Requirements

Here, you'll need a genius API token, and the following libraries:

- geniusr
- tidyverse
- tidytext
- ggplot2
- dplyr

Using the Genius API

To gain access to lyrics, you can use the *genius_token()* function from geniusr. Call it with *genius_token(TRUE)* and paste your token into the console where prompted. You should now be good to begin.

For this example, I'll be using The Darkness. Anyone who read my previous music post will not be surprised. At all. Since I'm looking at The Darkness, it's probably only right to begin with I Believe in a Thing Called Love, right? Let's start by finding the genius ID for the song.

```
# Song ID
thingCalledLove <- search_genius(search_term = "I Believe in a Thing Called Love")
thingCalledLove[[1]][[1]][6]
thingCalledLove$content[[1]]$id
```

In the above, both lines 3 and 4 will produce the song's ID. In our case, it's 81524. You can then get the lyrics for a song using the *get_lyrics_id()* function. This produces a table containing the lyrics, the section of the song, among other details.

```
# Lyrics
thingCalledLoveLyrics <- get_lyrics_id(song_id = 81524)
```

line	section_name	section_artist	song_name	artist_name	song_id
Can't explain all the feelings that you're making me feel	Verse 1	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
My heart's in overdrive and you're behind the steering wheel	Verse 1	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
Touching you	Pre-Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
Touching me	Pre-Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
Touching you, God, you're touching me	Pre-Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
I believe in a thing called love	Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
Just listen to the rhythm of my heart	Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
There's a chance we could make it, now	Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
We'll be rocking 'til the sun goes down	Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
I believe in a thing called love	Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
Ooh	Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524
Huh	Chorus	The Darkness	I Believe in a Thing Called Love	The Darkness	81524

"Ooh. Huh." – Justin Hawkins, 2003

Now that we see how to get a song's lyrics, let's get them for every song The Darkness have. We can do this easily enough. First, we'll find the artist ID for the band, and can then use a function to get all of their song titles, IDs, and links. We can then use a loop to add each song's lyrics to a dataframe – more details on that specific part below, it ends up a little less clean than you'd hope.

```
# Find artist ID
search_artist("The Dakrness") # 22667
songs <- get_artist_songs_df(22667)
```

```
# Get all song IDs
ids <- c(as.character(songs$song_id))

# Create empty dataframe to house them
allLyrics <- data.frame()

# Add lyrics to that df
for (id in ids) {
  allLyrics <- rbind(get_lyrics_id(id), allLyrics)
}
# Above loop behaves strange
```

Notice that last comment? It is why this blog post was abandoned in the WIP folder like I mentioned in the intro. Running that loop produces an error. It will add the lyrics of a few songs without issue and then fail on a song and end abruptly. No consistency as to which song causes the crash. The same song will work sometimes and fail others. Now, there is a solution, but it ain't pretty. I am a firm believer of "if it looks stupid but works, it ain't stupid" but even I am stretched here. Let's take a look.

So what's the fix?

R has a *tryCatch()* function. Usually used to change the output when an error appears, but this can be cheated a little bit. When setting how to handle the error, we can simply use a function that does nothing. See this StackOverflow post for more details. So if we were to put a *tryCatch()* inside of our loop with no error handling, it would work, right? Almost. this would just prevent the loop from coming to a halt when it reached an error, but whatever song produced the error would not be added to our lyrics df. The solution I've opted for, is to place our for loop inside a while loop, that will terminate once all songs are accounted for. Okay, here we go.

```
while (length(ids) > 0) {
  for (id in ids) {
    tryCatch({
      allLyrics <- rbind(get_lyrics_id(id), allLyrics)
      successful <- unique(allLyrics$song_id)
      ids <- ids[!ids %in% successful]
      print(paste("done - ", id))
      print(paste("New length is ", length(ids)))
    }, error = function(e){})
  }
}
```

So here we add a song to our df, add the song's ID to a variable named *successful*, remove the ID from the original list, and print to give us an update as to where we are. This can take a while, each song will take about 3 seconds, so for prolific artists you can use a package like beepr to let you know when this is finished. Since this takes a little while to run, I like to save the df as a csv to more quickly get back to this point in future.

Side note – I am aware that R prefers vectorised methods to loops. Perhaps this is possible using *apply()*. If you have a cleaner, faster, or plain alternative solution please let me know!

Extra Details

In order to make the text analysis portion a bit more complete, I'll add the album each song belongs to to the df. The *get_song_df()* function returns a dataframe with details on the song that is fed into it. I'll create a dataframe for each of the IDs, and the album they belong to.

```
allIds <- data.frame(song_id = unique(allLyrics$song_id))
allIds$album <- ""
```

And now, a loop to put it all together. *get_song_df()* returns a 1×13 df, of which position 12 is the album title.

```
for (song in allIds$song_id) {
  allIds[match(song,allIds$song_id),2] <- get_song_df(song)[12]
  print(allIds[match(song,allIds$song_id),])
}

allLyrics <- full_join(allIds, allLyrics)
```

```
> head(allIds)
  song_id                              album
1 2298182                  Last of Our Kind
2 1433219                         Hot Cakes
3 1669686 One Way Ticket to Hell... and Back
4 2142942                  Last of Our Kind
5 1548319                              <NA>
6 1363761 One Way Ticket to Hell... and Back
```

Using *head(allIds)* shows us a preview of the data, where we can see there is an NA in row 5. In fact, there are a lot of NAs in here. Not to worry though, this is just the case where Genius has cataloged a song that never received an official release on any album. We can replace the NA values with “Single Only” to reflect this. We can then combine our lyrics and albums dfs using a full join, from dplyr.

```
head(allIds)
allIds$album[is.na(allIds$album)] <- "Single Only"
head(allIds)

allLyrics2 <- full_join(allLyrics, allIds)
```

Text Analysis

Now that we have all of our text, let’s analyse it.

First thing to do is to tokenise thee words. This is super easy with tidytext and dplyr.

```
allLyricsTokenised <- allLyrics2 %>%
  #word is the new column, line the column to retrieve the information from
  unnest_tokens(word, line)
```

Now, we can count each word to see what is the most common – and knowing The Darkness, I bet it’s “love”.

```
# Count each word - I guarantee love is top
allLyricsTokenised %>%
  count(word, sort = TRUE)
```

```
> allLyricsTokenised %>%
+   count(word, sort = TRUE)
# A tibble: 2,811 x 2
   word      n
   <chr> <int>
 1 the     824
 2 i       783
 3 you     630
 4 a       491
 5 and     427
 6 to      390
 7 of      344
 8 my      326
 9 in      311
10 love    302
```

Oh right, duh.

Okay, maybe it's an idea to remove stopwords first. I hope you appreciate that I show my dumb moments in these posts. Removing stop words is a doddle though.

```
# Remove stopwords
tidyLyrics <- allLyricsTokenised %>%
  anti_join(stop_words)

# Top words again
tidyLyrics %>%
  count(word, sort = TRUE)
```

```
> tidyLyrics %>%
+   count(word, sort = TRUE)
# A tibble: 2,396 x 2
   word      n
   <chr> <int>
 1 love    302
 2 yeah     97
 3 gonna    92
 4 die      78
 5 time     67
 6 heart    54
 7 ooh      54
 8 life     52
 9 stop     52
10 black    46
```

Much more like it.

Preparing the top lyrics for Visualisation

In order to visuisalise the most frequent lyrics, we'll need to rework our dataframe to add a count for each one. dplyr's *group_by()* makes this easy.

```
topFew <- tidyLyrics %>%
  group_by(album, word) %>%
  mutate(n = row_number()) %>%
  ungroup()
```

The above adds a count column that increases every time a word appears (since we've grouped by the

word column). We've also grouped by album, meaning we count the number of times a word appears on a specific album – not the total number of times it appears in the band's discography (we'll add that later). Next, let's subset this to only include the total number of times a word appears on each album. For instance, the word "black" appears on Permission to Land 43 times. This means there are 43 rows that contain the count for the word "black", counting up from 1-43. We only need the row with the max figure. Let's first ditch any columns that aren't needed for this part.

```
# Remove extra cols
topFew <- topFew[,c("album", "word", "n")]

# Take only max for each word by album
topFew <- topFew %>%
  group_by(album, word) %>%
  summarise(n = max(n))%>%
  ungroup()
```

Now we can add the total column. This is seen in lines 73/74. Here, we're grouping by word and adding one every time it appears. We can subset to include only words that appear at least 40 times, and I'm going to remove the word "ooh".

```
# Subset
topFew <- topFew %>%
  group_by(word) %>%
  mutate(total = sum(n)) %>%
  filter(total >= 40,
         word != "ooh") %>%
  ungroup()
```

Visualising the top lyrics

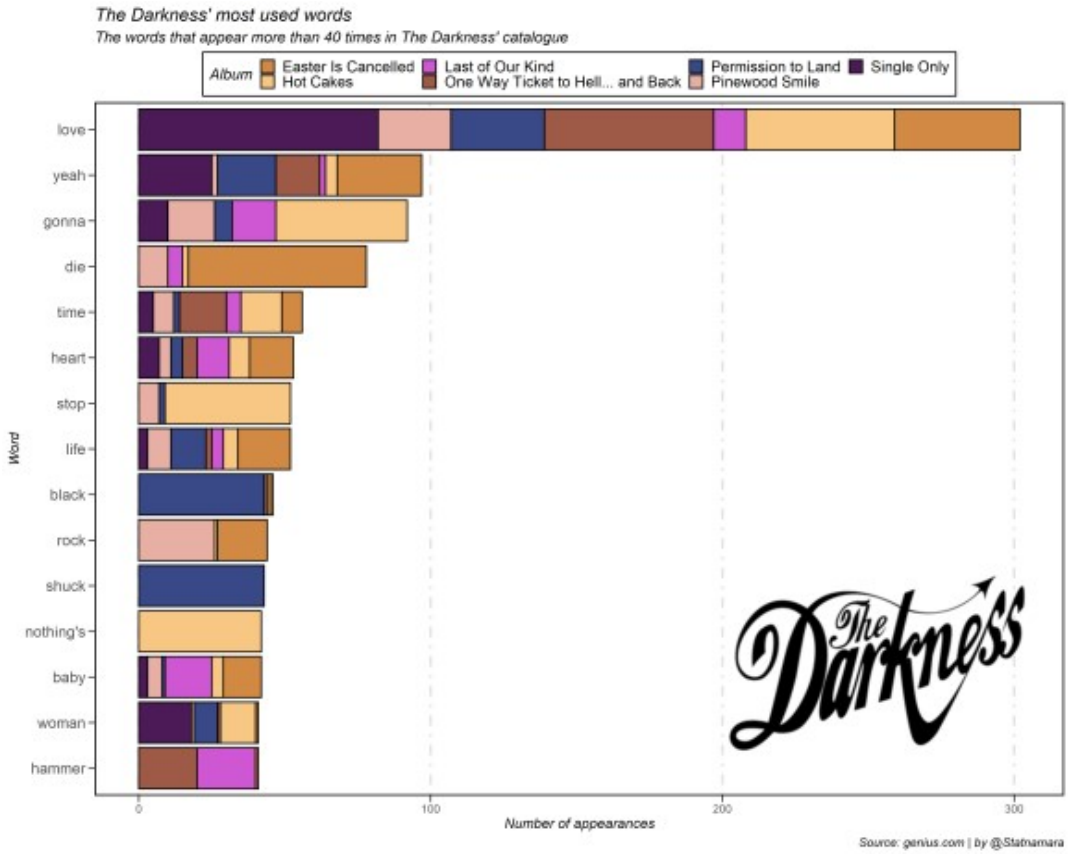
First, I am adding a vector with the colours I want to use for this viz. I used some colours from the artwork of each album to get these. We also need to give the colours names (of the albums they represent), and turn the album column of our word count df into a factor. The factor should contain the levels in reverse chronological order of the albums release date. This will put the albums in release order in our bar chart.

```
# colours for each album
albumCol <- c("#394887",      # PTL
              "#9e5a47",      # OWT
              "#f9c784",      # Hot cakes
              "#cf57d4",      # Last
              "#e8b0a5",      # PINE
              "#d18943",      # Easter
              "#4C1A57")      # singles
names(albumCol) <- c("Permission to Land", "One Way Ticket to Hell... and
Back",
                     "Hot Cakes", "Last of Our Kind", "Pinewood Smile",
"Easter Is Cancelled",
                     "Single Only")

# This ensures bars are stacked in order of release date
topFew$album <- factor(topFew$album, levels = c("Single Only",
                                                "Easter Is Cancelled",
                                                "Pinewood Smile",
                                                "Last of Our Kind",
                                                "Hot Cakes",
                                                "One Way Ticket to Hell... and
Back",
```

```
                                                    "Permission to Land"
))
```

Now we're ready to create a plot. Here, I'm creating a stacked bar chart, flipped to horizontal. The code can be seen below to create the ggplot. I've added the band's logo to the plot by following this blog post from The Mockup.

```
wordsPlot <- ggplot(topFew) +

  geom_bar(aes(x = reorder(word, total),
               y = n,
               fill = as.factor(album)),
           colour = "black",
           stat = "identity") +

  coord_flip() +

  labs(title = "The Darkness' most used words",
       subtitle = "The words that appear more than 40 times in The Darkness'
catalogue",
       caption = "Source: genius.com | by @Statnamara",
       y = "Number of appearances",
       x = "Word",
       fill = "Album")+

  scale_fill_manual(values = albumCol) +

  theme(title = element_text(face = "italic", size = 12),

        panel.border = element_rect(colour = "black", fill=NA, size=1),
        panel.background = element_rect(colour = "black", fill = "white"),
        panel.grid.major.x = element_line(colour="grey90",size = 1, linetype =
4),

        axis.title = element_text(face = "italic",size = 11, colour =
"black"),
        axis.ticks.length = unit(5, units = "pt"),

        legend.background = NULL,
        legend.position = "top",
        legend.key.size = unit(12,"pt"),
        legend.box.spacing = unit(5,"pt"),
        legend.text = element_text(size = 12),

        axis.text.y = element_text(size = 12))

wordsPlot

ggsave(filename = "DarknessWords.png", plot = wordsPlot, width = 30, height =
24, units = "cm",
type = "cairo")
```

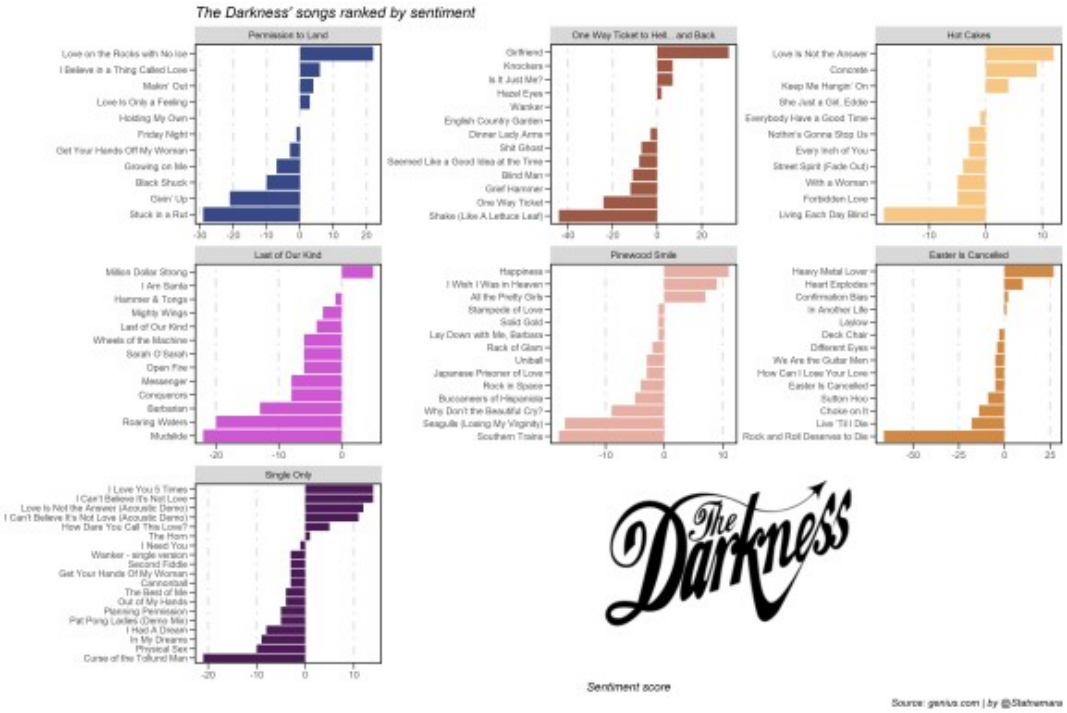


These guys really love love.

With the amount that these guys are signing about love, I have a feeling they're super positive guys in general. Why not check? Let's add a basic sentiment score to our lyrics dataframe and plot that.

```
# Create Sentiment df
darknessSentiments <- tidyLyrics %>%
  inner_join(get_sentiments("bing"))%>%
  count(album, song_name, sentiment) %>%
  spread(sentiment, n, fill = 0) %>%
  mutate(sentiment = positive - negative)

# Factor as we did above
darknessSentiments$album <- factor(darknessSentiments$album,
                                   levels = c("Permission to Land",
                                              "One Way Ticket to Hell... and
Back",
                                              "Hot Cakes",
                                              "Last of Our Kind",
                                              "Pinewood Smile",
                                              "Easter Is Cancelled",
                                              "Single Only"
                                   ))


# sent plot
sentPlot <- ggplot(darknessSentiments,
                   aes(reorder(song_name,
                               sentiment),
                       sentiment,
                       fill = album)) +

  geom_col(show.legend = FALSE) +
```

```
  facet_wrap(~album,
             ncol = 3,
             scales = "free")+

  scale_fill_manual(values = albumCol)+

  labs(title = "The Darkness' songs ranked by sentiment",
       caption = "Source: genius.com | by @Statnamara",
       y = "Sentiment score",
       fill = "Album")+

  theme(title = element_text(face = "italic", size = 12),

      panel.border = element_rect(colour = "black", fill=NA, size=1),
      panel.background = element_rect(colour = "black", fill = "white"),
      panel.grid.major.x = element_line(colour="grey90",size = 1, linetype =
4),

      axis.title.x = element_text(face = "italic",size = 11, colour =
"black"),
      axis.title.y = element_blank(),
      axis.ticks.length = unit(5, units = "pt"),

      legend.background = NULL,
      legend.position = "top",
      legend.key.size = unit(12,"pt"),
      legend.box.spacing = unit(5,"pt")) +

  coord_flip()

sentPlot

ggsave(filename = "DarknessSentiment.png", plot = sentPlot, width = 36, height
= 24, units = "cm",
type = "cairo")
```



Rock and Roll Deserves to Die being by far the most negative definitely makes sense here.