The Top 100 Songs on Spotify in 2019

Serena Everett (sle759) and Anika Pal (ap52553)

Introduction

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──
```

```
## ✓ ggplot2 3.3.5     ✓ purrr   0.3.4
## ✓ tibble  3.1.6     ✓ dplyr   1.0.8
## ✓ tidyr   1.1.4     ✓ stringr 1.4.0
## ✓ readr   2.1.2     ✓ forcats 0.5.1
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(readr)
Spotify_2010_2019_Top_100 <- read_csv("~/Downloads/archive (1)/Spotify 2010 - 2019 Top 1
00.csv")
```

```
## Rows: 1000 Columns: 17
```

```
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (5): title, artist, top genre, added, artist type
## dbl (12): year released, bpm, nrgy, dnce, dB, live, val, dur, acous, spch, p...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
View(Spotify_2010_2019_Top_100)
head(Spotify_2010_2019_Top_100) #first six observations of the dataset
```

```
## # A tibble: 6 × 17
##   title   artist `top genre` `year released` added   bpm  nrgy  dnce    dB  live
##   <chr>   <chr>  <chr>                 <dbl> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 STARST… 3OH!3  dance pop              2009 2022…   140    81    61    -6    23
## 2 My Fir… 3OH!3  dance pop              2010 2022…   138    89    68    -4    36
## 3 I Need… Aloe … pop soul               2010 2022…    95    48    84    -7     9
## 4 Airpla… B.o.B  atl hip hop            2010 2022…    93    87    66    -4     4
## 5 Nothin… B.o.B  atl hip hop            2010 2022…   104    85    69    -6     9
## 6 Magic … B.o.B  atl hip hop            2010 2022…    82    93    55    -4    35
## # … with 7 more variables: val <dbl>, dur <dbl>, acous <dbl>, spch <dbl>,
## #   pop <dbl>, top year <dbl>, artist type <chr>
```

```
spotify <- Spotify_2010_2019_Top_100 #renaming the imported dataset
spotify_2019 <- spotify %>%
  select(-added, -live, -acous, -spch, -`year released`, -val, -artist) %>% #removing va
riables we do not need
  filter(`top year` == "2019") #filter for 2019
```

The dataset we have chosen displays the top 100 songs across different genres for the years 2010 through 2019. We will use the year 2019 in this project because that year is very recent, so we know most of the songs. This dataset is interesting to us because we both use Spotify to listen to music and love using the app. This dataset was also found using the Kaggle website, where free datasets are available to the public. Source: (https://www.kaggle.com/datasets/muhmores/spotify-top-100-songs-of-20152019?resource=download (https://www.kaggle.com/datasets/muhmores/spotify-top-100-songs-of-20152019?resource=download))

There are 1000 observations in the dataset and 17 variables in the dataset. The variables are: title of the song (title), artist of the song (artist), genre of the song (genre), the year the song was released (year released), the day the song was added to the Top Hits (added), beats per minute (bpm), how energetic the song is (nrgy), how easy it is to dance to the song (dnce), decibel (dB), how likely the song is a live recording (live), how positive the mood of the song is (val), duration (dur), how acoustic the song is (acous), the more the song is focused on spoken word (spch), popularity of song (pop), year the song was a top hit (top year), and if it is a solo artist or group (artist type). Because there are so many variables, we removed 7 variables (acous, live, spch, added, artist, val, and year released) we did not think were necessary for the project and kept the other 10.

Some possible relationships we expect are that the variables energy, danceability, and song popularity will have a positive relationship. This is because if the song is easier to dance to and is more energetic, the song may be more popular than others. There was no joining or tidying needed for this dataset as it was already tidy when we found it.

After importing the csv file into R, we renamed the dataset “spotify” so it would be easier to perform future functions. We saved the dataset with only 2019 songs as “spotify_2019”.

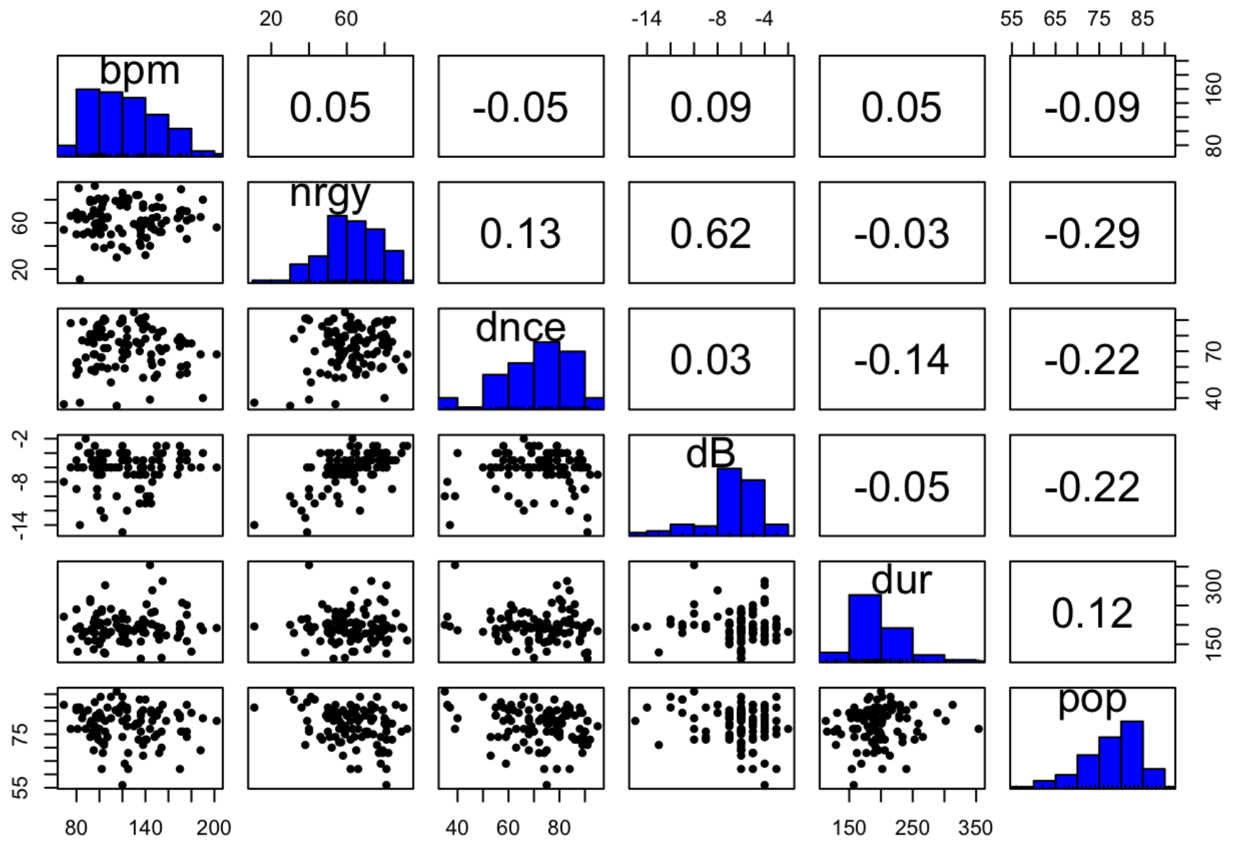
Exploratory Data Analysis

```
spotify_num <- spotify_2019 %>%
  select_if(is.numeric) %>% #selects only the numeric variables for the new dataset
  select(-`top year`) #removes top year
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
pairs.panels(spotify_num,
             method = "pearson", # correlation coefficient method
             hist.col = "blue", # color of histogram
             smooth = FALSE, density = FALSE, ellipses = FALSE)
```



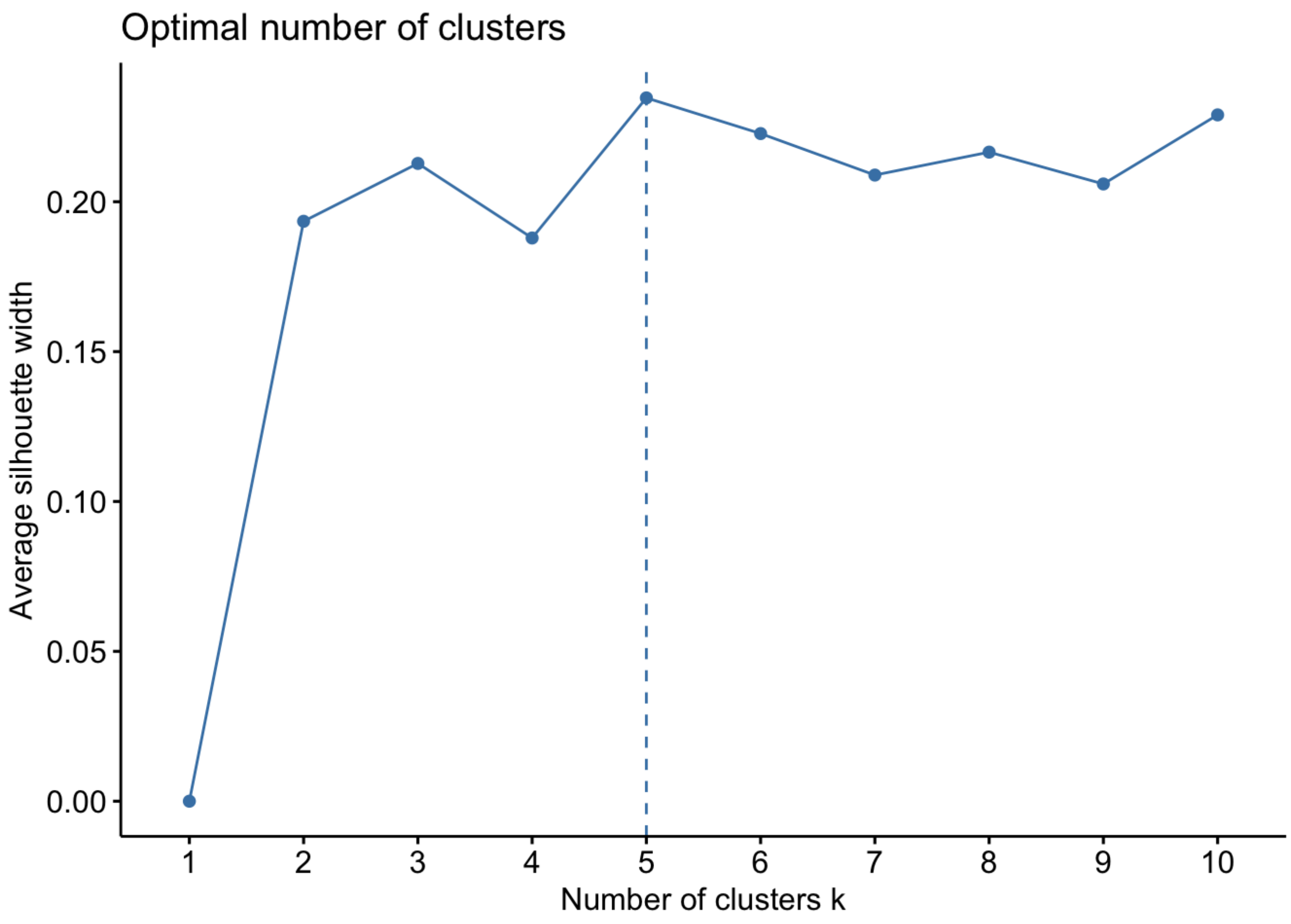
The two variables that are the most correlated are nrgy and dB with a correlation coefficient of 0.62. The two variables that are least correlated at dB and dnce with a correlation coefficient of 0.03. There does not appear to be any relationships or trends that are apparent from the correlation matrix as there are no strong correlation coefficients. The top.year will appear as NA as a correlation and have a single point on the bottom graphs because the year is 2019 for all 100 songs. Therefore, top.year needs to be removed after we utilize it to filter for 2019 songs.

Clustering

```
library(tidyverse)
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WB
a
```

```
library(cluster)

spotify_scaled <- spotify_num %>%
  select(nrgy, dnce, dur, pop) %>% #selecting 4 variables from the dataset with only num
eric variables
  scale #scale because some might have diff values, units, etc by subtracting mean and d
ividing by std dev

fviz_nbclust(spotify_scaled, kmeans, method = "silhouette") #number of clusters based on
silhouette width
```



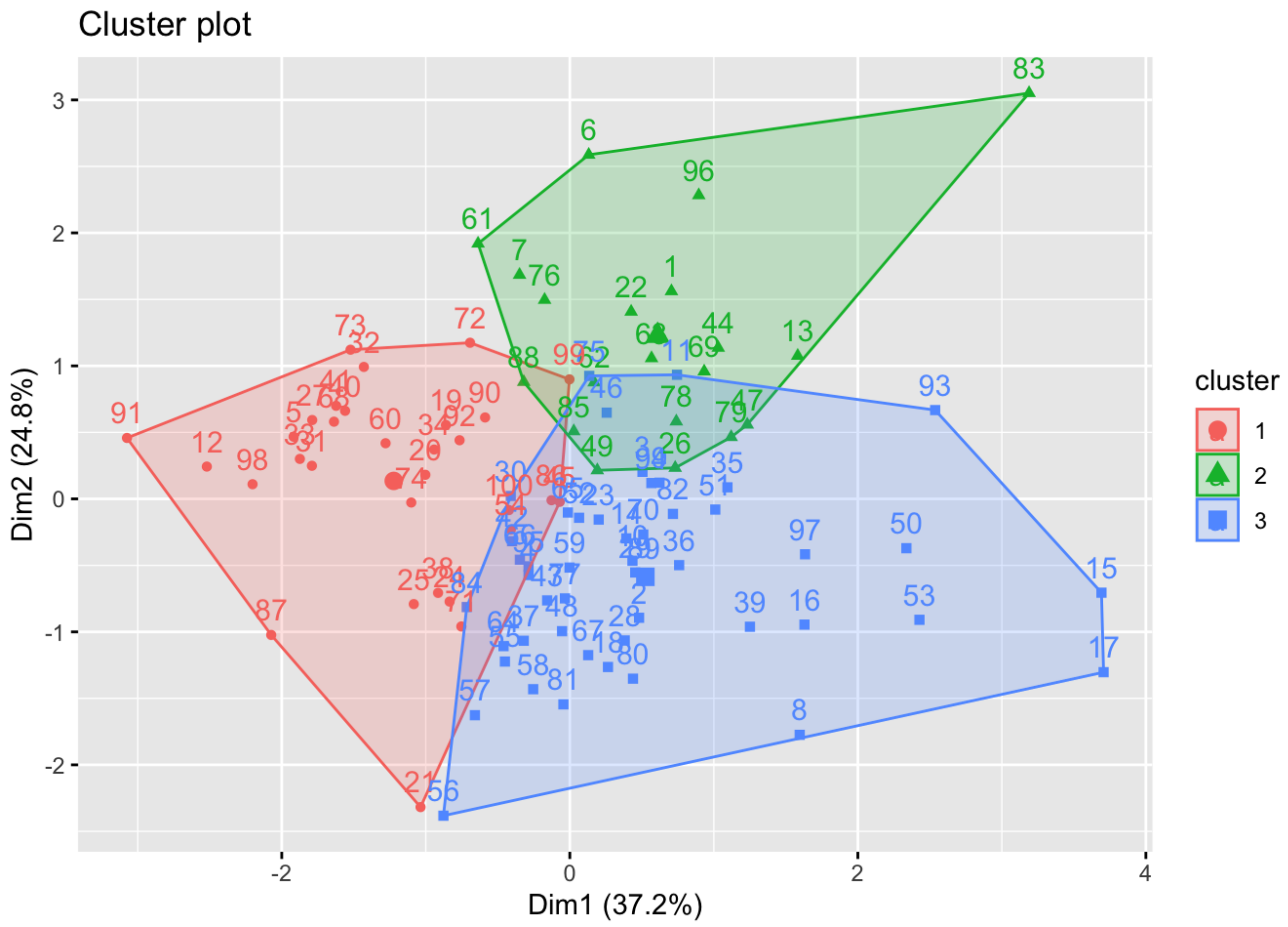
```
kmeans_results <- spotify_scaled %>% #saves the kmeans results
  kmeans(3) #using the optimal number of cluster via kmeans
kmeans_results
```

```
## K-means clustering with 3 clusters of sizes 31, 20, 49
## 
## Cluster means:
##         nrgy       dnce        dur        pop
## 1  0.5399412  0.3536828 -0.3305408 -1.0771059
## 2  0.1817982 -0.1129722  1.4584353  0.2845131
## 3 -0.4157988 -0.1776474 -0.3861620  0.5653065
## 
## Clustering vector:
##   [1] 2 3 3 3 1 2 2 3 3 3 3 1 2 3 3 3 3 3 1 1 1 2 3 1 1 2 1 3 3 3 1 1 1 1 3 3 3
##  [38] 1 3 1 1 3 3 2 1 3 2 3 2 3 3 3 3 1 3 3 3 3 3 1 2 2 2 3 3 3 3 1 2 3 1 1 1 1
##  [75] 3 2 3 2 2 3 3 3 2 3 2 1 1 2 3 1 1 1 3 3 3 2 3 1 1 1
## 
## Within cluster sum of squares by cluster:
## [1]  75.48990  55.92731 134.25573
##  (between_SS / total_SS =  32.9 %)
## 
## Available components:
## 
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
spotify_kmeans <- spotify_scaled %>%
  as.data.frame() %>% #dataset with kmeans results
  mutate(cluster = as.factor(kmeans_results$cluster)) #created a cluster column in datas
et
head(spotify_kmeans)
```

```
##          nrgy       dnce        dur         pop cluster
## 1  0.08117187  0.8553606  2.3510982  0.67720051       2
## 2 -1.12634354 -1.1986786 -0.9429538 -0.65645476       3
## 3  0.21534025 -1.0519615 -0.1634453  0.23264875       3
## 4 -0.25424908  0.4885679 -0.4148997 -0.06371909       3
## 5  1.42285565  1.2955119 -0.1131545 -0.80463868       1
## 6  1.22160308  0.4885679  2.6779889  0.23264875       2
```

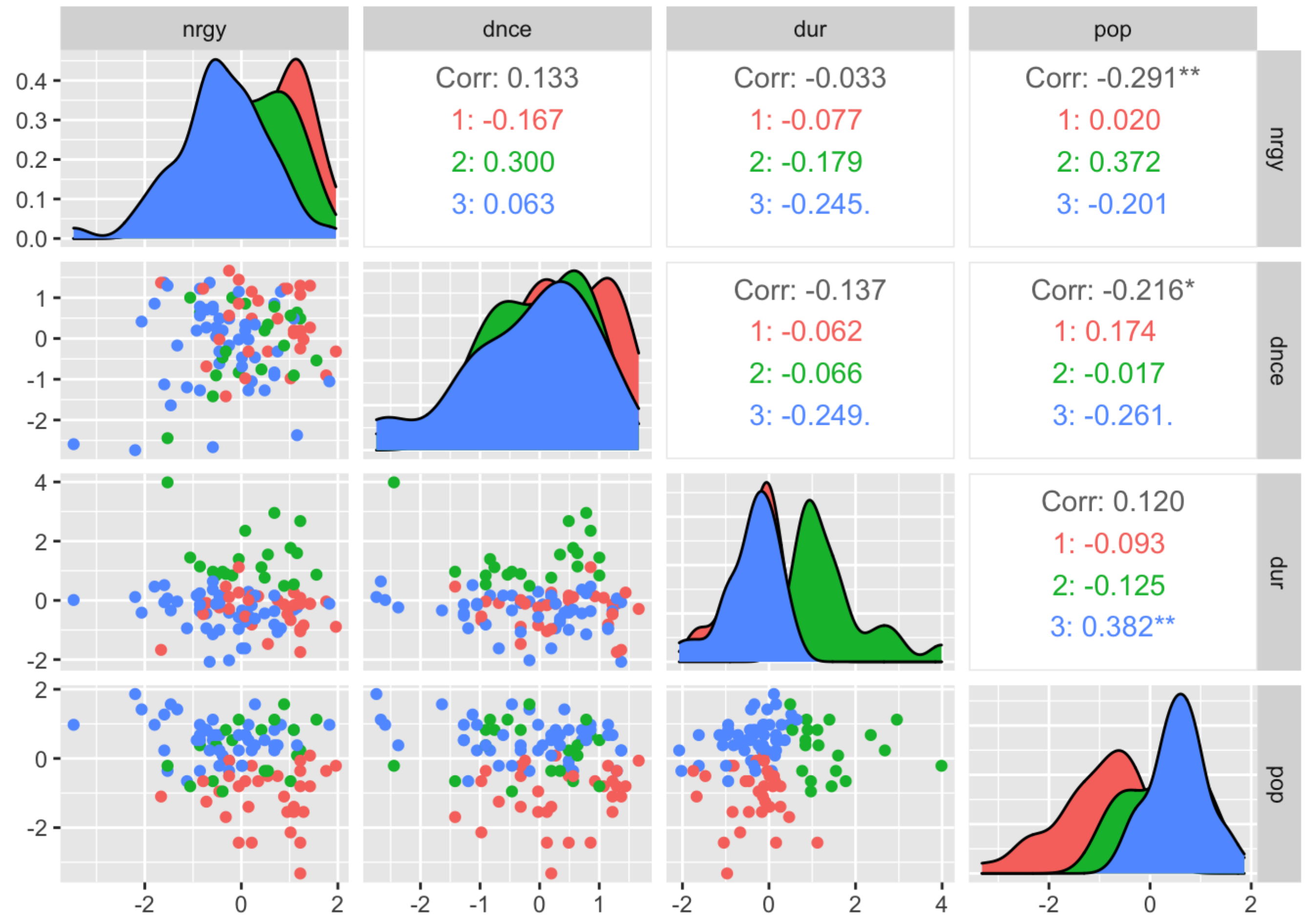
```
fviz_cluster(kmeans_results, data = spotify_scaled) #visualize our data by final cluster
assignment by fviz cluster
```



```
library(GGally)
```

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

```
ggpairs(spotify_kmeans, columns = 1:4, aes(color = cluster)) #visualize our data by fina
l cluster assignment by ggpairs
```



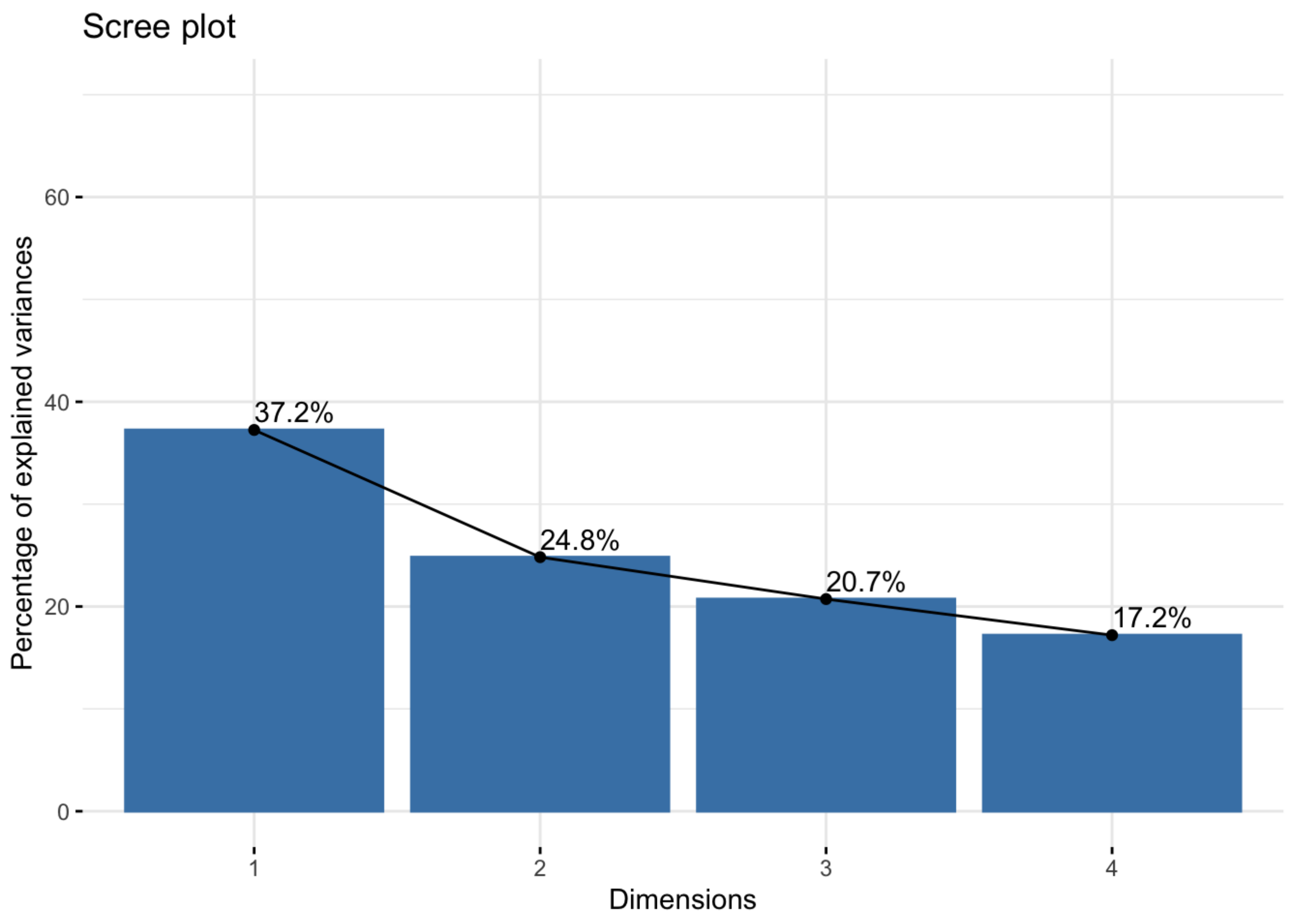
The spotify_num dataset, created in the previous section, needed to be scaled and was named spotify_scaled. After scaling the dataset, we found the optimal number of clusters using the silhouette width. Instead of using the optimal number of 5 clusters, Professor Guyot told us to use 3 clusters to help visualize the data better since it was close in silhouette width and would be easier to interpret. The average silhouette width was around 0.25; this is not a high average, which indicates no substantial structure has been found. There are 3 clusters with sizes of 31, 20, and 49. We then mutated a column to help us identify which cluster each song was in. After creating a cluster plot, we see there is overlap in the center of the graph, which could mean that certain observations belong to more than one cluster. Therefore, the clustering is not very effective in separating the different songs. With kmeans, the means of each variable represent which observation is at the center. This can be seen with the dnce variable, as cluster 2 and 3 have similar means of -0.1129722 and -0.1776474. In addition, the dur variable has the same issue as cluster 1 and 3 have similar means of -0.3305408 and -0.3861620. There might be some outliers in each group as well; for example, the green cluster extends so far out for the 83rd song. Even though the clustering fit is poor, the mean from each cluster provides an explanation for why they are clustered this way. For example, the mean for nrgy for each cluster is 0.5399412, 0.1817982, and -0.4157988, which are all different means. The mean for pop for each cluster are also different with -1.0771059, 0.2845131, and 0.5653065. We then used a ggpairs plot to compare the correlation coefficients with the cluster distributions. None of the correlation coefficients were very strong between each of the variables. The positive values indicate it is above the overall mean, and the negative values indicate that it is below the overall mean.

Dimensionality Reduction

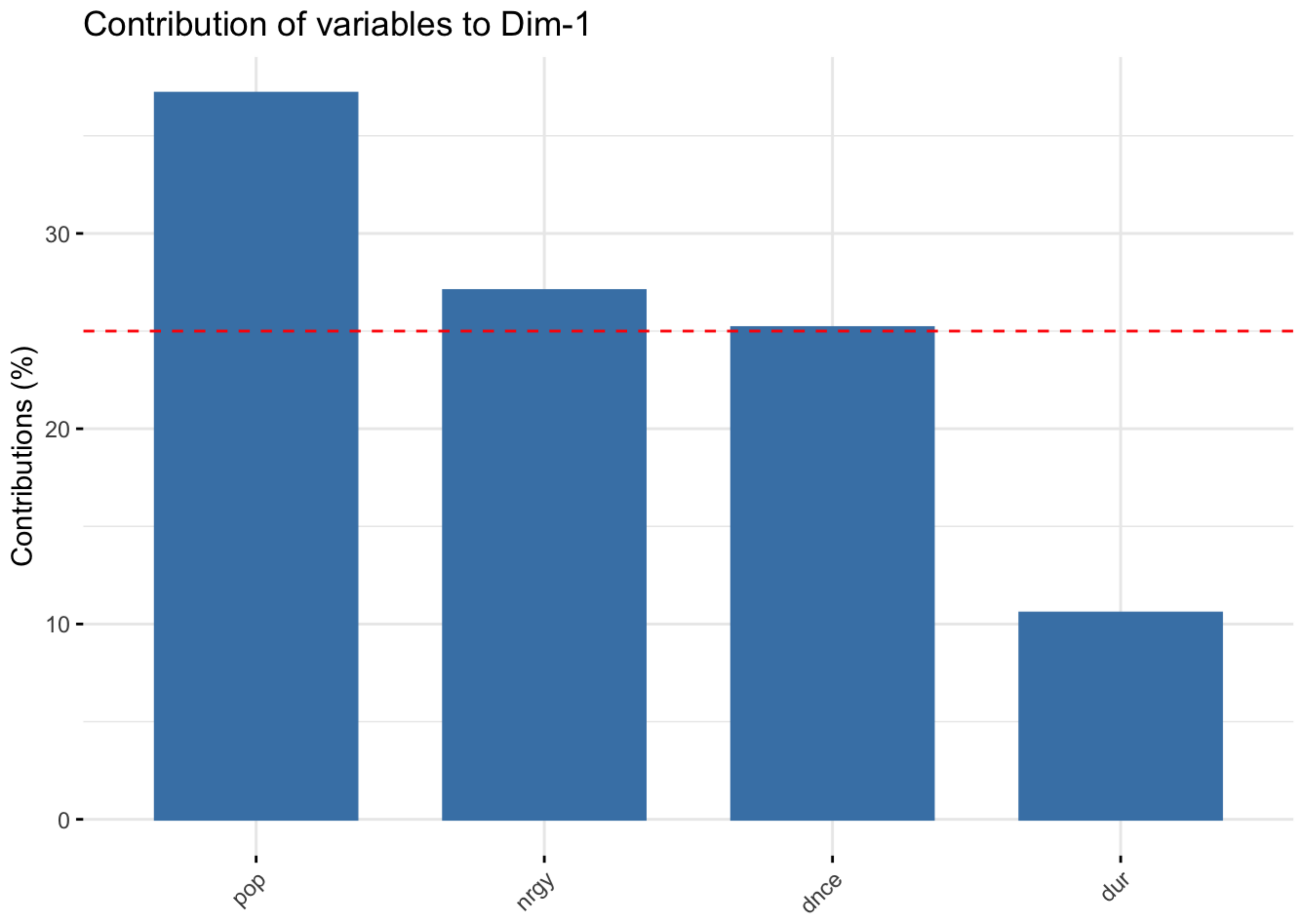
```
pca <- spotify_scaled %>% #finds the principle components
  prcomp()

get_eigenvalue(pca) #gives us number of principle components
```

```
##       eigenvalue variance.percent cumulative.variance.percent
## Dim.1  1.4899495         37.24874                    37.24874
## Dim.2  0.9928738         24.82185                    62.07058
## Dim.3  0.8291173         20.72793                    82.79851
## Dim.4  0.6880594         17.20149                   100.00000
```

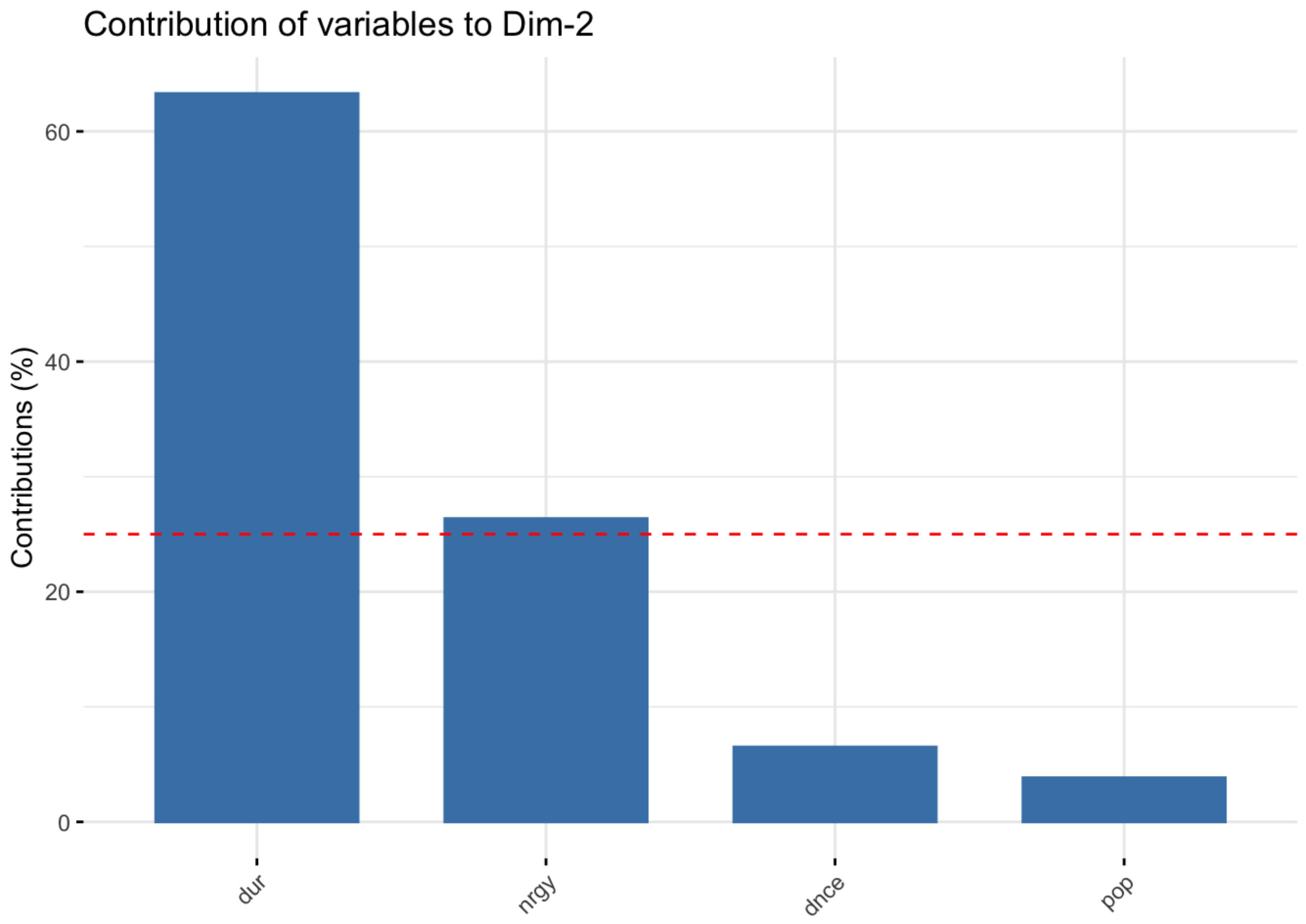
```
fviz_eig(pca, addlabels = TRUE, ylim = c(0, 70)) #visualize how much each PC contributes
```



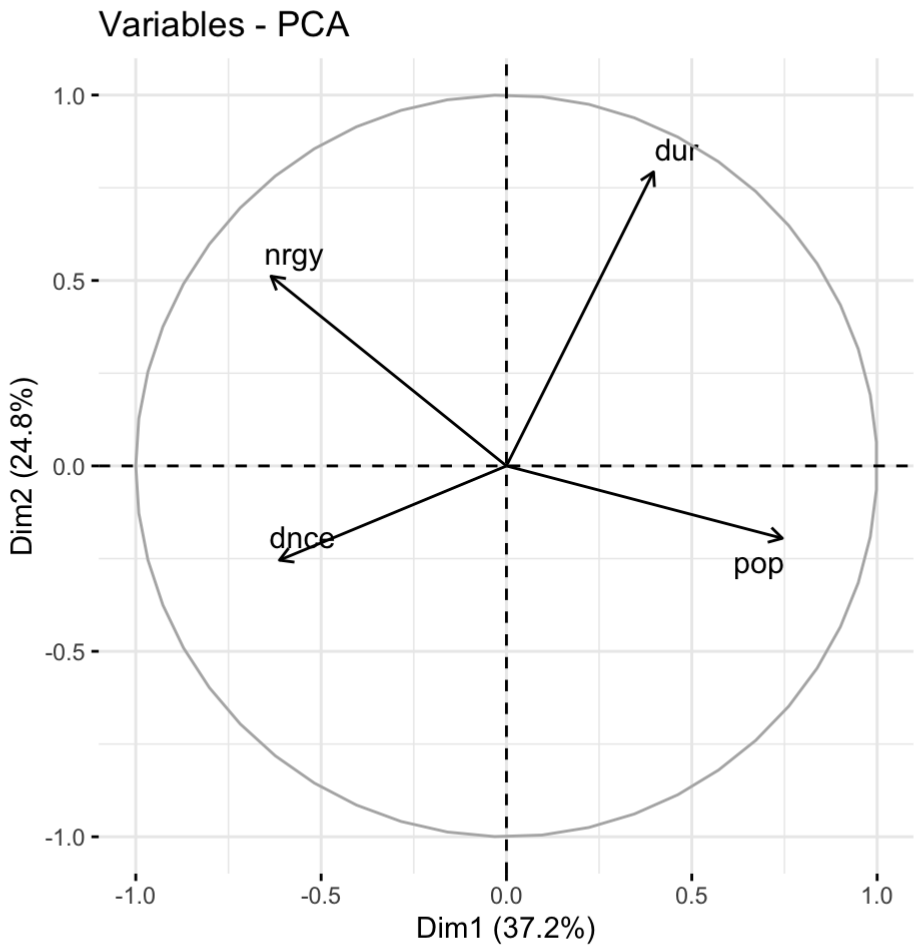
```
fviz_contrib(pca, choice = "var", axes = 1, top = 5) #how each variable contributed to P
C1
```



```
fviz_contrib(pca, choice = "var", axes = 2, top = 5) #how each variable contributed to P
C2
```



```
fviz_pca_var(pca, col.var = "black",
             repel = TRUE) # avoid text overlapping, plot for PC1 and PC2 together
```



We performed a PCA on four of our variables from the spotify_scaled dataset, and we used a prcomp() function to find the principal components. The eigenvalue function gives us the number of principal components. When looking at the PCs, we want to look at the first few that give us a total variance of roughly 80%. This means that we would use the first 3 PCs because the cumulative variance is 82%. This variance can be visualized by the scree plot. For PC1, we see that the variables contribute in order of greatest to least by pop, nrgy, dnce, and then dur. For PC2, we see that the variables continue in order of greatest to least by dur, nrgy, dnce, and then pop. We then created a graph to visualize all the variables in PC1 and PC2. For PC1, we see that pop and dur are positively correlated and nrgy and dnce are negatively correlated. Pop and nrgy are also further away from the center than dnce and dur, which means they do contribute more variance. For PC2, we see that nrgy and dur are positively correlated while dnce and pop are negatively correlated. Dur and nrgy are also significantly more further away from the center than dnce and pop, which means they do contribute more variance. Therefore, each PC has different variables that contribute the most and least.

Classification and Cross-Validation

```
library(tidyverse)
library(plotROC)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

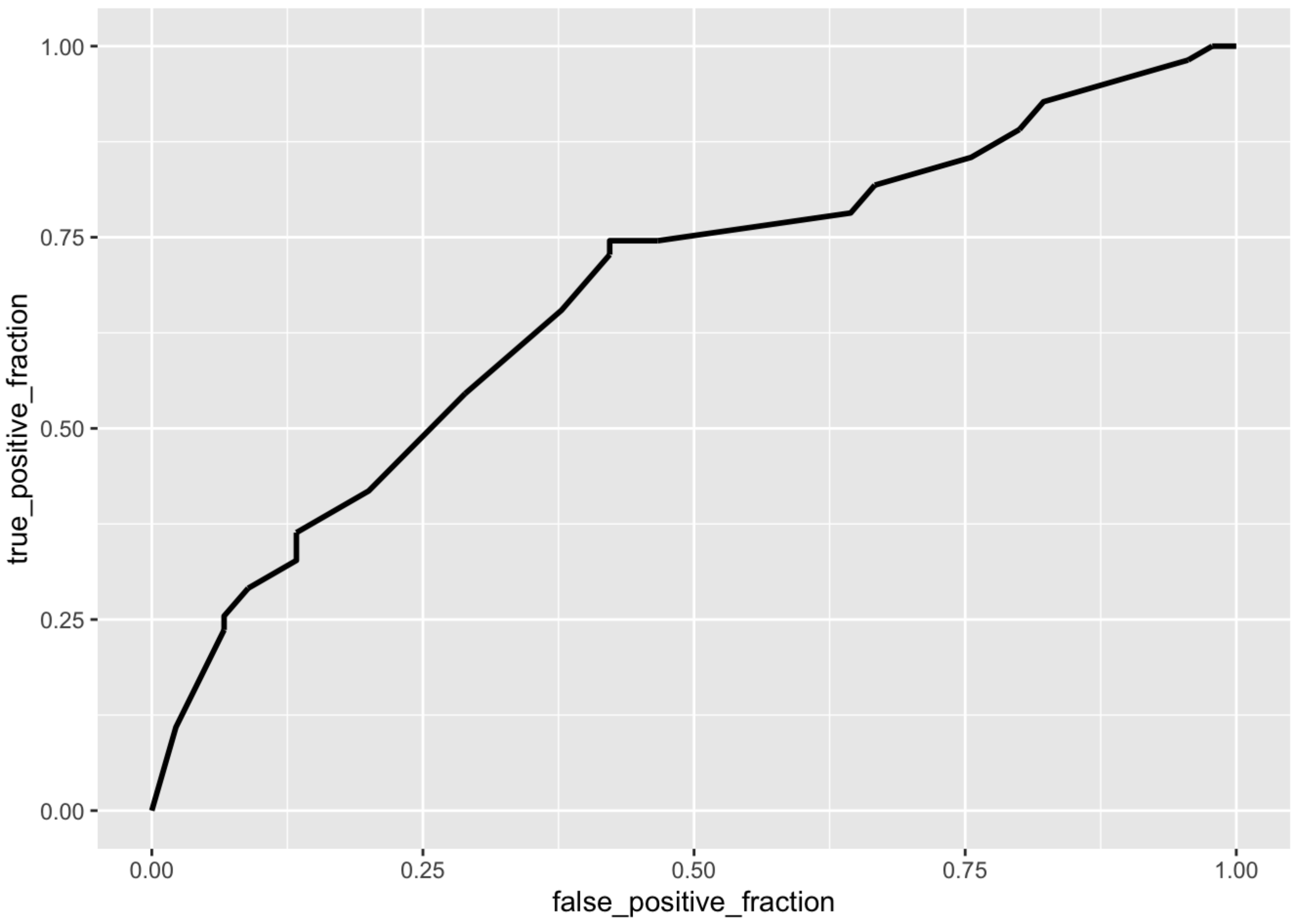
```
spotify_2019 %>%
  summarize(mean(dnce)) #find the mean danceability
```

```
## # A tibble: 1 × 1
##   `mean(dnce)`
##          <dbl>
## 1         72.3
```

```
s_2019 <- spotify_2019 %>%
  mutate(danceability = ifelse(dnce >72, 1,0)) #find if each observation falls above or
 below the mean

# kNN with k = 5
knn_fit <- knn3(factor(danceability == 1, #positivity is when danceability is set to 1
                       levels = c("TRUE","FALSE")) ~ nrgy, #depends on nrgy
                data = s_2019,
                k = 5)

# Calculate a proportion of nearest neighbors danceability
kNN_spotify <- s_2019 %>%
  mutate(proportion = predict(knn_fit, s_2019)[,1], #keep 1st column
         predicted = ifelse(proportion > 0.5, 1, 0)) %>%
  # Give a name to the rows
  rownames_to_column("Title") %>%
  select(title, nrgy, danceability, proportion, predicted)
head(kNN_spotify)
```

```
## # A tibble: 6 × 5
##   title                    nrgy danceability proportion predicted
##   <chr>                   <dbl>        <dbl>      <dbl>     <dbl>
## 1 a lot                      64            1      0.333         0
## 2 Easier                     46            0      0.667         1
## 3 Swervin (feat. 6ix9ine)    66            0      0.375         0
## 4 Look Back at It            59            1      0.571         1
## 5 Ladbroke Grove             84            1      0.7           1
## 6 China                      81            1      0.833         1
```

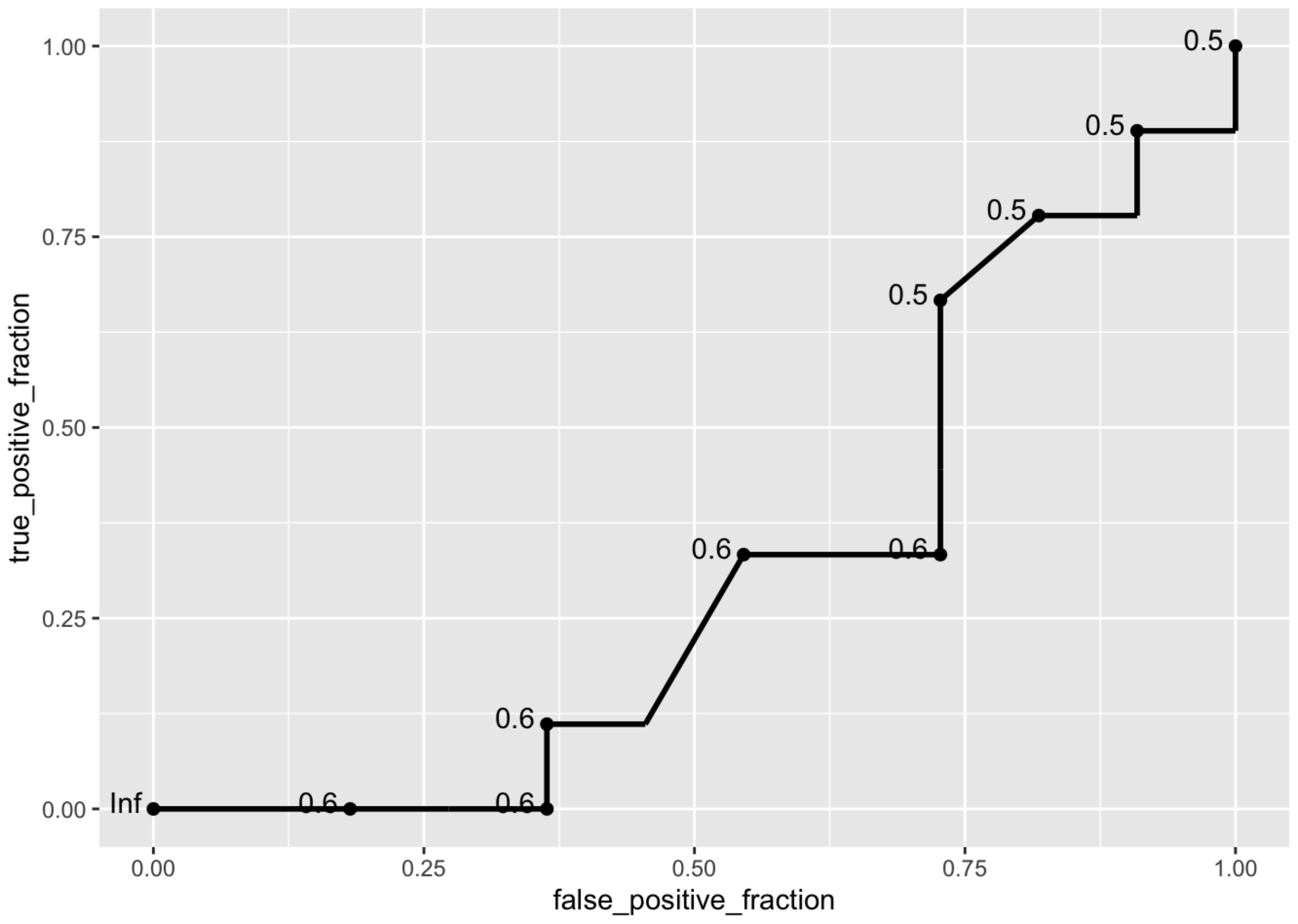
```
ROC <- ggplot(kNN_spotify) + #create ROC curve for kNN_spotify
  geom_roc(aes(d = danceability, m = proportion), n.cuts = 0)
ROC
```



```
calc_auc(ROC) #calculate the auc
```

```
##   PANEL group       AUC
## 1     1    -1 0.6731313
```

```
#k-fold cross-validation
k = 5

# Randomly order rows in the dataset
data <- s_2019[sample(nrow(s_2019)), ]

# Create k folds from the dataset
folds <- cut(seq(1:nrow(data)), breaks = k, labels = FALSE)

# Use a for loop to get diagnostics for each test set
diags_k <- NULL

for(i in 1:k){
  # Create training and test sets
  train <- data[folds != i, ] # all observations except in fold i
  test <- data[folds == i, ]  # observations in fold i

  # Train model on training set (all but fold i)
  fit <- glm(danceability ~ nrgy, data = s_2019, family = "binomial")

  # Test model on test set (fold i)
 df <- data.frame(
    probability = predict(fit, newdata = test, type = "response"),
    y = test$danceability)

  # Consider the ROC curve for the test dataset
  ROC1 <- ggplot(df) +
    geom_roc(aes(d = y, m = probability))

  # Get diagnostics for fold i (AUC)
  diags_k[i] <- calc_auc(ROC1)$AUC
}

ROC1 #displays ROC curve
```



```
# Average performance
mean(diags_k)
```

```
## [1] 0.4950712
```

Using the spotify_2019 dataset, we found the mean of the danceability variable (dnce), and then we split the data based on the mean of 72. If an observation for dnce fell above the mean of 72, it was classified as 1. If it was below 72, it was classified as 0. This was used to create a new dataset called s_2019. Using k nearest neighbors functions, we said that if danceability is equal to 1 then it is a positive value and would show up as TRUE. Anything else would show up as FALSE. This was done with a kNN of 5. Then we created a proportion, called kNN_spotify, of the nearest neighbors with danceability and classified anything above 50% as 1 and anything below as 0. We then created a ROC curve based on the kNN_spotify proportion. The ROC curve was poor since it had a value of 0.673.

We then performed a k-fold cross-validation with the same classifier and used 5 folds. This means the dataset was split into 5 parts. One part was considered the training set while the other part was the test set. This was repeated 5 times, so each part had been used once as a test set and the average performance was found after the 5 tests were performed. There is a noticeable decrease as the previous model had an AUC of 0.673 and the average five-fold AUC is 0.4950712. The ROC curve also looks worse when comparing them visually. Therefore, the model does show signs of overfitting.

Formatting

The references for our dataset can be found in the introduction section of the report. For this project, both of us worked together on each section by scheduling time to work together based on our availability. We were able to work more efficiently this way by working through errors quickly and ensuring we were both satisfied with the work. Overall, we had a great time working on this project together!