Group Project

Group 11

2024-10-15

Group member information:

Group 11

- Group 11 Leader: Xuan Bach Tran
- Number of team members: 4

Names - Student ID - Contribution (%)

- Xuan Bach Tran - 22027453 – 25%
- Thanh Thong Tran - 22027485 – 25%
- Celine Nguyen - 22027457 – 25%
- Dinh Tuan Tran - 22026986 – 25%

Declaration

By including this statement, we the authors of this work, verify that:

- We hold a copy of this assignment that we can produce if the original is lost or damaged.
- We hereby certify that no part of this assignment/product has been copied form any other student's work or from any other source except where due acknowledgement is made in the assignment.
- No part of this assignment/product has been written/produced for us by another person except where such collaboration has been authorised by the subject lecturer/ tutor concerned.
- We are aware that this work may be reproduced and submitted to plagiarism detection software programs to detect possible plagiarism **(which may retain a copy on its database for future plagiarism checking)**
- We hereby certify that we have read and understood what the School of Computing and Mathematics defines as minor and substantial breaches of misconduct as outlined in the learning guide for this unit.

Working Section

Prelimiary Task

By using Spotify API, we will get all the tracks and their audio features from a playlist of our choice.

Before collecting and loading the required data, we will need to install and load the required libraries and packages that will used throughout the project.

```r
# Installing and loading the libraries required for this task

#install.packages("devtools")
#devtools::install_github("charlie86/spotifyr")
#install.packages("ggpubr")
#install.packages("GGally")

library(spotifyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(ggpubr)
```

```
## Loading required package: ggplot2
```

```r
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

Next, we need to initialise our working environment by retrieving Spotify API credentials.

```r
# Initiating working environment
Sys.setenv(SPOTIFY_CLIENT_ID = '3a5930bcdbb744079f18ee5b827e52f5') # Spotify client ID
Sys.setenv(SPOTIFY_CLIENT_SECRET = '52b9bdfead98433c8d5169c10ed6ea1a') # Spotify client secret ID"

access_token <- get_spotify_access_token() # Get access token
```

Data collection and cleaning

Data collection

We will now collect all the tracks and their audio features within a playlist. First, we will need to collect the playlist's ID (usually at the end of the playlist's URL), which is readily available on user's Spotify dashboard, by using built-in 'spotifyr' function called 'get_playlist_audio_features()'

```r
playlist_id <- "5Sc2esIc6s0VsJTE5GgfDg"
audio_features <- get_playlist_audio_features(playlist_uris = "5Sc2esIc6s0VsJTE5GgfDg")

head(audio_features) # View the first six tracks and their features
```

```
## # A tibble: 6 x 62
##   playlist_id   playlist_name playlist_img playlist_owner_name playlist_owner_id
##   <chr>         <chr>         <chr>        <chr>               <chr>
## 1 5Sc2esIc6s0V~ Lifetime Pla~ https://ima~ Tran Xuan Bach      316ecxwlvl7i2gxb~
## 2 5Sc2esIc6s0V~ Lifetime Pla~ https://ima~ Tran Xuan Bach      316ecxwlvl7i2gxb~
## 3 5Sc2esIc6s0V~ Lifetime Pla~ https://ima~ Tran Xuan Bach      316ecxwlvl7i2gxb~
## 4 5Sc2esIc6s0V~ Lifetime Pla~ https://ima~ Tran Xuan Bach      316ecxwlvl7i2gxb~
## 5 5Sc2esIc6s0V~ Lifetime Pla~ https://ima~ Tran Xuan Bach      316ecxwlvl7i2gxb~
## 6 5Sc2esIc6s0V~ Lifetime Pla~ https://ima~ Tran Xuan Bach      316ecxwlvl7i2gxb~
## # i 57 more variables: danceability <dbl>, energy <dbl>, key <int>,
## #   loudness <dbl>, mode <int>, speechiness <dbl>, acousticness <dbl>,
## #   instrumentalness <dbl>, liveness <dbl>, valence <dbl>, tempo <dbl>,
## #   track.id <chr>, analysis_url <chr>, time_signature <int>, added_at <chr>,
## #   is_local <lgl>, primary_color <lgl>, added_by.href <chr>,
## #   added_by.id <chr>, added_by.type <chr>, added_by.uri <chr>,
## #   added_by.external_urls.spotify <chr>, track.preview_url <chr>, ...
```

```r
dim(audio_features) # View the total number of observations and columns
```

```
## [1] 2514   62
```

Select the desired audio features

```r
audio_features_selected <- audio_features %>%
  select(track.name, danceability, energy, speechiness, acousticness,
         instrumentalness, liveness, valence, tempo) %>%
  na.omit()
```

There were various audio features to choose from, we selected the ones that are audio-related, the selected features and their interpretations are as follows:

- **danceability**: Measures how suitable a track is for dancing based on tempo, rhythm, beat strength,.. Danceability ranges from 0.0 being the least danceable and 1.0 being the most danceable.
- **energy**: Energetic tracks feel loud, fast, upbeat, and fun.
- **speechiness**: Measures the presence of spoken words in a track. The more speech-like is the recording of tracks (e.g. talk show, audio book, poetry), the higher is the attribute value
- **acousticness**: Reflects how much of the track is acoustic.
- **instrumentalness**: Level of vocal presence in a track. For example, rap or hip-hop tracks are clearly not very instrumental, whereas orchestral play scores very high on the scale.
- **liveness**: Detects whether there's audience in the track. If the tracks scores high in the feature, there's an increased probability that the track was performed live.
- **valence**: Measures the positivity or emotional tone of a track.
- **tempo**: Measured in beats per minute (BPM), it is the estimated tempo of a track.

Data cleaning

As we go through have all the tracks and their audio features, we noticed there were a lot of tracks that had NA for their features, that is because those tracks are locally imported from a member's computer and they were not in Spotify database; therefore, we would need to remove them.

```
audio_features_selected <- na.omit(audio_features_selected)
head(audio_features_selected) # View the first six tracks and their selected features
```

```
## # A tibble: 6 x 9
##   track.name       danceability energy speechiness acousticness instrumentalness
##   <chr>                   <dbl>  <dbl>       <dbl>        <dbl>            <dbl>
## 1 1-800-273-8255          0.62   0.574      0.0479       0.569           0
## 2 Somebody That I~        0.865  0.521      0.0371       0.548           0.000115
## 3 Wildcard (feat.~        0.591  0.835      0.0977       0.0998          0
## 4 Payphone                0.743  0.752      0.0414       0.0188          0
## 5 Lalala                  0.843  0.39       0.0846       0.178           0
## 6 Mask Off                0.833  0.434      0.431        0.0102          0.0219
## # i 3 more variables: liveness <dbl>, valence <dbl>, tempo <dbl>
```

```
dim(audio_features_selected) # Now the number of observations is down to 869 tracks
```

```
## [1] 869   9
```

Scaling the data

The "tempo" attributes seem contain values that are far larger than the other measurements, most of which vary between 0 and 1, except for instrumentalness. Therefore, we are going to normalise the data using the min-max normalisation scheme.

```
# Normalization function: rescales data to the range [0,1]
minmax <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}

# Apply normalization to the numeric features
audio_features_selected <- as.data.frame(lapply(audio_features_selected[, c("danceability", "energy","sp

head(audio_features_selected) # View the first 6 tracks
```

```
##   danceability    energy speechiness acousticness instrumentalness   liveness
## 1    0.5756792 0.5714139  0.04019427   0.57530057      0.000000000 0.20252572
## 2    0.8926261 0.5170717  0.02210685   0.55406560      0.000119667 0.09600094
## 3    0.5381630 0.8390239  0.12359739   0.10085064      0.000000000 0.15341441
## 4    0.7347995 0.7539219  0.02930832   0.01894432      0.000000000 0.31594949
## 5    0.8641656 0.3827540  0.10165801   0.17992563      0.000000000 0.14055192
## 6    0.8512290 0.4278683  0.68179534   0.01024810      0.022788762 0.17329280
##     valence     tempo
## 1 0.3410197 0.3699574
## 2 0.7546012 0.5374335
## 3 0.4626613 0.5309274
## 4 0.5398773 0.4275901
## 5 0.4869896 0.5426996
## 6 0.2606304 0.6585763
```

Statistical Analysis

Question 1:

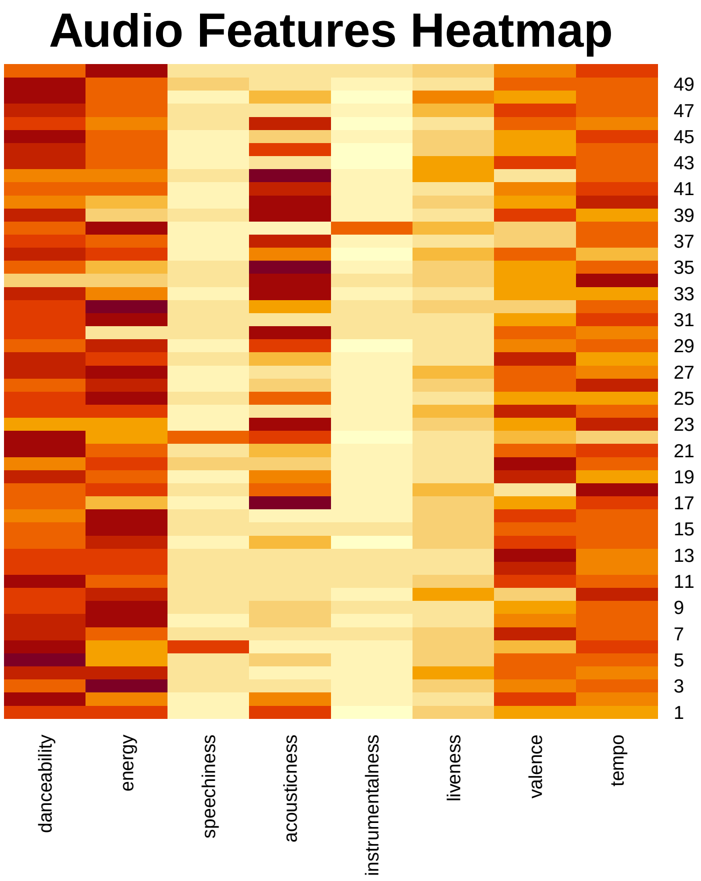
Research question: What insights can be drawn from visualisations of the tracks' audio features?

Heatmap for audio features

We wanted to inspect how the features would look like if they were presented as a heat map. The values would be encoded through color saturation, in this case, it's orange. The larger is the value, the darker is the color assigned to it, and vice versa. This will give us an overall understanding of the tracks' audio nature, and the 'heatmap()' function can help us achieve that.

```r
# Subsetting for the first 50 tracks
first_50_tracks <- audio_features_selected[1:50,]
```

```r
# Plotting the heatmap of audio features
heatmap(as.matrix(first_50_tracks),
        Colv = NA,
        Rowv = NA,
        main = "Audio Features Heatmap",
        cexRow = 0.75,
        cexCol = 0.75)
```



Interpretation: Based on the customised layout of the heatmap above, we can inspect the audio features of the first 50 tracks within the playlist. We used the color saturation of orange to indicate the level of each feature. For example, it seemed that the first 50 tracks all have high level of danceability, which is clearly obvious because most the tracks belong to pop/indie pop genre. Furthermore, almost all of them have very low level of instrumentalness, meaning very few of them are free of vocals and speech. Since nearly all of the tracks contain music as the primary component and not many speech-like feature, the speechiness level is very low accross the first 50 tracks.

Histogram for audio features

In terms of the frequency distribution of individual audio features from tracks, the histogram will be the best use. The chosen features are valence and acousticness as the range of those features are believed to vary, dependently of the tracks's nature and genres.

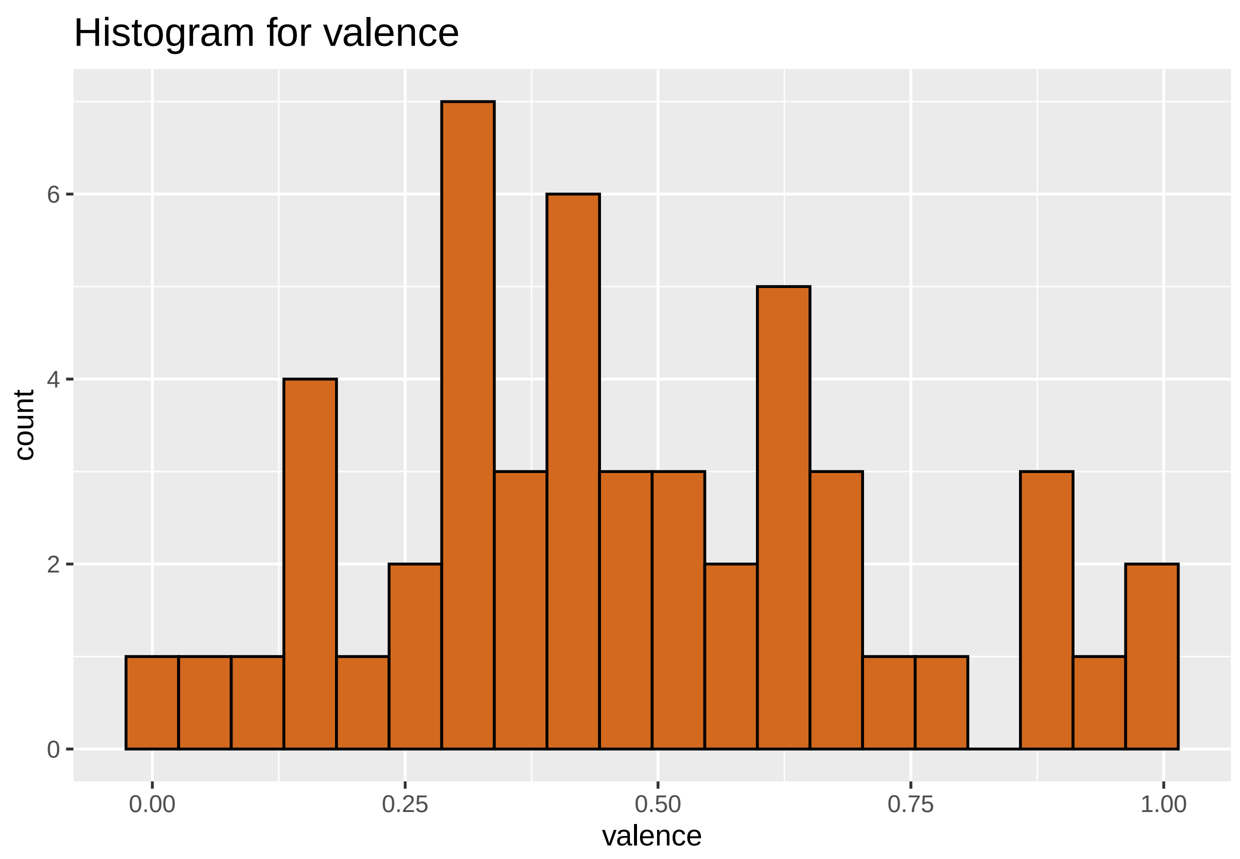
Firstly, we will shuffle the first 50 tracks to ensure to ensure the randomness.

```
# Initialising a for loop to shuffle the tracks
for (i in 1:nrow(first_50_tracks)){
  first_50_tracks[i,] = sample(first_50_tracks[i,])
  return(first_50_tracks)
}

head(first_50_tracks)
```

```
##   danceability    energy speechiness acousticness instrumentalness   liveness
## 1    0.5714139 0.3410197  0.20252572   0.57567917      0.369957433 0.57530057
## 2    0.8926261 0.5170717  0.02210685   0.55406560      0.000119667 0.09600094
## 3    0.5381630 0.8390239  0.12359739   0.10085064      0.000000000 0.15341441
## 4    0.7347995 0.7539219  0.02930832   0.01894432      0.000000000 0.31594949
## 5    0.8641656 0.3827540  0.10165801   0.17992563      0.000000000 0.14055192
## 6    0.8512290 0.4278683  0.68179534   0.01024810      0.022788762 0.17329280
##     valence      tempo
## 1 0.0000000 0.04019427
## 2 0.7546012 0.53743353
## 3 0.4626613 0.53092736
## 4 0.5398773 0.42759007
## 5 0.4869896 0.54269960
## 6 0.2606304 0.65857626
```

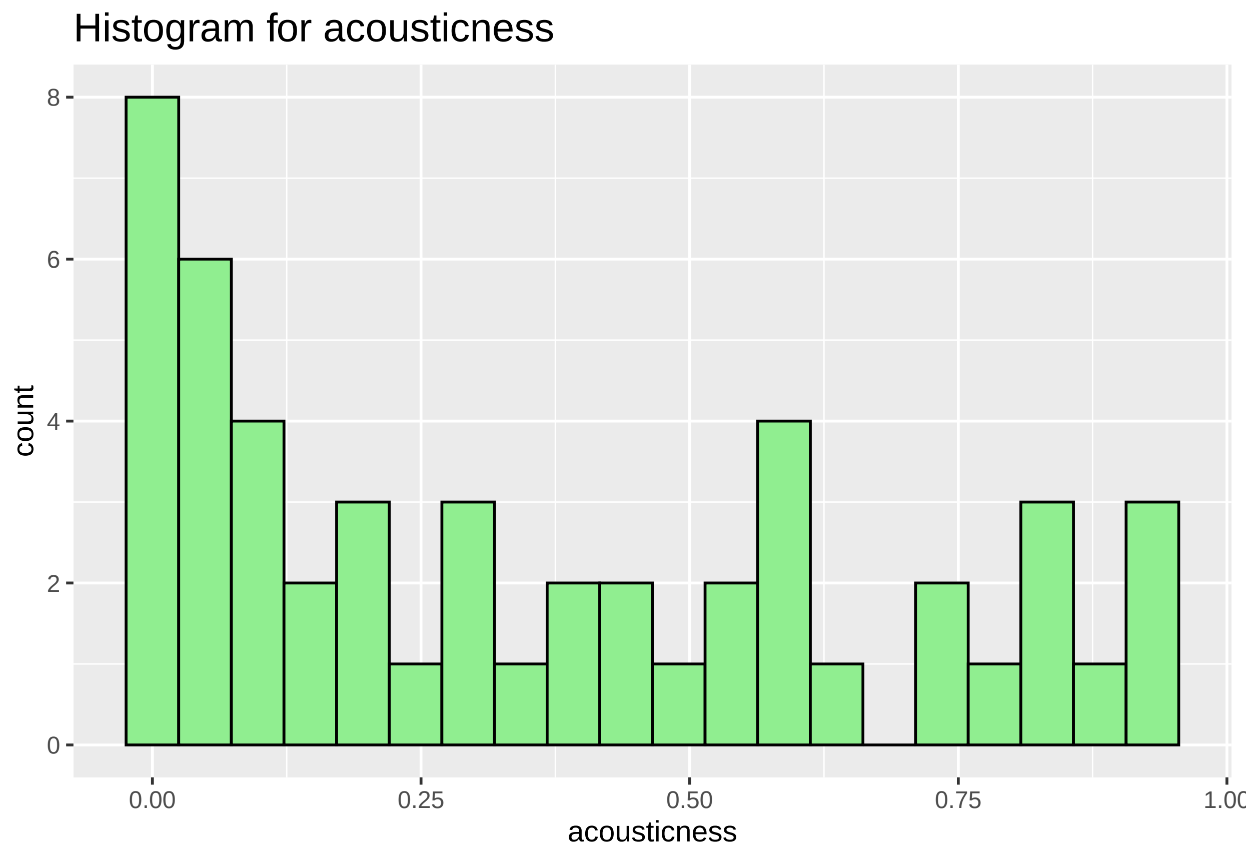
a. Histogram for valence

```
ggplot(first_50_tracks, aes(x = valence)) +
  geom_histogram(fill="chocolate", color="black", alpha=1, bins = 20)+
  ggtitle("Histogram for valence") +
    theme(plot.title = element_text(size=15))
```



b. Histogram for acousticness

```
ggplot(first_50_tracks, aes(x = acousticness)) +
  geom_histogram(fill="lightgreen", color="black", alpha=1, , bins = 20)+
  ggtitle("Histogram for acousticness") +
    theme(plot.title = element_text(size=15))
```



Interpretation: In terms of distribution for valence, high counts can be observed for songs that have valence between 0.3 to 0.45, and between 0.6 to 0.65. The same pattern is applied for tracks with acousticness between 0 and 0.2. Hence, we can infer that the first 50 songs are of mixed valence, meaning there seems to be an equal share between songs with positivity (happy, cheerful) and those with negative nature (sad, gloomy). Furthermore, tracks with medium or high valence all seem to have relatively high level of acousticness, which may suggest that acoustic songs in this playlist are likely to have positive energy.

Question 2: Construct K-means Clustering

Research question: How can we group tracks with similar audio features and assign their genres using K-means clustering technique?

By leveraging K-Means Clustering, we can identify the clusters to which the tracks may belong based on their audio features, and we can assign the clusters with potential genres.

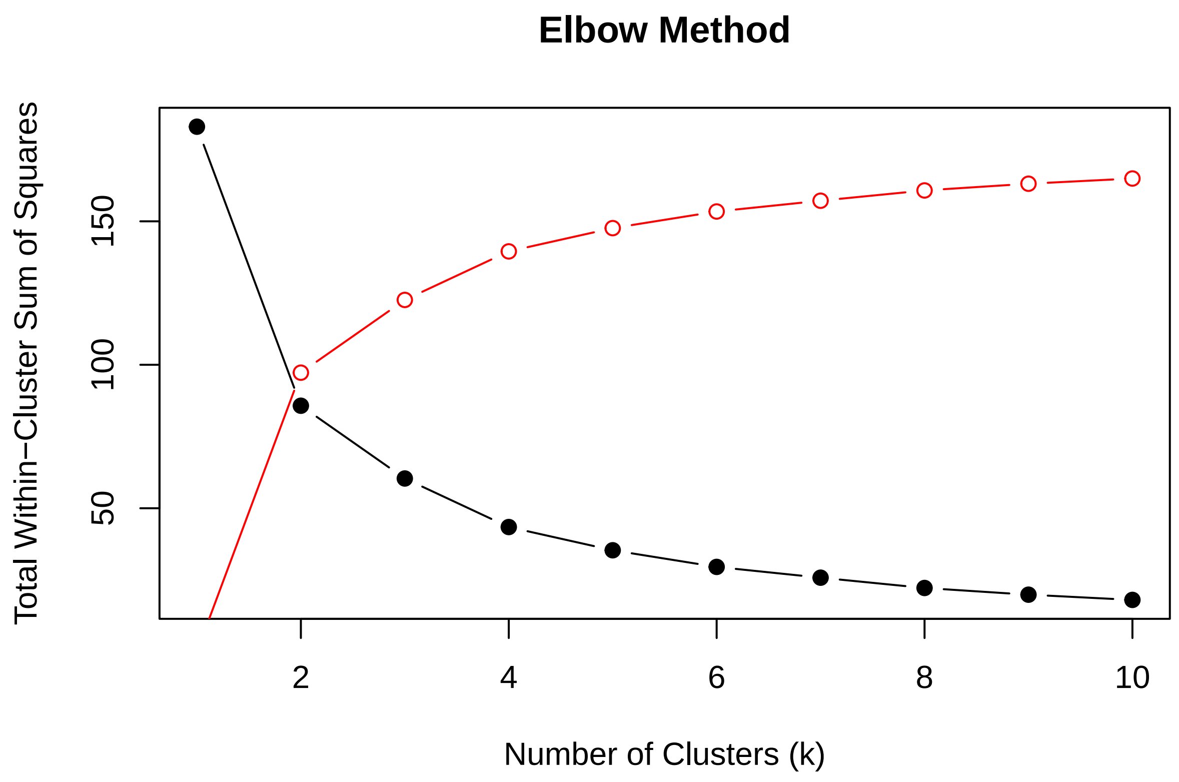
Since the dataset has multiple variables, and the distance between pairs of observations is present, we can scale the data to 2 dimensions to represent the distance between all pairs of observations. The K-means clustering is going to use Euclidean distance.

```r
# Construct a distance matrix on the audio_features_normalised
distance_matrix <- dist(audio_features_selected)

# Perform multi-dimensional scaling to 2D space
mds.euclidean <- cmdscale(distance_matrix) # Using Euclidean distance
```

Finding the optimal number of clusters (k)

We will proceed to plot the Sum of Square Withins and Sum of Square Betweens after performing K-means clustering with k ranging from 1 to 10 to visualise the optimal number of clusters that we need.

```r
set.seed(123)  # For reproducibility

# Define a range of cluster numbers
k <- 1:10

# Initialize a vector to hold the SSW and SSB values
ssw <- numeric(length(k))
ssb <- numeric(length(k))

# Loop through the range of k values and calculate WSS
for (i in k) {
  kmeans_result <- kmeans(mds.euclidean, centers = i, nstart = 25)
  ssw[i] <- kmeans_result$tot.withinss  # Store the SSW
  ssb[i] <- kmeans_result$betweenss # Store the SSB
}

# Plotting the result with Elbow Method
plot(1:10, ssw, type = "b", pch = 19,
     xlab = "Number of Clusters (k)",
     ylab = "Total Within-Cluster Sum of Squares",
     main = "Elbow Method")
lines(1:10, ssb, type = 'b', col = 'red')
```



Assessment of the SSW and SSB:

- We can see that the Within Sum of Square (SSW) declines steeply as the number of clusters go from 1 to 3, indicating that the distance between the cluster centroid and the data points is small enough to form its own distinct cluster.
- As for Between Sum of Square (SSB), it increases sharply when k ranges from 1 to 3, which suggest the distance between cluster centroids is large enough. Moreover, the Within Sum of Square (SSW) appears to become more stablilised at k = 3.

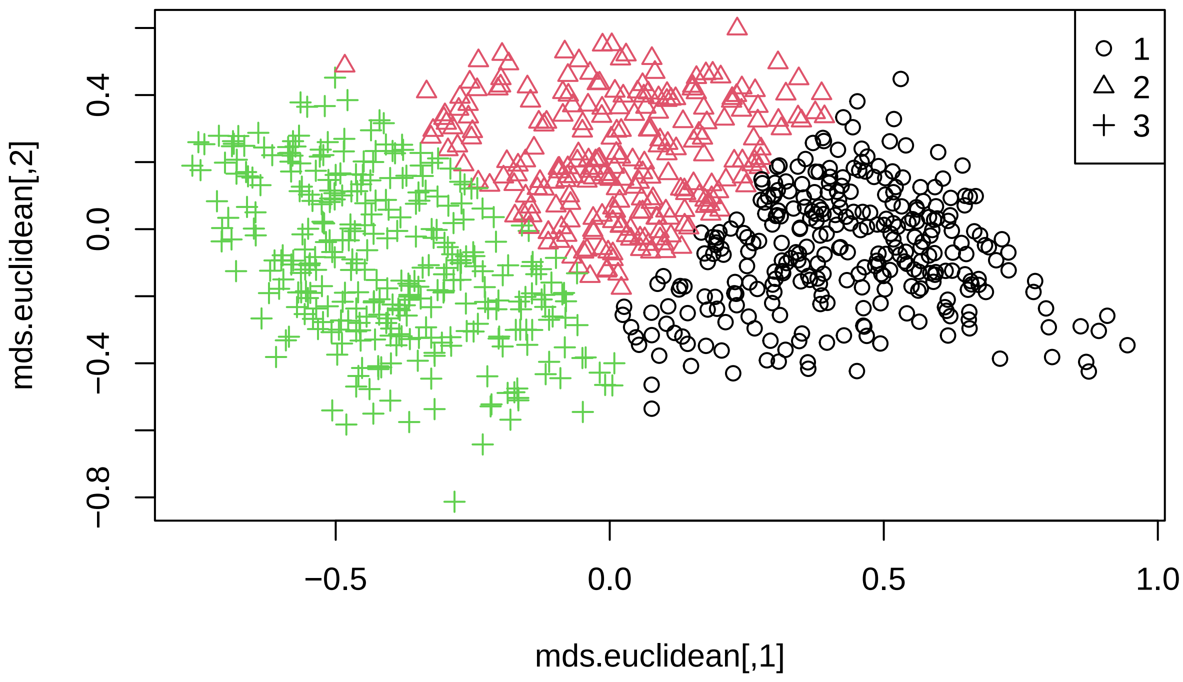
Hence, the most optimal number of cluster to perform K-Means clustering is 3.

Performing K-means Clustering with the optimal k

```
k <- 3 # Setting determined k value

# Perform K-Means clustering with the normalised data
kmeans_result <- kmeans(mds.euclidean, centers = k, nstart = 25) # try 25 different initialisations

# Assigning the result to a new "cluster" column
audio_features_selected$cluster <- kmeans_result$cluster

table(audio_features_selected$cluster) # Tabulate each cluster
```

```
##
##   1   2   3
## 297 244 328
```

```
plot(mds.euclidean, col = as.numeric(audio_features_selected$cluster),
     pch = as.numeric(audio_features_selected$cluster))
legend("topright", legend = c(1,2,3), pch = c(1,2,3))
```



```r
# Getting SSW and SSB
cat("Within Sum of Square: ", kmeans_result$withinss,
"\nBetween Sum of Square: ",kmeans_result$betweenss)
```

```
## Within Sum of Square:  19.07491 13.72539 27.57006
## Between Sum of Square:  122.6042
```

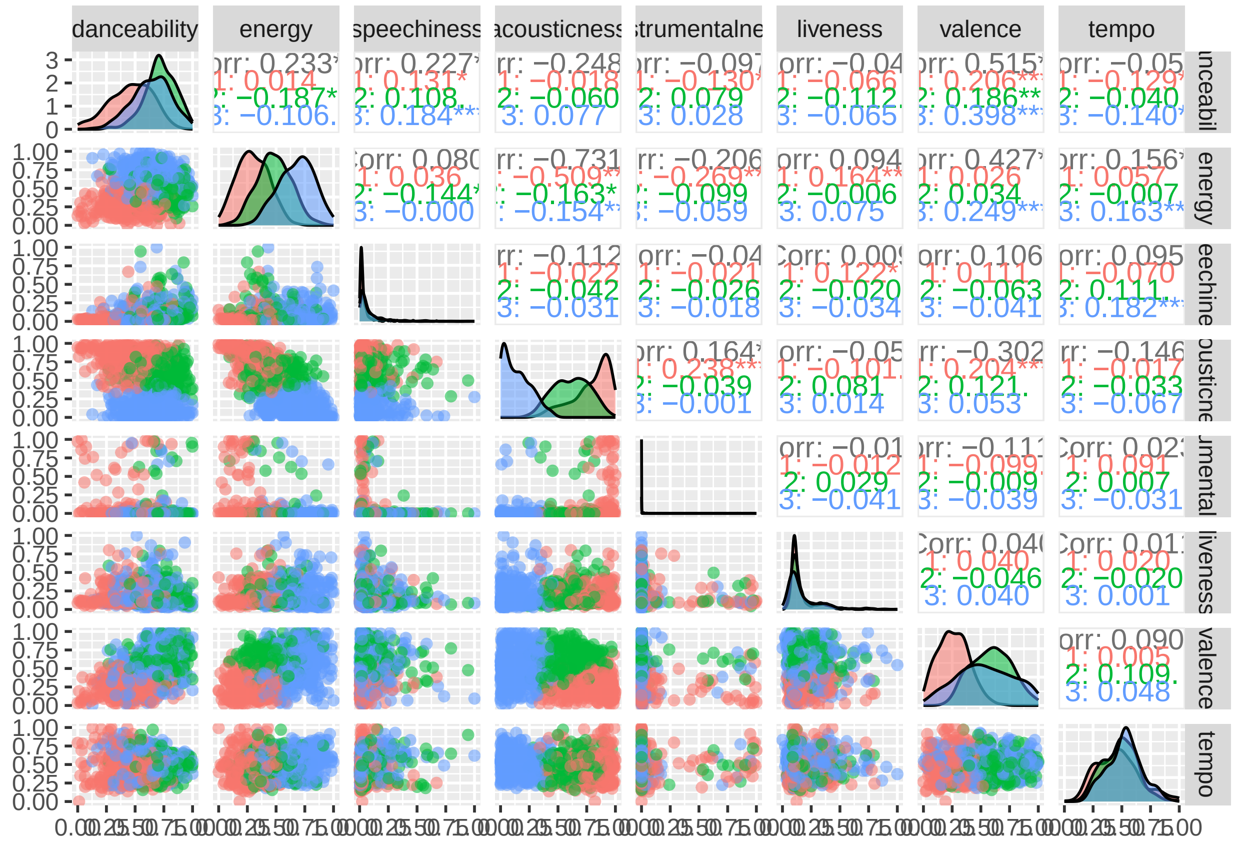
Interpretation

- The clusters seem well-separated, suggesting that the k-means algorithm has successfully grouped similar tracks based on their audio features.
- Data points concentration can be observed in Cluster 1 (circles), the reason might be that tracks in this group have very closely related audio features.
- As for Cluster 2 (triangles) and Cluster 3 (plus signs), the data points are more dispersed, with minor overlapping data points between the clusters, suggesting that tracks within these groups may have more variance in their features but are still distinguishable.

Genre classification

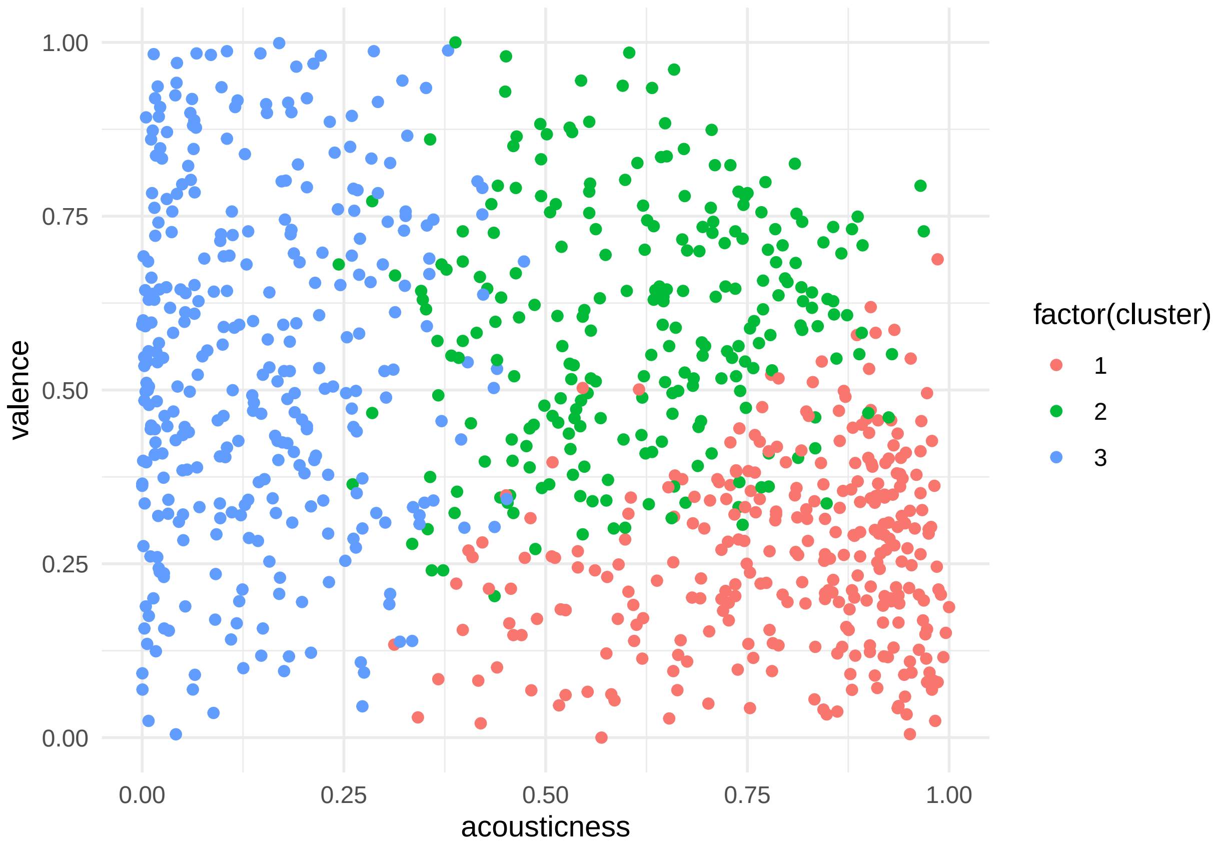
Our group's approach is creating a correlation plot between the audio features and see which pair of dimensions would separate the tracks best. The ggpairs() function will provide us with a plotting matrix from every possible pairs of dimensions.

```r
ggpairs(audio_features_selected,
        aes(color = as.factor(cluster), alpha = 0.5),  #controls the transparency (opacity) of the poin
        columns = 1:8) # Select the columns representing your features (1:8)
```



Based on the visualisation, it is clear that any pair of dimensions with accousticness as a component would the best cluster separators. Therefore, we will proceed to visualise the clusters with acousticness and valence:

```r
ggplot(audio_features_selected, mapping = aes(x = acousticness , y = valence, color = factor(cluster)))
  geom_point() +
  theme_minimal()
```



Based on the classification of music genres through their audio features, the K-Means clusters can be identified as below:

KM1: The tracks that have membership with this cluster is observed to have relatively high acousticness and valence; therefore, they are likely to be Pop/Indie Pop.

KM2: We find that tracks which belong to this cluster have very wide range of valence, and most of them have relatively low acousticness nature. Hence, the potential genere is Hip-hop/Rap .

KM3: Low valence and high acousticness can be observed in tracks within this cluster. Our best estimation of the genre is Blues.

Question 3: Hypothesis Testing

Research question: In this section of the report, we are interested testing whether the acousticness of a song has a linear relationship to its popularity score on Spotify. Therefore, the hypothesis test is implemented.

```r
# Loading the required packages the task
library(corrplot)
```

```
## corrplot 0.94 loaded
```

In this section of the report, we are interested testing whether the acousticness of a song has a linear relationship to its popularity score on Spotify. Therefore, the hypothesis test is implemented.

- Null hypothesis (H0): There is no linear relationship between the acousticness and popularity.

- Alternative hypothesis (H1): There is a linear relationship between the acousticness and popularity.

Data collection

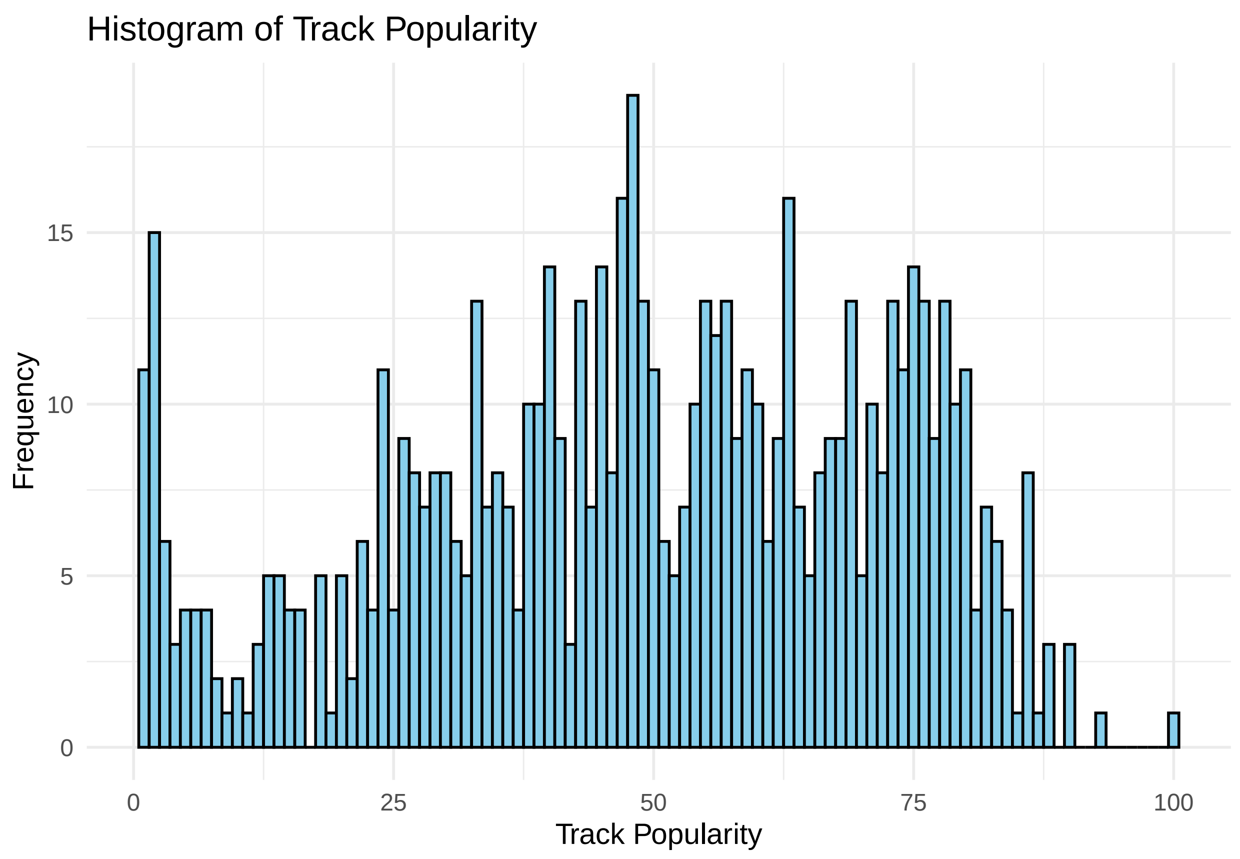
```r
# Get playlist tracks information
playlist_tracks = get_playlist_audio_features('', playlist_id)

# Select audio popularity and acousticness feature
hypo_data <- playlist_tracks %>%
  select(track.name, acousticness, track.popularity) %>%
  filter(track.popularity != 0) %>% #Remove all tracks have error in popularity (track.popularity=0)
  na.omit() #Remove all tracks have NA value

dim(hypo_data)
```

```
## [1] 683   3
```

```r
# There are 682 songs (observations) with 3 variables (track.name, acousticness and track.popularity)
```

We can see the distribution of track.popularity by plotting the data.

```r
ggplot(hypo_data, aes(x = track.popularity)) +
  geom_histogram(binwidth = 1, fill = "skyblue", color = "black") +
  labs(title = "Histogram of Track Popularity", x = "Track Popularity", y = "Frequency") +
  theme_minimal()
```



Methodology

We first build the Simple Linear Regression model by using lm() function

```r
hypo_model = lm(formula = track.popularity ~ acousticness, data = hypo_data)

# See the summary output
summary(hypo_model)
```

```
##
## Call:
## lm(formula = track.popularity ~ acousticness, data = hypo_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -55.636 -14.257   2.095  17.241  48.275
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    56.952      1.495  38.103  < 2e-16 ***
## acousticness  -16.969      2.656  -6.389  3.1e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 22.26 on 681 degrees of freedom
## Multiple R-squared:  0.05655,    Adjusted R-squared:  0.05516
## F-statistic: 40.82 on 1 and 681 DF,  p-value: 3.101e-10
```

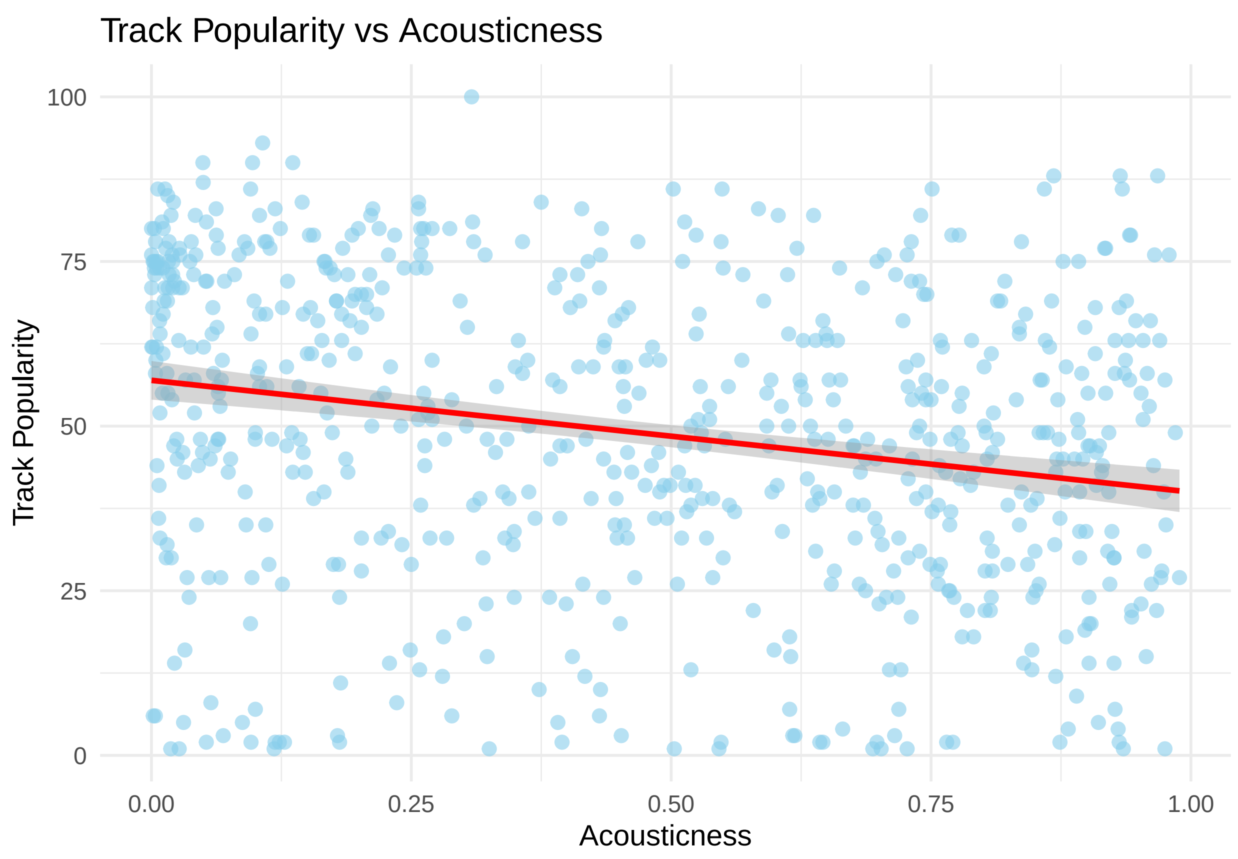
P-Value is 3.54e-11 (Refer to the output above), which is less than 0.05 (at 5% level of significance) so we can reject Null Hypothesis and and support the alternative hypothesis. Therefore, strong evidence to support that there is a significant linear relationship between track.popularity and acousticness.

Based on the output, the slope of the regression line is -17.795, which means for every additional acousticness point, the popularity of that song is predicted to decrease by -17.795 points. In other words, a negative slope value also indicates a negative relationship between track.popularity and acousticness.

Visualising the Regression model

```r
ggplot(hypo_data, aes(x = acousticness, y = track.popularity)) +
  geom_point(color = "skyblue", size = 2, alpha = 0.6) +  # Scatter plot of the data points
  geom_smooth(method = "lm", color = "red") +  # Regression line
  labs(title = "Track Popularity vs Acousticness",
       x = "Acousticness",
       y = "Track Popularity") +
  theme_minimal()  # Clean theme for aesthetics
```

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



Result Interpretation

The output of the hypothesis above depicts that the more acoustic a song is, the less popularity point it has (negative linear relationship). In other words, songs that use electronic musical instruments will be more popular than songs that use classical instruments. This also reflects the audience's music listening trend on the Spotify platform. Listeners tend to favor modern songs with non-classical rhythms. The insight can be valuable for music producers, artists to reach a larger audience segment and increase song recognition.

Question 4: Spotify Network Graph

```r
# Installing and loading the required libraries and packages for the task

#install.packages("devtools")
#devtools::install_github("charlie86/spotifyr")
#install.packages("httr")
#install.packages("rvest")
#install.packages("FactoMineR")
#install.packages("factoextra")
#install.packages("ggraph")
#install.packages("graphlayouts")

library(httr)
library(igraph)
```

```
##
```

```
## Attaching package: 'igraph'

## The following objects are masked from 'package:dplyr':
##
##     as_data_frame, groups, union

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following object is masked from 'package:igraph':
##
##     crossing
```

```
library(jsonlite)
library(ggraph)
library(FactoMineR)
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
library(graphlayouts)
```

Data Collection

To start data collecting for constructing a network to analyze the common linkages among playlists, we will get the five playlists on Spotify. Four of them would be the most popular playlist on Spotify and one of them is a personal playlist. This network aims to analyze the correlation between playlists based on genre and artist, thereby concentrating solely on "Song_Title," "Artist," and "Playlist" as our primary focal points.

```
playlist_ids = c('5Sc2esIc6s0VsJTE5GgfDg', '37i9dQZF1DWXXs9GFYnvLB', '37i9dQZF1DXcBWIGoYBM5M','37i9dQZE

# Initialize an empty data frame
all_playlists_data = data.frame()

# Loop through each playlist ID and combine the data
for (playlist_id in playlist_ids) {
  # Get playlist data
  playlist_data = get_playlist_tracks(playlist_id)

  # Extract song titles and artists
  song_artist_data = playlist_data %>%
    select(track.name, track.artists) %>%
    unnest_longer(track.artists) %>%
    unnest_wider(track.artists) %>%
    select(track.name, name) %>%
    rename(Song_Title = track.name, Artist = name)

  # Add playlist column
  song_artist_data$Playlist = playlist_id

  # Combine with the main data frame
  all_playlists_data = bind_rows(all_playlists_data, song_artist_data)
}

# View combined data from all playlists
head(all_playlists_data)
```

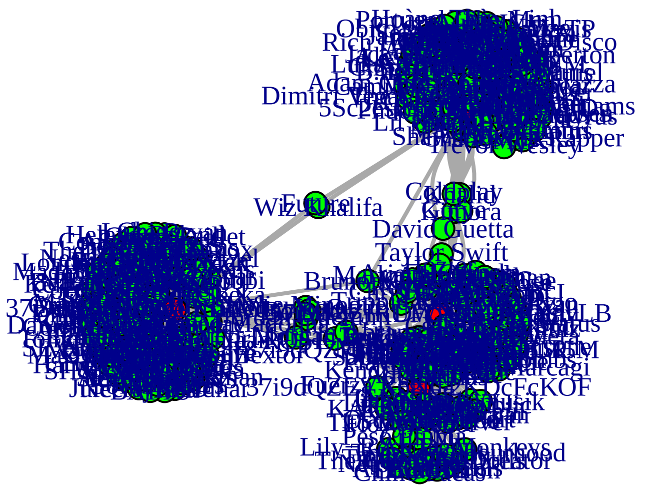
```
##                        Song_Title       Artist               Playlist
## 1                  1-800-273-8255        Logic 5Sc2esIc6s0VsJTE5GgfDg
## 2                  1-800-273-8255 Alessia Cara 5Sc2esIc6s0VsJTE5GgfDg
## 3                  1-800-273-8255       Khalid 5Sc2esIc6s0VsJTE5GgfDg
## 4   Somebody That I Used To Know        Gotye 5Sc2esIc6s0VsJTE5GgfDg
## 5   Somebody That I Used To Know       Kimbra 5Sc2esIc6s0VsJTE5GgfDg
## 6 Wildcard (feat. Sidnie Tipton)        KSHMR 5Sc2esIc6s0VsJTE5GgfDg
```

Methodology

Initially, we should analyze the artist connections among playlists. This R function constructs and examines a graph network derived from the Spotify playlist data, with playlists and artists depicted as nodes, and edges signifying the links between them.

```
edges_playlist_artist <- all_playlists_data %>%
  select(Playlist, Artist) %>%
  rename(from = Playlist, to = Artist)

# Create the graph object
g <- graph_from_data_frame(edges_playlist_artist, directed = FALSE)

# Assign colors to distinguish between Playlists and Artists
V(g)$color <- ifelse(V(g)$name %in% all_playlists_data$Playlist, "red", "green")

# Plot the graph with customized colors and sizes
plot(g, vertex.size = 10, vertex.label.cex = 0.8, edge.width = 2, layout = layout_with_fr, main="Spotif
```

Spotify Playlist to Artist Network



```
artist_betweenness <- betweenness(g, v = V(g)[name %in% all_playlists_data$Artist])
top_artists <- sort(artist_betweenness, decreasing = TRUE)
head(top_artists)
```

```
##    Bruno Mars   Wiz Khalifa        Future  Taylor Swift Gracie Abrams
##      7938.653      3663.823      3663.823      3280.034      3234.666
##      Coldplay
##      2454.708
```

We can see that each playlist is represented by a cluster of songs where they connected to the red dot in the middle. The notice point on this is that there are connection between playlists artists and we can also observe that in the network. We can also examine further by calculate the Betweeness Centrality to measures the number of times a node (artist) acts as a bridge along the shortest path between two other nodes. Higher betweenness indicates that the artist is more central to the network, playing an important role in connecting playlists.

According to betweenness centrality, the five musicians with the most connections in the playlists are "Halsey," "Ariana Grande," "Coldplay," "Taylor Swift," "Charli XCX," and "The Kid LAROI." This would signify the influence of these artists on current popular playlists.

Genre collection:

In SpotifyR, we cannot actually get the genre from the tracks but we can get the genre of the artists. That why we need a loop here to get a list of genres

```r
unique_artists <- unique(all_playlists_data$Artist)
artist_genres_df <- data.frame(Artist = character(), Genre = character(), stringsAsFactors = FALSE)

# Loop through the list of unique artists and get their genres
for (artist in unique_artists) {
  # Search for the artist on Spotify to get the artist ID
  artist_search <- search_spotify(artist, type = "artist")

  if (nrow(artist_search) > 0) {
    artist_id <- artist_search$id[1]  # Get the first matching artist's ID

    # Get detailed information about the artist, including genres
    artist_info <- get_artist(artist_id)

    # Store artist name and genres as a comma-separated string
    artist_genres <- paste(artist_info$genres, collapse = ", ")

    # Append to the artist_genres_df
    artist_genres_df <- rbind(artist_genres_df, data.frame(Artist = artist, Genre = artist_genres, stri
  }
}
```

```
## Request failed [503]. Retrying in 1.5 seconds...
```

```
## Error in curl::curl_fetch_memory(url, handle = handle): Recv failure: Connection was reset
## Request failed [ERROR]. Retrying in 1.9 seconds...
```

```r
# View the result (optional)
head(artist_genres_df)
```

```
##         Artist
## 1        Logic
## 2 Alessia Cara
## 3       Khalid
## 4        Gotye
## 5       Kimbra
## 6        KSHMR
##                                                                                                     G
## 1                                                                conscious hip hop, hip hop, pop rap
## 2                                                            canadian contemporary r&b, canadian pop
## 3                                                                                            pop, po
## 4                                                                                          australia
## 5                                                                        bergen indie, electropop, n
## 6 big room, dutch house, edm, electro house, indian edm, pop dance, progressive electro house, slap
```

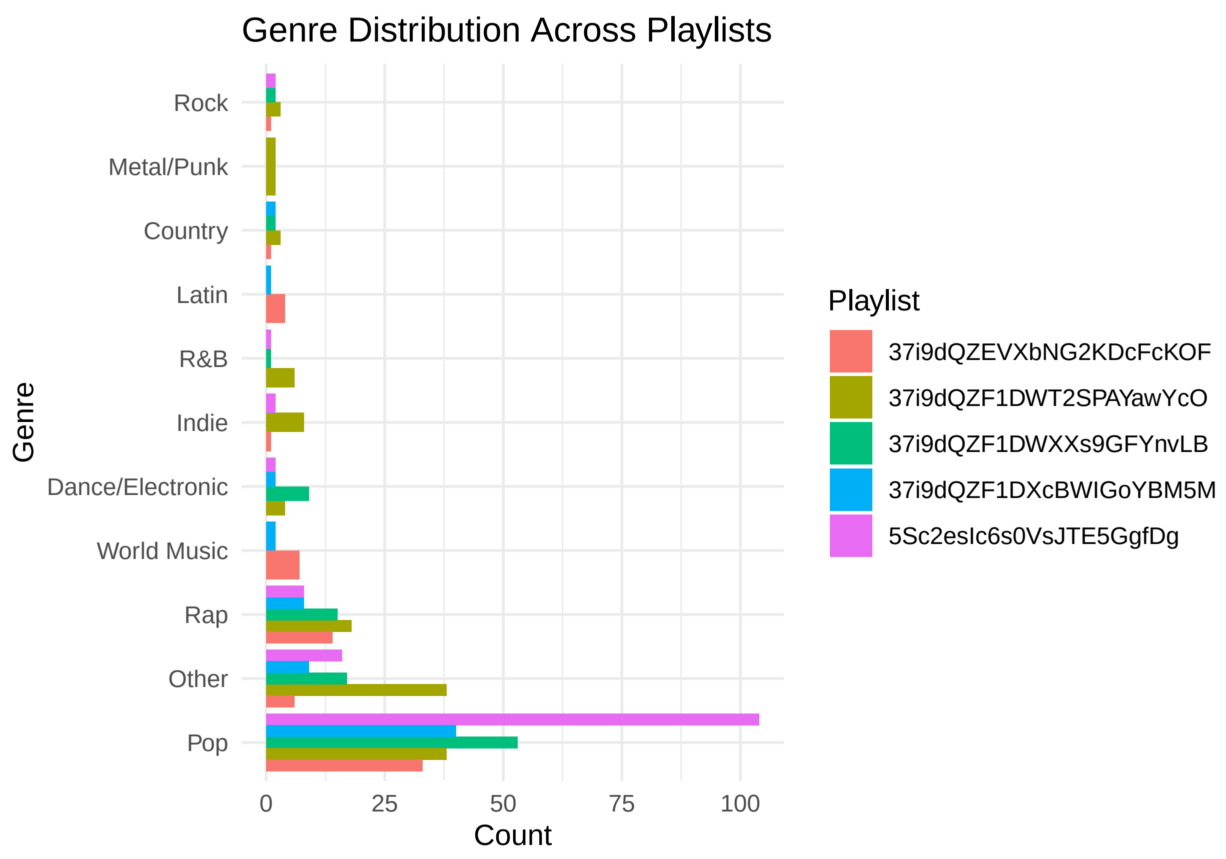
```r
## Side note: this chunk of code takes some time to run
```

However, the genre list that we retrieved has so many variance in its genre so we might want to simplify it to a more general genre like "Pop", "Rock", "R&B", etc.

```r
all_playlists_data_with_genres <- merge(all_playlists_data, artist_genres_df, by = "Artist", all.x = TRU

# View genre distribution
simplify_genre <- function(genre) {
  genre <- tolower(genre)  # Convert to lowercase for uniformity

  # Map sub-genres to basic genres
  if (grepl("pop", genre)) {
    return("Pop")
  } else if (grepl("rap|hip hop", genre)) {
    return("Rap")
  } else if (grepl("rock", genre)) {
    return("Rock")
  } else if (grepl("dance|edm|house|electro|tech", genre)) {
    return("Dance/Electronic")
  } else if (grepl("country", genre)) {
    return("Country")
  } else if (grepl("indie", genre)) {
    return("Indie")
  } else if (grepl("r&b", genre)) {
    return("R&B")
  } else if (grepl("k-pop", genre)) {
    return("K-Pop")
  } else if (grepl("v-pop", genre)) {
    return("V-Pop")
  } else if (grepl("reggaeton|latino", genre)) {
    return("Latin")
  } else if (grepl("metal|punk", genre)) {
    return("Metal/Punk")
  } else if (grepl("folk", genre)) {
    return("Folk")
  } else if (grepl("amapiano|corrido", genre)) {
    return("World Music")
  } else {
    return("Other")  # Default to "Other" for uncategorized genres
  }
}

# Apply the function to the Genre column in your dataset
all_playlists_data_with_genres$Basic_Genre <- sapply(all_playlists_data_with_genres$Genre, simplify_gen
genre_distribution <- all_playlists_data_with_genres %>%
  separate_rows(Basic_Genre, sep = ", ") %>%
  group_by(Playlist, Basic_Genre) %>%
  summarise(Genre_Count = n()) %>%
  arrange(desc(Genre_Count))
```

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

```r
# Visualize the genre distribution
library(ggplot2)
ggplot(genre_distribution, aes(x = reorder(Basic_Genre, -Genre_Count), y = Genre_Count, fill = Playlist
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip() +
  labs(title = "Genre Distribution Across Playlists", x = "Genre", y = "Count") +
  theme_minimal()
```



The graph indicates that the pop genre is the most prevalent in terms of quantity. The most prevalent genre currently is pop music due to its significant presence in several playlists. A salient aspect is that in playlist number 5, the "Pop" genre far surpasses the other categories within the same playlist. The rationale is that Playlist Five is a personal compilation belonging to an individual, generally encompassing all the songs they have saved, so reflecting their musical tastes and preferences as indicated by the playlist analytics.

Construct Genre Network:

Genres coexisting inside the same playlist are interrelated, creating a graph of genre associations. We will implement that on every playlist. This would yield a combination of genre interactions across the playlist. A genre present in many playlists will be associated with diverse genres across different playlists, establishing cross-playlist linkages.

```
edges <- data.frame(from = character(), to = character(), stringsAsFactors = FALSE)

# Loop through the playlists and create edges between genres in the same playlist
unique_playlists <- unique(all_playlists_data_with_genres$Playlist)

for (playlist in unique_playlists) {
  # Get the genres in the current playlist
  genres_in_playlist <- unique(all_playlists_data_with_genres$Basic_Genre[all_playlists_data_with_genre

  # Create edges between all pairs of genres in the same playlist
```

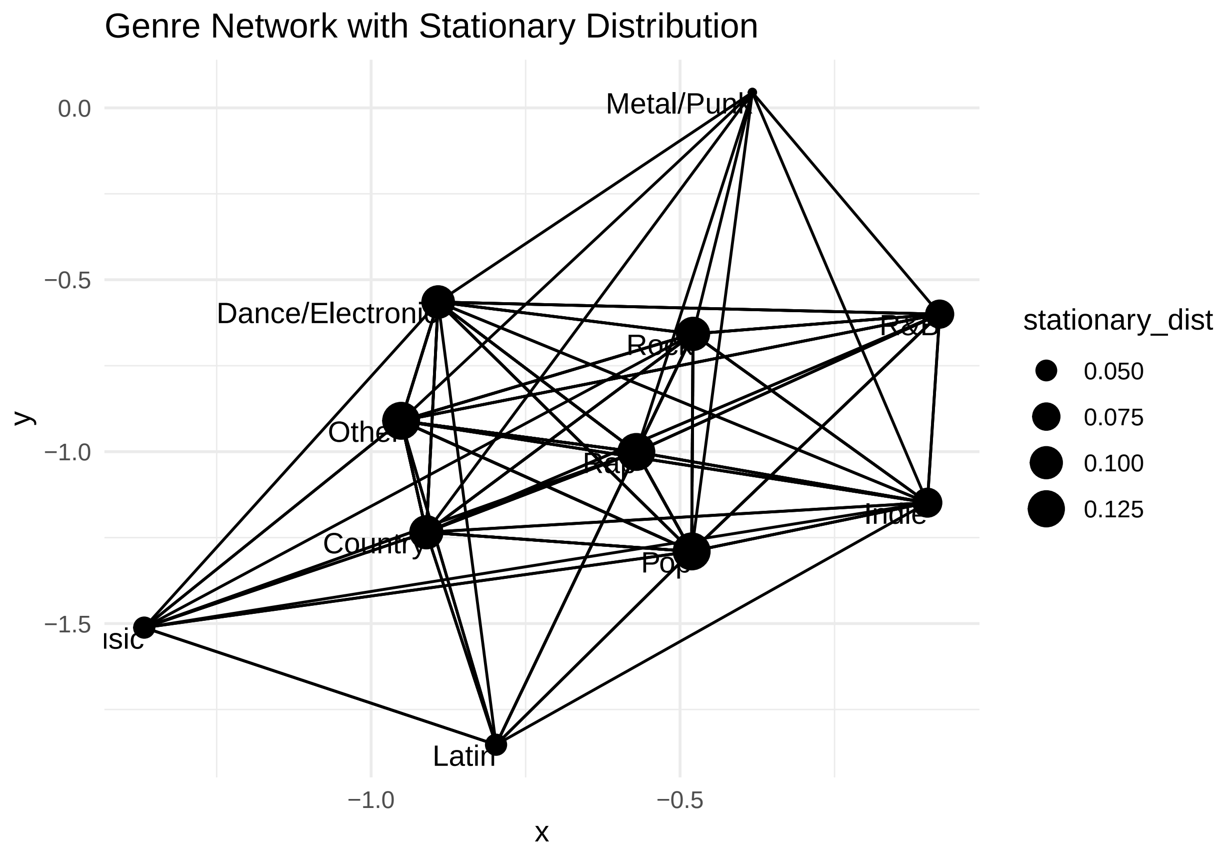
```r
  if (length(genres_in_playlist) > 1) {
    for (i in 1:(length(genres_in_playlist) - 1)) {
      for (j in (i + 1):length(genres_in_playlist)) {
        edges <- rbind(edges, data.frame(from = genres_in_playlist[i], to = genres_in_playlist[j]))
      }
    }
  }
}
g <- graph_from_data_frame(edges, directed = FALSE)
```

We can compute centrality to ascertain which genres are most essential in the network, indicating their significance.

```r
transition_matrix <- as_adjacency_matrix(g, sparse = FALSE)
transition_matrix <- transition_matrix / rowSums(transition_matrix)

# Compute the stationary distribution using eigenvalue method
eigen_result <- eigen(t(transition_matrix))

# Extract the eigenvector corresponding to eigenvalue 1 (stationary distribution)
stationary_dist <- abs(eigen_result$vectors[,1])
stationary_dist <- stationary_dist / sum(stationary_dist)  # Normalize
names(stationary_dist) <- V(g)$name

set.seed(123)
# Print the stationary distribution
print(stationary_dist)
```

```
##              Rap           Other             Pop             R&B
##       0.12992126      0.12992126      0.12992126      0.07874016
##            Indie Dance/Electronic            Rock      Metal/Punk
##       0.08267717      0.10236220      0.10629921      0.03149606
##      World Music           Latin         Country
##       0.05118110      0.05118110      0.10629921
```

```r
ggraph(g, layout = "fr") +
  geom_edge_link() +
  geom_node_point(aes(size = stationary_dist)) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_minimal() +
  ggtitle("Genre Network with Stationary Distribution")
```



In most force-directed architectures, such as Fruchterman-Reingold, nodes with multiple connections are often situated in proximity to one another. In the graph, genres that regularly co-occur in playlists will be positioned in proximity to one another. Genres that are infrequently associated or share limited playlists with others are positioned further from center clusters, making them visually distinguishable as more specialist genres. The size of each node in the graph reflect its stationary distribution value. Larger nodes signify genres that are more "central" or commonly found in playlists.

The genres "Rock," "Pop," "Rap," and "Other" form a compact cluster near the center of the graph. The proximity indicates a strong connection between them. Another cluster may be seen with "Pop" in its center, surrounded by "Rap", "Rock", "R&B", "Dance", "Country" and "Other." This suggests that Pop is the most participatory genre within this cluster, rendering it one of today's most flexible genres.

Genres positioned at the periphery of the network, characterized by smaller nodes such as "Metal/Punk," "World Music," "Indie," and "Latin," have lower connectivity. They may signify narrow or specialized genres that seldom share playlists with other categories.

Question 5: Artist Collaboration Network

In this section of the report, we will examine the collaboration between the artists and who possesses the most collaborations throughout their songs. Therefore, we can investigate by building a collaboration network and finding out top artists having the most edges with orthe artists.

```r
# Loading the required libraries for the task
library(corrplot)
library(purrr)
```

```
##
```

```
## Attaching package: 'purrr'

## The following object is masked from 'package:jsonlite':
##
##     flatten

## The following objects are masked from 'package:igraph':
##
##     compose, simplify
```

```r
library(future)
```

```
##
## Attaching package: 'future'

## The following objects are masked from 'package:igraph':
##
##     %->%, %<-%
```

```r
library(future.apply)
```

Data collection

We first define an artist that we concentrate on. His id could be taken by the last part of URL when we access to that artist on Spotify platform. The function *get_collaborators* has input as the track and the artist name and return the list of collaborator artists in that track (beside the main artist). The second function *build_collaboration_network* is built to conduct a process: collect that artist albums, get the list of tracks of each album and use *get_collaborators* function to get collaborative artists for further network building. The available function *future_lapply* in R is a parallel computing function that enables to efficiently execute functions over elements of a list or vector across multiple cores or machines to efficiently deal with multiple API calls.

```r
plan(multisession)  # Use parallel processing

# Optimized function to get collaborators from a track
get_collaborators <- function(track, artist_name) {
  # Use lapply to extract collaborator names while avoiding loops
  collaborators <- unlist(lapply(track[[1]], function(artist) artist$name[artist$name != artist_name]))
  return(unique(collaborators))
}

# Build a network of artist collaborations
build_collaboration_network <- function(artist_id) {
  # Cache artist name
  artist_name <- get_artist(artist_id)$name

  # Fetch albums and limit the number of albums if necessary (limit can vary)
  albums <- get_artist_albums(artist_id, include_groups = 'album', limit = 10)

  if (length(albums$id) == 0) {
    return(NA)
  }
  # Parallel processing to get collaborators across albums
  collaborators <- future_lapply(albums$id, function(album) {
    album_tracks <- get_album_tracks(album)
    get_collaborators(album_tracks, artist_name)
  })

  # Flatten and get unique collaborators
  collaborators <- unique(unlist(collaborators))

  # Return only the first 20 collaborators
  collaborators <- collaborators[1:min(20, length(collaborators))]

  if (length(collaborators) == 0) {
    return(NA)
  }

  return(collaborators)
}
```

```r
# Define first artist
artist_id <- "2RdwBSPQiwcmiDo9kixcl8"

# Get the first layer of collaborators
collaborator_1 <- build_collaboration_network(artist_id)

# Get artist name
artist_name <- get_artist(artist_id)$name

# Fetch more collaborators for each of the first-level collaborators
more_collaborators_id <- map(collaborator_1, function(collab_name) {
  search_spotify(collab_name, type="artist")$id[1]
})

# Parallel processing with rate-limiting and seed control
more_collaborators <- future_lapply(more_collaborators_id, function(collab_id) {
  Sys.sleep(1)  # Add delay to prevent hitting rate limits
  build_collaboration_network(collab_id)
}, future.seed = TRUE)
```

```
## Request failed [429]. Retrying in 2 seconds...

## Request failed [429]. Retrying in 1 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".

## Request failed [429]. Retrying in 2 seconds...
## Request failed [429]. Retrying in 1 seconds...
## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".

## Request failed [429]. Retrying in 2 seconds...
## Request failed [429]. Retrying in 1 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".

## Request failed [429]. Retrying in 2 seconds...
## Request failed [429]. Retrying in 1 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".

## Request failed [429]. Retrying in 2 seconds...
## Request failed [429]. Retrying in 1 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".

## Request failed [429]. Retrying in 1 seconds...
## Request failed [429]. Retrying in 1 seconds...

## Request failed [429]. Retrying in 2 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".
## Request failed [429]. Retrying in 2 seconds...

## Request failed [429]. Retrying in 1 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".

## Request failed [429]. Retrying in 2 seconds...
## Request failed [429]. Retrying in 1 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".

## Request failed [429]. Retrying in 2 seconds...
## Request failed [429]. Retrying in 1 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".

## Request failed [429]. Retrying in 2 seconds...
## Request failed [429]. Retrying in 1 seconds...

## Warning: UNRELIABLE VALUE: One of the 'future.apply' iterations
## ('future_lapply-1') unexpectedly generated random numbers without declaring so.
## There is a risk that those random numbers are not statistically sound and the
## overall results might be invalid. To fix this, specify 'future.seed=TRUE'. This
## ensures that proper, parallel-safe random numbers are produced via the
## L'Ecuyer-CMRG method. To disable this check, use 'future.seed = NULL', or set
## option 'future.rng.onMisuse' to "ignore".
```

```r
# View collaborative artists of artist1
print("Collaborative artists of artist1")
```

```
## [1] "Collaborative artists of artist1"
```

```r
print(collaborator_1)
```

```
##  [1] "Princess Anne High School Fabulous Marching Cavaliers"
##  [2] "Tyler, The Creator"
##  [3] "N.E.R.D"
##  [4] "Justin Timberlake"
##  [5] "Wreckx-N-Effect"
##  [6] "N.O.R.E."
##  [7] "JAY-Z"
##  [8] "No Doubt"
##  [9] "Snoop Dogg"
## [10] "Clipse"
## [11] "Uncle Charlie Wilson"
## [12] "Daft Punk"
## [13] "Nile Rodgers"
## [14] "Hans Zimmer"
## [15] "Benjamin Wallfisch"
## [16] "Alicia Keys"
## [17] "Gwen Stefani"
## [18] "Jamie Cullum"
## [19] "Slim Thug"
## [20] "Pusha T"
```

```r
# View collaborative artists of artist1's collaborators
print("Collaborative artists of artist1's collaborators")
```

```
## [1] "Collaborative artists of artist1's collaborators"
```

```r
print(more_collaborators)
```

```
## [[1]]
## [1] NA
##
## [[2]]
##  [1] "DJ Drama"                 "42 Dugg"
##  [3] "YoungBoy Never Broke Again" "Ty Dolla $ign"
##  [5] "Lil Wayne"                "Teezo Touchdown"
##  [7] "Domo Genesis"             "Brent Faiyaz"
##  [9] "Fana Hues"                "DAISY WORLD"
## [11] "Lil Uzi Vert"             "Pharrell Williams"
## [13] "Vince Staples"            "A$AP Rocky"
## [15] "Rex Orange County"        "Frank Ocean"
## [17] "Kali Uchis"               "Jaden"
## [19] "Estelle"                  "Anna of the North"
##
## [[3]]
##  [1] "Rihanna"           "Pharrell Williams" "Gucci Mane"
##  [4] "Wale"              "Future"            "Kendrick Lamar"
##  [7] "André 3000"        "M.I.A."            "Ed Sheeran"
## [10] "T.I."              "Nelly Furtado"     "Kanye West"
## [13] "Lupe Fiasco"       "Pusha T"           "Benji Madden"
## [16] "Joel Madden"       "Lenny Kravitz"     "Vita"
## [19] "Lee Harvey"        "Kelis"
##
## [[4]]
##  [1] "Fireboy DML"        "Tobe Nwigwe"        "*NSYNC"
##  [4] "Alicia Keys"        "Chris Stapleton"    "Drake"
##  [7] "JAY-Z"              "Dirty Vegas"        "Tiësto"
## [10] "Basement Jaxx"      "Paul Oakenfold"     "Hani"
## [13] "Steve Angello"      "Sebastian Ingrosso" "Timbaland"
## [16] "John Clark"         "Linus Loves"        "Justice"
## [19] "Sébastien Léger"    "Sander Kleinenberg"
##
## [[5]]
## [1] "DJ Kool"
##
## [[6]]
##  [1] "Fabolous"          "Fat Joe"           "The-Dream"
##  [4] "Yung Reallie"      "Sevyn Streeter"    "Phokus"
##  [7] "Jadakiss"          "Wyclef Jean"       "Kent Jones"
## [10] "Pharrell Williams" "Royal Flush"       "Nature"
## [13] "Kool G Rap"        "Aziz Ansari"       "DJ Khaled"
## [16] "Manolo Rose"       "French Montana"    "Noah"
## [19] "PnB Rock"          "Capone"
##
## [[7]]
##  [1] "Gloria Carter"         "Frank Ocean"           "Beyoncé"
##  [4] "Damian Marley"         "Justin Timberlake"     "Rick Ross"
##  [7] "Kanye West"            "The-Dream"             "Otis Redding"
## [10] "Mr Hudson"             "Curtis Mayfield"       "Eminem"
## [13] "Faith Evans"           "The Notorious B.I.G." "Dr. Dre"
## [16] "Rakim"                 "Truth Hurts"           "Luke Steele"
## [19] "Rihanna"               "Alicia Keys"
##
## [[8]]
## [1] "Bounty Killer" "Lady Saw"      "Busy Signal"   "Major Lazer"
##
## [[9]]
##  [1] "Jaywillz"                "Bosco Soundtrack"
##  [3] "MyStylez"                "Anyta"
##  [5] "JeMarcus Bridges"        "Dave East"
##  [7] "WHOISTEVENYOUNG"         "Xtofa"
##  [9] "Allie McIntosh"          "Kg3"
## [11] "Georgie Leahy"           "ErrolSpace"
## [13] "RJMrLA"                  "Sofila"
## [15] "Jay Millian"             "Reekado Banks"
## [17] "Yinka Bernie"            "Tatiana Ladymay Mayfield"
## [19] "Yellopain"               "Quawntay 'Bosco' Adams"
##
## [[10]]
##  [1] "Cam'ron"           "Pharrell Williams"  "Kanye West"
##  [4] "Yo Gotti"          "Ab-Liva"            "Keri Hilson"
##  [7] "Nicole Hurst"      "Slim Thug"          "Ab Liva"
## [10] "Re-Up Gang"        "Bilal"              "Rosco P. Coldchain"
## [13] "Faith Evans"       "Fam-Lay"            "Fabolous"
## [16] "Jermaine Dupri"    "Jadakiss"           "Style P"
## [19] "N.O.R.E."           "Birdman"
##
## [[11]]
## [1] NA
##
## [[12]]
##  [1] "Julian Casablancas"     "Pharrell Williams"      "Paul Williams"
##  [4] "Nile Rodgers"           "Todd Edwards"           "Panda Bear"
##  [7] "The Voidz"              "The Glitch Mob"         "M83 VS Big Black Delta"
## [10] "M83"                    "Big Black Delta"        "The Crystal Method"
## [13] "Teddybears"             "Ki:Theory"              "Paul Oakenfold"
## [16] "Moby"                   "Boys Noize"             "Kaskade"
## [19] "Com Truise"             "Tame Impala"
##
## [[13]]
##  [1] "CHIC"              "Mura Masa"        "Cosha"            "VIC MENSA"
##  [5] "Nao"               "Craig David"      "Stefflon Don"     "LunchMoney Lewis"
##  [9] "Hailee Steinfeld" "Philippe Saisse"  "Elton John"       "Emeli Sandé"
## [13] "Lady Gaga"
##
## [[14]]
##  [1] "Klaus Badelt"             "Lisa Gerrard"
##  [3] "Gavin Greenaway"          "The Lyndhurst Orchestra"
##  [5] "Djivan Gasparyan"         "Omer Benyamin"
##  [7] "Steven Doar"              "Kara Talve"
##  [9] "Steve Mazzaro"            "Jacob Shea"
## [11] "Sara Barone"              "Bastille"
## [13] "Anže Rozman"              "Andrew James Christie"
## [15] "David Fleming"
##
## [[15]]
##  [1] "Eurielle"          "Benjamin Botkin"   "NCSOUND"
##  [4] "Saulius Petreikis" "Jang-Won Lee"      "Michal Cielecki"
##  [7] "Max Cameron"       "Judah & the Lion"  "Nate Ruess"
## [10] "Chris Egan"
##
## [[16]]
##  [1] "Maleah Joi Moon"                    "Shoshana Bean"
##  [3] "Chris Lee"                          "Hell's Kitchen Cast"
##  [5] "Vanessa Ferguson"                   "Jackie Leon"
##  [7] "Brandon Victor Dixon"               "Kecia Lewis"
##  [9] "Tony! Toni! Toné!"                  "Jermaine Paul"
## [11] "Lellow"                             "Queen Charlotte's Global Orchestra"
## [13] "DJ Min One"                         "Hani"
## [15] "IZA"                                "Kris Bowers"
## [17] "Tayla Parx"                         "Caleb Chan"
## [19] "Brian Chan"                         "Vitamin String Quartet"
##
## [[17]]
##  [1] "Blake Shelton"     "Mon Laferte"       "Eve"
##  [4] "Johnny Vulture"    "André 3000"        "Fetty Wap"
##  [7] "Akon"              "Pharrell Williams" "Diplo"
## [10] "Elan"              "Tony Kanal"
##
```

```
## [[18]]
## [1] "Kansas Smitty’s"   "The Vernon Spring" "Lady Blackbird"
## [4] "Robbie Williams"   "Michel Legrand"    "Laura Mvula"
## [7] "Gregory Porter"    "Roots Manuva"
##
## [[19]]
##  [1] "Phill Wade"       "OTB Fastlane"     "Z-Ro"              "Lil’ Keke"
##  [5] "Swishahouse"      "DJ Michael Watts" "Le$"               "Lenora"
##  [9] "DJ X.O."          "Killa Kyleon"     "Propain"           "Mr. Lee"
## [13] "Scarface"         "Yung Al"          "Le$ Mug"
##
## [[20]]
##  [1] "Kanye West"        "Kid Cudi"           "Nigo"
##  [4] "No Malice"         "Clipse"             "Labrinth"
##  [7] "Lil Uzi Vert"      "Don Toliver"        "JAY-Z"
## [10] "Pharrell Williams" "Rick Ross"          "The-Dream"
## [13] "A$AP Rocky"        "Ab-Liva"            "Beanie Sigel"
## [16] "Kehlani"           "Jill Scott"         "Chris Brown"
## [19] "Jeezy"             "Kevin Cossom"
```

```r
# Note: Please ignore the warnings within the output as they don't affect the result of the task
```

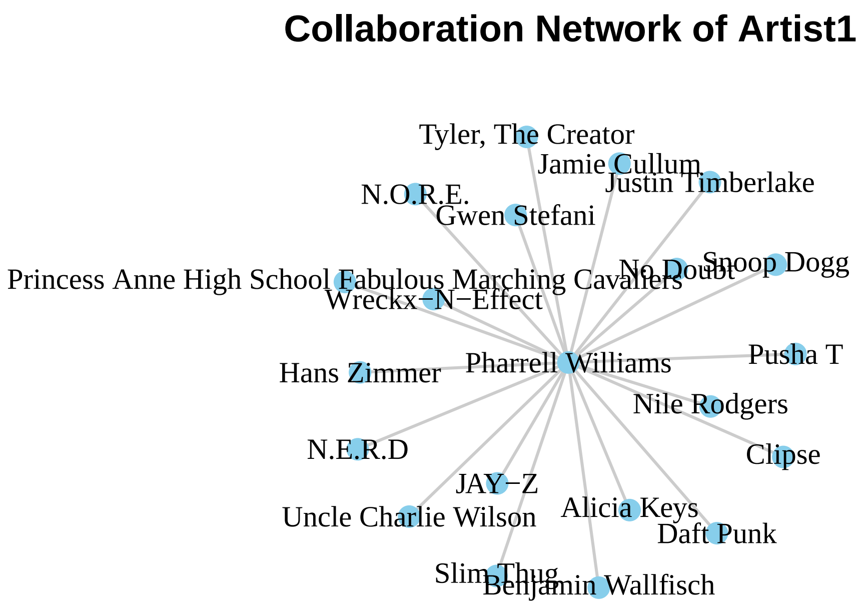
Methodology

To building network, the matrix is created by replicating the name of artists at first column and his collaborators at the second one.

```r
# Combine artist and collaborators into a single data frame
artist1 <- cbind(rep(artist_name, length(collaborator_1)), collaborator_1)

# Use lapply to create the artist collaboration network
artist1_collab <- lapply(1:length(collaborator_1), function(i) {
  name_rep <- rep(collaborator_1[i], length(more_collaborators[[i]]))
  cbind(name_rep, more_collaborators[[i]])
})

# Combine all collaborations into a single matrix
artist1_more <- do.call(rbind, artist1_collab)

# Filter out rows with NA values and convert to character in one step
artist1 <- apply(artist1[!is.na(artist1[, 2]), ], 2, as.character)
artist_total <- apply(artist1_more[!is.na(artist1_more[, 2]), ], 2, as.character)
```

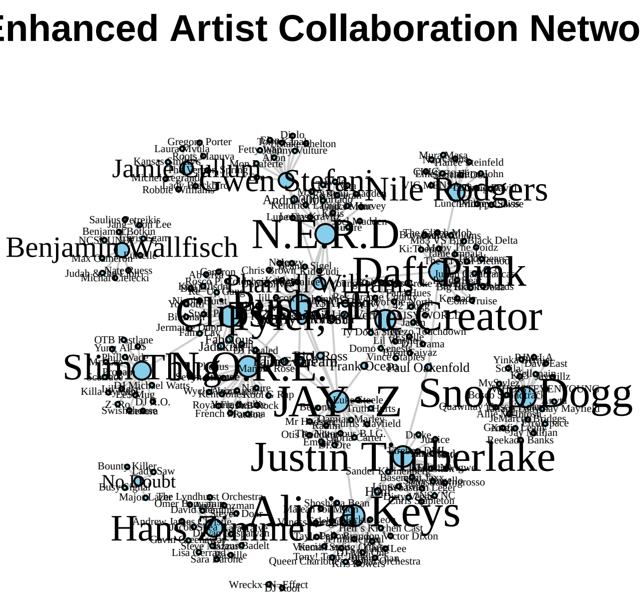
Building collaborations network of the artist1 base on the created matrix. The given output show 20 artists collaborating with artist1 throughout his tracks on Spotify.

```r
# Create graph from edgelist
g1 <- graph_from_edgelist(artist1, directed = FALSE)

# Visualize artist1 collaboration network
plot(
  g1,
  layout = layout_with_fr,             # Fruchterman-Reingold layout
  vertex.size = 10,                    # Increase vertex size for visibility
  vertex.color = "skyblue",             # Set vertex color to a more aesthetic shade
  vertex.label.color = "black",        # Label color
  vertex.label.cex = 0.9,              # Label size
  vertex.frame.color = NA,             # Remove borders around vertices
  edge.width = 1.5,                    # Thicker edges for better clarity
  edge.color = "gray80",               # Set edge color to a light gray
  edge.arrow.size = 0.5,               # Set arrow size
  main = "Collaboration Network of Artist1",  # Add title to the plot
)
```



The expand network is built below to show more collaborations. The label size of each vertex depends on the number of connection with other artists, meaning the bigger the artist label is, the more collaborations that artist has.

```
# Create graph from edgelist
g <- graph_from_edgelist(artist_total, directed = FALSE)

# Visualize artist1 collaborators' network
plot(
  g,
  layout = layout_with_fr,              # Fruchterman-Reingold layout
  vertex.size = sqrt(degree(g))*2,        # Scale vertex size with degree
  vertex.label.cex = sqrt(degree(g)) * 0.3,  # Scale label size with degree
  vertex.color = "skyblue",             # Set vertex color to a more aesthetic shade
  vertex.label.color = "black",        # Label color
  edge.color = "gray80",                # Light gray edges for aesthetics
  edge.arrow.size = 0.5,                # Set arrow size for directed edges
  main = "Enhanced Artist Collaboration Network",  # Add a main title
)
```



Degree centrality

```
sort(degree(g), decreasing=TRUE)[1:15]
```

```
##        Alicia Keys                JAY-Z             Pusha T  Justin Timberlake
##                 22                   22                  21                 21
##           N.O.R.E.               Clipse Tyler, The Creator             N.E.R.D
##                 21                   21                  20                 20
##         Snoop Dogg            Daft Punk           Slim Thug        Hans Zimmer
##                 20                   20                  16                 15
##        Nile Rodgers         Gwen Stefani Benjamin Wallfisch
##                 14                   11                  10
```

- Degree centraility counts the number of connection a node has in the network. In order words, nodes with high degree centrality are often influentials in that network. According to the degree centrality **Alicia Keys, JAY-Z, Pusha T, Justin Timberlake, N.O.R.E., Clipse** are the most central collaborators, meaning they collaborated with the most other artists.

Result Interpretation

- To sum up, they are the most interconnected artists in this collaboration network, meaning they played a significant role in connecting other artists and are likely to have had many musical collaborations.

Project Limitations

This research possesses multiple limitations. First, we consolidated Spotify's varied genre classifications into more general categories, potentially masking the intricacies of sub-genres. The analysis relies on only five playlists, constraining its scope and possibly overlooking wider trends across the site. Third, genres were allocated according to performers instead of specific tracks, perhaps failing to encompass genre diversity. The study is a temporal snapshot, and playlist dynamics may evolve, necessitating additional longitudinal investigation.

The Spotify API is also a big limitation for this project because it does not allow access to much user information. A deep analysis of consumer behavior and their network on Spotify therefore can not be implemented efficiently. Instead, it focuses on researching the relationships between public artists on this platform. Moreover, using spotifyr also faces API Rate Limits, which means that it imposes strict rate limits on API requests, preventing excessive usage that could strain their servers. Hence, processing large amounts of data through requests to APIs is time-sensitive or even limited in functionality for a period of time.

In terms of Song Similarity Network visualization, while clustering provides a useful framework for understanding relationships among songs based on audio features, the interpretability of these clusters can be limited without additional context or descriptive information. Each cluster represents a group of songs that share similar audio characteristics, such as energy, tempo, and acousticness. However, without further insight into the specific criteria used to form these clusters—such as musical genre, mood, or lyrical themes—users may find it challenging to derive actionable insights. To be more specific, if Cluster 1 contains energetic tracks but lacks a clear description indicating that these songs are predominantly from the electronic genre, decision-makers might overlook the nuances that differentiate these songs from others in the dataset. Therefore, providing richer descriptions or metadata for each cluster could significantly enhance the interpretability and usefulness of the analysis, allowing users to make more informed decisions based on the clustering results.