# Spotify Playlist Manager- Sta 323 Final Project

Garrett Allen

## Setup

## Introduction

```
#setting up API. Make sure when get_spotify_authorization_code runs, you
#verify with spotify and select "No" when it asks if you'd like to cache results in Console.
#RUN THIS BEFORE ALL OTHER CODE
id <- "0798c2bac40840aeb6759bf3a039f609"
secret <- "8cf2bcb368b543c8b198e477b3a3dd23"
Sys.setenv(SPOTIFY_CLIENT_ID = id)
Sys.setenv(SPOTIFY_CLIENT_SECRET = secret)
access_token <- get_spotify_access_token()
auth_code <- get_spotify_authorization_code()
```

For this project, I wanted to make something really practical that I would use. After I thought for awhile, I realized a common issue I have is that there's not a lot of nice tools for Spotify to manage playlists. I store all my music in one large mega-playlist (about ~3000 songs), which, while I like, doesn't let me easily analyze my music or communicate my taste in music to others. For example, if I wanted to send all of my music of a certain genre, or a certain feel, I'd have no way of doing so easily without manually doing so. This is clearly intractable for me now with the amount of music in my playlist, and it's even worse for some of my friends, who have playlists with $>$ 10,000 songs.

As a result, I resolved that I wanted to make a ShinyApp that made it easy to do a few key tasks with playlists. For one, I wanted to be able to filter my playlists by a variety of features easily, as well as have visualizations that let me understand both the distribution of my music and how to effectively filter it. I also wanted to make the app capable of handling Liked Songs, an internal playlist that Spotify doesn't normally let you turn into public playlists. My friend has all of his music in his Liked Songs, and it makes it really difficult for him to share music in a mass-way.

Finally, I wanted to have a clean implementation to combine playlists together, as the current method of dragging and dropping in the Spotify client becomes really cumbersome for large datasets.

## Methods / Implementation

To implement all of this, I used a variety of packages. SpotifyR is a nice wrapper around the Spotify API so that I don't have to deal with some of deeper intricacies of the Spotify API. While I considered foregoing the package at first so I could have more flexiility, I found that the wrapper had more than enough flexibility to be useful for my purposes.

I also used a small package called ggoutlier to make a nice histogram that avoided being too skewed due to high follower counts of a few artists. The rest of the packages are fairly standard or self explanatory.

One of the most annoying aspects of this project was figuring out how to use an authorization code with Spotify's OAuth 2.0 system. This is necessary to access data from Spotify users that is not public, as well as necessary to create playlists from song ids. The details are not so important now, but the documentation on Spotify's website is rather poor.

It may be necessary for me to grant you access to my Spotify app in development mode for the OAuth to work properly; if that is true, please email me so I can add both your email and name registered to your Spotify account to Spotify's API manager. It should all work and be reproducible, as I tested it both on my computer and a friend's, but getting the browser to properly authenicate with R has been a bit of a difficulty, so fingers crossed.

After getting the API keys from Spotify and implementing OAuth 2.0 in my Spotify App Developer page, I decided that I wanted to make a few functions that would make my life easier (specifically around Pagination) and then leverage these functions in my app to make the code cleaner. The pagination functions are all rather similar, although there had to be minor adjustments based on the specific operation happening. They are all variations of grabbing x results at a time and then shifting upwards by x in position to grab the next x results.

The main meat of the project is the ShinyApp itself; I did end up implementing every aspect that I wanted, and I am now able to filter Spotify Playlists/Liked Songs by tempo, danceability, energy, valence (happiness), speechiness, acousticness, number of followers the artist has, and the genre, as well as combine playlists together and extract these playlists to my Spotify account.

A few notes on using the app: to get a playlist_id, one needs to first create a playlist on Spotify, then get the share link. After getting the link to share it, you must extract the part between playlist/ and ?. I have loaded some default values for ease of verifying that the app works.

For example, in the link below, the ID would be 5SNDR83wfKeMvWhcablylH.

https://open.spotify.com/playlist/5SNDR83wfKeMvWhcablylH?si=c5f38043f81b4114

The app is rather robust to failed inputs due to a lot of tryCatch loops that validate various sections of code. It should not crash easily.

Spotify's requests per day is very generous, and I never ran into an issue where Spotify ratelimited my usage of their API, so I have not put in to place any safeguards to protect against ratelimiting; I don't anticipate this will be an issue, as I made upwards of hundreds of calls in a short amount of time without triggering any ratelimiting.

I used a navPage to make the app easier to navigate, along with several tabPanels that made the main panel significantly less cluttered. In order to generate the data seen in the main panel under "Songs", I had to extract songs from a playlist using get_all_playlist_tracks, then extract more information from the tracks with get_all_x_features using the function get_track_audio_features, and then pull artist information from get_all_x_features with the function get_artists. Note that tracks with more than 1 artists are duplicated in the output, as Spotify classes the genre of a song by the artists' genre. That is, Spotify does no track level genre classification, so its important to look at the genre of all artists collaborating on a track in order to get a clear view of the genres present. I then joined all the dataframes together with joining functions, using various unique IDS to combine them together.

After getting the dataset, the code is mostly straightforward; it applies filtering if the user selects it, generates a playlist if the user selects it using the methods create_playlist and add_all_tracks_to_playlist, and generates a few visualizations. Updates only occur upon pressing the action button at the bottom, as we would hope.

If you try a large playlist with the app, give it some time to generate results: pagination makes some of the operations quite slow, particularly the artist one, as results can only be grabbed 20 at a time. Results are still fairly fast on playlists of thousands, however.

```
#gets all songs in a playlist
get_all_playlist_tracks <- function(id){
  my_tracks <- get_playlist_tracks(playlist_id = id)
  new_page = 0
  prev_tracks = 1
  #handles pagination for indeterminate size of playlist by going until my_tracks doesn't increase in s
```

```
  while(nrow(my_tracks) != prev_tracks){
    prev_tracks = nrow(my_tracks)
    new_page = new_page + 100
    new_tracks <- get_playlist_tracks(playlist_id = id, offset = new_page)
    my_tracks <- rbind(my_tracks,new_tracks)
    print(glue("get_all_playlist_tracks: Processing page {new_page / 100} of playlist"))
  }
  my_tracks %>% #filtering bad row
    select(-(2:4),-(6:8),-13,-17,-21,-23,-33,-40)

}

#gets saved tracks from user's library, similar to previous method
get_all_saved_tracks <- function(auth_code){
  my_tracks <- get_my_saved_tracks(limit = 20)
  new_page = 0
  prev_tracks = 1
  while(nrow(my_tracks) != prev_tracks){
    prev_tracks = nrow(my_tracks)
    new_page = new_page + 20
    new_tracks <- get_my_saved_tracks(offset = new_page, authorization = auth_code)
    my_tracks <- rbind(my_tracks,new_tracks)
    print(glue("Processing page {new_page / 20}"))
  }
  my_tracks

}

#gets all "features" from a type of thing. written generically to allow
# for getting album features, track features, artist features, such as
#genre, followers, internal spotify data, etc. handles pagination
#enforced by spotifyR package.
get_all_x_features <- function(all_ids,limit,func){

  if(length(all_ids) <= limit){
    return(func(ids = all_ids))
  }

  features <- func(ids = all_ids[1:limit])
  len_ids <- length(all_ids)
  num_pages <- floor(len_ids / limit)
  num_last_page <- len_ids - num_pages*limit
  page <- 1
  while(page != num_pages){
    print(glue("get_all_x_features: Processing page {page} of features"))
    new_features <- func(ids = all_ids[(page*limit + 1):(page*limit + limit)])
    features <- rbind(features,new_features)
    page = page + 1
  }
  print(glue("get_all_x_features: Processing page {page} of features"))
  new_features <- func(ids = all_ids[(num_pages*limit + 1):(num_pages*limit + num_last_page)])
  rbind(features,new_features)
}
```

```r
#function POSTs tracks to a playlist that is owned by a certain spotify account
#very similar to previous pagination methods, except it adds tracks to a playlist
#instead of extracting info
add_all_tracks_to_playlist <- function(user_id, all_ids,auth_code,playlist_id,name){

  if(length(all_ids) <= 80){
    return(add_tracks_to_playlist(playlist_id = playlist_id, uris = all_ids))
  }
  print(glue("add_all_tracks_to_playlist: Adding songs 1-80 to playlist {name}"))
  add_tracks_to_playlist(playlist_id = playlist_id,
                         uris = all_ids[1:80],
                         authorization = auth_code)
  len_ids <- length(all_ids)
  num_pages <- floor(len_ids / 80)
  num_last_page <- len_ids - num_pages*80
  page <- 1
  while(page != num_pages){
    print(glue("add_all_tracks_to_playlist: Adding songs {page*80}-{page*80 + 80} to playlist {name}"))
    add_tracks_to_playlist(playlist_id = playlist_id,
                           uris = all_ids[(page*80+ 1):(page*80 + 80)],
                           authorization = auth_code)
    page = page + 1
  }
  print(glue("add_all_tracks_to_playlist: Adding songs {num_pages*80}-{num_pages*80 + num_last_page} to
  add_tracks_to_playlist(playlist_id = playlist_id,
                         uris = all_ids[(num_pages*80 + 1):(num_pages*80 + num_last_page)],
                         authorization = auth_code)
}
```

```r
run_app <- function(auth_code){
  #handles ui logic for two apps; an app that combines playlists, and an
  #app that visualizes and filters playlist data.
  ui <- navbarPage(title = "Spotify Playlist Manager",
                   tabPanel("Visualizations and Filtering",
                       sidebarLayout(
                         sidebarPanel = sidebarPanel(
                           selectInput("type","Type of Playlist",choices = c("Liked Songs","Playlist")),
                           conditionalPanel(
                             condition = "input.type == 'Playlist'",
                             helpText("A playlist ID can be found by getting the share link for a Spotif
                             textInput("id","Playlist ID", value = "2EgjrgDSeAF25yPq7No1mG"),
                           ),
                           checkboxInput("filter","Filter results?", value = FALSE),
                           conditionalPanel(
                             condition = "input.filter",
                             numericInput("followers_min","Minimum Followers",value = 0, min = 0),
                             numericInput("followers_max", "Maximum Followers",value = 9999999999, min =
                             helpText("Enter all for genre to not filter by genre in output."),
                             textInput("genre","Genre"),
                             sliderInput("danceability","Danceability", min = 0, max = 1,value = c(0,1))
                             sliderInput("energy","Energy", min = 0, max = 1, value = c(0,1)),
                             sliderInput("speechiness","Speechiness", min = 0, max = 1, value = c(0,1)),
                             sliderInput("valence","Valence",min = 0, max = 1, value = c(0,1)),
```

```
                        sliderInput("acousticness","Acousticness",min = 0, max = 1, value = c(0,1)),
                        sliderInput("tempo","Tempo (bpm)",min = 0, max = 500, value = c(0,500))
                      ),
                      checkboxInput("playlist","Create Playlist of Results?"),
                      conditionalPanel(
                        condition = "input.playlist",
                        textInput("user_name","Username"),
                        textInput("name","Name of Playlist")
                      ),
                      actionButton("run_viz","Get Songs")
                      ),
                    mainPanel = mainPanel(
                      tabsetPanel(
                        tabPanel("Songs",
                                 DTOutput("songs")),
                        tabPanel("Spotify Variable Summary",
                                 plotOutput("graph_sum"),
                                 plotOutput("followers"),
                                 DTOutput("summary")),
                        tabPanel("Genre Summary",
                                 DTOutput("genre"))
                      )

                    )
                  )
                ),
                tabPanel("Combine Two Playlists", #app for combining playlists
                         sidebarLayout(
                           sidebarPanel = sidebarPanel(
                            textInput("playlist1","Playlist 1 ID", value = "76ZauyHiAEI4ftqo6hfyt4")
                            textInput("playlist2", "Playlist 2 ID", value = "3Nb7Xx6ohBfOHqOFVMMgO7
                            textInput("userid","User_Name"),
                            textInput("new_name","New Playlist Name"),
                            actionButton("run_comb","Combine")),
                           mainPanel = textOutput("complete")
                         )
                         )
            )

server <- function(input,output,session){

  observe({
    #following code produces the dataframe used for all of proceeding calculations
    #has genre information, spotify internal data (i.e. valence), date,
    #number of followers for an artist, etc.

    if(input$type == "Liked Songs")
      tracks <- get_all_saved_tracks(auth_code = auth_code)
    else{
      tryCatch({ #used from midterm2 to check for errors,
      tracks <- get_all_playlist_tracks(id = input$id)
    },
    warning = function(warn){
```

```
        shinyalert(paste0(warn),
                   type = "warning",
                   size = "l")
    },
    error = function(err){
        shinyalert(paste0(paste0(err), "\n Check to make sure the Playlist ID was entered correctly.")
                   type = 'error',
                   size = "l")
      validate(need(exists("playlist"), message = FALSE))


    })
  }
    validate(need(exists("tracks"), message = FALSE)) #checks that result worked

    feats <- get_all_x_features(tracks$track.id, func = get_track_audio_features, limit = 100) %>%
      select(-12,-(14:17))
    more_info_tracks <- full_join(tracks,feats, by = c("track.id" = "id")) %>%
      unnest(track.artists)

    print("Processed all track features, grabbing genres of artists")
    artists <- get_all_x_features(more_info_tracks$id, func = get_artists, limit = 50) %>%
      select(genres,followers.total,id)

    print("Finished processing genres of artists, cleaning up and combining")
    all_info_tracks <- unique(full_join(more_info_tracks,artists, by = "id"))
    print("Done!")

    all_info_tracks <- all_info_tracks %>%
      mutate(added_at = as.Date(added_at),
             track.album.release_date = as.Date(track.album.release_date)
         ) %>%
      rename(Artist = name,
               Track_Title = track.name,
               Album = track.album.name,
               Genre = genres,
               Artist_Followers = followers.total)

    if(input$filter){ #if filtering is turned on, does the appropriate filtering
      all_info_tracks <- all_info_tracks %>%
        filter(valence > input$valence[1] & valence < input$valence[2]) %>%
        filter(energy > input$energy[1] & energy < input$energy[2]) %>%
        filter(acousticness > input$acousticness[1] & acousticness < input$acousticness[2]) %>%
        filter(speechiness > input$speechiness[1] & speechiness < input$speechiness[2]) %>%
        filter(danceability > input$danceability[1] & danceability < input$danceability[2]) %>%
        filter(tempo > input$tempo[1] & tempo < input$tempo[2]) %>%
        filter(Artist_Followers > input$followers_min & Artist_Followers < input$followers_max) %>%
        filter(str_detect(Genre, input$genre) | str_to_lower(input$genre) == "all")

    }
    if(nrow(all_info_tracks) == 0){ #if genre has no rows, tells the user before errors occur
      shinyalert(title = "Error",
                 "Result has no rows. Make your filter conditions more lenient.",
```

```
                type = "error",
                size = "l")
  }

  validate(need(nrow(all_info_tracks) > 0, message = FALSE))

  if(input$playlist){ #if user has asked for results to be put into playlist, checks to make sure i
    tryCatch({
    playlist_filter <- create_playlist(user_id = input$user_name,
                        name = input$name,
                        authorization = auth_code)
  },
  warning = function(warn){
      shinyalert(paste0(warn),
                 type = "warning",
                 size = "l")
  },
  error = function(err){
      shinyalert(paste0(paste0(err), "\n Make sure Username was entered correctly and playlist name
                 type = 'error',
                 size = "l")
  })
  validate(need(exists("playlist_filter"), message = FALSE))
  add_all_tracks_to_playlist(user_id = input$user_name, #adding tracks
                             playlist_id = playlist_filter$id,
                             all_ids = all_info_tracks$track.id,
                             auth_code = auth_code,
                             name = input$name)
}

  output$songs <- renderDT({ #renders key aspects of playlist returned
    all_info_tracks %>%
      select(Artist, Track_Title, Album, Genre, Artist_Followers)
    })

  output$graph_sum <- renderPlot({ #CDF plot of internal variables
  all_info_tracks %>%
    pivot_longer(cols = c(acousticness,speechiness,valence,danceability,energy),
                 names_to = "Spotify Internal Variables",
                 values_to = "spotify_val") %>%
    ggplot(aes(x = spotify_val, color = `Spotify Internal Variables`)) +
             stat_ecdf() +
    theme_bw() +
    labs(x = "Values",
         y = "Proportion of Songs Below Value",
         title = "Empirical CDFs of Internal Spotify Variables")
  })

  output$summary <- renderDT({ #summary stats
    all_info_tracks %>%
        summarize(mean_danceability = mean(danceability),
        mean_speechiness = mean(speechiness),
        mean_energy = mean(energy),
```

```
          mean_valence = mean(valence),
          mean_tempo = mean(tempo),
          mean_acousticness = mean(acousticness),
          median_follower_count_of_artists = median(Artist_Followers)
          )
    })
    output$followers <- renderPlot({ #followers histogram
      ggoutlier_hist(all_info_tracks,
                     var_name = "Artist_Followers",
                     cut_off_ceiling =
                       quantile(all_info_tracks$Artist_Followers,.9)) +
        labs(x = "Followers", y = "Count", title = "Distribution of Artist Follows Below 90th Percenti
        theme_bw()
    })
    output$genre = renderDT({ #summarizes genre of playlists
      all_info_tracks %>%
        count(Genre) %>%
        arrange(desc(n)) %>%
        rename(`Number of Songs` = n)
        })

  }) %>%
    bindEvent(input$run_viz)


  observe({ #for the 2nd app
    tryCatch({ #used from midterm2
      play1 <- get_all_playlist_tracks(id = input$playlist1) #reads in both playlists
      play2 <- get_all_playlist_tracks(id = input$playlist2)
      playlist <- create_playlist(user_id = input$userid,
                          name = input$new_name,
                          authorization = auth_code)
    },
    warning = function(warn){
        shinyalert(paste0(warn),
                   type = "warning",
                   size = "l")
    },
    error = function(err){
        shinyalert(paste0(paste0(err), "\n Check to make sure the Playlist IDs/Username was entered c
                   type = 'error',
                   size = "l")
    })

    #validates that playlist creation and grabbing each playlist was successful
    validate(need(exists("play1"), message = FALSE))
    validate(need(exists("play2"), message = FALSE))
    validate(need(exists("playlist"), message = FALSE))
    new_play <- rbind(play1,play2)
    #combines tracks
    add_all_tracks_to_playlist(user_id = input$userid,
                               playlist_id = playlist$id,
                               all_ids = new_play$track.id,
```

```
                                auth_code = auth_code,
                                name = input$new_name)
      #prints textoutput notifying user task succeeded.
      output$complete <- renderPrint({"Successfully combined playlists"})
    }) %>%
       bindEvent(input$run_comb)
  }

  observe({

  })

  shinyApp(ui,server)
}
```

```
run_app(auth_code)
```

## Discussion & Conclusions

While the app does everything I wanted to do for the most part, there is still some room for improvement. In particular, there could be more visualization/analysis of the data itself (such as time-series analysis with the Date column) but I wanted to keep the app focused on playlist management rather than detailed analysis. Additionally, the genre filter could be more robust, as it does not work well if you want to filter for several different genres together; however, you can filter by a superset (e.g. genre = indie will return indie rock, indie pop, indietronic, etc.) so this is not as big a concern as it might appear at first. I also know that I could have reduced the scope of my ask for OAuth 2.0, but I didn't feel like playing around with what specific scopes I needed for my analysis. In the future I plan to improve this, as I certainly need a lot less permissions than I ask for to make my app work.

Overall, I feel satisfied with my app, although I could potentially make it more pretty so it looks less like a ShinyApp. I do intend to return to this later and improve it more, so any feedback would be appreciated! Thank you and all the TAs for such an interesting/useful course, its filled in a lot of gaps in my knowledge and I enjoyed it a ton. Have a great summer!