




fitter happier

Last updated on Mar 1, 2017 12 min read  Music (/tags/music), R (/tags/r), Spotify (/tags/spotify)



(<https://twitter.com/intent/tweet?text=fitter%20happier&url=%2fpost%2ffitter-happier%2f>)



(<https://www.facebook.com/sharer.php?u=%2fpost%2ffitter-happier%2f>)



(<https://www.linkedin.com/shareArticle?mini=true&url=%2fpost%2ffitter-happier%2f&title=fitter%20happier>)



(<http://service.weibo.com/share/share.php?url=%2fpost%2ffitter-happier%2f&title=fitter%20happier>)



(<mailto:?subject=fitter%20happier&body=%2fpost%2ffitter-happier%2f>)

Radiohead has been my favorite band for a while, so I am used to people politely suggesting that I

play something “less depressing.” Much of Radiohead’s music is undeniably sad, and this post catalogs my journey to quantify that sadness, concluding in a data-driven determination of their most depressing song.

Getting Data

Spotify’s Web API (<https://developer.spotify.com/web-api/>) provides detailed audio statistics for each song in their library. One of these metrics, “valence,” measures a song’s positivity. From the official API documentation:

A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

So valence provides a measure of how sad a song *sounds* from a musical perspective. Another key component of a song’s sentiment is its lyrics, and it just so happens that Genius Lyrics also has an API (<https://docs.genius.com/>) to pull track-level data. To determine a song’s sadness, I calculated a weighted average of valence and lyrical sentiment. But first, I had to get the data.

[Click here to jump straight to the analysis!](#) (skip API setup and web scraping)

Spotify Web API

Spotify’s API is well documented, but it’s still a pretty involved process to grab all songs for a given artist. In short, Spotify has separate API endpoints for tracks, albums, and artists, each of which needs their own identifying “uri” to access. To simplify the process of grabbing an artist’s entire discography, I created the `spotifyr` (<http://www.rcharlie.com/spotifyr>) package, which you can install from GitHub.

```
devtools::install_github('charlie86/spotifyr')  
library(spotifyr)
```

First, you’ll need to set up a dev account with Spotify here (<https://developer.spotify.com/my-applications/#/applications>). When you have your “client id” and “client secret”, you can authorize your account by setting them to your environment variables.

```
Sys.setenv(SPOTIFY_CLIENT_ID = [YOUR_CLIENT_ID])  
Sys.setenv(SPOTIFY_CLIENT_SECRET = [YOUR_CLIENT_SECRET])
```

Now, you can pull audio features for Radiohead’s entire discography with just one line.

```
spotify_df <- get_artist_audio_features('radiohead')

str(spotify_df)

Classes 'tbl_df', 'tbl' and 'data.frame': 101 obs. of 22 variables:
 $ danceability      : num  0.223 0.515 0.185 0.212 0.364 0.294 0.256 0.384 0.25 0.284
 ---
 $ energy            : num  0.706 0.43 0.964 0.696 0.37 0.813 0.906 0.717 0.62 0.825 ..
 -
 $ key              : num  9 7 9 2 7 4 2 6 0 7 ...
 $ loudness          : num  -12.01 -9.94 -8.32 -10.06 -14.13 ...
 $ mode             : num  1 1 1 1 1 0 1 1 1 1 ...
 $ speechiness       : num  0.0581 0.0369 0.084 0.0472 0.0331 0.0547 0.0548 0.0339 0.06
 11 0.0595 ...
 $ acousticness      : num  0.000945 0.0102 0.000659 0.000849 0.704 0.000101 NA 0.00281
 0.000849 0.00968 ...
 $ instrumentalness  : num  0.0068 0.000141 0.879 0.0165 NA 0.000756 0.366 0.569 0.0848
 0.3 ...
 $ liveness          : num  0.109 0.129 0.107 0.129 0.0883 0.333 0.322 0.187 0.175 0.11
 8 ...
 $ valence           : num  0.305 0.096 0.264 0.279 0.419 0.544 0.258 0.399 0.278 0.269
 ---
 $ tempo            : num  112.9 91.8 147.4 122.4 103.4 ...
 $ track_uri        : chr   "1MyqLTRhgyWPw7v107BEuI" "6b2oQwSGFkzsMtQruIWm2p" "71wIOaoa
 VMUwskK5yCXZL4" ...
 $ duration_ms      : num  208667 238640 132173 325627 161533 ...
 $ time_signature   : num  3 4 4 4 4 4 4 4 4 ...
 $ album_uri        : chr   "6400dnyeDyD2mIFHfkwHXN" "6400dnyeDyD2mIFHfkwHXN" "6400dnye
 DyD2mIFHfkwHXN" ...
 $ track_number     : num  1 2 3 4 5 6 7 8 9 10 ...
 $ track_name       : chr   "You" "Creep" "How Do You?" "Stop Whispering" ...
 $ album_name       : chr   "Pablo Honey" "Pablo Honey" "Pablo Honey" "Pablo Honey" ..
 -
 $ album_img        : chr   "https://i.scdn.co/image/e17011b2aa33289dfa6c0828a0e40d6b56
 ad8820 (https://i.scdn.co/image/e17011b2aa33289dfa6c0828a0e40d6b56ad8820)" ...
 $ album_release_date: chr   "1993-02-22" "1993-02-22" "1993-02-22" "1993-02-22" ...
 $ album_release_year: num   1993 1993 1993 1993 1993 ...
 $ artist_img       : chr   "https://i.scdn.co/image/afcd616e1ef2d2786f47b3b4a8a6aeea24
 a72adc (https://i.scdn.co/image/afcd616e1ef2d2786f47b3b4a8a6aeea24a72adc)" ...
```

Note that this returns more fields than necessary for this particular analysis, but I figured it was worth keeping those extra metrics in the function for future use.

Also, as this analysis focused on just the band's studio albums, I eliminated remixes and EPs.

```
non_studio_albums <- c('TKOL RMX 1234567', 'In Rainbows Disk 2', 'Com Lag: 2+2=5', 'I M
ight Be Wrong', 'OK Computer OKNOTOK 1997 2017')
spotify_df <- filter(spotify_df, !album_name %in% non_studio_albums)
```

Genius Lyrics API

While this data proved to be slightly easier to pull, it was still a multi-step process. Similar to with Spotify, I first used the `search` API call to get the `artist_id`. Go here (https://genius.com/signup_or_login) to set up a dev account to get an API token.

```
token <- 'xxxxxxxxxxxxxxxxxxxx'

genius_get_artists <- function(artist_name, n_results = 10) {
  baseURL <- 'https://api.genius.com/search?q=' (https://api.genius.com/search?q=)
  requestURL <- paste0(baseURL, gsub(' ', '%20', artist_name),
                        '&per_page=', n_results,
                        '&access_token=', token)

  res <- GET(requestURL) %>% content %>% .$response %>% .$hits

  map_df(1:length(res), function(x) {
    tmp <- res[[x]]$result$primary_artist
    list(
      artist_id = tmp$id,
      artist_name = tmp$name
    )
  }) %>% unique
}

genius_artists <- genius_get_artists('radiohead')
genius_artists

# A tibble: 1 × 2
  artist_id artist_name
  <int>      <chr>
1      604    Radiohead
```

Next, I looped through the contents of the `songs` endpoint (the limit is 50 per page), pulling down each result (a list containing the url of the tracks' lyrics) until the `next_page` parameter was null.

```

baseURL <- 'https://api.genius.com/artists/ (https://api.genius.com/artists/)'
requestURL <- paste0(baseURL, genius_artists$artist_id[1], '/songs')

track_lyric_urls <- list()
i <- 1
while (i > 0) {
  tmp <- GET(requestURL, query = list(access_token = token, per_page = 50, page = i))
  %>% content %>% .$response
  track_lyric_urls <- c(track_lyric_urls, tmp$songs)
  if (!is.null(tmp$next_page)) {
    i <- tmp$next_page
  } else {
    break
  }
}

length(track_lyric_urls)
[1] 219

summary(track_lyric_urls[[1]])

```

	Length	Class	Mode
annotation_count	1	-none-	numeric
api_path	1	-none-	character
full_title	1	-none-	character
header_image_thumbnail_url	1	-none-	character
header_image_url	1	-none-	character
id	1	-none-	numeric
lyrics_owner_id	1	-none-	numeric
path	1	-none-	character
pyongs_count	1	-none-	numeric
song_art_image_thumbnail_url	1	-none-	character
stats	3	-none-	list
title	1	-none-	character
url	1	-none-	character
primary_artist	8	-none-	list

From here, I used `rvest` to scrape the “lyrics” elements from the urls provided above.

```

library(rvest)

lyric_scraper <- function(url) {
  read_html(url) %>%
    html_node('lyrics') %>%
    html_text
}

genius_df <- map_df(1:length(track_lyric_urls), function(x) {
  # add in error handling
  lyrics <- try(lyric_scraper(track_lyric_urls[[x]]$url))
  if (class(lyrics) != 'try-error') {
    # strip out non-lyric text and extra spaces
    lyrics <- str_replace_all(lyrics, '\\[(Verse [[:digit:]]|Pre-Chorus [[:digit:]]|Hook [[:digit:]]|Chorus|Outro|Verse|Refrain|Hook|Bridge|Intro|Instrumental)\\]|[:digit:]]|\\.|!|?|\\(|\\)|\\[|\\]|,|'|''')
    lyrics <- str_replace_all(lyrics, '\\n', ' ')
    lyrics <- str_replace_all(lyrics, '([A-Z])', ' \\1')
    lyrics <- str_replace_all(lyrics, '{2,}', ' ')
    lyrics <- tolower(str_trim(lyrics))
  } else {
    lyrics <- NA
  }

  tots <- list(
    track_name = track_lyric_urls[[x]]$title,
    lyrics = lyrics
  )

  return(tots)
})

str(genius_df)

Classes 'tbl_df', 'tbl' and 'data.frame': 219 obs. of 2 variables:
 $ track_name: chr "15 Step" "2 + 2 = 5" "4 Minute Warning" "Airbag" ...
 $ lyrics : chr "how come i end up where i started how come i end" ...

```

After bit of name-standardizing between Spotify and Genius, I left joined `genius_df` onto `spotify_df` by `track_name` (The album info will come in handy later).

```

genius_df$track_name[genius_df$track_name == 'Packt Like Sardines in a Crushd Tin Box']
<- 'Packt Like Sardines in a Crushed Tin Box'
genius_df$track_name[genius_df$track_name == 'Weird Fishes / Arpeggi'] <- 'Weird Fishes
/ Arpeggi'
genius_df$track_name[genius_df$track_name == 'A Punchup at a Wedding'] <- 'A Punch Up a
t a Wedding'
genius_df$track_name[genius_df$track_name == 'Dollars and Cents'] <- 'Dollars & Cents'
genius_df$track_name[genius_df$track_name == 'Bullet Proof...I Wish I Was'] <- 'Bullet
Proof ... I Wish I was'

genius_df <- genius_df %>%
  mutate(track_name_join = tolower(str_replace(track_name, '[:punct:]]', ''))) %>%
  filter(!duplicated(track_name_join)) %>%
  select(-track_name)

track_df <- spotify_df %>%
  mutate(track_name_join = tolower(str_replace(track_name, '[:punct:]]', ''))) %>%
  left_join(genius_df, by = 'track_name_join') %>%
  select(track_name, valence, duration_ms, lyrics, album_name, album_release_year, al
bum_img)

str(track_df)

Classes 'tbl_df', 'tbl' and 'data.frame': 101 obs. of 8 variables:
 $ track_name      : chr  "You" "Creep" "How Do You?" "Stop Whispering" ...
 $ track_number    : num  1 2 3 4 5 6 7 8 9 10 ...
 $ valence         : num  0.305 0.096 0.264 0.279 0.419 0.544 0.258 0.399 0.278 0.269
 ---
 $ duration_ms     : num  208667 238640 132173 325627 161533 ...
 $ lyrics         : chr  "you are the sun and moon and stars are you and i could" ..
 -
 $ album_name      : chr  "Pablo Honey" "Pablo Honey" "Pablo Honey" "Pablo Honey" ..
 -
 $ album_release_year: num  1993 1993 1993 1993 1993 ...
 $ album_img       : chr  "https://i.scdn.co/image/e17011b2aa33289dfa6c0828a0e40d6b5
(https://i.scdn.co/image/e17011b2aa33289dfa6c0828a0e40d6b5)" ...

```

Now onto the analysis!

Quantifying Sentiment

Using valence alone, calculating the saddest song is pretty straightforward - the song with the lowest valence wins.

```
track_df %>%
  select(valence, track_name) %>%
  arrange(valence) %>%
  slice(1:10)
```

	valence	track_name
1	0.0378	We Suck Young Blood
2	0.0378	True Love Waits
3	0.0400	The Tourist
4	0.0425	Motion Picture Soundtrack
5	0.0458	Sail To The Moon
6	0.0468	Videotape
7	0.0516	Life In a Glasshouse
8	0.0517	Tinker Tailor Soldier Sailor...
9	0.0545	The Numbers
10	0.0585	Everything In Its Right Place

Would that it were so simple. “True Love Waits” and “We Suck Young Blood” tie here, each with a valence of 0.0378, further illustrating the need to factor in lyrics.

While valence serves as an out-of-the box measure of musical sentiment, the emotions behind song lyrics are much more elusive. To find the most depressing song, I used sentiment analysis to pick out words associated with sadness. Specifically, I used `tidytext` and the NRC lexicon, based on a crowd-sourced project (<http://saifmohammad.com/WebPages/NRC-Emotion-Lexicon.htm>) by the National Research Council Canada. This lexicon contains an array of emotions (sadness, joy, anger, surprise, etc.) and the words determined to most likely elicit them.

To quantify sad lyrics, I calculated the share of “sad” words per song, filtering out “stopwords” (e.g. “the,” “and,” “I”).


```
library(tidytext)

sad_words <- sentiments %>%
  filter(lexicon == 'nrc', sentiment == 'sadness') %>%
  select(word) %>%
  mutate(sad = T)

sent_df <- track_df %>%
  unnest_tokens(word, lyrics) %>%
  anti_join(stop_words, by = 'word') %>%
  left_join(sad_words, by = 'word') %>%
  group_by(track_name) %>%
  summarise(pct_sad = round(sum(sad, na.rm = T) / n(), 4),
            word_count = n()) %>%
  ungroup

sent_df %>%
  select(pct_sad, track_name) %>%
  arrange(-pct_sad) %>%
  head(10)
```

	pct_sad <dbl>	track_name <chr>
1	0.3571	High And Dry
2	0.2955	Backdrifts
3	0.2742	Give Up The Ghost
4	0.2381	True Love Waits
5	0.2326	Exit Music (For a Film)
6	0.2195	Karma Police
7	0.2000	Planet Telex
8	0.1875	Let Down
9	0.1842	A Punch Up At a Wedding
10	0.1800	Identikit

By the percentage of non-stopwords that were sad, “High And Dry” wins, with about 36% of its lyrics containing sad words. Specifically, the algorithm picked out the words “broke,” “fall,” “hate,” “kill,” and “leave” - the last of which was repeated 15 times in the chorus (“Don’t leave me high, don’t leave me dry.”)

Lyrical Density

To combine lyrical and musical sadness I turned to an analysis (<https://www.r-bloggers.com/everything-in-its-right-place-visualization-and-content-analysis-of-radiohead-lyrics/>) by Myles Harrison, a fellow R Blogger, which coincidentally also dealt with Radiohead lyrics. He explored the concept of “lyrical density,” which is, according to his definition - “the number of lyrics per song over the track length.” One way to interpret this is how “important” lyrics are to a given song, making it the perfect weighting metric for my analysis. Note that my version of lyrical density is slightly modified as it excludes stopwords.

Using track duration and word count, I calculated lyrical density for each track. To create the final “gloom index,” I took the average of valence and the percentage of sad words per track, weighted by lyrical density.

$$gloomIndex = \frac{(1 - valence) + pctSad * (1 + lyricalDensity)}{2}$$

I also rescaled the metric to fit within 1 and 100, so that the saddest song had a score of 1 and the least sad song scored 100.

```
library(scales)

track_df <- track_df %>%
  left_join(sent_df, by = 'track_name') %>%
  mutate_at(c('pct_sad', 'word_count'), funs(ifelse(is.na(.), 0, .))) %>%
  mutate(lyrical_density = word_count / duration_ms * 1000,
         gloom_index = round(rescale(1 - ((1 - valence) + (pct_sad * (1 + lyrical_density)))) / 2, to = c(1, 100)), 2))
```

Drum roll...

```
track_df %>%
  select(gloom_index, track_name) %>%
  arrange(gloom_index) %>%
  head(10)
```

	gloom_index	track_name
	<dbl>	<chr>
1	1.00	True Love Waits
2	6.46	Give Up The Ghost
3	9.35	Motion Picture Soundtrack
4	13.70	Let Down
5	14.15	Pyramid Song
6	14.57	Exit Music (For a Film)
7	15.29	Dollars & Cents
8	15.69	High And Dry
9	15.80	Tinker Tailor Soldier ...
10	16.03	Videotape

We have a winner! “True Love Waits” is officially the single most depressing Radiohead song to date. Rightly so, given that it tied for lowest valence (0.0378) and ranked fourth for highest percentage of sad words (24%). If the numbers still don’t convince you, just listen to it (<https://vimeo.com/170620454>).

To see how sadness evolved across all nine albums, I calculated the average gloom index per album and plotted each song by album release date. To spice up the `highcharter` plot a bit, I created a custom tooltip incorporating the `album_img` from Spotify.

```

library(RColorBrewer)
library(highcharter)

plot_df <- track_df %>%
  rowwise %>%
    mutate(tooltip = paste0('<a style = "margin-right:', max(max(nchar(track_name), nchar(album_name)) * 7, 55), 'px">', # dynamic sizing
                           '<img src=', album_img, ' height="50" style="float:left;margin-right:5px">',
                           '<b>Album:</b> ', album_name,
                           '<br><b>Track:</b> ', track_name)) %>%
  ungroup

avg_line <- plot_df %>%
  group_by(album_release_year, album_name, album_img) %>%
  summarise(avg = mean(gloom_index)) %>%
  ungroup %>%
  transmute(x = as.numeric(as.factor(album_release_year)),
            y = avg,
            tooltip = paste0('<a style = "margin-right:55px">',
                           '<img src=', album_img, ' height="50" style="float:left;margin-right:5px">',
                           '<b>Album:</b> ', album_name,
                           '<br><b>Average Gloom Index:</b> ', round(avg, 2),
                           '</a>'))

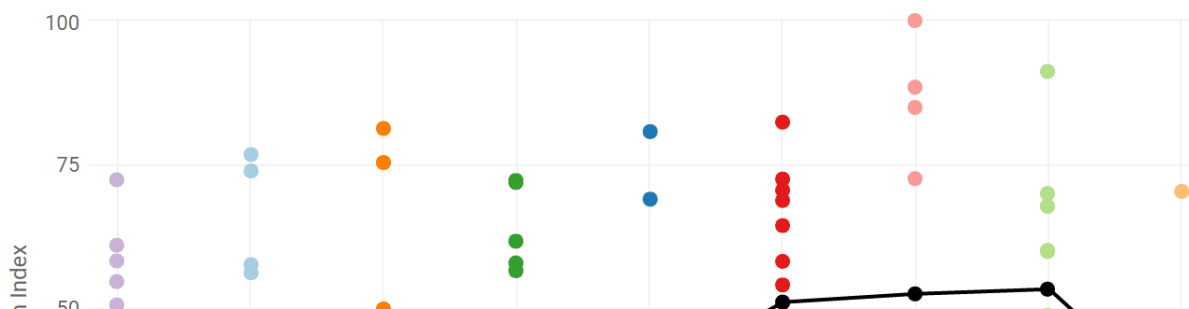
plot_track_df <- plot_df %>%
  mutate(tooltip = paste0(tooltip, '<br><b>Gloom Index:</b> ', gloom_index, '</a>'),
         album_number = as.numeric(as.factor(album_release_year))) %>%
  ungroup

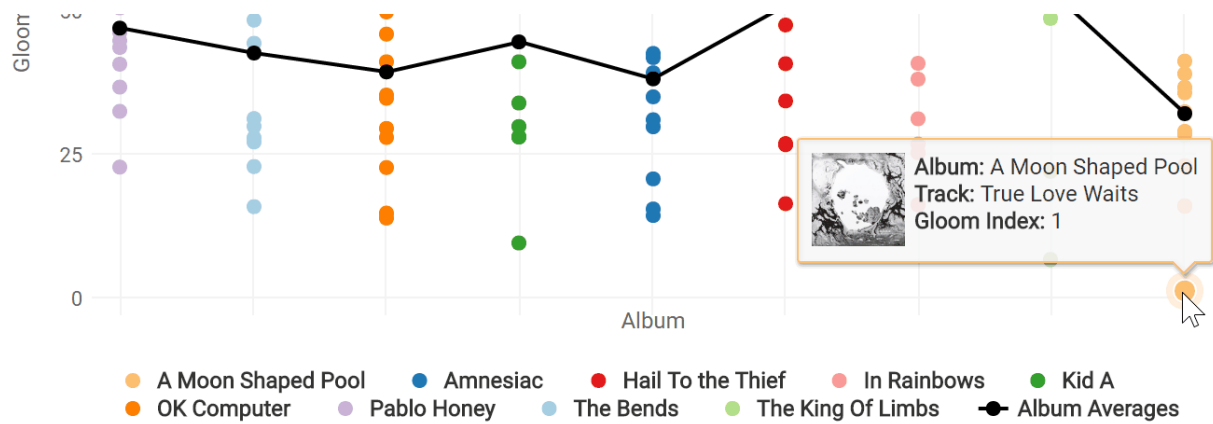
album_chart <- hchart(plot_track_df, 'scatter', hcaes(x = as.numeric(as.factor(album_release_year)), y = gloom_index, group = album_name)) %>%
  hc_add_series(data = avg_line, type = 'line') %>%
  hc_tooltip(formatter = JS(paste0("function() {return this.point.tooltip;}")), useHTML = T) %>%
  hc_colors(c(sample(brewer.pal(n_distinct(track_df$album_name), 'Paired')), 'black')) %>%
  hc_xAxis(title = list(text = 'Album'), labels = list(enabled = F)) %>%
  hc_yAxis(max = 100, title = list(text = 'Gloom Index')) %>%
  hc_title(text = 'Data Driven Depression') %>%
  hc_subtitle(text = 'Radiohead song sadness by album') %>%
  hc_add_theme(hc_theme_smpl())
album_chart$x$hc_opts$series[[10]]$name <- 'Album Averages'
album_chart

```

Data Driven Depression

Radiohead song sadness by album





(/img/posts/fitter-happier/album_chart.html)

Click to view plot in new window (/img/posts/fitter-happier/album_chart.html)

Of all nine studio albums, Radiohead's latest release, "A Moon Shaped Pool," boasts the lowest average gloom index. This is driven largely by the fact that its finale, "True Love Waits," was the gloomiest song overall. It's also apparent that "A Moon Shaped Pool" broke a trend of increasingly less depressing albums since 2003's "Hail to the Thief" and directly followed the band's least sad album, "The King of Limbs."

This was a really fun dataset to work with, and there are plenty of other interesting things to explore here (artist comparisons, within album sadness, additional song features, etc.). In fact, Andrew Clark made this awesome shiny app (<https://mytinyshinys.shinyapps.io/spotifyFlexDB/>) that allows you to explore the valence, danceability, and instrumentality of any artist of your choosing - check it out!

To further explore the emotion of your own favorite artists and Spotify playlists, I've made a musical sentiment app (<http://www.rcharlie.net/sentify>) with RShiny to supplement this article (<https://www.1843magazine.com/data-graphic/the-daily/youre-happy-and-they-know-it>) from The Economist's 1843 Magazine.

Update: March 1, 2017

The gloom index for "Weird Fishes/ Arpeggi" in an earlier version of this post was erroneously calculated with missing lyrics due to a naming discrepancy between the Genius and Spotify APIs. Resolving the issue dropped the track's gloom index from 40.43 to 25.1 and the "In Rainbows" album average from 53.85 to 52.31. The graph and text have been updated to reflect this change, and the `hchart` code is now compatible with version 0.5.0 of `highcharter`. The code for scraping Genius Lyrics has also been updated to include error handling and better eliminate punctuation and meta-info contained in the lyrics, neither of which significantly affected the gloom index in the previous version.

Related

- [Sentify \(/project/sentify/\)](/project/sentify/)

[CoachellaR - A Cluster Analysis → \(/post/coachellar/\)](/post/coachellar/)

0 Comments

rCharlie

 Login Recommend 2 Tweet Share

Sort by Best



Start the discussion...

LOG IN WITH



OR SIGN UP WITH DISQUS ?

Be the first to comment.

ALSO ON RCHARLIE

fitter happier

154 comments • 2 years ago

Abhinav — This made my day, thank you!**CoachellaR - A Cluster Analysis**

8 comments • a year ago

J Walt Granecki — What if I don't want to use the best partition and I want to force it to use 4 clusters in your plot? Subscribe  Add Disqus to your siteAdd DisqusAdd Disqus' Privacy PolicyPrivacy PolicyPrivacy