Crank It Up: Chart Level

Turn up the volume on your analysis by breaking it down to the chart level using the Bing lexicon. Create a graph of the polar sentiment per chart level. Use spread() to separate the sentiments into columns and mutate() to create a polarity (positive - negative) field and a percent\_positive field (positive/totalsentiment∗100positive/totalsentiment∗100), for a different perspective. For the polarity graph, add a yintercept with geom\_hline(). Plot the graphs side by side with grid.arrange().

prince\_polarity\_chart <- prince\_bing %>%

count(sentiment, chart\_level) %>%

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

mutate(polarity = positive - negative,

percent\_positive = positive / (positive + negative) \* 100)

#Polarity by chart

plot1 <- prince\_polarity\_chart %>%

ggplot( aes(chart\_level, polarity, fill = chart\_level)) +

geom\_col() +

scale\_fill\_manual(values = my\_colors[3:5]) +

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

theme\_lyrics() + theme(plot.title = element\_text(size = 11)) +

xlab(NULL) + ylab(NULL) +

ggtitle("Polarity By Chart Level")

#Percent positive by chart

plot2 <- prince\_polarity\_chart %>%

ggplot( aes(chart\_level, percent\_positive, fill = chart\_level)) +

geom\_col() +

scale\_fill\_manual(values = c(my\_colors[3:5])) +

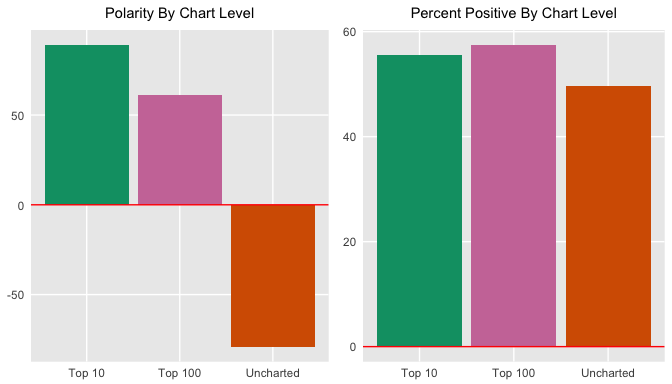
geom\_hline(yintercept = 0, color = "red") +

theme\_lyrics() + theme(plot.title = element\_text(size = 11)) +

xlab(NULL) + ylab(NULL) +

ggtitle("Percent Positive By Chart Level")

grid.arrange(plot1, plot2, ncol = 2)



Does this say that charted songs are typically more positive than negative? If so, what does this tell you about what society wants to hear? Can you even make these assumptions? Looking at the positive sentiment relative to total sentiment, it seems like the charted songs are just slightly more positive than the negative. This is interesting given that the Bing lexicon itself has more negative than positive words.

### Polar Melting: So Blue

Since you're looking at sentiment from a polar perspective, you might want to see weather or not it changes over time (geek humor). This time use geom\_smooth() with the loess method for a smoother curve and another geom\_smooth() with method = lm for a linear smooth curve.

prince\_polarity\_year <- prince\_bing %>%

count(sentiment, year) %>%

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

mutate(polarity = positive - negative,

percent\_positive = positive / (positive + negative) \* 100)

polarity\_over\_time <- prince\_polarity\_year %>%

ggplot(aes(year, polarity, color = ifelse(polarity >= 0,my\_colors[5],my\_colors[4]))) +

geom\_col() +

geom\_smooth(method = "loess", se = FALSE) +

geom\_smooth(method = "lm", se = FALSE, aes(color = my\_colors[1])) +

theme\_lyrics() + theme(plot.title = element\_text(size = 11)) +

xlab(NULL) + ylab(NULL) +

ggtitle("Polarity Over Time")

relative\_polarity\_over\_time <- prince\_polarity\_year %>%

ggplot(aes(year, percent\_positive , color = ifelse(polarity >= 0,my\_colors[5],my\_colors[4]))) +

geom\_col() +

geom\_smooth(method = "loess", se = FALSE) +

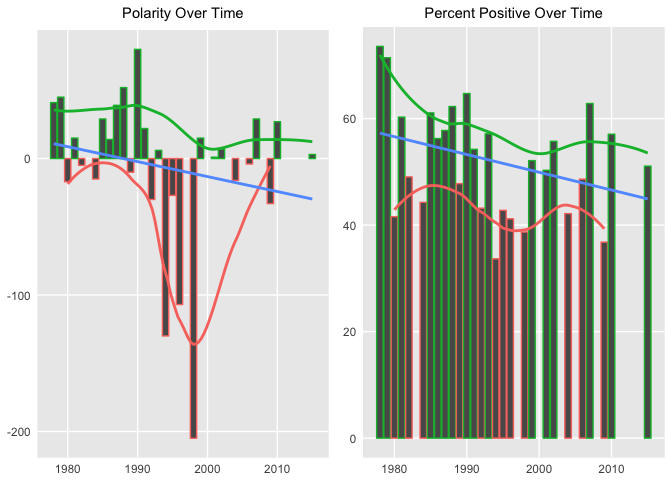
geom\_smooth(method = "lm", se = FALSE, aes(color = my\_colors[1])) +

theme\_lyrics() + theme(plot.title = element\_text(size = 11)) +

xlab(NULL) + ylab(NULL) +

ggtitle("Percent Positive Over Time")

grid.arrange(polarity\_over\_time, relative\_polarity\_over\_time, ncol = 2)



A few extremes were adjusted in the second graph above, but the overall polarity trend over time is negative in both cases.

### Mood Ring

You'll again use the power of the chordDiagram() to examine the relationships between NRC sentiments and decades. Note that sentiment categories appear on the top part of the ring and decades on the bottom.

grid.col = c("1970s" = my\_colors[1], "1980s" = my\_colors[2], "1990s" = my\_colors[3], "2000s" = my\_colors[4], "2010s" = my\_colors[5], "anger" = "grey", "anticipation" = "grey", "disgust" = "grey", "fear" = "grey", "joy" = "grey", "sadness" = "grey", "surprise" = "grey", "trust" = "grey")

decade\_mood <- prince\_nrc %>%

filter(decade != "NA" & !sentiment %in% c("positive", "negative")) %>%

count(sentiment, decade) %>%

group\_by(decade, sentiment) %>%

summarise(sentiment\_sum = sum(n)) %>%

ungroup()

circos.clear()

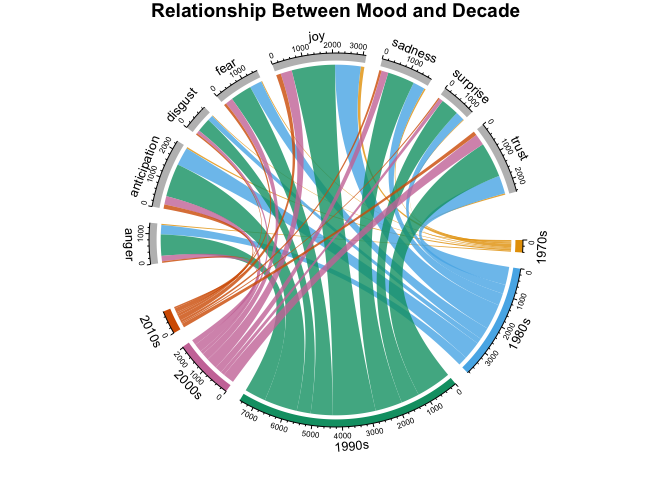
#Set the gap size

circos.par(gap.after = c(rep(5, length(unique(decade\_mood[[1]])) - 1), 15,

rep(5, length(unique(decade\_mood[[2]])) - 1), 15))

chordDiagram(decade\_mood, grid.col = grid.col, transparency = .2)

title("Relationship Between Mood and Decade")



This shows the counts of words per NRC category per decade. It's a lot to take in on a small graph, but it provides tons of information on relationships between categories and time. These diagrams are incredibly customizable and can be as simple or informative as desired.

### Real-Time Sentiment

Given all the wonderful courses you've taken on DataCamp, you're probably itching to apply those skills to real world situations. So do that now by mapping your analysis of Prince's lyrics to something real over a period of time. However, please be skeptical and cautious when performing a simplified analysis on such a complex subject.

I have created a list of Prince's life events, collected from popular sources such as Rolling Stone Magazine, Biography.com, etc. I selected highly public years that match songs that have release dates in our dataset. Read in those events from princeEvents.csv now. Then use prince\_bing and spread() to create a polarity score per year. Join on the events data frame and create a sentiment field so you can fill in colors on your bar chart. As always, use coord\_flip() when you're showing large text labels.

events <- read.csv('princeEvents.csv', stringsAsFactors = FALSE)

year\_polarity\_bing <- prince\_bing %>%

group\_by(year, sentiment) %>%

count(year, sentiment) %>%

spread(sentiment, n) %>%

mutate(polarity = positive - negative,

ratio = polarity / (positive + negative)) #use polarity ratio in next graph

events %>%

#Left join gets event years with no releases

left\_join(year\_polarity\_bing) %>%

filter(event != " ") %>% #Account for bad data

mutate(event = reorder(event, year), #Sort chart by desc year

sentiment = ifelse(positive > negative,

"positive", "negative")) %>%

ggplot(aes(event, polarity, fill = sentiment)) +

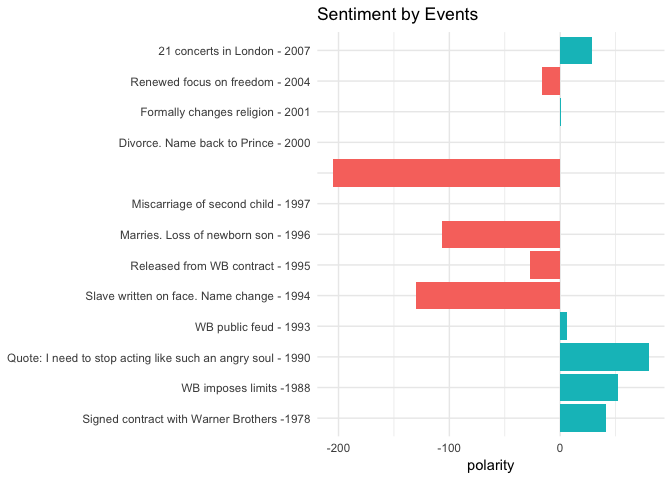
geom\_bar(stat = "identity") +

theme\_minimal() + theme(legend.position = "none") +

xlab(NULL) +

ggtitle("Sentiment by Events") +

coord\_flip()



**Tip**: you can find princeEvents.csv here.

I think it's fascinating to compare the events with the sentiment. Although subjective, I found them to be very correlated. Now granted, I created the event dataset myself. I could have easily biased it to match the lyrics. However, I did use multiple sources for each year to determine the most commonly reported fact. These results should motivate you to look even more deeply into a word level analysis to see what Prince was singing about.

### The Black Album: 1994 - 1996

In 1994, Prince released **The Black Album** as an attempt to regain his African-American audience. Given the fact that 1994 and the following two years stand out in the analysis, you can look at the top words that match the NRC lexicon for this period using geom\_label\_repel() from the ggrepel package along with geom\_point(). This is a tricky graph, so I suggest you play around with the configuration of the code below.

plot\_words\_94\_96 <- prince\_nrc %>%

filter(year %in% c("1994", "1995", "1996")) %>%

group\_by(sentiment) %>%

count(word, sort = TRUE) %>%

arrange(desc(n)) %>%

slice(seq\_len(8)) %>% #consider top\_n() from dplyr also

ungroup()

plot\_words\_94\_96 %>%

#Set `y = 1` to just plot one variable and use word as the label

ggplot(aes(word, 1, label = word, fill = sentiment )) +

#You want the words, not the points

geom\_point(color = "transparent") +

#Make sure the labels don't overlap

geom\_label\_repel(force = 1,nudge\_y = .5,

direction = "y",

box.padding = 0.04,

segment.color = "transparent",

size = 3) +

facet\_grid(~sentiment) +

theme\_lyrics() +

theme(axis.text.y = element\_blank(), axis.text.x = element\_blank(),

axis.title.x = element\_text(size = 6),

panel.grid = element\_blank(), panel.background = element\_blank(),

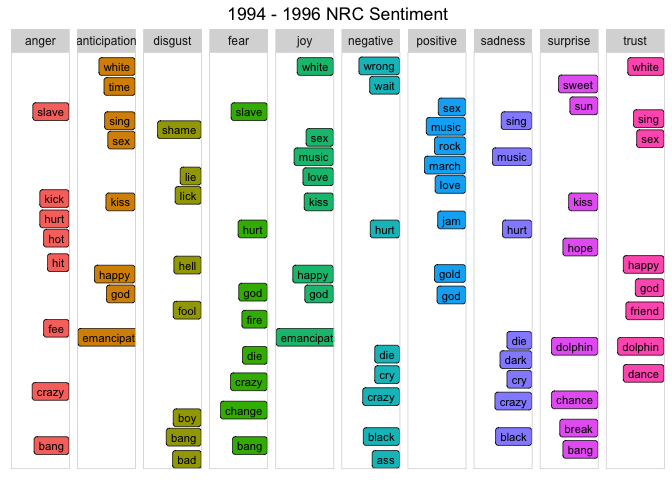
panel.border = element\_rect("lightgray", fill = NA),

strip.text.x = element\_text(size = 9)) +

xlab(NULL) + ylab(NULL) +

ggtitle("1994 - 1996 NRC Sentiment") +

coord\_flip()



In 1994, Prince quoted to Rolling Stone magazine,

*"When you stop a man from dreaming, he becomes a slave. That's where I was."*

So he appeared in public with the word "slave" prominently penned on his face outwardly declaring his anti-corporate sentiment with his current music label. He also changed his name to a symbol. How well do the words above match these events? Notice the words "slave", "emancipation", "black" and "change" in the graph?

(Remember, this is not just about Prince. You can apply it to any text! For example, what were the most trust sentiment words spoken by President Trump in his first year in office?)

### This Time It's Personal: 1998

The Sentiment By Events chart above indicated that between 1996 and 1998, Prince married, lost two children, and was said to predict 9/11 on stage. (Do a YouTube search on his 1998 concert in The Netherlands to listen for yourself!) Using the same steps as with the previous graph, look at the top 10 words in each NRC category during 1998.

plot\_words\_1998 <- prince\_nrc %>%

filter(year == "1998") %>%

group\_by(sentiment) %>%

count(word, sort = TRUE) %>%

arrange(desc(n)) %>%

slice(seq\_len(10)) %>%

ungroup()

#Same comments as previous graph

plot\_words\_1998 %>%

ggplot(aes(word, 1, label = word, fill = sentiment )) +

geom\_point(color = "transparent") +

geom\_label\_repel(force = 1,nudge\_y = .5,

direction = "y",

box.padding = 0.05,

segment.color = "transparent",

size = 3) +

facet\_grid(~sentiment) +

theme\_lyrics() +

theme(axis.text.y = element\_blank(), axis.text.x = element\_blank(),

axis.title.x = element\_text(size = 6),

panel.grid = element\_blank(), panel.background = element\_blank(),

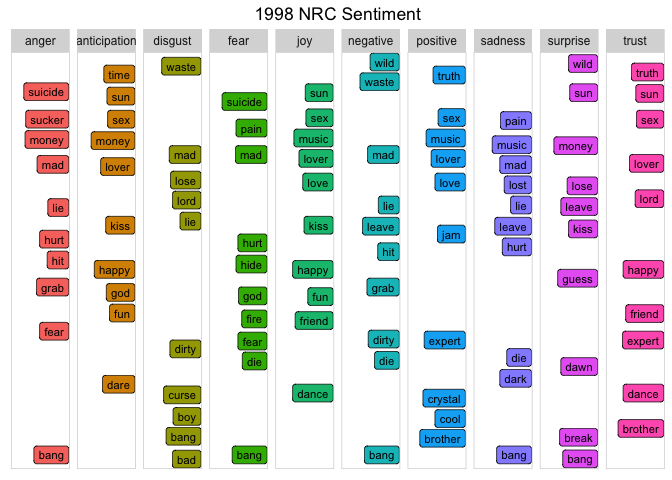
panel.border = element\_rect("lightgray", fill = NA),

strip.text.x = element\_text(size = 9)) +

xlab(NULL) + ylab(NULL) +

ggtitle("1998 NRC Sentiment") +

coord\_flip()



Given the personal loss and the looming threat of terrorism, words like "suicide", "waste", "mad", "hurt", and many more not captured in this graph are very indicative of the actual events. These words are powerful and provide real insight into the sentiment of that time period.

### On The Radar: Radar Charts

Another great way to compare sentiment across categories is to use a radar chart, which is also known as a spider chart. You can make this type of charts with the radarchart package. These are useful for seeing which variables have similar values or if there are any outliers for each variable.

You will break this analysis into three different levels: year, chart, and decade. (To save space, I'll only include code for the specific years.)

* Use the prince\_nrc\_sub dataset which does not contain the positive and negative sentiments so that the other ones are more visible. This time you will first calculate the total count of words by sentiment per year, as well as the total sentiment for the entire year and obtain a percentage (countofsentimentwordsperyear/totalperyear∗100countofsentimentwordsperyear/totalperyear∗100).
* Filter for the specific years 1978, 1994, 1995, and remove the unneeded fields with select().
* Finally, you'll need to spread() the year and percent values (key/value pairs) into multiple columns so that you have one row for each sentiment and a column for each year. Then use chartJSRadar() to generate an interactive HTML widget. You can pass an argument to display dataset labels in the mouse over. (FYI, sometimes the J and Y are cropped from the word "joy" by radarchart and it looks like "iov".)

#Get the count of words per sentiment per year

year\_sentiment\_nrc <- prince\_nrc\_sub %>%

group\_by(year, sentiment) %>%

count(year, sentiment) %>%

select(year, sentiment, sentiment\_year\_count = n)

#Get the total count of sentiment words per year (not distinct)

total\_sentiment\_year <- prince\_nrc\_sub %>%

count(year) %>%

select(year, year\_total = n)

#Join the two and create a percent field

year\_radar\_chart <- year\_sentiment\_nrc %>%

inner\_join(total\_sentiment\_year, by = "year") %>%

mutate(percent = sentiment\_year\_count / year\_total \* 100 ) %>%

filter(year %in% c("1978","1994","1995")) %>%

select(-sentiment\_year\_count, -year\_total) %>%

spread(year, percent) %>%

chartJSRadar(showToolTipLabel = TRUE,

main = "NRC Years Radar")

You'll have to scroll up and down to compare these graphs because I wanted to leave them full size. It's interesting to note that with a smaller dataset like year, you are able to see the variance in each sentiment more distinctly using this type of visualization. You may also notice that in contrast to previous exercises, using a percentage, "joy" has the highest score on all charts (especially at the beginning of Prince's career in 1978).

### Sign O' The Times

Spotify uses machine learning to create song recommendations for its 140 million active users. It doesn't use lyrical content yet. Imagine if it could... So now, take an even deeper look into the mood of specific songs.

In 1987, Prince wrote a song called **"Sign O' the Times"**. A Billboard article by Kenneth Partridge stated:

*"As he tackles the topics of AIDS, gangs, drugs, natural disasters, and even the Challenger explosion, Prince plays it cool and detached. 'Sign O' the Times' is a status update, not a call to action. His solution isn't marching in the streets or phoning your congressman."*

So how would a machine interpret the mood of this song? Is it highly emotional or simply informative? Does it represent Prince as a person or society in general? Graph the NRC categories with ggplot2.

prince\_nrc %>%

filter(song %in% "sign o the times") %>%

group\_by(sentiment) %>%

summarise(word\_count = n()) %>%

ungroup() %>%

mutate(sentiment = reorder(sentiment, word\_count)) %>%

ggplot(aes(sentiment, word\_count, fill = -word\_count)) +

geom\_col() +

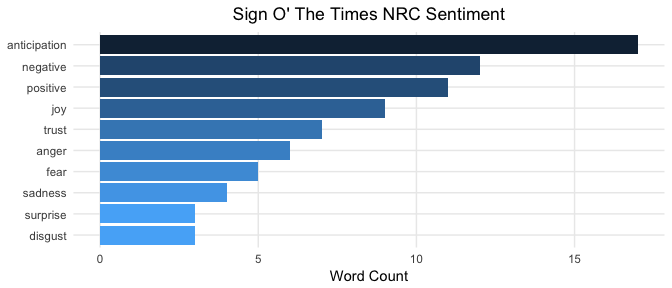
guides(fill = FALSE) +

theme\_minimal() + theme\_lyrics() +

labs(x = NULL, y = "Word Count") +

ggtitle("Sign O' The Times NRC Sentiment") +

coord\_flip()



Although once again highly subjective, your results appear to confirm that Prince "plays it cool and detached", as stated by Partridge. This may be interpreted by the observation that anticipation words are more prevalent than emotional categories such as sadness, fear, and anger. Here is an article that states these three categories are **emotions** whereas anticipation is a **sentiment**. The Billboard author also stated the song is simply a "status update, almost devoid of emotion". It seems as that the machine agrees. Do you interpret it this way as well? I know that this is an R tutorial, but sentiment analysis is not purely technical. It is said to be the place where AI meets psychology.

Using ggplot2 to create a slightly different chart, look at the words for each category.

prince\_tidy %>%

filter(song %in% 'sign o the times') %>%

distinct(word) %>%

inner\_join(get\_sentiments("nrc")) %>%

ggplot(aes(x = word, fill = sentiment)) +

facet\_grid(~sentiment) +

geom\_bar() + #Create a bar for each word per sentiment

theme\_lyrics() +

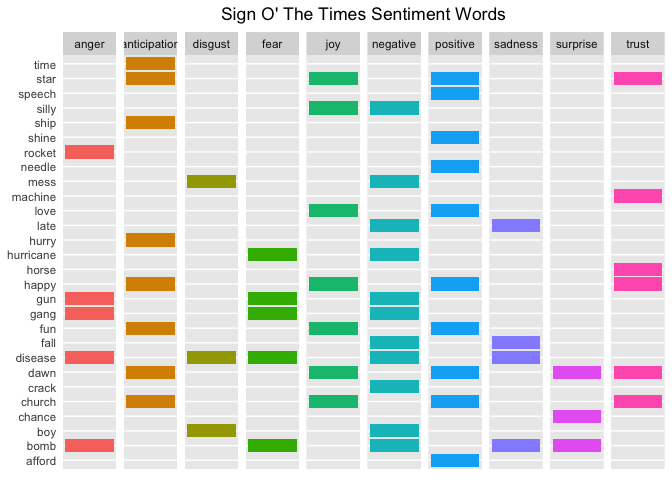
theme(panel.grid.major.x = element\_blank(),

axis.text.x = element\_blank()) + #Place the words on the y-axis

xlab(NULL) + ylab(NULL) +

ggtitle("Sign O' The Times Sentiment Words") +

coord\_flip()



Can you match these words with the topics mentioned in the Billboard article? It's pretty easy to see the connection: Challenger disaster -> "rocket", AIDS -> "disease", Drugs -> "crack", Natural Disasters -> "hurricane", Gangs -> "gang", etc.

It's almost like Partridge wrote some R code and created this chart before writing the article!

### More Songs

Look at the sentiment categories for a few more songs with distinctive titles and see if they appear to be correlated.

prince\_nrc\_sub %>%

filter(song %in% c("so blue", "controversy", "raspberry beret",

"when doves cry", "the future", "1999")) %>%

count(song, sentiment, year) %>%

mutate(sentiment = reorder(sentiment, n), song = reorder(song, n)) %>%

ggplot(aes(sentiment, n, fill = sentiment)) +

geom\_col() +

facet\_wrap(year ~ song, scales = "free\_x", labeller = label\_both) +

theme\_lyrics() +

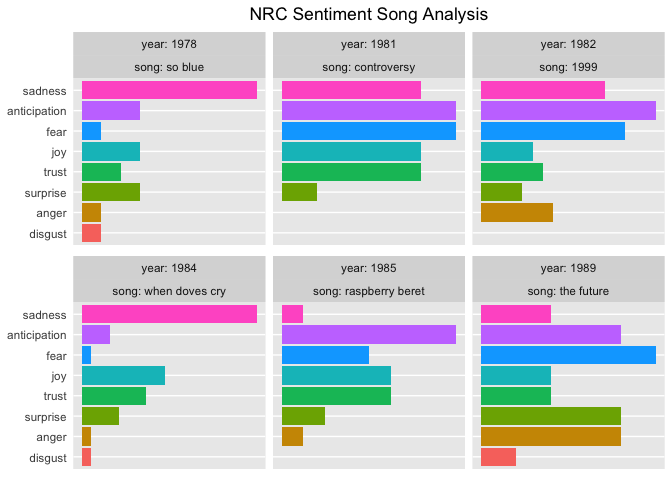
theme(panel.grid.major.x = element\_blank(),

axis.text.x = element\_blank()) +

labs(x = NULL, y = NULL) +

ggtitle("NRC Sentiment Song Analysis") +

coord\_flip()



NRC sentiments show high anticipation and fear for a song about **the future**, and the same thing plus high trust for a song about **controversy**. The songs that are prevalently sad seem to match their titles as well.

### Bigrams Per Decade

So far you have only been looking at unigrams or single words. But if **"love"** is a common word, what precedes it? Or follows it? Looking at single words out of context could be misleading. So, now it's time to look at some bigrams or word pairs.

Conveniently, the tidytext package provides the ability to unnest pairs of words as well as single words. In this case, you'll call unnest\_tokens() passing the token argument ngrams. Since you're just looking at bigrams (two consecutive words), pass n = 2. Use prince\_bigrams to store the results.

The tidyr package provides the ability to separate the bigrams into individual words using the separate() function. In order to remove the stop words and undesirable words, you'll want to break the bigrams apart and filter out what you don't want, then use unite() to put the word pairs back together. This makes it easy to visualize the most common bigrams per decade.

prince\_bigrams <- prince\_data %>%

unnest\_tokens(bigram, lyrics, token = "ngrams", n = 2)

bigrams\_separated <- prince\_bigrams %>%

separate(bigram, c("word1", "word2"), sep = " ")

bigrams\_filtered <- bigrams\_separated %>%

filter(!word1 %in% stop\_words$word) %>%

filter(!word2 %in% stop\_words$word) %>%

filter(!word1 %in% undesirable\_words) %>%

filter(!word2 %in% undesirable\_words)

#Because there is so much repetition in music, also filter out the cases where the two words are the same

bigram\_decade <- bigrams\_filtered %>%

filter(word1 != word2) %>%

filter(decade != "NA") %>%

unite(bigram, word1, word2, sep = " ") %>%

inner\_join(prince\_data) %>%

count(bigram, decade, sort = TRUE) %>%

group\_by(decade) %>%

slice(seq\_len(7)) %>%

ungroup() %>%

arrange(decade, n) %>%

mutate(row = row\_number())

## Joining, by = c("song", "year", "album", "peak", "us\_pop", "us\_rnb", "decade", "chart\_level", "charted")

bigram\_decade %>%

ggplot(aes(row, n, fill = decade)) +

geom\_col(show.legend = FALSE) +

facet\_wrap(~decade, scales = "free\_y") +

xlab(NULL) + ylab(NULL) +

scale\_x\_continuous( # This handles replacement of row

breaks = bigram\_decade$row, # Notice need to reuse data frame

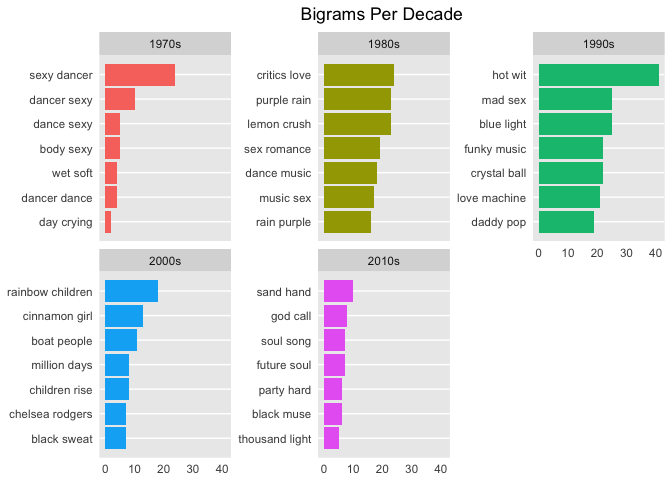
labels = bigram\_decade$bigram) +

theme\_lyrics() +

theme(panel.grid.major.x = element\_blank()) +

ggtitle("Bigrams Per Decade") +

coord\_flip()



Using bigrams, you can almost see the common phrases shift from sex, dance and romance to religion and (rainbow) children. In case you didn't know, the term "rainbow baby" is sometimes used by parents who are expecting another child after losing a baby to miscarriage. Interestingly, I could not find a real review of Prince's Rainbow Children album that made note of this.

### Sentiment with Bigrams

So how do bigrams affect sentiment? This time use the AFINN lexicon to perform sentiment analysis on word pairs, looking at how often sentiment-associated words are preceded by "not" or other negating words.

AFINN <- get\_sentiments("afinn")

not\_words <- bigrams\_separated %>%

filter(word1 == "not") %>%

inner\_join(AFINN, by = c(word2 = "word")) %>%

count(word2, score, sort = TRUE) %>%

ungroup()

not\_words %>%

mutate(contribution = n \* score) %>%

arrange(desc(abs(contribution))) %>%

head(20) %>%

mutate(word2 = reorder(word2, contribution)) %>%

ggplot(aes(word2, n \* score, fill = n \* score > 0)) +

geom\_col(show.legend = FALSE) +

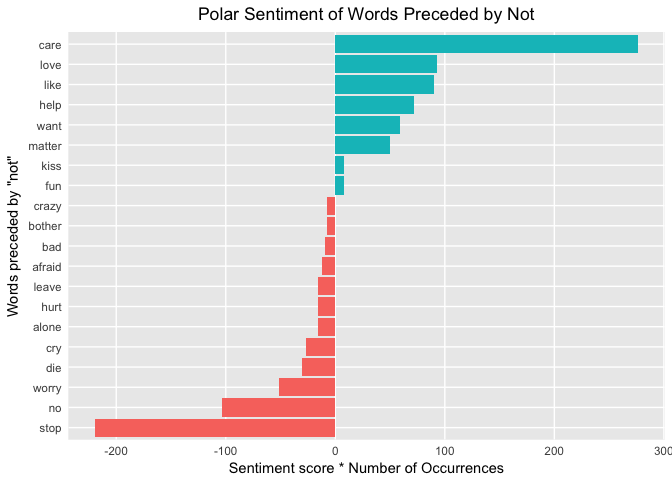
theme\_lyrics() +

xlab("Words preceded by \"not\"") +

ylab("Sentiment score \* Number of Occurrences") +

ggtitle("Polar Sentiment of Words Preceded by Not") +

coord\_flip()



On the first line of the graph, **care** is given a false positive sentiment because the **"not"** is ignored with single-word analysis. Do the false positive bigrams cancel out the false negative bigrams?

Yet another question that I can't answer, but it's a good topic for further exploration. **Tip**: you can also pass the parameters of "trigrams" and "n = 3" to unnest\_tokens() to look at even more consecutive words!

There are other negation words to consider as well. This time you will create a network graph using the ggraph and igraph packages. You'll arrange the words into connected nodes with the negation words at the centers. Create the first object from the tidy dataset using graph\_from\_data\_frame() and then use ggraph() to plot it. You can highlight the main nodes with a call to geom\_edge\_density(). You can get more details of a similar example in Julia Silge and David Robinson's book on Tidy Text Mining.

negation\_words <- c("not", "no", "never", "without")

negation\_bigrams <- bigrams\_separated %>%

filter(word1 %in% negation\_words) %>%

inner\_join(AFINN, by = c(word2 = "word")) %>%

count(word1, word2, score, sort = TRUE) %>%

mutate(contribution = n \* score) %>%

arrange(desc(abs(contribution))) %>%

group\_by(word1) %>%

slice(seq\_len(20)) %>%

arrange(word1,desc(contribution)) %>%

ungroup()

bigram\_graph <- negation\_bigrams %>%

graph\_from\_data\_frame() #From `igraph`

set.seed(123)

a <- grid::arrow(type = "closed", length = unit(.15, "inches"))

ggraph(bigram\_graph, layout = "fr") +

geom\_edge\_link(alpha = .25) +

geom\_edge\_density(aes(fill = score)) +

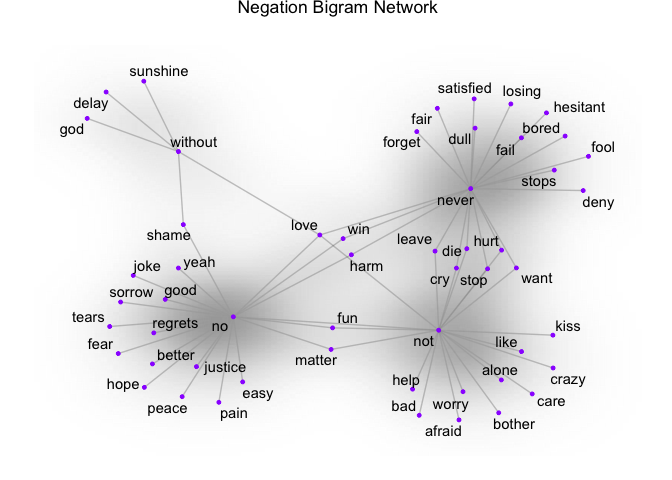
geom\_node\_point(color = "purple1", size = 1) + #Purple for Prince!

geom\_node\_text(aes(label = name), repel = TRUE) +

theme\_void() + theme(legend.position = "none",

plot.title = element\_text(hjust = 0.5)) +

ggtitle("Negation Bigram Network")



Here, you can see the word pairs associated with negation words. So if your analysis is based on unigrams and "alone" comes back as negative, the bigram "not alone" as you see above will have a reverse effect. Some words cross over to multiple nodes which can be seen easily in a visual like this one: for example, "never hurt" and "not hurt".

### Pairwise Comparisons

Since you've now looked at n-grams, take a look at the correlation between words. Which words are most highly correlated? Use the pairwise\_count() function from the widyr package to identify co-occurrence counts. That is, you count the number of times each pair of words appear together within a song. The widyr package takes a tidy dataset, and **temporarily** widens it before returning it to a tidy structure for visualization and further analysis.

To keep it simple, I've chosen four interesting words in Prince's lyrics.

pwc <- prince\_tidy %>%

filter(n() >= 20) %>% #High counts

pairwise\_count(word, song, sort = TRUE) %>%

filter(item1 %in% c("love", "peace", "gangster", "hate")) %>%

group\_by(item1) %>%

slice(seq\_len(7)) %>%

ungroup() %>%

mutate(row = -row\_number()) #Descending order

pwc %>%

ggplot(aes(row, n, fill = item1)) +

geom\_bar(stat = "identity", show.legend = FALSE) +

facet\_wrap(~item1, scales = "free") +

scale\_x\_continuous( #This handles replacement of row

breaks = pwc$row, #Notice need to reuse data frame

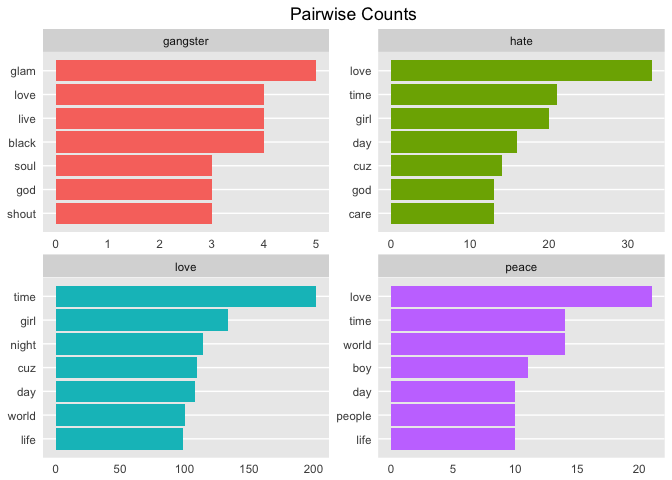
labels = pwc$item2) +

theme\_lyrics() + theme(panel.grid.major.x = element\_blank()) +

xlab(NULL) + ylab(NULL) +

ggtitle("Pairwise Counts") +

coord\_flip()



Compare that to pairwise correlation. This refers to how often words appear together relative to how often they appear separately. Use pairwise\_cor() to determine the correlation between words based on how often they appear in the same song.

prince\_tidy %>%

group\_by(word) %>%

filter(n() >= 20) %>%

pairwise\_cor(word, song, sort = TRUE) %>%

filter(item1 %in% c("love", "peace", "gangster", "hate")) %>%

group\_by(item1) %>%

top\_n(7) %>%

ungroup() %>%

mutate(item2 = reorder(item2, correlation)) %>%

ggplot(aes(item2, correlation, fill = item1)) +

geom\_bar(stat = 'identity', show.legend = FALSE) +

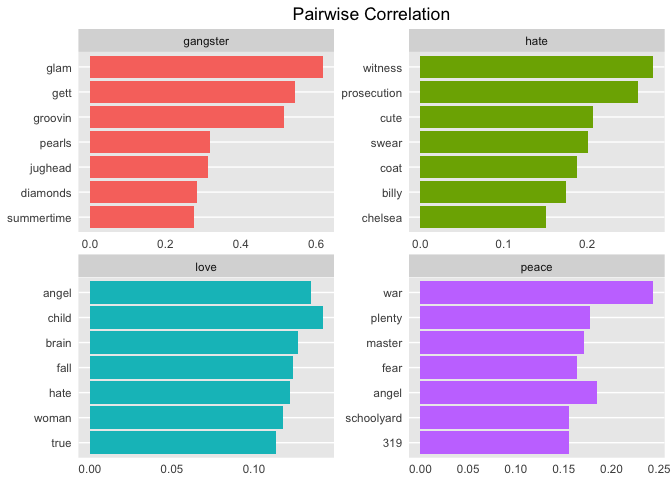
facet\_wrap(~item1, scales = 'free') +

theme\_lyrics() + theme(panel.grid.major.x = element\_blank()) +

xlab(NULL) + ylab(NULL) +

ggtitle("Pairwise Correlation") +

coord\_flip()



I think these are fascinating insights into the lyrics, just based on these four words alone! Looking at these results you can begin to see some themes emerge. This provides a great segue into the next tutorial on Topic Modeling!

## Conclusion

In this tutorial, you created a tidy dataset and analyzed basic information such as the lexical diversity and song counts per release year, and you examined the relationship between release decade and whether a song hit the charts. You then explored some sentiment lexicons and how well they matched the lyrics. Next, you performed sentiment analysis on all songs in the dataset, sentiment over time, song level sentiment, and the impact of bigrams. You did this using a wide variety of interesting graphs, each giving a different perspective.

So is it possible to write a program to determine mood in lyrics? Why yes, it is! How reliable are your results? It depends on a wide range of criteria such as the amount of data preparation, the choice of lexicon, the method of analysis, the quality of the source data, and so on. Comparing real life events, both personal and societal, can illuminate the mood of any lyric. Prince's polar sentiment seemed to slightly decline over time, yet overall, joy does seem to stand out. Charted songs seem to be more positive than uncharted songs. The claims of predicting 9/11 and his own death seem to eerily match his words.

But lyrics are complex and too many assumptions can cause problems. How much NLP can be performed on lyrics which appear without punctuation or sentence structure? What was Prince really singing about? Now that you understand the mood, you're more prepared to investigate the possible topics. And finally, can you apply machine learning techniques to predict the decade or chart level of a song? Join me in the next few tutorials to find out!