Modeling Spotify User Behaviour

Luka Vukovic

11/05/2021

Import Libraries

```
library(tidyverse)
library(data.table)
library(glue)
library(lubridate)

# PCA
library("FactoMineR")
library("factoextra")
#library(psych)

# ML
library(randomForest)

# For visualization
library(cowplot)
library(ggplot2)
library(quantreg)
library(tidyquant) ## rolling average for geom_smooth
library(grid)

# Themes
library(ggthemes)
library(egg)
library(RColorBrewer)
```

Data Cleaning and Exploration

Import API constructed user library and streaming data

```
# User 1
usr1_lib <- read.csv('Enhanced_Data/User1/usr1_lib.csv')
usr1_streams <- read.csv('Enhanced_Data/User1/usr1_streams.csv')

# User 2
usr2_lib <- read.csv('Enhanced_Data/User2/usr2_lib.csv')
usr2_streams <- read.csv('Enhanced_Data/User2/usr2_streams.csv')

# put in a single list
data <- list(usr1_lib, usr1_streams, usr2_lib, usr2_streams)
```

Minor data cleaning

Notes: - Most undefined rows that the API couldn't retrieve information for have been removed in the original data acquisition script - The cleanup function removes any leftover rows containing NA values. (It seems just a few rows still contain NAs)

```
cleanup <- function(df) {
  # Keep only rows without NA values
  df <- df[!rowSums(is.na(df)) >= 1, ];  df
}


clean_data <- lapply(data, cleanup)

paste('Lefover NAs in each library/streaming set: ')
```

```
## [1] "Lefover NAs in each library/streaming set: "
```

```
unlist(lapply(data, nrow)) - unlist(lapply(clean_data, nrow))
```

```
## [1]  1  0 16 16
```

```
# user 1 libary / user 1 streams / user 2 library / user 2 streams
```

Functions for Top Songs and Genres

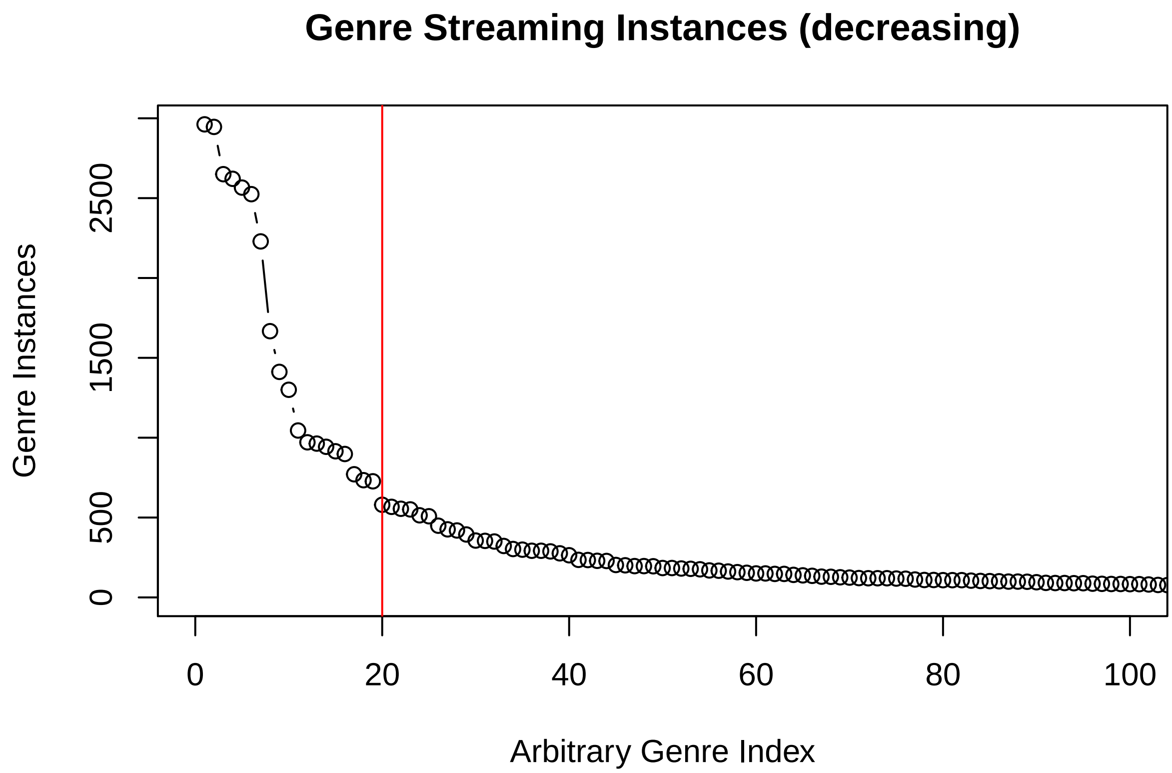
```
## Function to retrieve a user's top X songs
## Requires a streaming dataframe
top_x_songs <- function(streaming_frame, x) {
  summary_streams <- streaming_frame %>%
    group_by(trackName, artistName, # main vars
             danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, li
    summarise(plays = n(), hoursPlayed = sum(msPlayed)/(1000*60*60))

  top_tracks <- head(summary_streams[order(summary_streams$hoursPlayed, decreasing = TRUE), ], x)
  top_tracks
}

## Function to retrieve a user's top X genres
## Requires either a streaming or library dataframe
top_x_genres <- function(df, x = NULL, library = FASLE) {
  summary_lib <- df %>%
    group_by(genres) %>%
    summarise(instances = n())

  # need the total songs or streams to get a genre's %occurrence
  if (library == FALSE) {
    n_songs <- df %>% nrow() # n streams
  } else {
    n_songs <- df %>% group_by(id) %>% summarise() %>% nrow()  # n songs
  }

  summary_lib$`%occurence` <- round(100*summary_lib$instances / n_songs, 2)
  if (is.null(x)) {
    top_genres <- summary_lib[order(summary_lib$instances, decreasing = TRUE), ]
    return(top_genres)
```

```r
  } else {
    top_genres <- head(summary_lib[order(summary_lib$instances, decreasing = TRUE), ], x)
    return(top_genres)
  }
}
```

```r
## User 1 top 20 songs
top_x_songs(clean_data[[2]], 10)
```

```
## `summarise()` has grouped output by 'trackName', 'artistName', 'danceability', 'energy', 'key', 'lou

## # A tibble: 10 x 16
## # Groups:   trackName, artistName, danceability, energy, key, loudness, mode,
## #   speechiness, acousticness, instrumentalness, liveness, valence, tempo [10]
##    trackName    artistName  danceability energy   key loudness  mode speechiness
##    <chr>        <chr>              <dbl>  <dbl> <int>    <dbl> <int>       <dbl>
##  1 Marigold     Periphery          0.788  0.533     4    -8.81     1      0.0573
##  2 Doomsday     Architects         0.856  0.841     6    -7.74     0      0.335
##  3 Reptile      Periphery          0.277  0.91      7    -5.72     1      0.117
##  4 Hereafter    Architects         0.513  0.973     6    -4.12     0      0.0849
##  5 Blood Eagle  Periphery          0.492  0.982     1    -5.25     1      0.147
##  6 Ludens       BringMeThe~        0.295  0.679     1    -6.18     1      0.15
##  7 Satellites   Periphery          0.669  0.9      10    -3.53     0      0.178
##  8 Stranded     Gojira             0.513  0.88      7    -4.69     0      0.0309
##  9 Take Me Out  FranzFerdi~        0.277  0.663     4    -8.82     0      0.0377
## 10 The Hand Th~ NineInchNa~        0.587  0.99      0    -4.50     1      0.0783
## # ... with 8 more variables: acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, time_signature <int>,
## #   plays <int>, hoursPlayed <dbl>
```

```r
## User 1 all genres from streaming
urs1_all_genres <- top_x_genres(clean_data[[2]], library = FALSE)

# determine a cutoff point
plot(urs1_all_genres$instances, type = 'b', ylab = 'Genre Instances', xlab = 'Arbitrary Genre Index', ma
abline(v = 20, col = 'red')
```



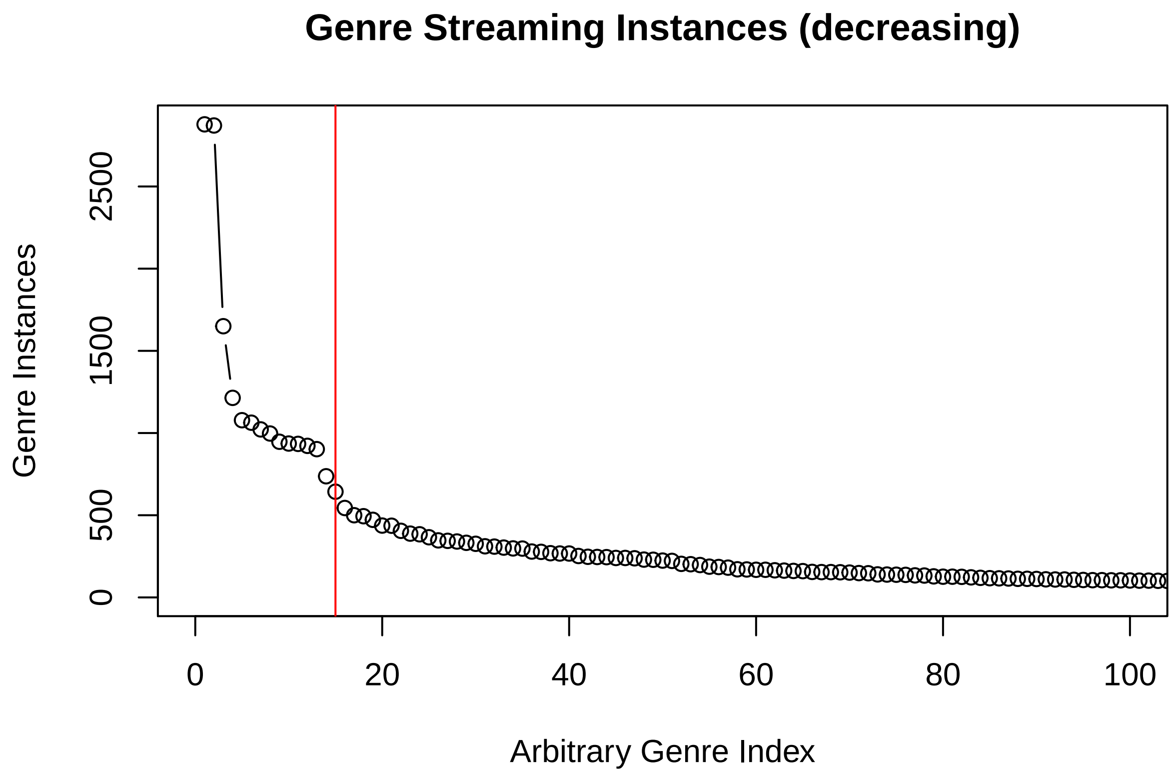

```r
# view top genres from cutoff point (20)
top_x_genres(clean_data[[2]], 20, library = FALSE)
```

```
## # A tibble: 20 x 3
##    genres                 instances `%occurence`
##    <chr>                      <int>        <dbl>
##  1 Unknown                     2962         5.36
##  2 alternative metal           2946         5.33
##  3 rock                        2650         4.79
##  4 modern rock                 2621         4.74
##  5 nu metal                    2566         4.64
##  6 metalcore                   2525         4.57
##  7 melodic metalcore           2229         4.03
##  8 pop punk                    1667         3.02
##  9 uk metalcore                1412         2.55
## 10 screamo                     1300         2.35
## 11 trancecore                  1045         1.89
## 12 punk                         971         1.76
## 13 alternative rock             963         1.74
## 14 post-grunge                  943         1.71
## 15 rap metal                    915         1.66
## 16 post-screamo                 898         1.62
## 17 permanent wave               771         1.39
## 18 progressive metalcore        734         1.33
## 19 rap rock                     727         1.32
## 20 skate punk                   580         1.05
```

```r
## User 2 top 20 songs
top_x_songs(clean_data[[4]], 10)
```

```
## `summarise()` has grouped output by 'trackName', 'artistName', 'danceability', 'energy', 'key', 'lou

## # A tibble: 10 x 16
## # Groups:   trackName, artistName, danceability, energy, key, loudness, mode,
## #   speechiness, acousticness, instrumentalness, liveness, valence, tempo [10]
##    trackName     artistName danceability energy   key loudness  mode speechiness
##    <chr>         <chr>             <dbl>  <dbl> <int>    <dbl> <int>       <dbl>
##  1 Summer 99     Tchami            0.81   0.968     2    -3.98     1      0.103
##  2 Red Light Gr~ DukeDumont        0.811  0.825     2    -5.48     1      0.0654
##  3 Deceiver      ChrisLake         0.881  0.883    11    -5.49     1      0.0633
##  4 Brothers In ~ DireStrai~        0.22   0.44      8    -9.46     0      0.0352
##  5 Conjure Drea~ MaceoPlex         0.687  0.744     2    -8.09     1      0.0545
##  6 Lies, Decept~ ChrisLake         0.809  0.935     1    -4.81     1      0.06
##  7 IM GONE       JOYRYDE           0.253  0.676    10    -6.13     1      0.0455
##  8 Walk          Pantera           0.871  0.701     5    -5.59     0      0.0458
##  9 San Frandisc~ DomDolla          0.762  0.826     1    -4.87     1      0.0406
## 10 Never Let Yo~ SNBRN             0.728  0.94      4    -5.03     1      0.0368
## # ... with 8 more variables: acousticness <dbl>, instrumentalness <dbl>,
## #   liveness <dbl>, valence <dbl>, tempo <dbl>, time_signature <int>,
## #   plays <int>, hoursPlayed <dbl>
```

```r
## User 2 all genres from streaming
urs2_all_genres <- top_x_genres(clean_data[[4]], library = FALSE)

# determine a cutoff point
plot(urs2_all_genres$instances, type = 'b', ylab = 'Genre Instances', xlab = 'Arbitrary Genre Index', ma
abline(v = 15, col = 'red')
```




```r
# view top genres from cutoff point (15)
top_x_genres(clean_data[[4]], 15, library = FALSE)
```

```
## # A tibble: 15 x 3
##    genres              instances `%occurence`
##    <chr>                   <int>        <dbl>
##  1 electro house            2878         5.51
##  2 edm                      2871         5.49
##  3 pop dance                1650         3.16
##  4 rap                      1214         2.32
##  5 hip hop                  1078         2.06
##  6 Unknown                  1063         2.03
##  7 pop                      1022         1.96
##  8 house                     997         1.91
##  9 modern rock               947         1.81
## 10 tropical house            936         1.79
## 11 electronic trap           934         1.79
## 12 bass house                922         1.76
## 13 rock                      902         1.73
## 14 progressive house         737         1.41
## 15 dance pop                 643         1.23
```

Important Notes - Since we know our users top genres, we have a probabilistic method to predict what genre of music they will listen to in the future although this method of modeling takes on a bad assumption. - This way of modeling user music preference assumes that a user's taste in music is static, when in reality it can and does changes over time, whether it be over years or over the span of a week. - As such we need to use genre frequencies over time to predict what music they may like to listen to. - We may also use the occurrence of particular sound/audio features to help predict what genres may be best suited for a certain time.

Visualizing User Play Time

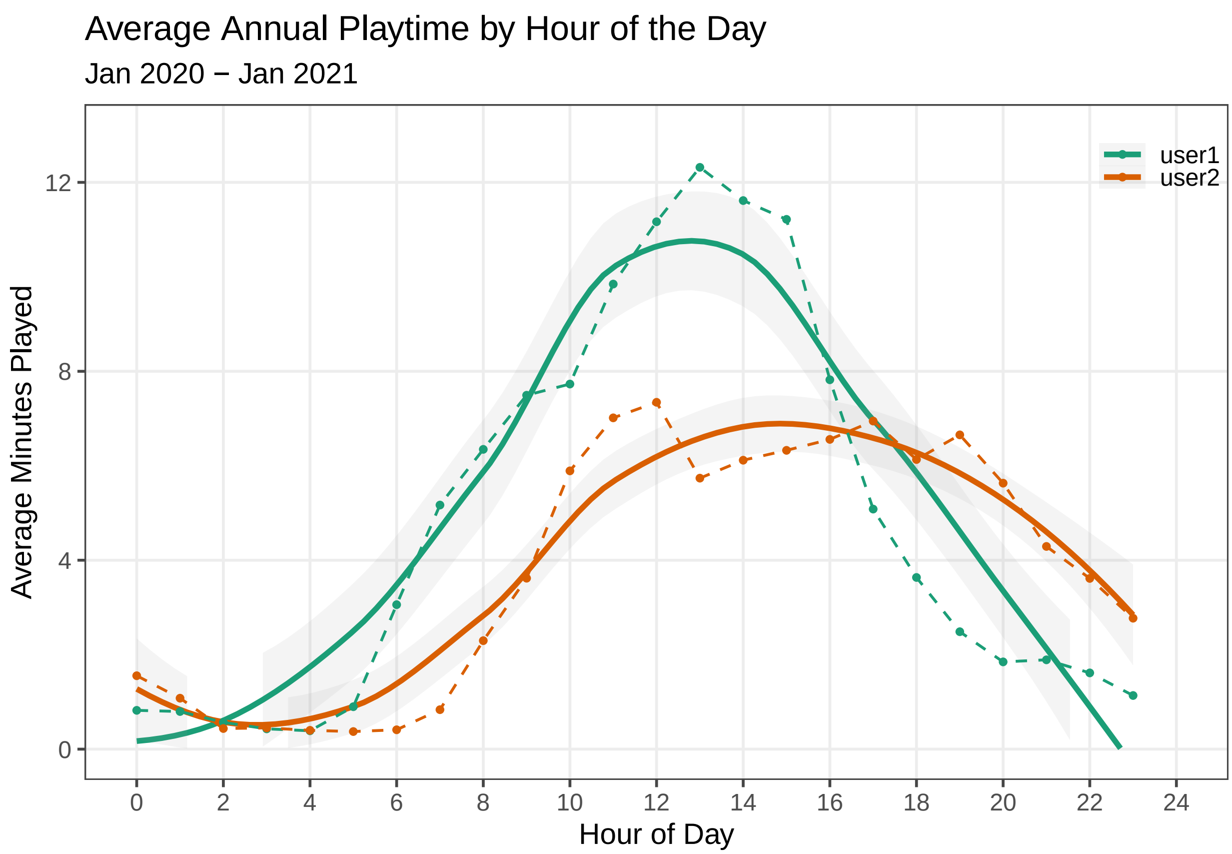
```
## More data processing

## Function for binding user streaming data
stream_together <- function(frames) {
  n <- length(frames)
  for (i in 1:n) {
    frames[[i]] <- frames[[i]] %>%
      group_by(endTime, trackName, artistName, msPlayed, # main vars
               danceability, energy, key, loudness, mode,
               speechiness, acousticness, instrumentalness,
               liveness, valence, tempo, time_signature) %>%
      summarise()

    frames[[i]]$minPlayed <- frames[[i]]$msPlayed/(1000*60)
    frames[[i]] <- frames[[i]][-4]
    frames[[i]]$user <- as.factor(glue('user{i}'))
  }
  frames <- rbindlist(frames)
  frames
}


all_streams_UTC <- stream_together(clean_data[c(2,4)]) # Universal Standard Time


# Manual adjustment for user timezones
# This can only be done if one knows where the user has been when
all_streams <- rbind(
  # User 2 Ontario
  all_streams_UTC %>% filter(user == "user2" & month(endTime) < 9) %>%
    mutate(endHour = (hour(endTime)-5) %% 24,
           endDate = date(endTime),
           weekDay = wday(endTime, label = FALSE, abbr=TRUE),
           week_of_year = week(date(endTime))),
  # User 2 British Columbia
  all_streams_UTC %>% filter(user == "user2" & month(endTime) >= 9) %>%
    mutate(endHour = (hour(endTime)-8) %% 24,
           endDate = date(endTime),
           weekDay = wday(endTime, label = FALSE, abbr=TRUE),
           week_of_year = week(date(endTime))),
  # User 1 Ontario
  all_streams_UTC %>% filter(user == "user1" & month(endTime) < 5) %>%
    mutate(endHour = (hour(endTime)-5) %% 24,
           endDate = date(endTime),
           weekDay = wday(endTime, label = FALSE, abbr=TRUE),
           week_of_year = week(date(endTime))),
  # User 1 British Columbia
  all_streams_UTC %>% filter(user == "user1" & month(endTime) >= 5) %>%
    mutate(endHour = (hour(endTime)-8) %% 24,
```

```r
            endDate = date(endTime),
            weekDay = wday(endTime, label = FALSE, abbr=TRUE),
            week_of_year = week(date(endTime)))
  )
```

Daily

```r
# Plotting average playtime per hour of the day over the last year
tot_days <- as.numeric(max(all_streams$endDate) - min(all_streams$endDate))

all_streams %>%
  group_by(endHour, endDate, user) %>%
  summarise(hourdayPlayTime = sum(minPlayed)) %>% # total minutes played by hour by day
  group_by(endHour, user) %>%
  summarise(avg = sum(hourdayPlayTime)/tot_days) %>% # avg minutes played by hour over all days
            #n = n())  # days played at given hour
            #q1 = quantile(c(hourdayPlayTime, rep(0, tot_days-n())), 0.5),  # quantiles including days w
            #q3 = quantile(c(hourdayPlayTime, rep(0, tot_days-n())), 0.8)) %>%

  ggplot(aes(y = avg, x = endHour, color = user)) +

  geom_smooth(level = 0.95, method = loess, alpha = 0.1) +
  geom_line(alpha = 1, size = 0.5, linetype  = 'dashed') +
  geom_point(alpha = 1.2, shape = 16, size=1.2) +

  scale_x_continuous(name = 'Hour of Day', breaks=seq(0,24,2), limits = c(0,24)) +
  scale_y_continuous(name = 'Average Minutes Played', breaks=seq(0, 12, 4), limits = c(0,13)) +
  ggtitle('Average Annual Playtime by Hour of the Day', subtitle = 'Jan 2020 - Jan 2021') +
  scale_color_brewer(palette = 'Dark2') +
  theme_article() +
  theme(legend.title = element_blank(),
        legend.position = c(0.94,0.92),
        panel.grid.major = element_line(color = '#ededed'))
```

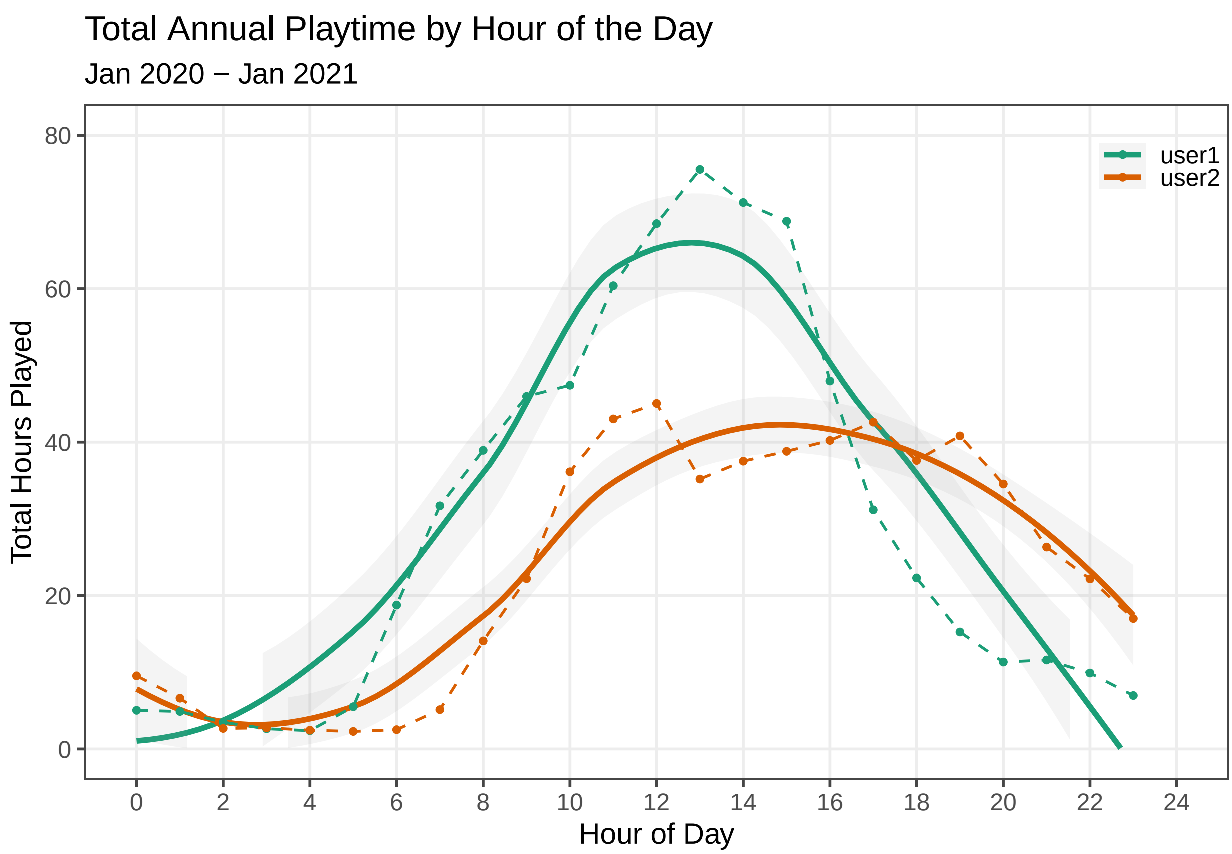
```
## `summarise()` has grouped output by 'endHour', 'endDate'. You can override using the `.groups` argume

## `summarise()` has grouped output by 'endHour'. You can override using the `.groups` argument.

## `geom_smooth()` using formula 'y ~ x'

## Warning: Removed 1 rows containing missing values (geom_smooth).
```




```r
# Plotting total playtime per hour of the day over the last year
all_streams %>%
  group_by(endHour, user) %>%
  summarise(totalPlayTime = sum(minPlayed)/60) %>%

  ggplot(aes(y = totalPlayTime, x = endHour, color = user)) +

  geom_smooth(level = 0.95, method = loess, alpha = 0.1) +
  geom_line(alpha = 1, size = 0.5, linetype  = 'dashed') +
  geom_point(alpha = 1.2, shape = 16, size=1.2) +

  scale_x_continuous(name = 'Hour of Day', breaks=seq(0,24,2), limits = c(0,24)) +
  scale_y_continuous(name = 'Total Hours Played', breaks=seq(0, 80, 20), limits = c(0,80)) +

  ggtitle('Total Annual Playtime by Hour of the Day',
          subtitle = 'Jan 2020 - Jan 2021') +
  scale_color_brewer(palette = 'Dark2') +
  theme_article() +
  theme(legend.title = element_blank(),
        legend.position = c(0.94,0.92),
        panel.grid.major = element_line(color = '#ededed'))
```

```
## `summarise()` has grouped output by 'endHour'. You can override using the `.groups` argument.
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 1 rows containing missing values (geom_smooth).
```

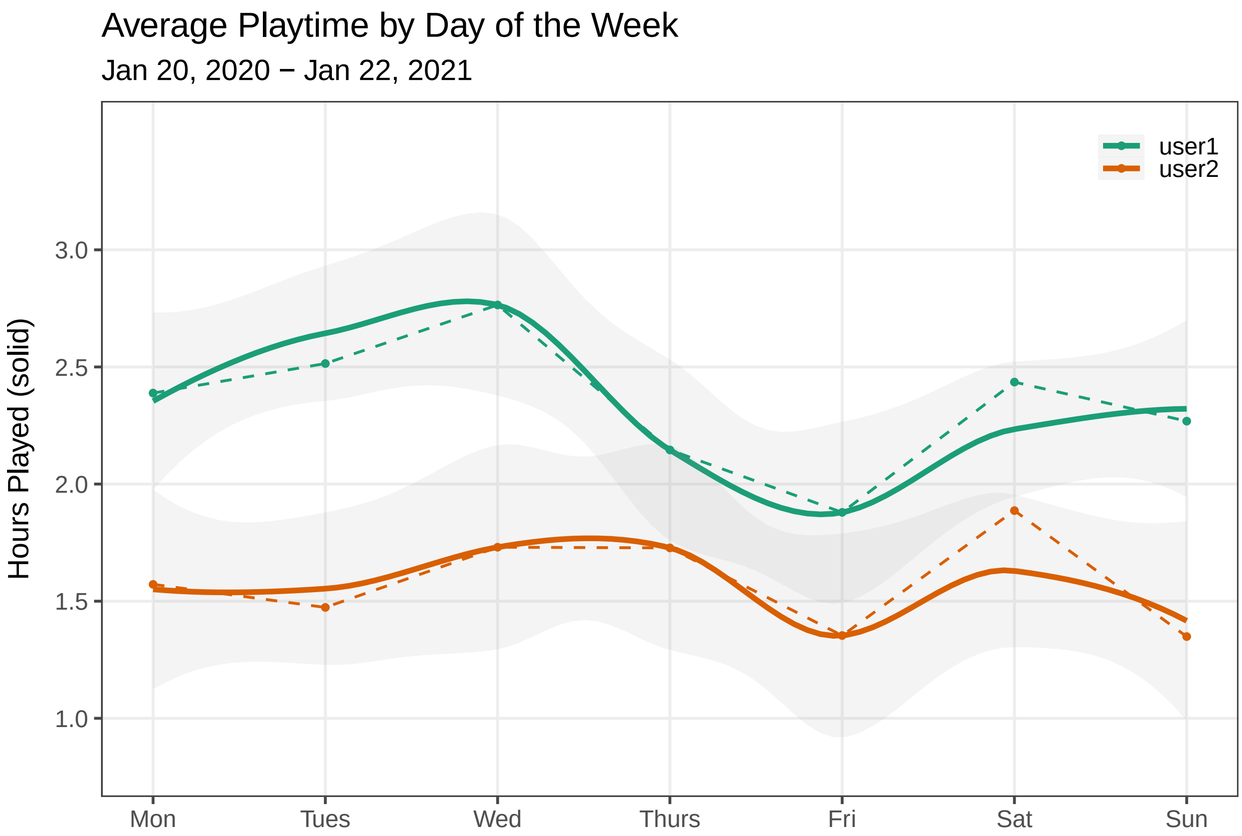


Weekly

```
week_days <- c('Mon', 'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun')

tot_weeks <- length(unique(all_streams$week_of_year))

# Plot average daily play time for each day of the week
all_streams %>%
  group_by(weekDay, endDate, user) %>%
  summarise(dailyPlayTime = sum(minPlayed)/60) %>%
  group_by(weekDay, user) %>%
  summarise(avg_dailyPlayTime = mean(dailyPlayTime)) %>%

  ggplot(aes(y = avg_dailyPlayTime, x = weekDay, color = user)) +

  geom_smooth(level = 0.5, method = loess, alpha = 0.1) + # 50% confidence interval used
  geom_line(alpha = 1, size = 0.5, linetype  = 'dashed') +
  geom_point(alpha = 1.2, shape = 16, size=1.2) +

  scale_x_continuous(name = element_blank(), labels = week_days, breaks = seq(1, 7, 1)) +
  scale_y_continuous(name = 'Hours Played (solid)', limits = c(0.8,3.5), breaks = seq(0,3,0.5)) +

  ggtitle('Average Playtime by Day of the Week',
          subtitle = 'Jan 20, 2020 - Jan 22, 2021') +
  scale_color_brewer(palette = 'Dark2') +
```

```
  theme_article() +
  theme(legend.title = element_blank(),
        legend.position = c(0.93,0.93),
        axis.title.y.left = element_text(vjust = 3),
        axis.title.y.right = element_text(vjust = 3, angle = -90),
        panel.grid.major = element_line(color = '#ededed'))
```

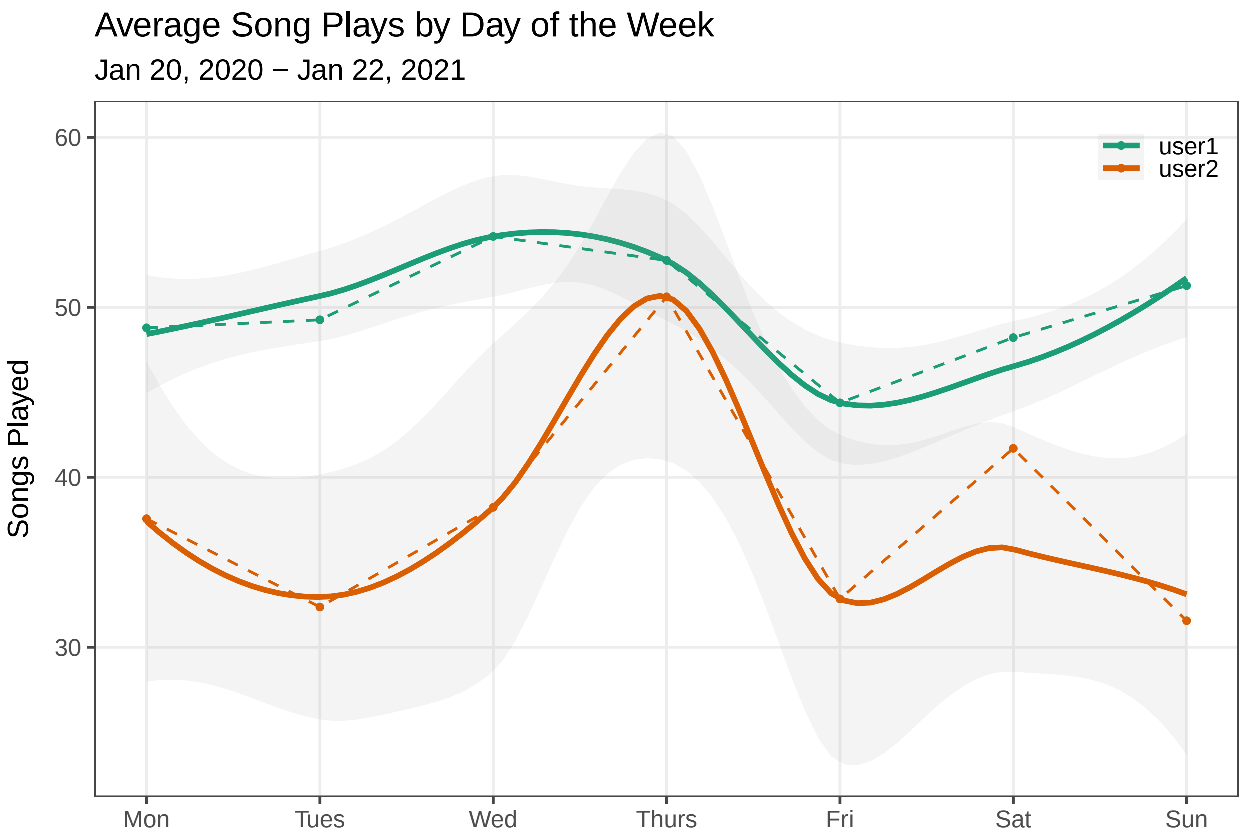
```
## `summarise()` has grouped output by 'weekDay', 'endDate'. You can override using the `.groups` argume

## `summarise()` has grouped output by 'weekDay'. You can override using the `.groups` argument.

## `geom_smooth()` using formula 'y ~ x'
```




```
# Plot daily play counts for each day of the week
all_streams %>%
  group_by(weekDay, endDate, user) %>%
  summarise(dailyPlays = n()) %>%
  group_by(weekDay, user) %>%
  summarise(avg_dailyPlays = mean(dailyPlays)) %>%

  ggplot(aes(y = avg_dailyPlays, x = weekDay, color = user)) +

  geom_smooth(level = 0.5, method = loess, alpha = 0.1) + # 50% confidence interval used
  geom_line(alpha = 1, size = 0.5, linetype  = 'dashed') +
  geom_point(alpha = 1.2, shape = 16, size=1.2) +

  scale_x_continuous(name = element_blank(), labels = week_days, breaks = seq(1, 7, 1)) +
```

```r
  scale_y_continuous(name = 'Songs Played') +

  ggtitle('Average Song Plays by Day of the Week',
          subtitle = 'Jan 20, 2020 - Jan 22, 2021') +
  scale_color_brewer(palette = 'Dark2') +
  theme_article() +
  theme(legend.title = element_blank(),
        legend.position = c(0.93,0.93),
        axis.title.y.left = element_text(vjust = 3),
        axis.title.y.right = element_text(vjust = 3, angle = -90),
        panel.grid.major = element_line(color = '#ededed'))
```

```
## `summarise()` has grouped output by 'weekDay', 'endDate'. You can override using the `.groups` argume
## `summarise()` has grouped output by 'weekDay'. You can override using the `.groups` argument.
## `geom_smooth()` using formula 'y ~ x'
```

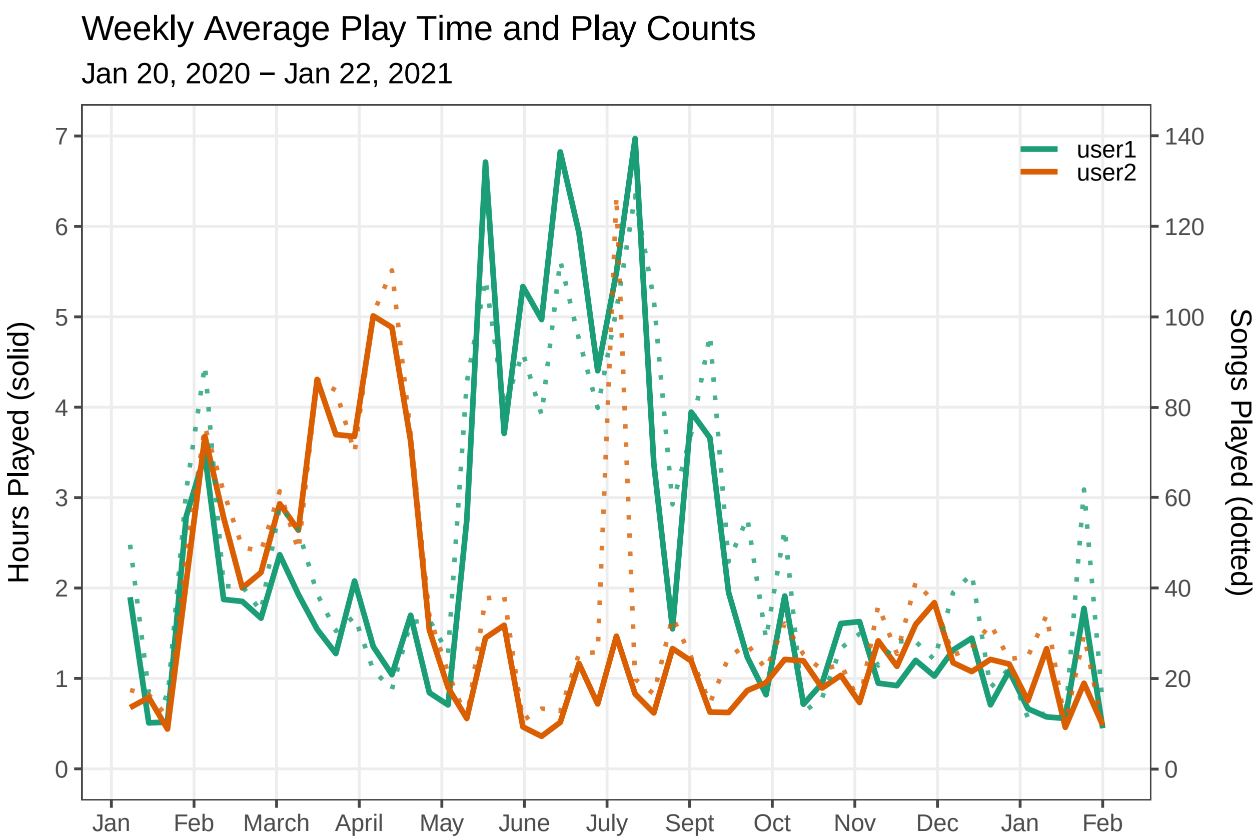


Monthly

```r
months <- c('Jan', 'Feb', 'March', 'April', 'May', 'June', 'July', 'Sept',
            'Oct', 'Nov', 'Dec', 'Jan', 'Feb')

tot_weeks <- length(unique(all_streams$week_of_year))

# Plot weekly average play time
all_streams %>%
```

```r
  group_by(week_of_year, endDate, user) %>%
  summarise(dailyPlayTime = sum(minPlayed)/60,
            dailyPlays = n()) %>%
  group_by(week_of_year, user) %>%
  summarise(avg_weeklyPlayTime = mean(dailyPlayTime),
            avg_weeklyPlays = mean(dailyPlays)) %>%

  ggplot(aes(x = week_of_year, color = user)) +

  geom_ma(aes(y=avg_weeklyPlayTime), n = 1, linetype = 1, size = 1, alpha = 1.0) +
  geom_ma(aes(y=avg_weeklyPlays/20), n = 1, linetype = 3, size = 0.8, alpha = 0.8) +

  scale_x_continuous(name = element_blank(), labels = months, breaks = seq(0, 57, 4.4167)) +
  scale_y_continuous(name = 'Hours Played (solid)', limits = c(0,7), breaks = seq(0,7,1),
                     sec.axis = sec_axis(name = 'Songs Played (dotted)', ~.*20, breaks = seq(0,140,20),

  ggtitle('Weekly Average Play Time and Play Counts',
          subtitle = 'Jan 20, 2020 - Jan 22, 2021') +
  scale_color_brewer(palette = 'Dark2') +
  theme_article() +
  theme(legend.title = element_blank(),
        legend.position = c(0.93,0.93),
        axis.title.y.left = element_text(vjust = 3),
        axis.title.y.right = element_text(vjust = 3, angle = -90),
        panel.grid.major = element_line(color = '#ededed'))
```

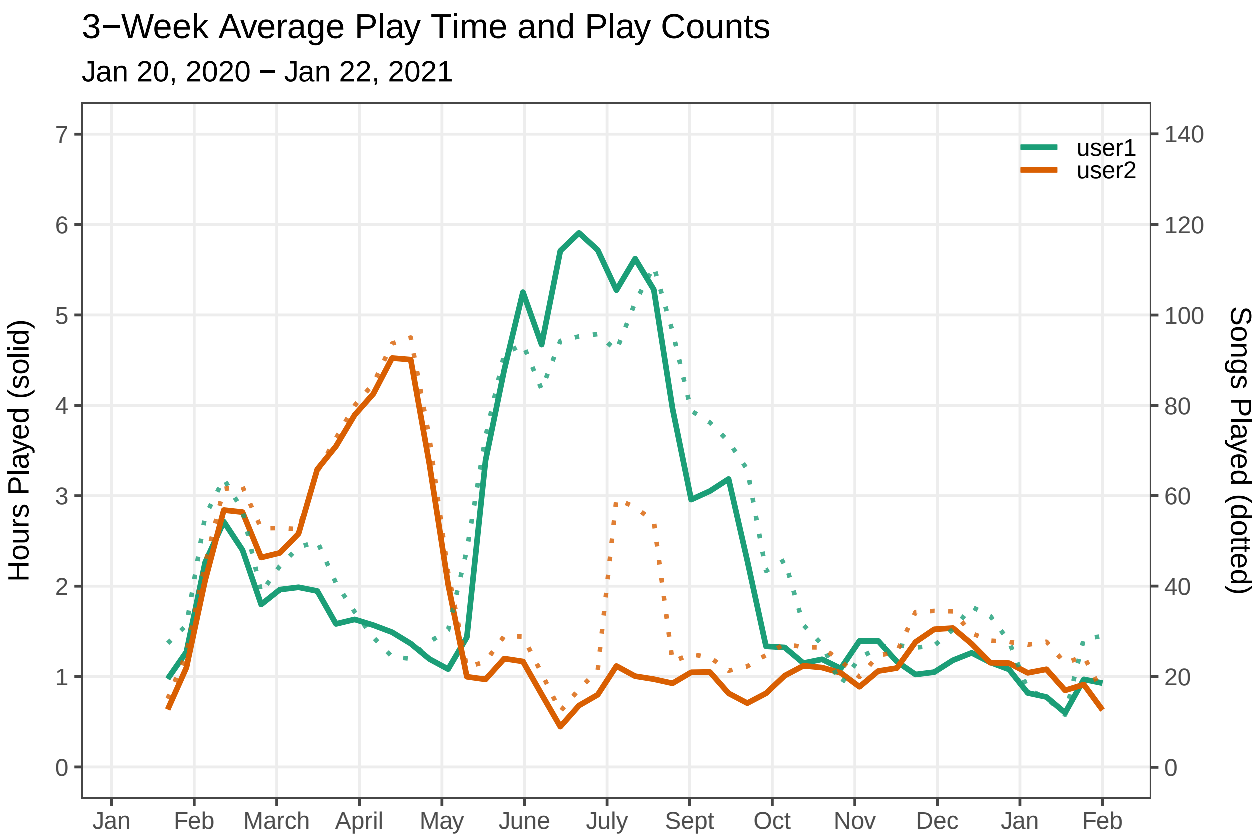
```
## `summarise()` has grouped output by 'week_of_year', 'endDate'. You can override using the `.groups` a

## `summarise()` has grouped output by 'week_of_year'. You can override using the `.groups` argument.
```




```r
# Plot 3 week average play time
all_streams %>%
  group_by(week_of_year, endDate, user) %>%
  summarise(dailyPlayTime = sum(minPlayed)/60,
            dailyPlays = n()) %>%
  group_by(week_of_year, user) %>%
  summarise(avg_weeklyPlayTime = mean(dailyPlayTime),
            avg_weeklyPlays = mean(dailyPlays)) %>%

  ggplot(aes(x = week_of_year, color = user)) +

  #geom_ma(es(y=avg_weeklyPlayTime), n = 1, linetype = 1, size = 1, alpha = 1.0) +
  #geom_ma(aes(y=avg_weeklyPlayTime), n = 2, linetype = 1, size = 1, alpha = 1.0) +
  geom_ma(aes(y=avg_weeklyPlayTime), n = 3, linetype = 1, size = 1, alpha = 1.0) +
  geom_ma(aes(y=avg_weeklyPlays/20), n = 3, linetype = 3, size = 0.8, alpha = 0.8) +

  scale_x_continuous(name = element_blank(), labels = months, breaks = seq(0, 57, 4.4167)) +
  scale_y_continuous(name = 'Hours Played (solid)', limits = c(0,7), breaks = seq(0,7,1),
                     sec.axis = sec_axis(name = 'Songs Played (dotted)', ~.*20, breaks = seq(0,140,20),

  ggtitle('3-Week Average Play Time and Play Counts',
          subtitle = 'Jan 20, 2020 - Jan 22, 2021') +
  scale_color_brewer(palette = 'Dark2') +
  theme_article() +
  theme(legend.title = element_blank(),
        legend.position = c(0.93,0.93),
```

```
        axis.title.y.left = element_text(vjust = 3),
        axis.title.y.right = element_text(vjust = 3, angle = -90),
        panel.grid.major = element_line(color = '#ededed'))
```

```
## `summarise()` has grouped output by 'week_of_year', 'endDate'. You can override using the `.groups` a
## `summarise()` has grouped output by 'week_of_year'. You can override using the `.groups` argument.
```

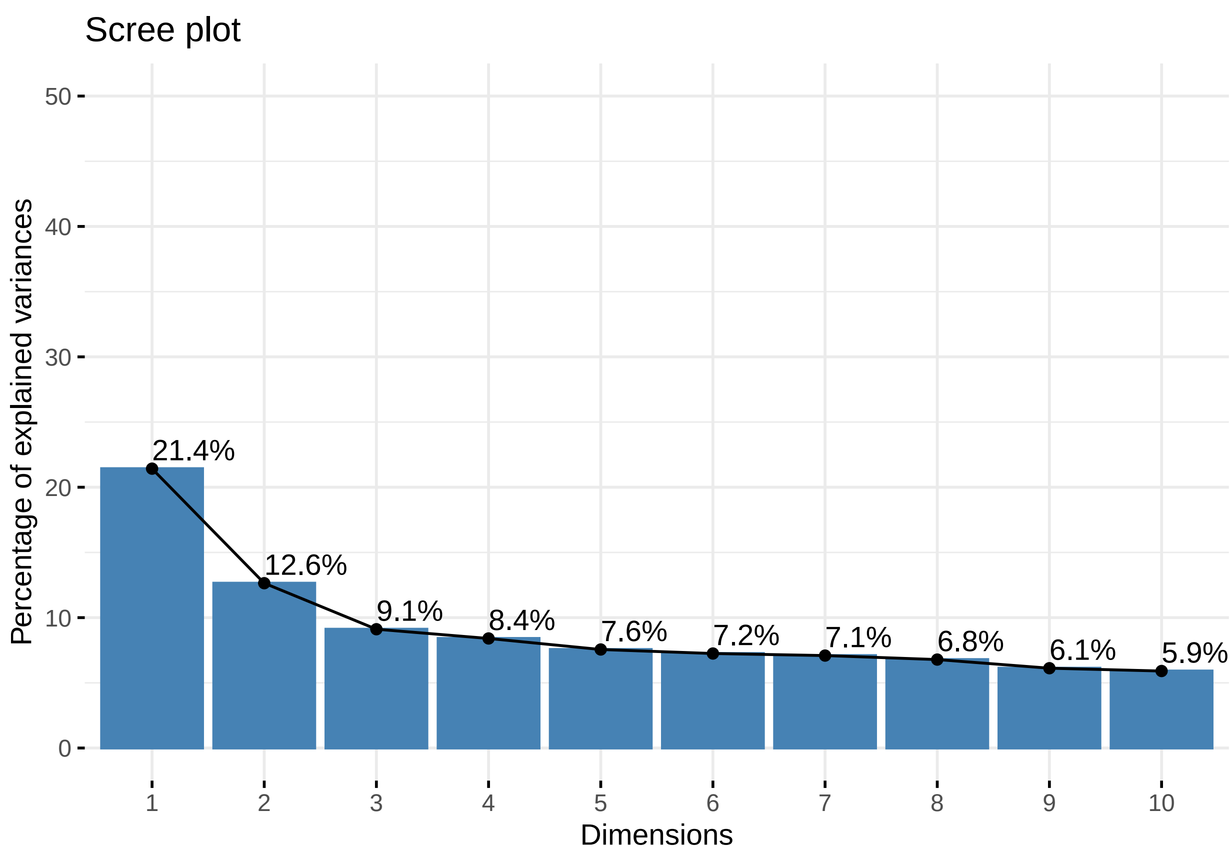


Unsupervised Exploration to find if Emotional Groupings in Libraries

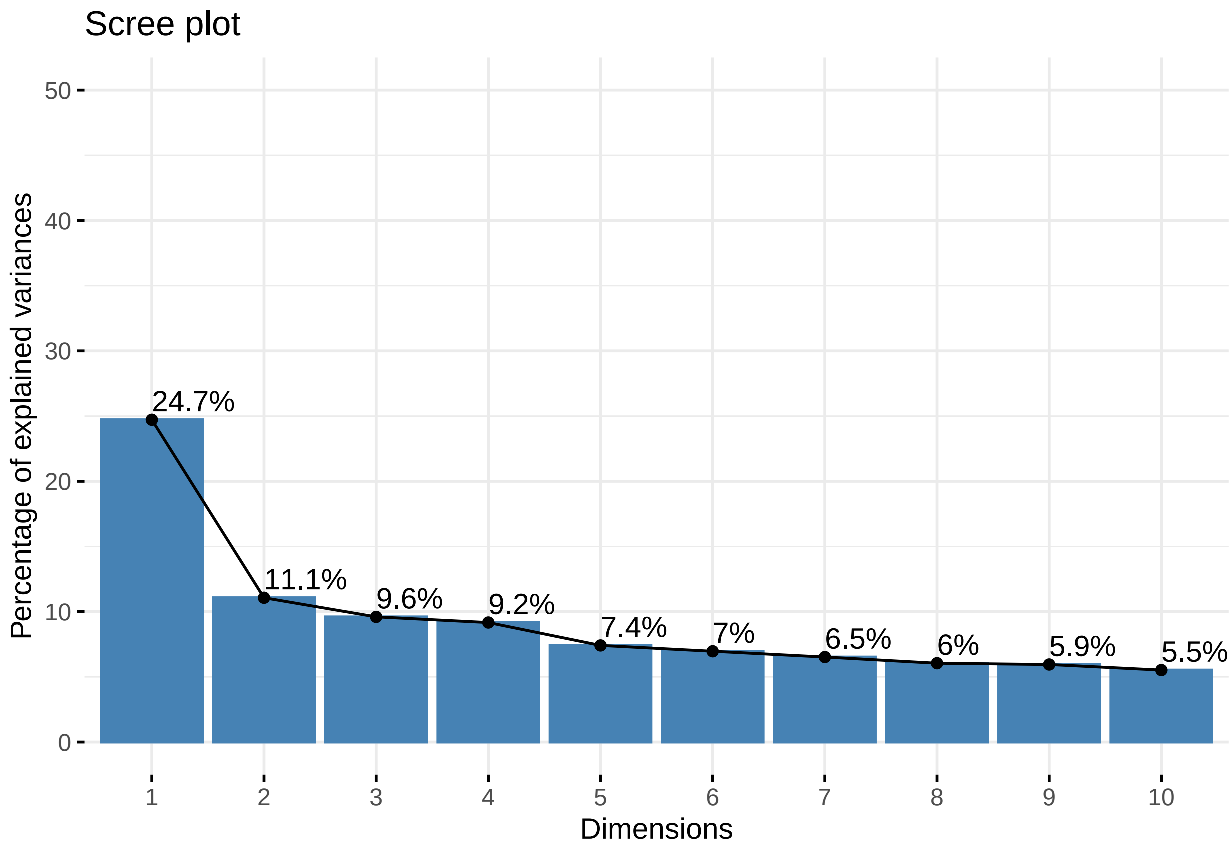
- Genres do not form clear clusters
- PC1 is related to energy/positivity/loudness in one direction and calmness in the other.
- PC2 is related to duration/energy in one direction and mode in the other.
- Ultimately, we won't be able to cluster music tastes into specific subgroups for either user.
- Perhaps clusters may appear when comparing user data but it's not clear if they exist within user data.
- Therefore, further clustering methods won't be performed.

```
# Function to scale and remove non-numeric rows
convenient_scale <- function(df) {
  # Save genre classes
  genres <- as.factor(df$genres)
  # Scale numeric columns
  df <- df %>% mutate_if(is.numeric, scale) %>% select_if(is.numeric)
  # Readd genre classes
  df$genres <- cbind(genres, df)
}
```

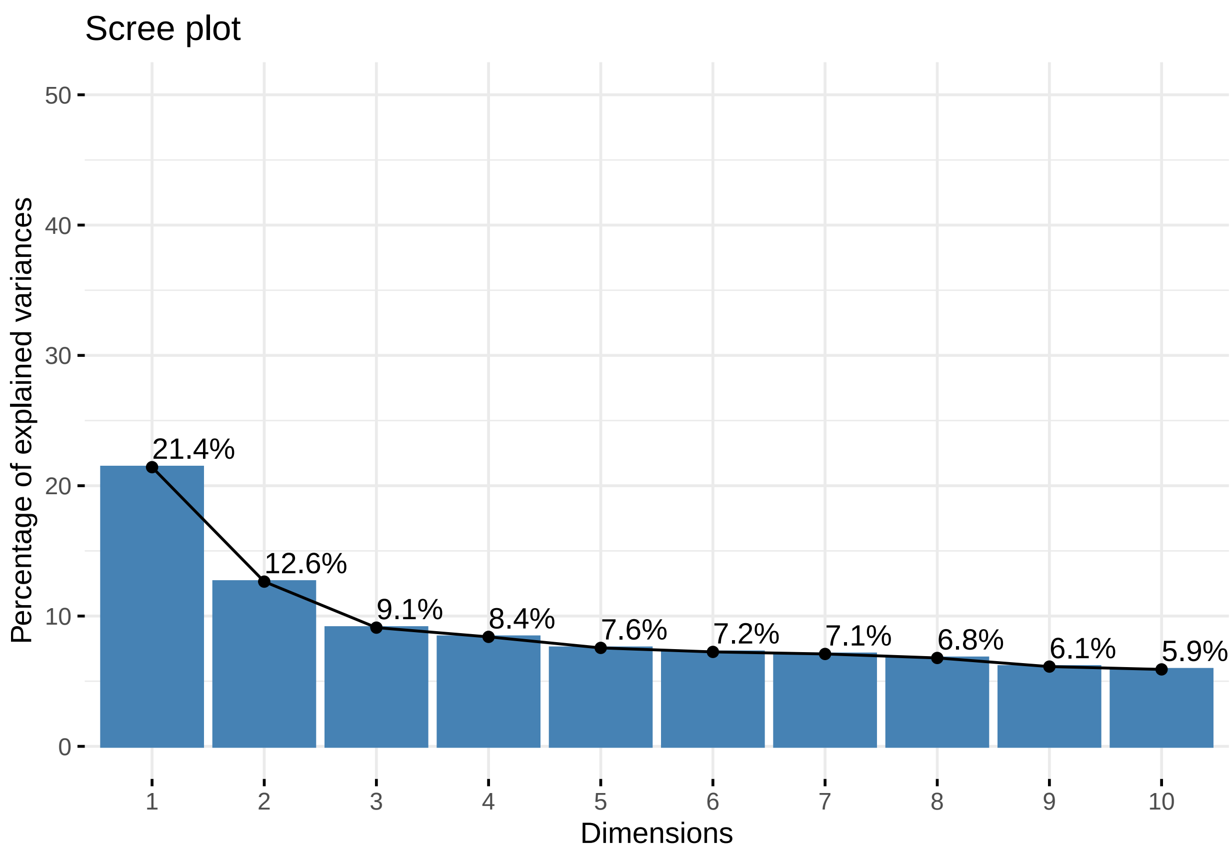
```r
par(mfrow = c(2,2))

# Scale data
usr1_lib_scaled <- convenient_scale(clean_data[[1]])
usr2_lib_scaled <- convenient_scale(clean_data[[3]])
usr1_streams_scaled <- convenient_scale(clean_data[[2]])[, -2]
usr2_streams_scaled <- convenient_scale(clean_data[[4]])[, -2]

# PCA
usr1_libpca <- PCA(usr1_lib_scaled[,-1], graph = FALSE)
usr2_libpca <- PCA(usr2_lib_scaled[,-1], graph = FALSE)
usr1_strmpca <- PCA(usr1_lib_scaled[,-1], graph = FALSE)
usr2_strmpca <- PCA(usr2_lib_scaled[,-1], graph = FALSE)

# Skree plot
fviz_eig(usr1_libpca, addlabels = TRUE, ylim = c(0, 50))
```



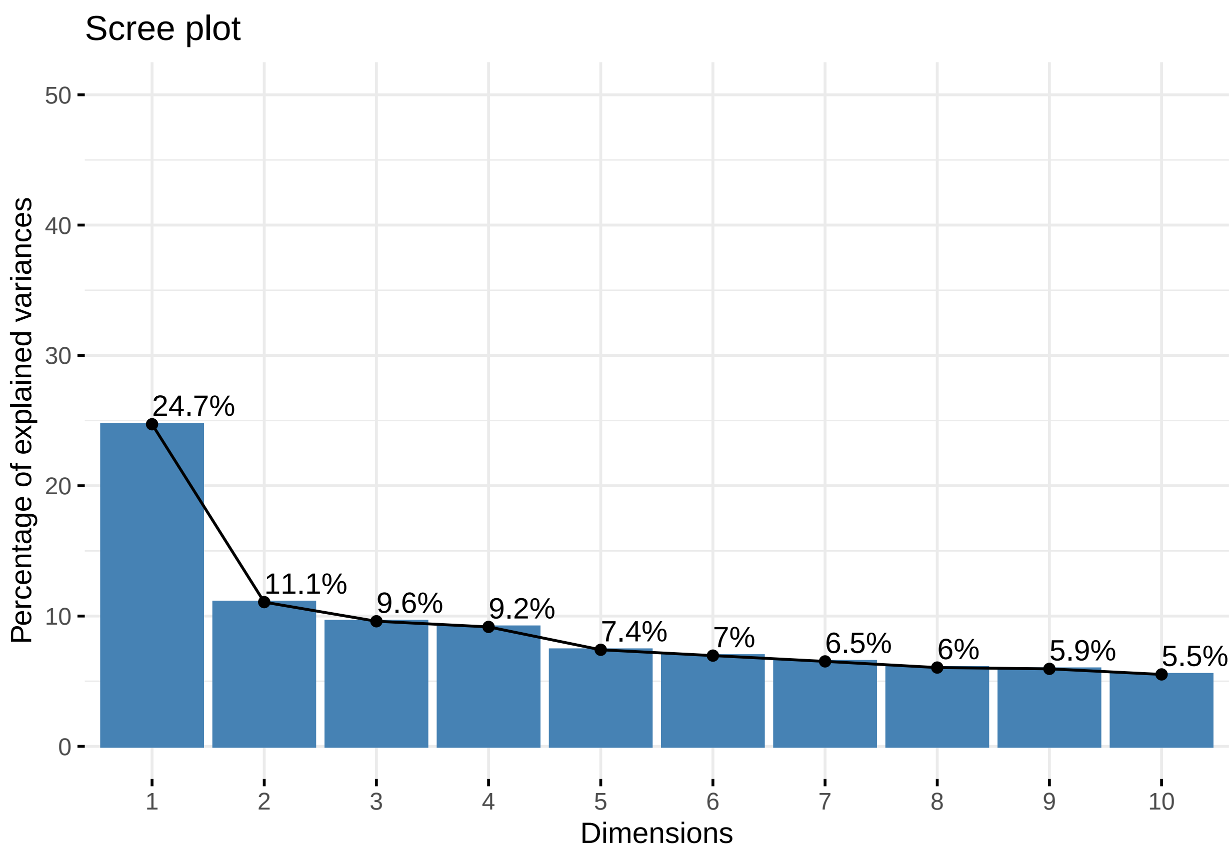

```r
fviz_eig(usr2_libpca, addlabels = TRUE, ylim = c(0, 50))
```



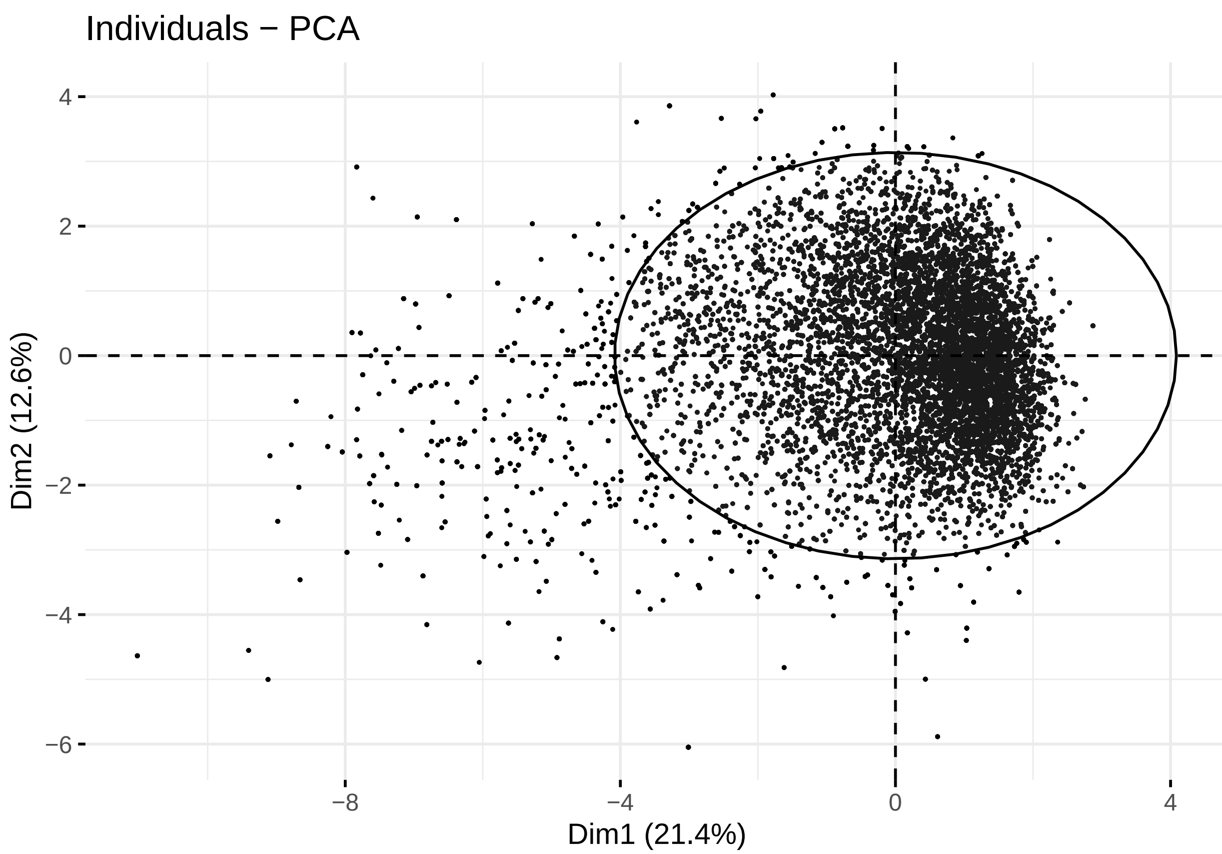

```
fviz_eig(usr1_strmpca, addlabels = TRUE, ylim = c(0, 50))
```



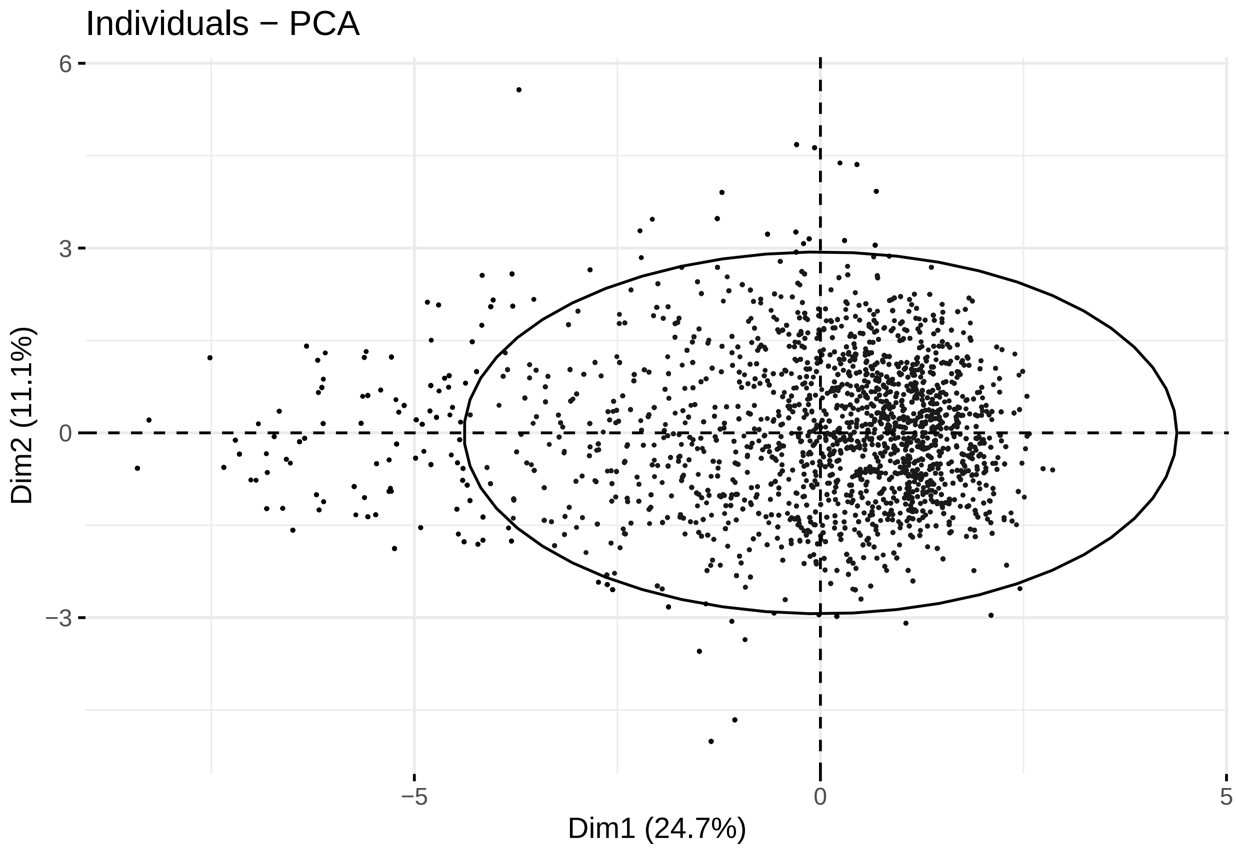

```r
fviz_eig(usr2_strmpca, addlabels = TRUE, ylim = c(0, 50))
```



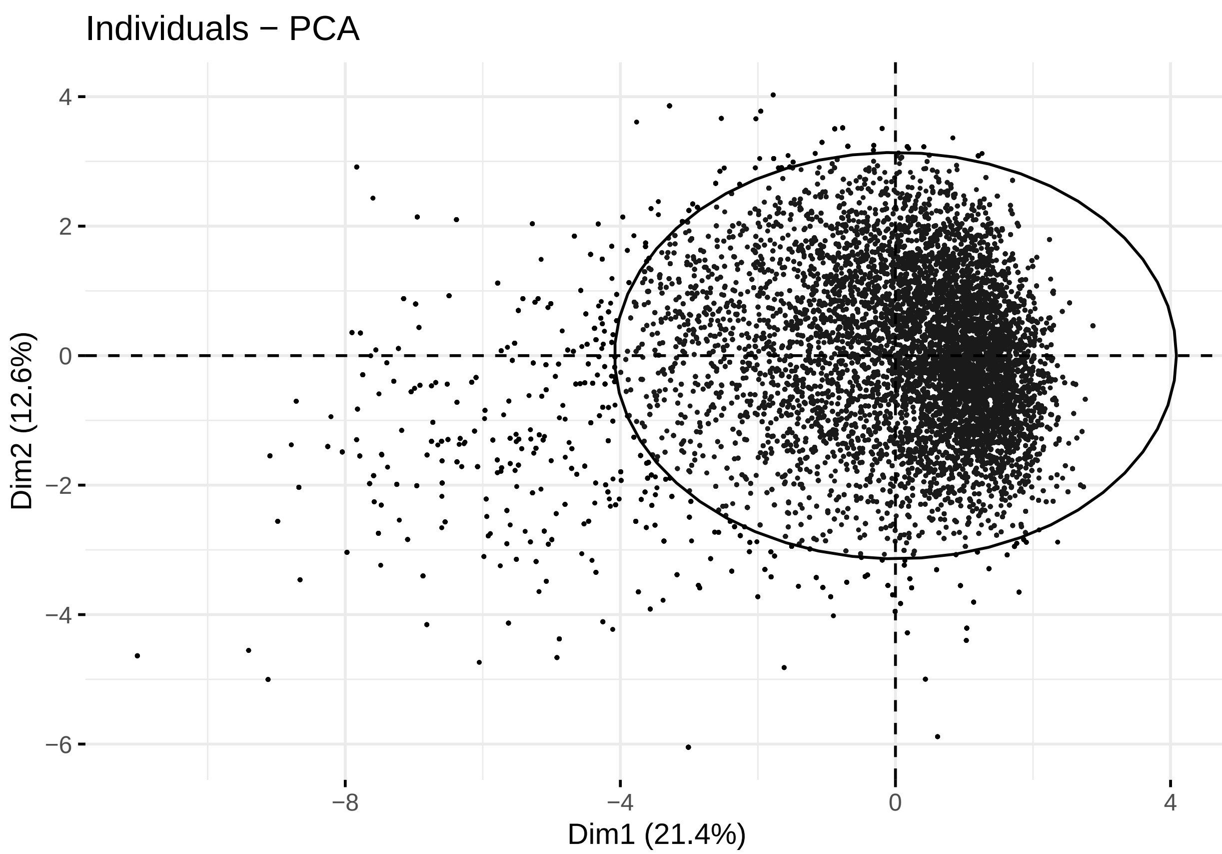

```
# PC1/PC2 Plot
fviz_pca_ind(usr1_libpca, geom.ind = "point", pointsize = 0.2, addEllipses = TRUE,)
```



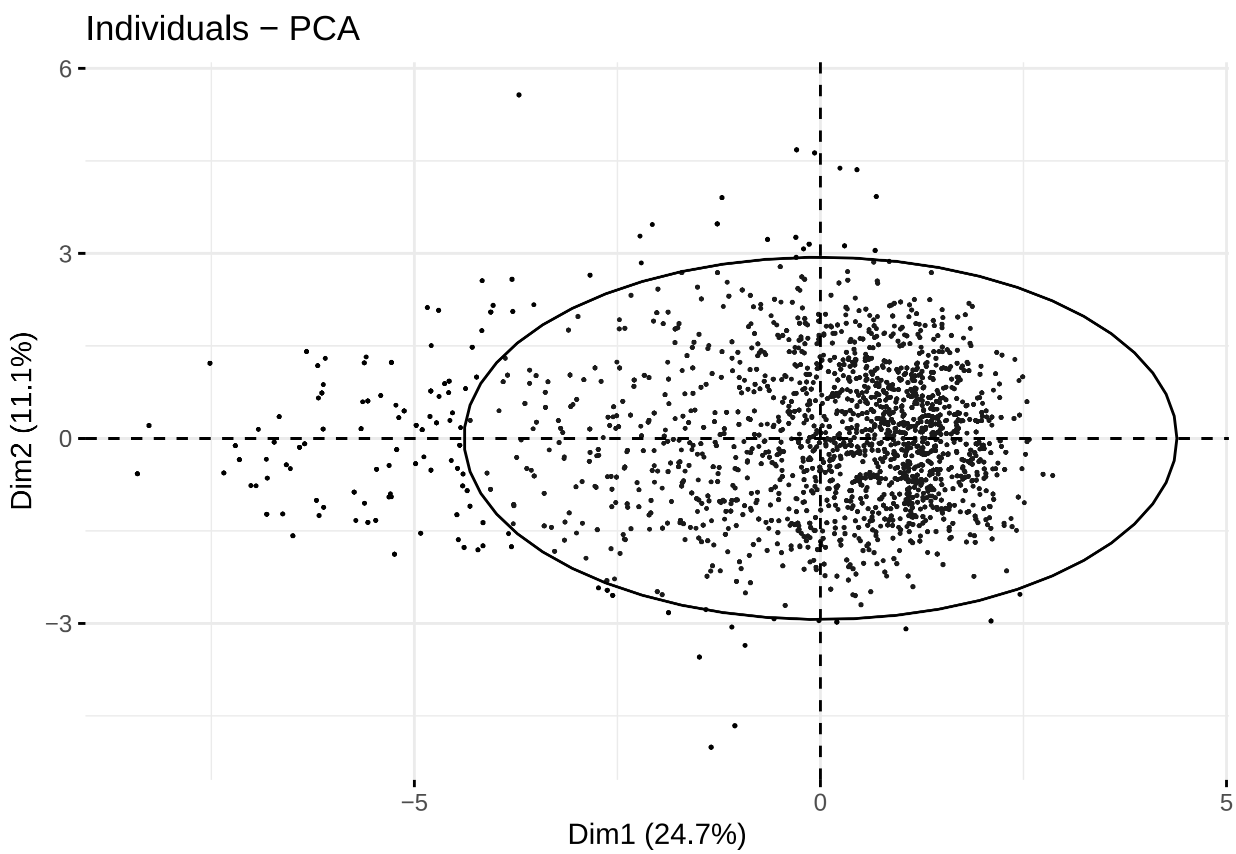

```
fviz_pca_ind(usr2_libpca, geom.ind = "point", pointsize = 0.2, addEllipses = TRUE,)
```




```r
fviz_pca_ind(usr1_strmpca, geom.ind = "point", pointsize = 0.2, addEllipses = TRUE,)
```

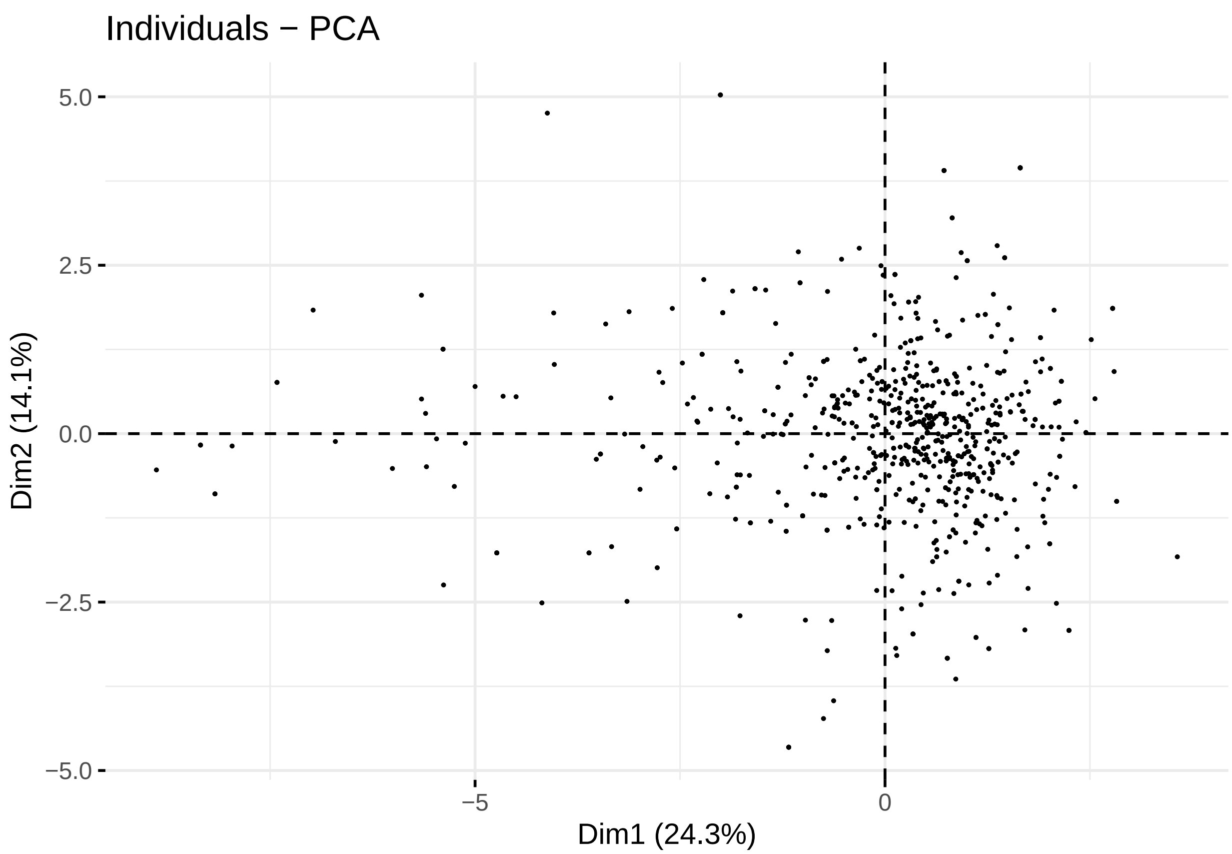



```
fviz_pca_ind(usr2_strmpca, geom.ind = "point", pointsize = 0.2, addEllipses = TRUE,)
```

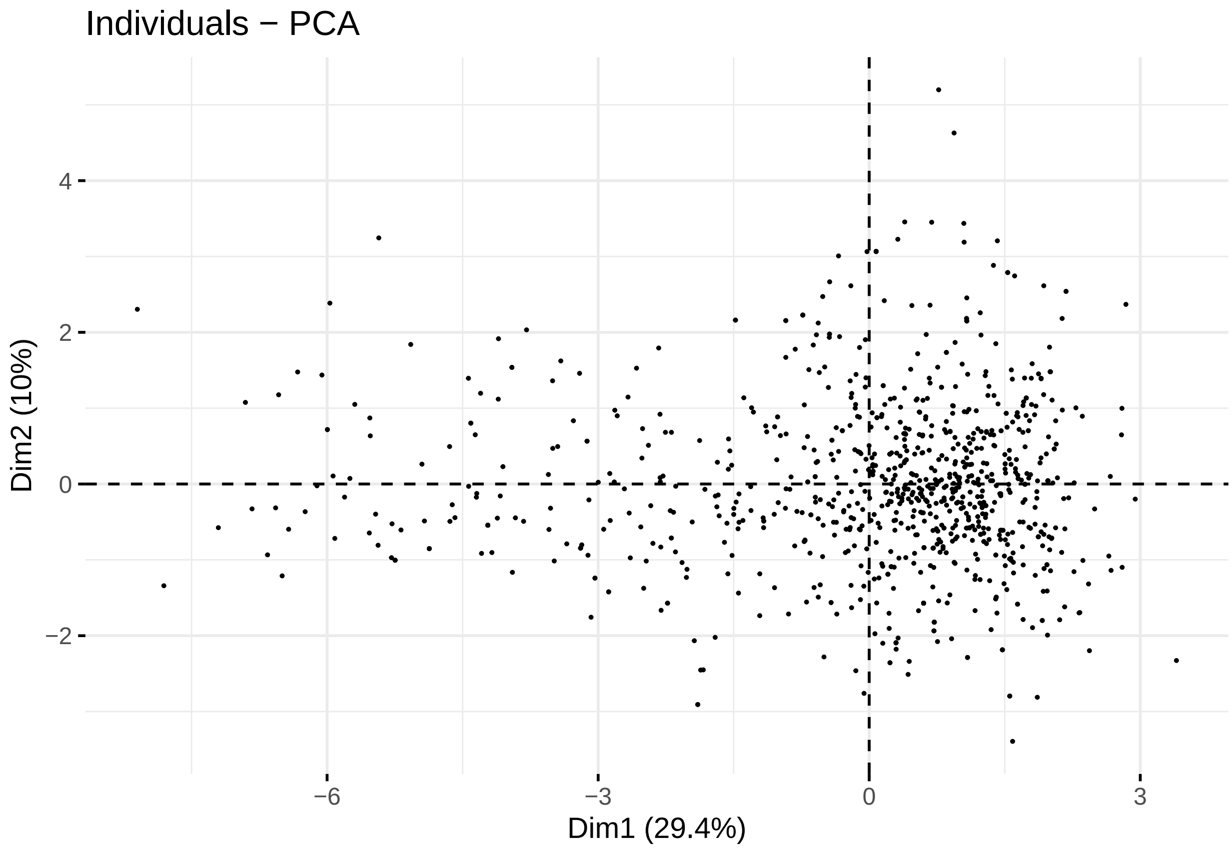


Since there are no obvious clusters across the first 2 PCs, let's see if genres themselves cluster into groups based on audio features.

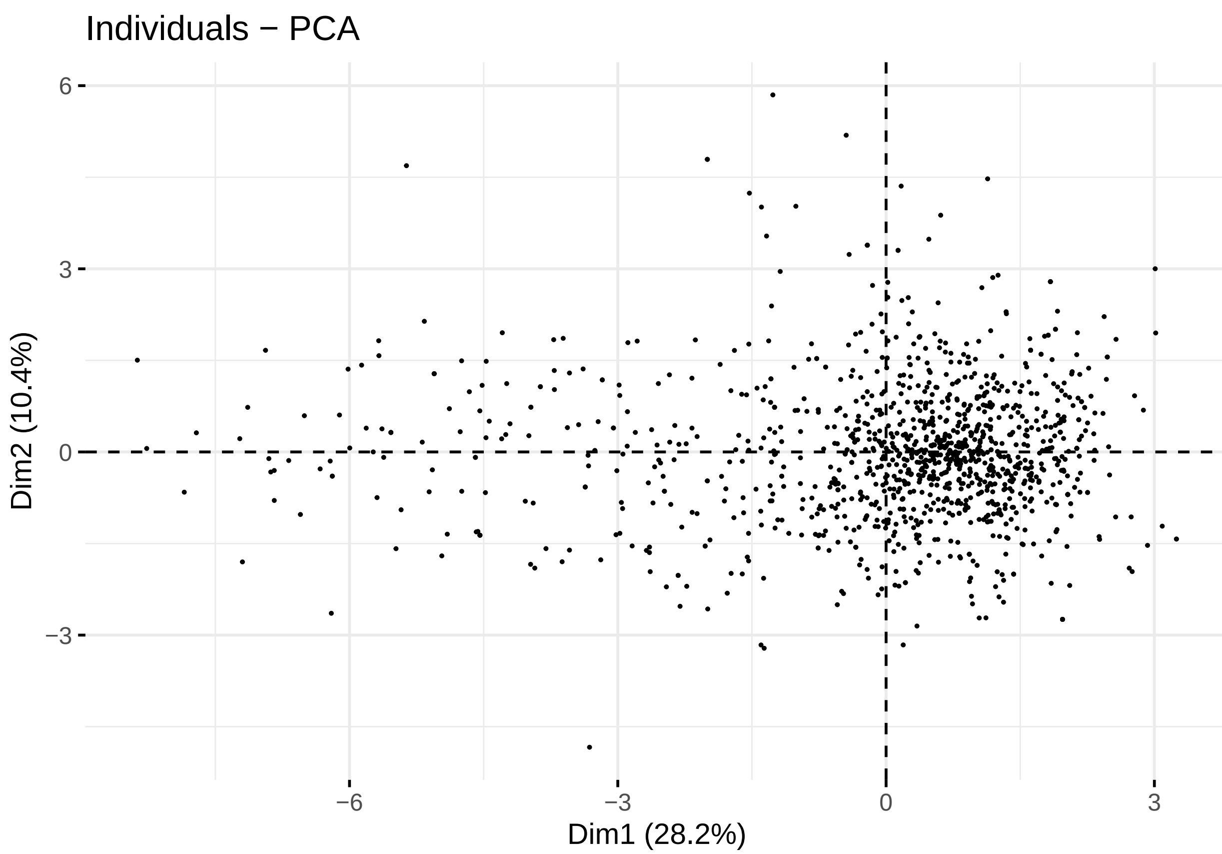
```
# Check if average genre features alone cluster based on audio features
# (summarize average audio features by genre)
gens1 <- usr1_streams_scaled %>%
  group_by(genres) %>%
  summarise_all(mean)

gens2 <- usr2_streams_scaled %>%
  group_by(genres) %>%
  summarise_all(mean)

combined_streams <- rbind(usr1_streams_scaled, usr2_streams_scaled)
gens3 <- combined_streams %>%
  group_by(genres) %>%
  summarise_all(mean)

# PCA on average genre features
gens1_pca <- PCA(gens1[,-1], ncp = 10, graph = FALSE)
gens2_pca <- PCA(gens2[,-1], ncp = 10, graph = FALSE)
gens3_pca <- PCA(gens3[,-1], ncp = 10, graph = FALSE)


# PC1/PC2 Plots
fviz_pca_ind(gens1_pca, geom.ind = "point", pointsize = 0.2)
```




```r
fviz_pca_ind(gens2_pca, geom.ind = "point", pointsize = 0.2)
```




```r
fviz_pca_ind(gens3_pca, geom.ind = "point", pointsize = 0.2)
```



It's still not clear if there's any clustering on genre types. If there exist hundreds of small clusters, we can perform explorative modeling to verify this. Let's see if random forest can identify nuances in audio features to predict genres correctly. This may involve merging similar genre groups into 1.

Explorative Supervised Learning

If we simply train on all genres, we quickly run into huge problems. An error rate of 100%!!!

This is because there are simply too many classes to try and predict off of only 13 predictors. Even if specific genre names provide highly nuanced insight into user music preference, with the current features it may be unfeasible to make accurate predictions at such a specific scale. Instead we can opt to predict a smaller set of larger overarching genres, **sacrificing user specificity for prediction accuracy.**

```r
# we'll use the streaming data since it has more data points than library data
# we'll also use the combined streaming data since it further adds diversity in training
# note: 13 audio features = 13 predictors
usr_streams <- rbind(usr1_streams_scaled, usr2_streams_scaled)

#genre_rf <- randomForest(genres~., data = usr_streams, ntree = 500, mtry = 4, importance = TRUE)

# misclassifications
#genre_rf # error rate = 100%

# number of classes
paste('Classes to train on: ', length(unique(usr_streams$genres)))
```

```
## [1] "Classes to train on:  1203"
```

```r
paste('predictors: 13')
```

```
## [1] "predictors: 13"
```

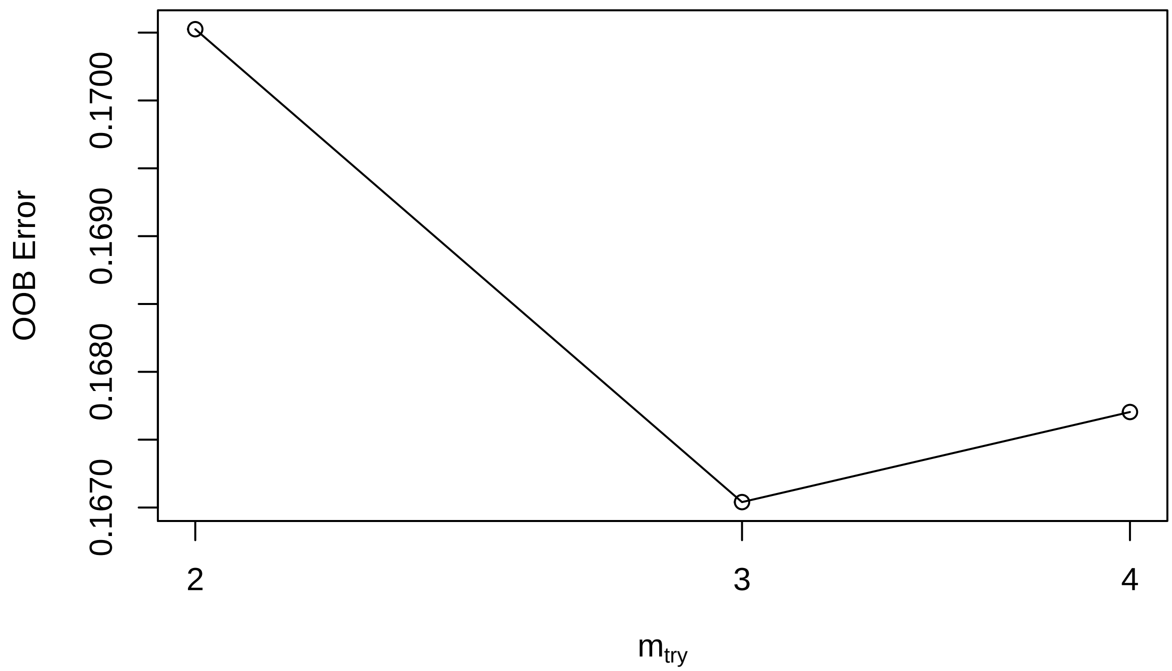
Grouping into fewer classes reduced the error rate from 100% to 16%! If we develop a method to further group genres we may be able to further improve accuracy. This would be considered a very datacentric approach to model building.

```r
# lets use general 12 classes
new_classes <- c('hip hop', 'jazz', 'edm', 'rock', 'punk', 'r&b', 'pop', 'rap', 'country', 'metal', 'la

# drop levels that aren't being used anymore
shortened_usr_streams <- usr_streams[usr_streams$genres %in% new_classes, ]
shortened_usr_streams$genres[shortened_usr_streams$genres == 'hip hop'] <- 'rap'
shortened_usr_streams <- droplevels(shortened_usr_streams)

genre_rf2 <- randomForest(genres~., data = shortened_usr_streams, ntree = 500, importance = TRUE)

# misclassifications
genre_rf2
```

```
##
## Call:
##  randomForest(formula = genres ~ ., data = shortened_usr_streams,      ntree = 500, importance = TRU
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 16.95%
## Confusion matrix:
##           classical country  edm jazz metal pop punk r&b  rap rock latin
## classical       233       0    2    0     0   0    0   0    0    1     0
## country           0      43    1    0     0   4    0   0    1    5     0
## edm               2       0 2600    1     0 348    0   0   29   49     0
## jazz              0       0    1   32     0   0    0   0    1    3     0
## metal             0       0    0    0   132   0    1   0    0  177     0
## pop               1       4  330    0     1 831    6  17   21  226     4
## punk              0       0    0    0     0   5  796   0    0  176     0
## r&b               0       0    1    0     0  17    0   8   43    0     0
## rap               0       0   16    0     2  11    0  11 2258    2     0
## rock              2       1   54    0   163 174  113   0    9 3036     0
## latin             0       0    2    0     0   4    0   0    0    1    39
##           class.error
## classical  0.01271186
## country    0.20370370
## edm        0.14163090
## jazz       0.13513514
## metal      0.57419355
## pop        0.42331714
## punk       0.18526100
## r&b        0.88405797
## rap        0.01826087
## rock       0.14527027
## latin      0.15217391
```

```r
# we should use an mtry of 4 for best restuls
tuneRF(shortened_usr_streams[-1], shortened_usr_streams$genres, ntreeTry = 500,
       stepFactor = 1.5, improve = 0.01, trace = TRUE, plot = TRUE)
```

```
## mtry = 3  OOB error = 16.7%
## Searching left ...
## mtry = 2     OOB error = 17.05%
## -0.02086438 0.01
## Searching right ...
## mtry = 4     OOB error = 16.77%
## -0.003974168 0.01
```




```
##       mtry  OOBError
## 2.OOB    2 0.1705253
## 3.OOB    3 0.1670401
## 4.OOB    4 0.1677039
```

```r
genre_rf2 <- randomForest(genres~., data = shortened_usr_streams, ntree = 500, mtry = 4, importance = T
```

```r
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.0.5
```

```
##
## Attaching package: 'xgboost'
```

```
## The following object is masked from 'package:dplyr':
##
```

```
##     slice
```

Modeling daily, weekly, and monthly listening habits

Modeling play time, and specific audio feature play time