Decoding Popularity: A Data-Driven Analysis of Hit Songs on Spotify

Uchenna Ekeh, Lauren Ellis, Orhan Gozutok, and Ava Meissner

May 2024

Introduction

In the dynamic landscape of the music industry, achieving success requires a comprehensive understanding of the intricate elements that contribute to a song's popularity. Artists, songwriters, and record labels alike are continuously challenged to decipher the formula that distinguishes a hit song from the vast array of musical compositions. This pursuit is fueled by the inherent variability observed among songs within the top music charts, prompting a fundamental question: What constitutes a hit song? Our endeavor is to delve into this inquiry and leverage analysis of recent hit songs as a means to unearth invaluable insights and discern recurring patterns. By analyzing the characteristics of successful compositions, we endeavor to unravel the web of factors that wield influence over music popularity. Through this examination, we aspire to shed light on the determinants of musical success and form recommendations for those in the music industry to create and release a hit song.

The dataset for our project is a comprehensive list on Kaggle of nine hundred and forty-three of the most famous songs of 2023 as listed on Spotify, using the charts system. There are twenty-four variables in total. Six specify details of the song including track name, artist name(s), artist count, release year, release month, and release day. Eight quantitative variables denote the track's success on streaming platforms with number of streams on Spotify as well as number of playlists the song is in, and presence and rank of the song on top charts for Apple Music, Spotify, Deezer, and Shazam. The remaining variables are audio features. Mode of song (major or minor) and key are qualitative. Beats per minute, danceability percentage, valence percentage, energy percentage, acoustic percentage, instrumental content percentage, liveliness percentage, and speech percentage are all quantitative and measured by Spotify's audio intelligence.

Data Cleaning

```
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
```

```
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.
0.0 ──
## ✓ forcats   1.0.0     ✓ readr     2.1.4
## ✓ ggplot2   3.4.3     ✓ stringr   1.5.0
## ✓ lubridate 1.9.2     ✓ tibble    3.2.1
## ✓ purrr     1.0.1     ✓ tidyr     1.3.0
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflict
s() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors
```

```
library(ggcorrplot)
library(tree)
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:ggplot2':
##
##     margin
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
spotify <- read.csv("spotify-2023.csv")
spotify$streams <- as.numeric(spotify$streams)
```

```
## Warning: NAs introduced by coercion
```

```
spotify$numeric_key <- as.numeric(factor(spotify$key))
spotify$in_shazam_charts  <- as.numeric(spotify$in_shazam_charts)
```

```
## Warning: NAs introduced by coercion
```

```
spotify$in_deezer_playlists  <- as.numeric(spotify$in_deezer_playlists)
```

```
## Warning: NAs introduced by coercion
```

```r
spotify<- na.omit(spotify)
summary(spotify)
```

```
##   track_name        artist.s._name      artist_count   released_year
##  Length:829         Length:829         Min.   :1.000   Min.   :1930
##  Class :character   Class :character   1st Qu.:1.000   1st Qu.:2021
##  Mode  :character   Mode  :character   Median :1.000   Median :2022
##                                        Mean   :1.581   Mean   :2019
##                                        3rd Qu.:2.000   3rd Qu.:2022
##                                        Max.   :8.000   Max.   :2023
##  released_month    released_day   in_spotify_playlists in_spotify_charts
##  Min.   : 1.000   Min.   : 1.00   Min.   :   31        Min.   :  0.00
##  1st Qu.: 3.000   1st Qu.: 6.00   1st Qu.:  801        1st Qu.:  0.00
##  Median : 6.000   Median :13.00   Median : 1788        Median :  2.00
##  Mean   : 6.154   Mean   :14.11   Mean   : 3116        Mean   : 10.82
##  3rd Qu.: 9.000   3rd Qu.:22.00   3rd Qu.: 3879        3rd Qu.: 15.00
##  Max.   :12.000   Max.   :31.00   Max.   :29499        Max.   :147.00
##     streams          in_apple_playlists in_apple_charts  in_deezer_playlis
ts
##  Min.   :2.762e+03   Min.   :  0.00     Min.   :  0.00   Min.   :  0.00
##  1st Qu.:1.304e+08   1st Qu.: 11.00     1st Qu.:  6.00   1st Qu.: 12.00
##  Median :2.451e+08   Median : 27.00     Median : 30.00   Median : 32.00
##  Mean   :3.738e+08   Mean   : 49.69     Mean   : 47.15   Mean   : 96.75
##  3rd Qu.:4.817e+08   3rd Qu.: 68.00     3rd Qu.: 80.00   3rd Qu.: 99.00
##  Max.   :2.808e+09   Max.   :492.00     Max.   :275.00   Max.   :974.00
##  in_deezer_charts in_shazam_charts      bpm          key
##  Min.   : 0.000   Min.   :  0.00   Min.   : 65   Length:829
##  1st Qu.: 0.000   1st Qu.:  0.00   1st Qu.:100   Class :character
##  Median : 0.000   Median :  2.00   Median :121   Mode  :character
##  Mean   : 2.341   Mean   : 48.71   Mean   :123
##  3rd Qu.: 1.000   3rd Qu.: 33.00   3rd Qu.:142
##  Max.   :45.000   Max.   :953.00   Max.   :206
##      mode           danceability_.   valence_.        energy_.
##  Length:829         Min.   :23.0   Min.   : 4.00   Min.   :14.00
##  Class :character   1st Qu.:58.0   1st Qu.:32.00   1st Qu.:53.00
##  Mode  :character   Median :70.0   Median :51.00   Median :65.00
##                     Mean   :67.4   Mean   :51.37   Mean   :64.24
##                     3rd Qu.:78.0   3rd Qu.:70.00   3rd Qu.:77.00
##                     Max.   :96.0   Max.   :97.00   Max.   :97.00
##  acousticness_  instrumentalness_   liveness_    speechiness_.
##  Min.   : 0.00   Min.   : 0.000     Min.   : 3.00   Min.   : 2.00
##  1st Qu.: 6.00   1st Qu.: 0.000     1st Qu.:10.00   1st Qu.: 4.00
##  Median :18.00   Median : 0.000     Median :12.00   Median : 6.00
##  Mean   :27.07   Mean   : 1.654     Mean   :18.33   Mean   :10.43
##  3rd Qu.:42.00   3rd Qu.: 0.000     3rd Qu.:23.00   3rd Qu.:12.00
##  Max.   :97.00   Max.   :91.000     Max.   :97.00   Max.   :64.00
##   numeric_key
##  Min.   : 1.000
##  1st Qu.: 4.000
##  Median : 6.000
```

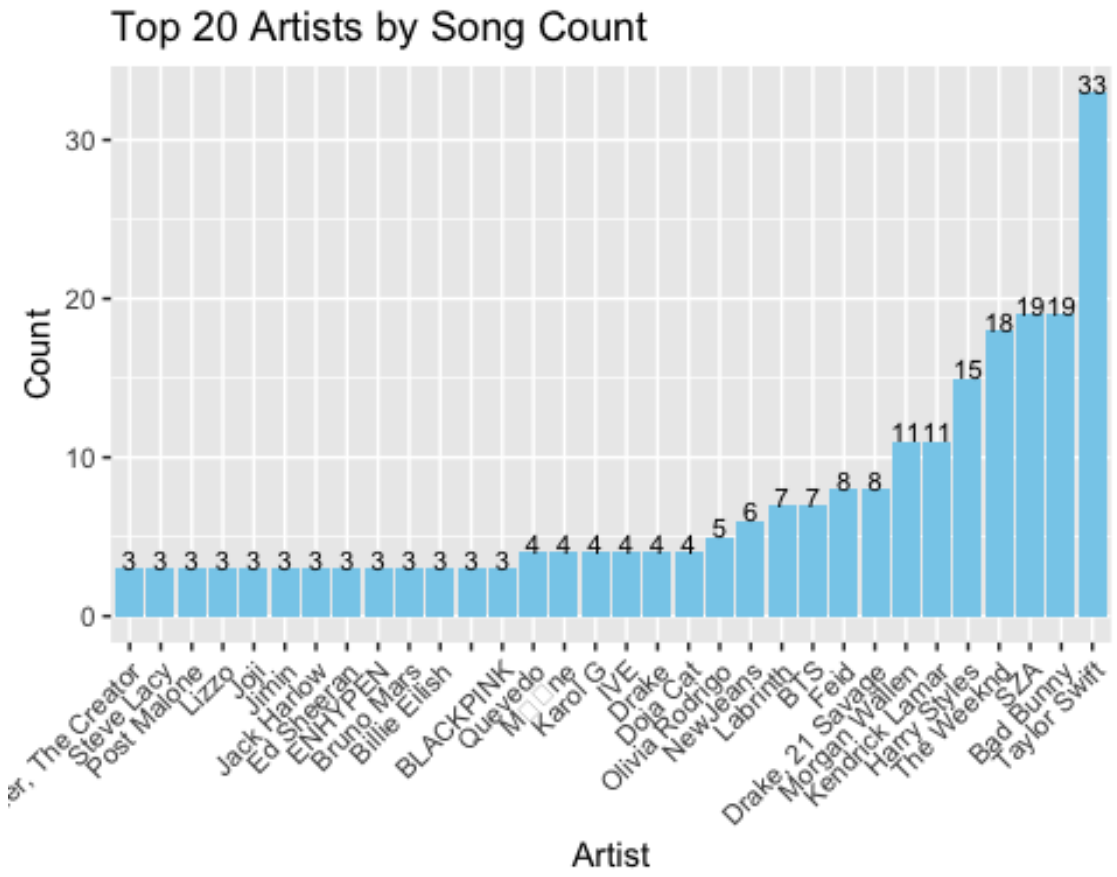
```
##  Mean   : 6.537
##  3rd Qu.:10.000
##  Max.   :12.000
```

In the cleaning of the data, the variables of streams, in_shazam_charts, and in_deezer_playlists were converted to numeric variables as they were previously not. The categorical variable of key was assigned to factor values for efficiency in later modeling. Lastly, all null values were omitted from the data set.

Data Exploration

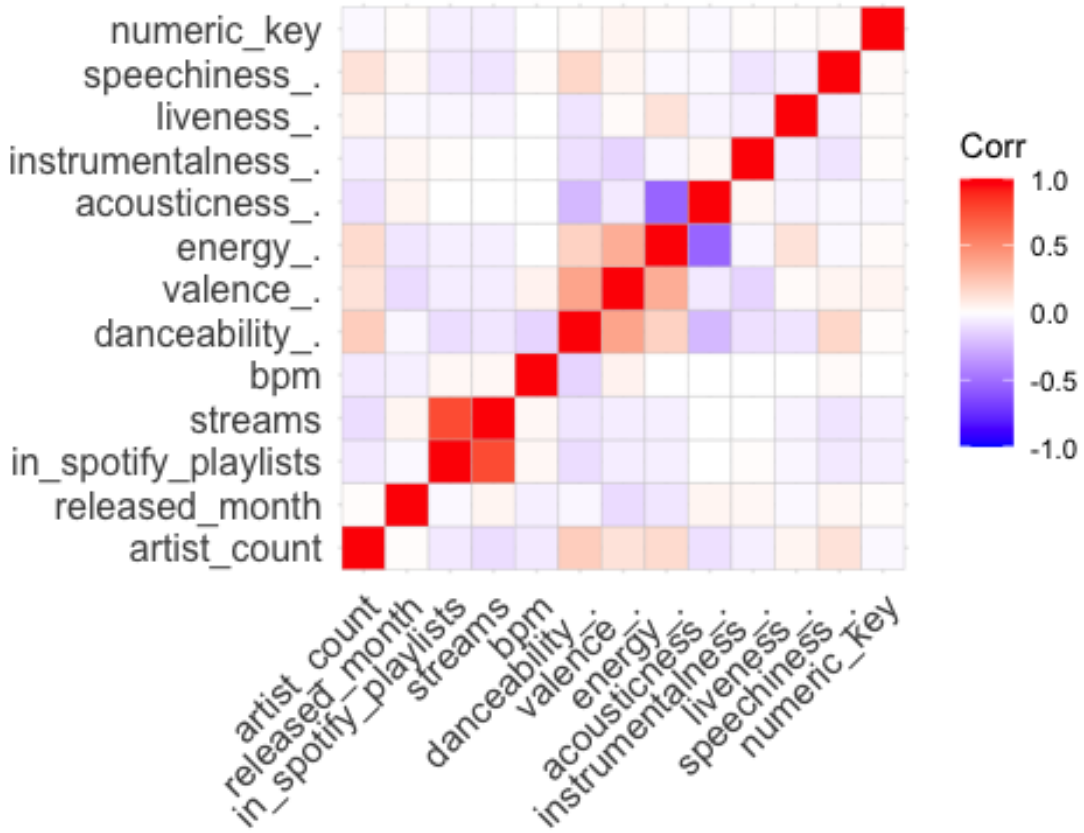
To start, we conducted exploratory analysis to look at some trends amongst the variables.

```
top_artists <- spotify %>%
  count(artist.s._name) %>%
  top_n(20, n)

top_artists <- top_artists %>%
 rename(artist = artist.s._name) %>%
 arrange(desc(n))


top_artists$artist <- factor(top_artists$artist, levels = rev(top_artists$artist))


top_artists %>%
 ggplot(aes(artist, n)) +
 geom_bar(stat = "identity", fill = "skyblue", width = .9) +
 geom_text(aes(label = n), vjust = 0, size = 3) +
 theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
 labs(x = "Artist", y = "Count", title = "Top 20 Artists by Song Count")
```



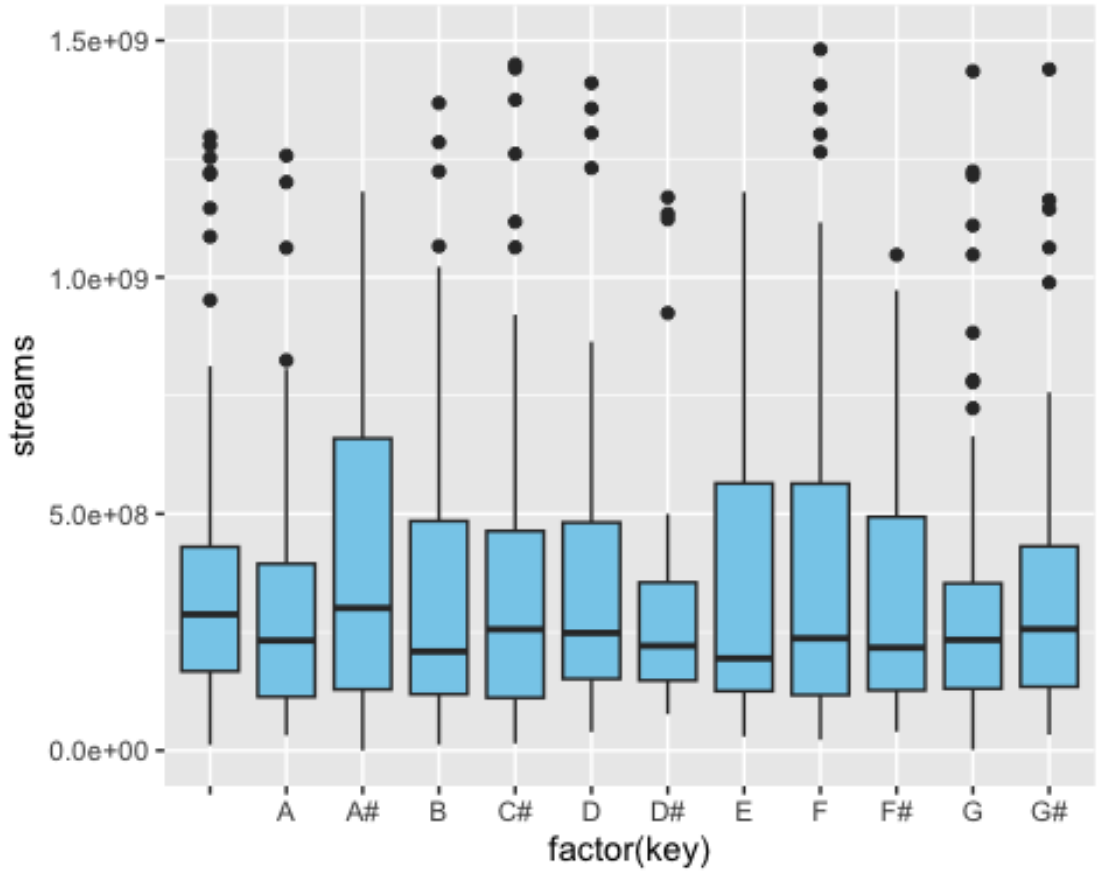
We identified the top twenty artists based on their frequency of appearances on the music charts to identify which artists are most popular among listeners on Spotify and are able to consistently perform well and maintain chart presence. We found that Taylor Swift is the most popular artist, followed by Bad Bunny and SZA. The artist name variable had frequent errors and special characters that R did not recognize. We attempted to replace these characters with underscores, but could not get the transformation to work correctly across the entire dataset. As a result, we excluded the artist name variable from our analysis.

```
spotify2 <- subset(spotify, select = -c(track_name, artist.s._name, in_apple_
playlists, in_apple_charts, in_deezer_playlists, in_deezer_charts, in_shazam_
charts, key, in_spotify_charts, released_day, released_year))
numeric <-  spotify2[,unlist(lapply(spotify2, is.numeric))]
cor <- cor(numeric)
ggcorrplot(cor)
```



We generated a correlation heatmap for the numeric variables of interest in our linear modeling. The majority of correlations were relatively weak.

```
key_streams <- subset(spotify, !is.na(key), select = c(key, streams))
key_streams <- na.omit(key_streams)


ggplot(key_streams, aes(x = factor(key), y = streams)) +
  geom_boxplot(fill = "skyblue") +
  ylim(0, 1500000000)

## Warning: Removed 13 rows containing non-finite values (`stat_boxplot()`).
```




```
  labs(x = "Key", y = "Streams", title = "Distribution of Streams by Key")
```

```
## $x
## [1] "Key"
##
## $y
## [1] "Streams"
##
## $title
## [1] "Distribution of Streams by Key"
##
## attr(,"class")
## [1] "labels"
```

We were curious to investigate potential variation in stream counts across different keys. We observed slight fluctuations in the medians of the distributions as we limited the y-axis more. We conducted an analysis of variance test, yielding a p-value of 0.668. This indicates that, within the scope of our analysis, there is insufficient evidence to suggest significant differences in stream counts across different keys.

Considered Approaches

Initially, we considered creating a logistic regression model to predict a song's virality prior to normalizing the data. Our idea was that logistic regression is a suitable method for binary classification tasks, like predicting whether a song will go viral or not. However, a logistic regression model is influenced by the distribution of the response variable, and we discovered the high variability of streams in our preliminary analysis. Even with normalization, our target variable streams would not have the balanced distribution between positive and negative classes that is needed for logistic regression. All of the songs in our dataset hit the charts and thus are considered "viral" to some extent, which may lead to struggles for our model to discern between viral and non-viral songs and reducing model performance.

One of the algorithms we were interested in was K-NN. Initially, we hypothesized that the K-Nearest Neighbors (KNN) method might be a good fit for our data, given its ability to capture complex patterns and its success in various classification and regression tasks, especially when dealing with multi-dimensional feature spaces like ours. There are several considerations when using k-NN for this type of data:

- Scalability: k-NN can be computationally expensive, especially for large datasets. This is because the algorithm needs to compute the distance between each test data point and every training data point, which can be time-consuming.
- Choice of k: The choice of the parameter "k" is crucial in the k-NN algorithm. A smaller "k" value can lead to noisy predictions, as outliers might influence the algorithm. On the other hand, a larger value of "k" can lead to smoother decision boundaries but might also cause the algorithm to lose local patterns in the data.
- Sensitivity to Outliers: k-NN can be sensitive to outliers in the data, which can significantly affect its performance and was particularly relevant to our data as there were some drastic outliers in the data. In the context of music popularity prediction, several studies have used other machine learning methods and found them to be effective. For example, a study by Arora & Rani (2024) used Linear Regression, Lasso Regression, Ridge Regression, Elastic Net, Random Forest, GBM, and neural networks to predict whether a song will be a hit. Another study by Sebastian, Jung, & Mayer (2024) used Ordinary Least Squares (OLS), Multivariate Adaptive Regression Splines (MARS), Random Forest, and XGBoost algorithms to analyze song characteristics and their impact on popularity. These studies suggest that other methods may be more suitable for this type of data due to their ability to handle high-dimensional data, deal with noise and outliers, and model complex relationships. Before conducting the literature review, we attempted to apply the K-Nearest Neighbors (KNN) method to our dataset. However, despite adjusting the parameters, we encountered high Mean Squared Error (MSE) values. The literature review subsequently suggested the use of alternative methods. Given these circumstances, we decided to discontinue using the KNN method for our analysis.

Previous Music Analyses and Methods

- "Songs with more artists are generally more popular": A study published in the Journal of Marketing Behavior found that songs featuring other artists have a greater likelihood of making it into the top 10 than songs not featuring other artists (Ordanini, Nunes, & Valsesia, 2018). This study used a dataset of 26,000 songs from the Billboard Hot 100 chart from 1990 to 2010. They used regression analysis to examine the relationship between the number of featured artists and a song's chart performance. Another study conducted by Stanford University suggested that the presence of guest artists improved the popularity of a track, but not with the same degree of significance (Pham, Kyauk, & Park, 2015). This study used a dataset of over 380,000 songs and used machine learning algorithms to predict song popularity.

- Song release season and popularity "correlation": While there is a common belief that songs released towards the summer (summer hits) tend to be more popular, we couldn'' find a study that confirmed this. However, a study published on Medium suggested that the duration of songs has been decreasing over the years, and this trend coincides with the increasing importance of AI metrics and personalized recommendations, which could indirectly influence the seasonal popularity of songs (Parth, 2021). This study used descriptive statistics and data visualization to analyze trends in song duration over time.

- Effect of Release Day on Popularity: The choice of release day can have an impact on a song's visibility and streams. While there are no specific studies on this topic, it's generally accepted in the music industry that releasing a song on a day other than Friday might help it stand out from the crowd and get more attention (Lee & Lee, 2018). This study used a dataset of over 50,000 songs from the Melon music streaming service. They used machine learning algorithms, including decision trees and random forests, to predict song popularity based on various features, including release day.

Methods and Analysis

We intend to predict streams based on various song attributes, thus we opted for a supervised learning approach. Our response variable of interest, streams, is continuous, therefore we chose to conduct regression analysis to predict the virality of songs based on several song attributes in an efficient and interpretable way. Based on our own analysis goals and the methods of previous analysts, we opted to try various methods—linear regression, decision trees, and random forests—to derive the strongest conclusions and find the most effective model for our dataset.

A new dataframe was created for the cleaned data that eliminated unnecessary variables. Key was converted from a categorical to a factor variable due to the multitude of levels. Because the categorical variables of artists_name and track_name had an extensive number

of levels, they were omitted from the lasso and ridge regression. Additionally, variables that were not of interest for our analysis were excluded—including variables relating to songs being in the Deezer, Shazam, Apple, and Spotify charts and playlists as well as the date metrics of release year and release day of month.

```
# Make new
library(car)

## Loading required package: carData

##
## Attaching package: 'car'

## The following object is masked from 'package:purrr':
##
##     some

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

spotify_new <- subset(spotify, select = -c(track_name, artist.s._name, stream
s, in_apple_playlists, in_apple_charts, in_deezer_playlists, in_deezer_charts
, in_shazam_charts, key, in_spotify_charts, released_day, released_year)) # R
emove unneeded variables
```

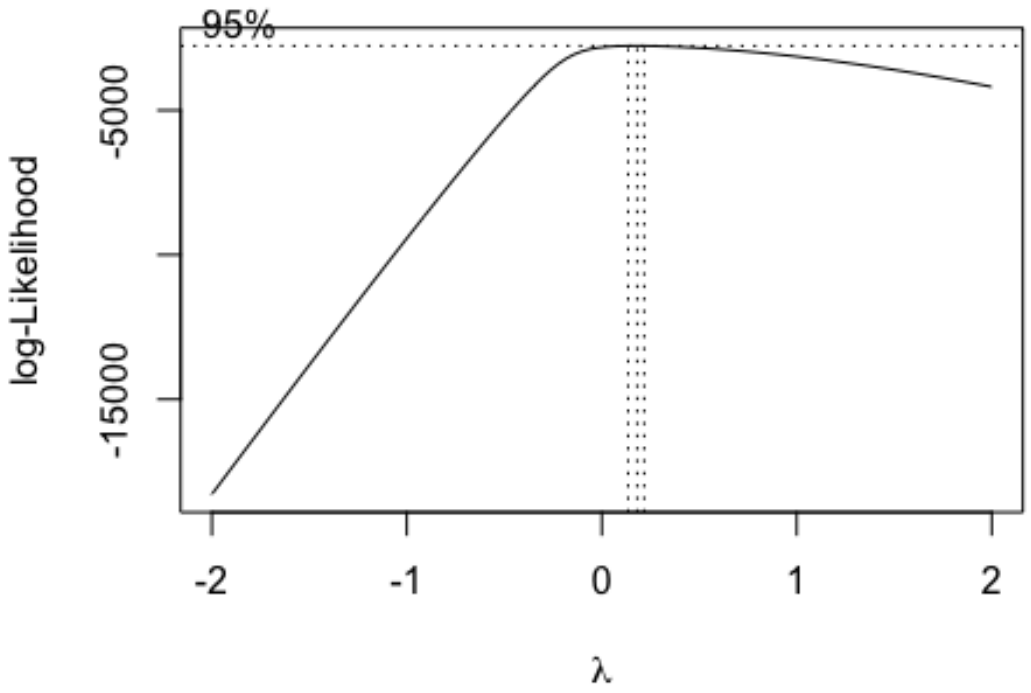
Decision trees

We attempted a decision tree to find the most optimal weight for each predictor to accurately predict the number of streams a song would get, and which predictors proved to be most optimal for finding the number of streams a song would get. We started with a conservative approach toward the decision tree, keeping the default values of best =5 and running the function. Overall, we gained an MSE of 478.45, which was very high. In an attempt to minimize this MSE, we attempted pruning by cross validating and plotting the cv error to find the optimal number of terminal nodes to have in our model to gain the lowest MSE while maintaining an aesthetically pleasing tree. The plot revealed that 4 may be the optimal amount of terminal nodes to use in the decision tree. We later ran the decision tree function with best = 4. The resulting MSE of the pruned tree was 475.4. It was clear that the decision tree had far too high of an MSE to be used in our final analysis towards the project. Reasons for this may be due to our model underfitting our data and therefore increasing the bias in our predictions.

```
set.seed(1)
# Normalize streams
library(MASS)

##
## Attaching package: 'MASS'
```

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

```
b <- boxcox(lm(streams ~ 1, data = spotify))
```



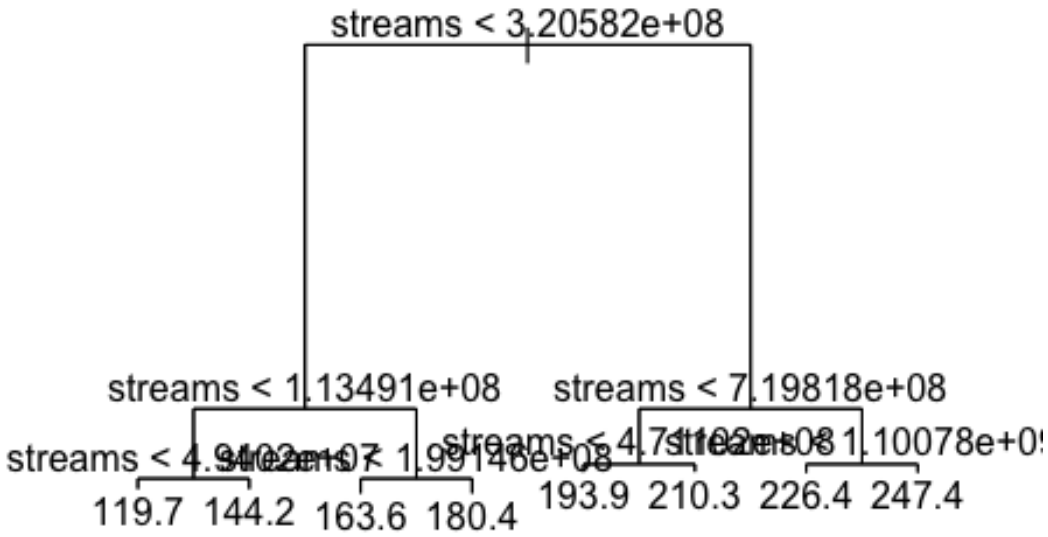

```
# Extract lambda
lambda <- b$x[which.max(b$y)]
lambda
```

```
## [1] 0.1818182
```

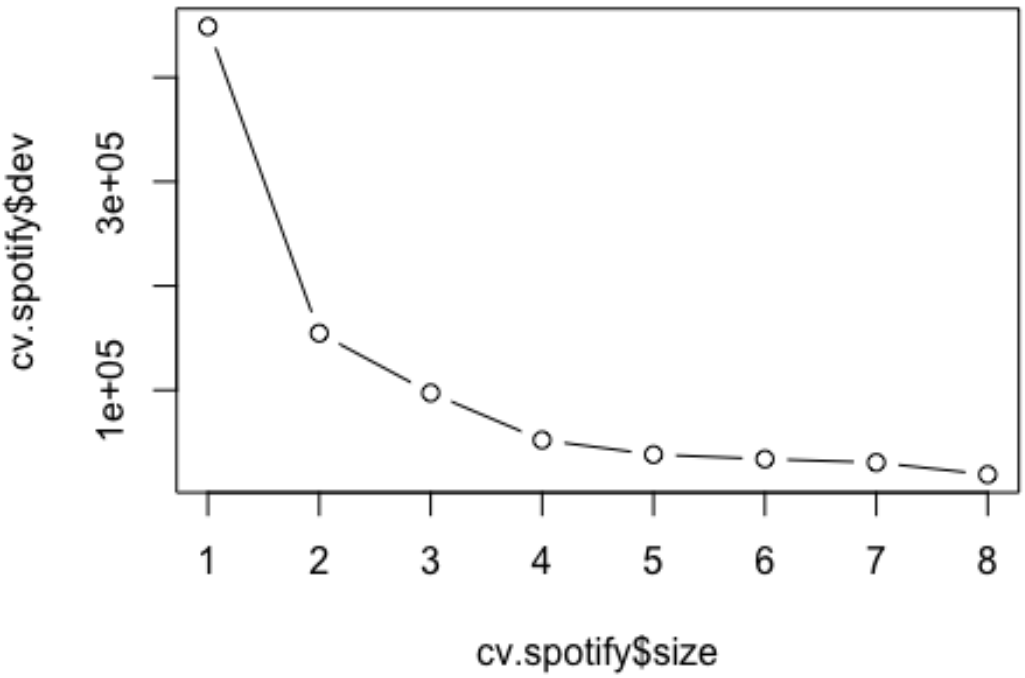
```
spotify$norm_streams <- (spotify$streams ^ lambda - 1) / lambda
spotify_new <-  spotify[,unlist(lapply(spotify, is.numeric))]
train <- sample(1:nrow(spotify_new), nrow(spotify_new) / 2)
tree.spotify <-tree(norm_streams ~., data = spotify_new, subset = train) #sma
ller tree
#, control = tree.control(nobs = length(train), mindev = 0) for big tree
summary(tree.spotify)
```

```
##
## Regression tree:
## tree(formula = norm_streams ~ ., data = spotify_new, subset = train)
## Variables actually used in tree construction:
```

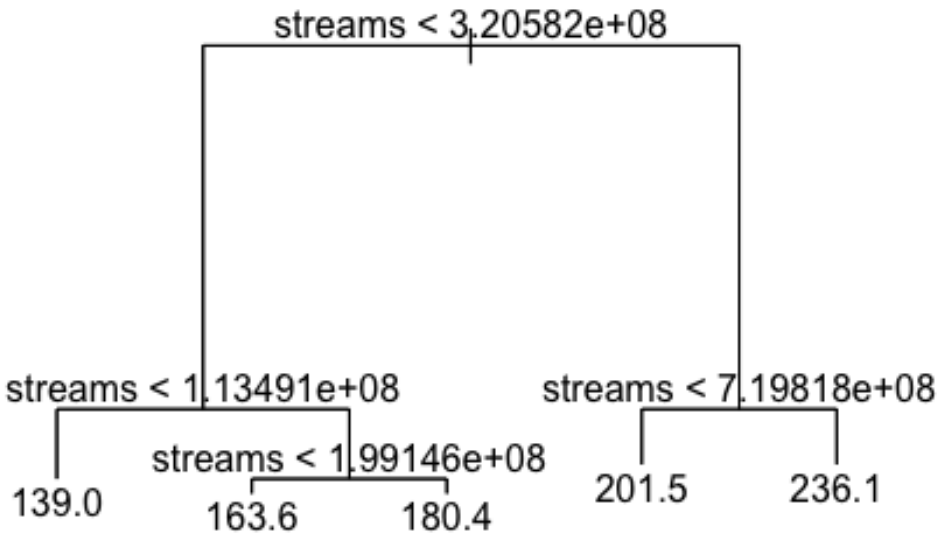
```
## [1] "streams"
## Number of terminal nodes:  8
## Residual mean deviance:  40.99 = 16640 / 406
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -53.4900  -4.2360  -0.1077   0.0000   4.6340  34.3600

plot(tree.spotify, cex.lab = 0.8)
text(tree.spotify, pretty = 0)
```



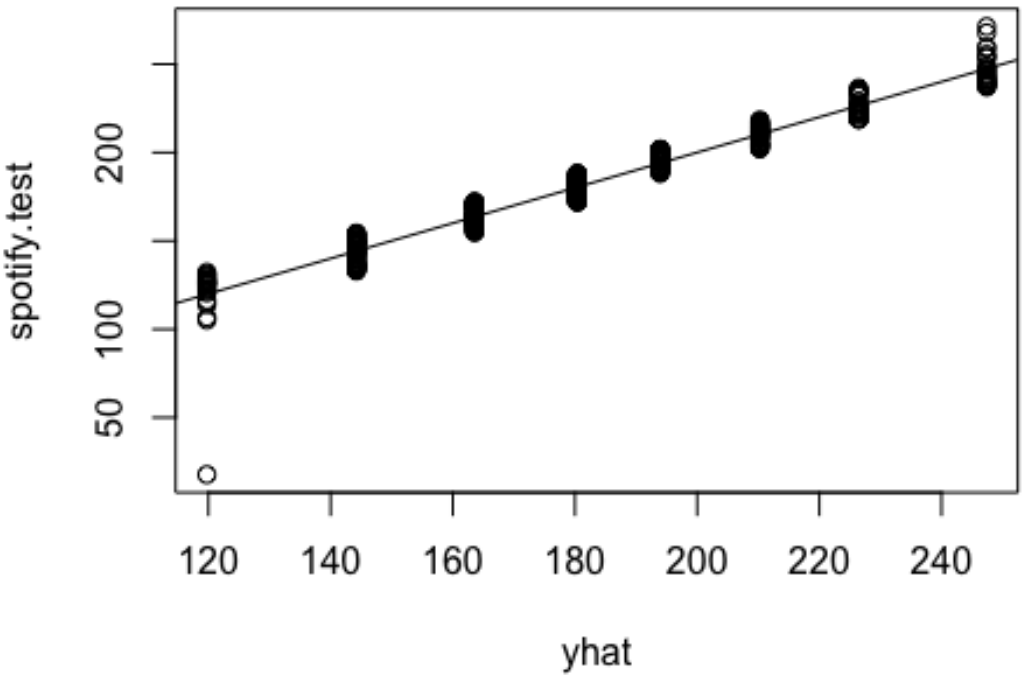

```
#cross validation
cv.spotify <- cv.tree(tree.spotify)
plot(cv.spotify$size, cv.spotify$dev, type = "b")
```




```r
#our attempt at pruning
prune.spotify <- prune.tree(tree.spotify, best = 5)
plot(prune.spotify)
text(prune.spotify, pretty = 0)
```




```
yhat <- predict(tree.spotify, newdata = spotify_new[-train,])
spotify.test <- spotify_new[-train, "norm_streams"]
plot(yhat, spotify.test)
abline(0,1)
```




```
mean((yhat- spotify.test)^2)
```

```
## [1] 57.93916
```

Random forest

Upon getting a very high MSE from our decision tree, we made an effort to see if we could create a random forest to account for our high number of variables. We ran the random forest with a variety of mtry values and our best performance was with an mtry = 5. Our resulting error was 423.39. With our error still being very high, our curiosity led us to check the importance of the variables used for these tests. Our results of the importance showed the overwhelming amplifier of our MSE to be in_spotify_playlists. In an attempt to find a better MSE, we ran the decision tree without the in_spotify_playlists variable and received an MSE of 1161. At this point we realized that the MSE would be high for our data regardless if done with a random forest tree. Reasons for our random forest performance performing poorly can be linked to our data not being linear.

```
set.seed(1)
rf.spotify <- randomForest(norm_streams ~ ., data = spotify_new, subset = tra
in, mtry = 5, importance = TRUE)
yhat.rf <-predict(rf.spotify, newdata = spotify_new[-train, ])
mean((yhat.rf- spotify.test)^2)
```

```
## [1] 91.70593
```

```
importance(rf.spotify)
```

```
##                         %IncMSE IncNodePurity
## artist_count         -1.7297871     2574.5000
## released_year        15.8004601    29024.8294
## released_month        6.0592055     4368.0272
## released_day          3.0402982     3557.0140
## in_spotify_playlists 19.7452345    75665.6049
## in_spotify_charts     9.3105190    10628.6832
## streams              43.0710053   187340.1320
## in_apple_playlists   11.6968974    30281.5980
## in_apple_charts       6.9916602    10098.3792
## in_deezer_playlists  13.3707830    51475.0677
## in_deezer_charts      5.4510107     3800.6492
## in_shazam_charts      8.2804590     6953.2122
## bpm                   0.7607666     3450.0722
## danceability_.       -0.3061399     3840.0344
## valence_.             2.7302709     3361.3548
## energy_.              1.5140473     3221.7109
## acousticness_.        0.8683214     2795.0843
## instrumentalness_.    0.5313377      469.6265
## liveness_.           -1.3138933     3133.8826
## speechiness_.         1.5554961     1948.4189
## numeric_key           1.3247327     2214.0112
```

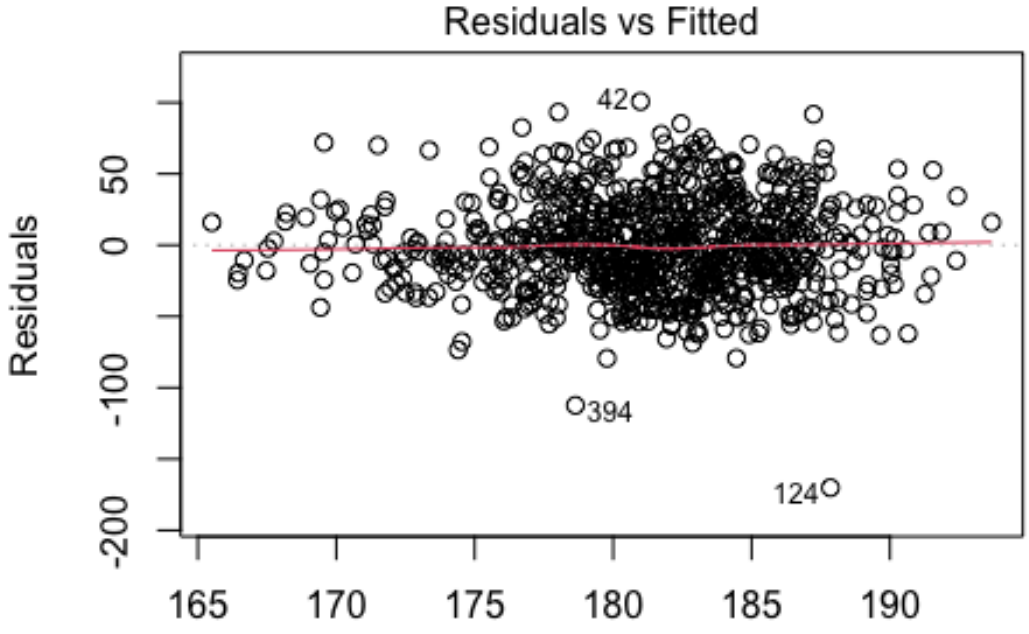
Linear Regression Analysis

In order to conduct multiple linear regression, variable transformation was done to normalize the data that failed to meet linear assumptions. Utilizing the boxcox() function, a lambda value was found to transform the streams variable. Upon the transformation, a preliminary model was used to confirm that the transformation improved the diagnostics and the residual standard error was greatly reduced with the utilization of a normalized response variable.

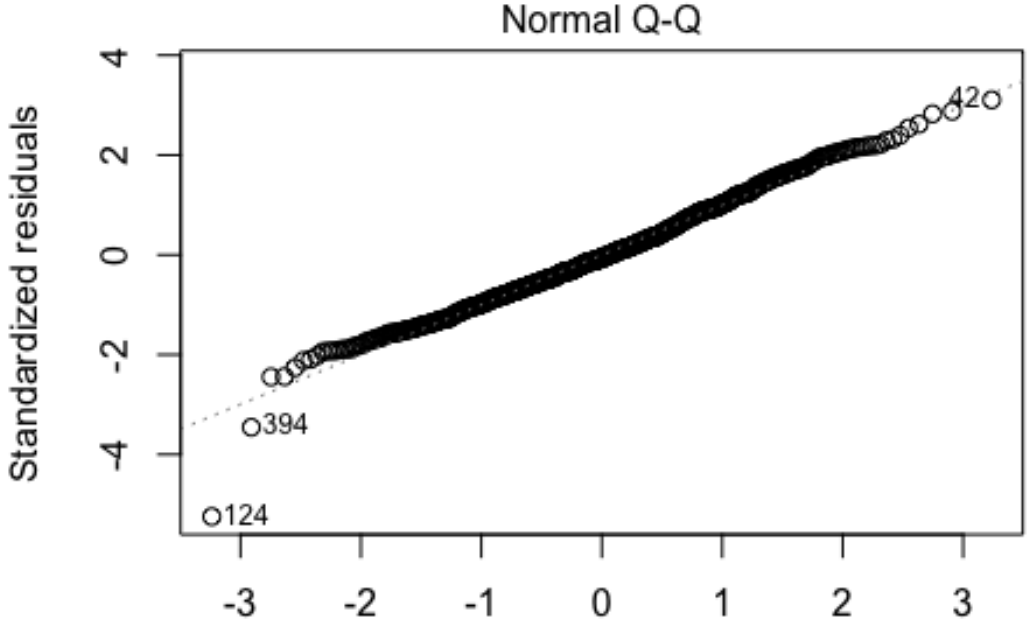
```
mod <- lm(norm_streams ~ bpm + danceability_. + liveness_. + acousticness_. +
valence_. + energy_. + instrumentalness_. + speechiness_., data = spotify)
summary(mod)
```

```
##
## Call:
## lm(formula = norm_streams ~ bpm + danceability_. + liveness_. +
##     acousticness_. + valence_. + energy_. + instrumentalness_. +
##     speechiness_., data = spotify)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -170.119  -21.931   -2.031   21.807  100.758
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
```

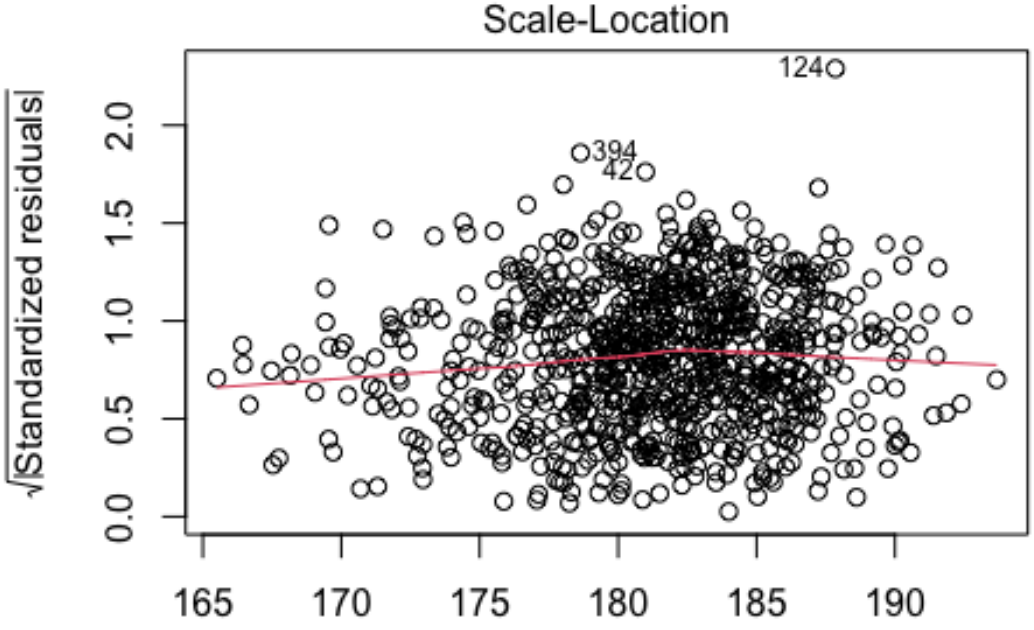
```
## (Intercept)        199.88488   10.61624  18.828   <2e-16 ***
## bpm                  0.04633    0.04133   1.121   0.2626
## danceability_.      -0.08540    0.09108  -0.938   0.3487
## liveness_.          -0.11134    0.08283  -1.344   0.1792
## acousticness_.      -0.09543    0.05578  -1.711   0.0875 .
## valence_.           -0.01595    0.05774  -0.276   0.7824
## energy_.            -0.15508    0.09094  -1.705   0.0885 .
## instrumentalness_.  -0.02088    0.13235  -0.158   0.8747
## speechiness_.       -0.28756    0.11387  -2.525   0.0117 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 32.61 on 820 degrees of freedom
## Multiple R-squared:  0.01971,    Adjusted R-squared:  0.01015
## F-statistic: 2.061 on 8 and 820 DF,  p-value: 0.03722

plot(mod)
```



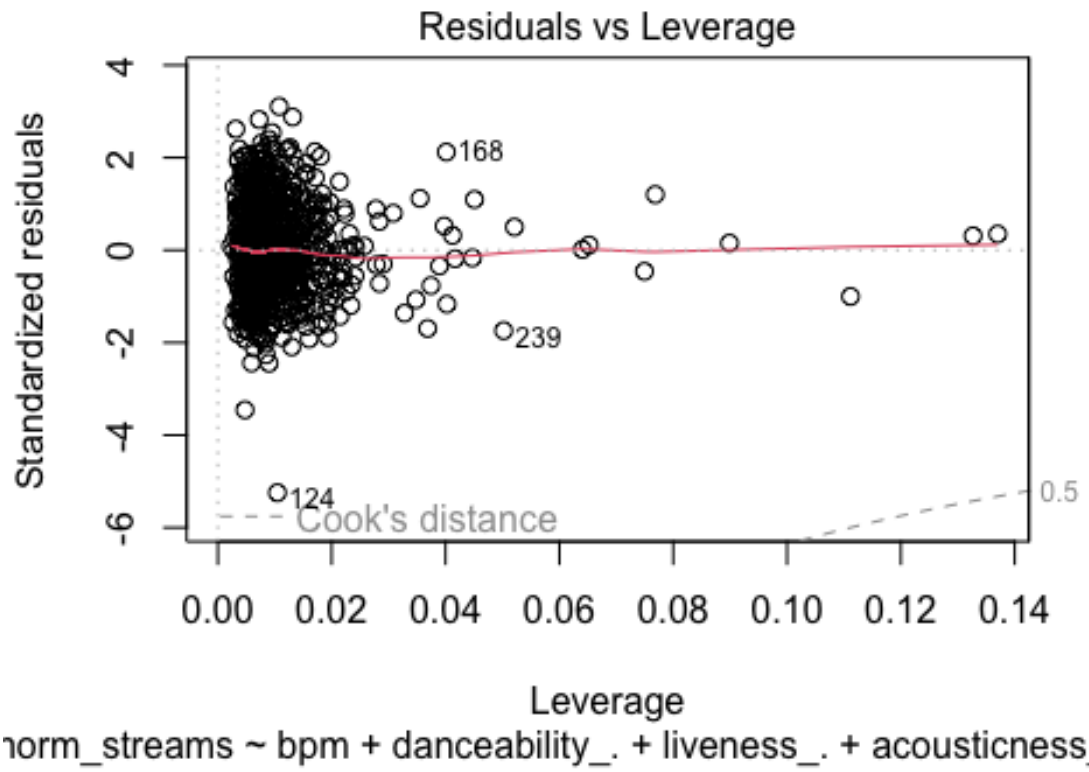
norm_streams ~ bpm + danceability_. + liveness_. + acousticness_.



norm_streams ~ bpm + danceability_. + liveness_. + acousticness_.



norm_streams ~ bpm + danceability_. + liveness_. + acousticness_.



Shrinkage Methods

Both LASSO and ridge regression were conducted to compare their outcomes and determine if the true model is non-sparse and most of the true coefficients are not zero. Ridge regression is effective in reducing multicollinearity and preventing overfitting by shrinking the coefficients while LASSO additionally performing variable selection, making the model simpler and more interpretable. Which technique to use depends on the specific characteristics of the data and the goals of the analysis; thus, both were conducted to determine the better shrinkage method.

```
# LASSO
library(glmnet)
```

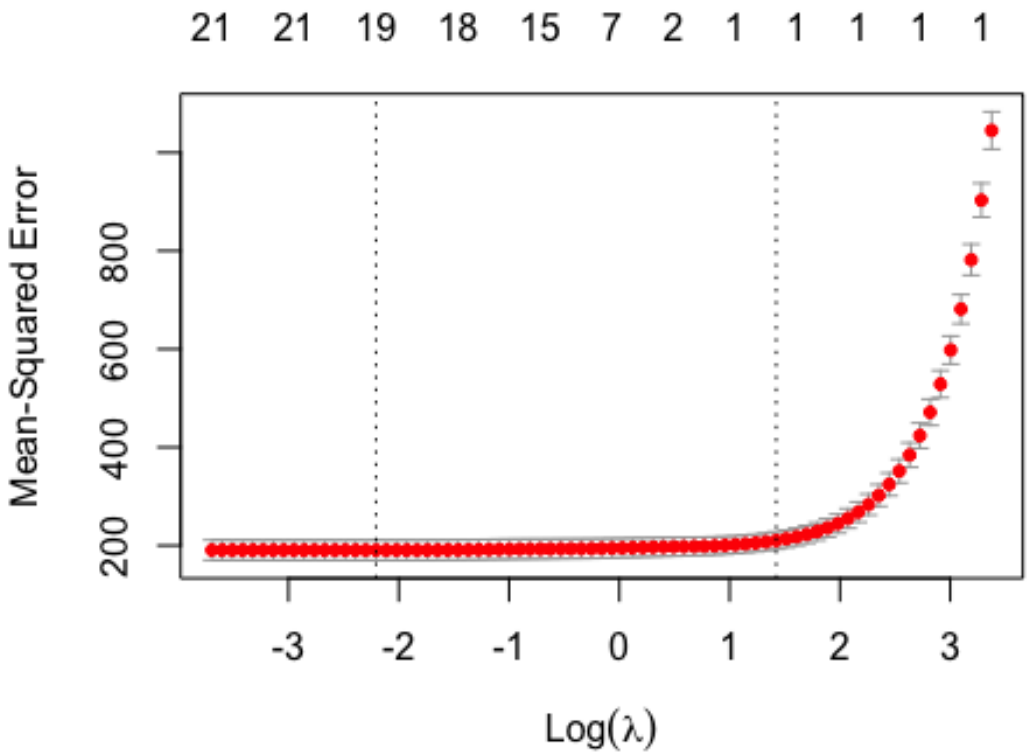
```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
```

```
## Loaded glmnet 4.1-8
```

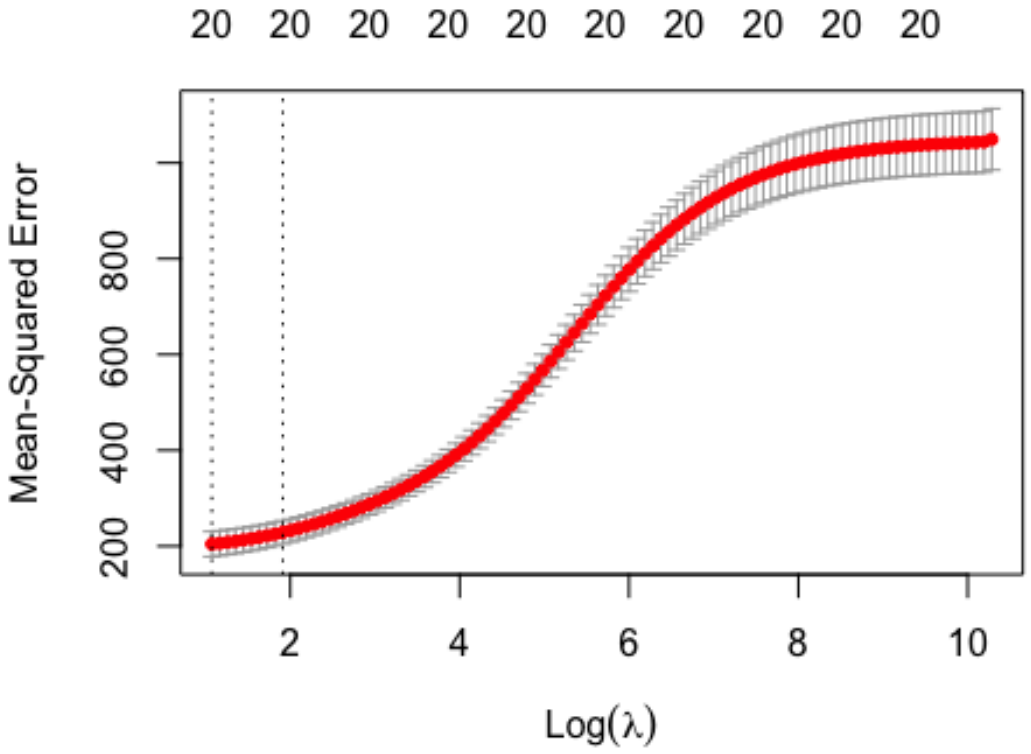
```
library(dplyr)
set.seed(8)
samp <- sample(seq_len(nrow(spotify_new)),  size = 0.75*nrow(spotify_new))
train <- spotify_new[samp,]
test <- spotify_new[-samp,]
xtrain <- model.matrix(norm_streams~., data=train)[,-1]
ytrain <- train$norm_streams
xtest <-model.matrix(norm_streams~., data=test)[,-1]
ytest <- test$norm_streams
lasso <- glmnet(xtrain, ytrain, alpha=1)
cv.lasso <- cv.glmnet(xtrain, ytrain,alpha=1)
plot(cv.lasso)
```




```
bestlam <- cv.lasso$lambda.min
lasso.pred <- predict(lasso ,s=bestlam ,newx=xtest)
lasso.err <- mean((lasso.pred - ytest)^2)
lasso.err

## [1] 260.2689

predict(lasso, type = "coefficients", s = bestlam)

## 22 x 1 sparse Matrix of class "dgCMatrix"
##                                      s1
```

```
## (Intercept)           5.018278e+02
## artist_count         -1.050530e+00
## released_year        -1.738803e-01
## released_month        2.758338e-01
## released_day          1.305924e-01
## in_spotify_playlists  4.059176e-04
## in_spotify_charts    -6.973486e-02
## streams               7.791515e-08
## in_apple_playlists    .
## in_apple_charts       2.877082e-02
## in_deezer_playlists  -1.828292e-02
## in_deezer_charts      1.519490e-01
## in_shazam_charts      2.942126e-03
## bpm                   1.580128e-02
## danceability_.        3.082077e-02
## valence_.             2.252360e-02
## energy_.             -5.679575e-02
## acousticness_.       -3.952974e-02
## instrumentalness_.    5.066302e-02
## liveness_.           -4.025903e-02
## speechiness_.        -3.940405e-02
## numeric_key           .
```

```r
# Ridge Regression
set.seed(8)
xtrain <- model.matrix(norm_streams~., data=train[,-1])
ytrain <- train$norm_streams
ridge <- glmnet(xtrain,ytrain,alpha=0)
cvridge <- cv.glmnet(xtrain,ytrain,alpha=0)
plot(cvridge)
```




```
# Best Lambda
bestlam <- cvridge$lambda.min
ridgepred <- predict(ridge, s = bestlam, newx = xtest)
ridge.error <- mean((ridgepred - ytest)^2)
ridge.error

## [1] 297.1808

predict(ridge, type = "coefficients", s = bestlam) # Final model

## 22 x 1 sparse Matrix of class "dgCMatrix"
##                                 s1
## (Intercept)           4.023008e+02
## (Intercept)           .
## released_year        -1.236778e-01
## released_month        3.338098e-01
## released_day          1.608996e-01
## in_spotify_playlists  1.006475e-03
## in_spotify_charts    -2.039836e-02
## streams               6.175129e-08
## in_apple_playlists    1.135566e-02
## in_apple_charts       3.343837e-02
## in_deezer_playlists  -3.753543e-03
## in_deezer_charts      2.029535e-01
```

```
## in_shazam_charts     -5.309317e-03
## bpm                   2.015117e-02
## danceability_.        2.111338e-02
## valence_.             2.000621e-02
## energy_.             -8.219460e-02
## acousticness_.       -4.558887e-02
## instrumentalness_.    5.754343e-02
## liveness_.           -5.144439e-02
## speechiness_.        -7.000117e-02
## numeric_key           6.431284e-03
```

LASSO had a lower test mean squared error (MSE) than ridge regression; ergo, LASSO outperformed ridge regression and subset selection was determined to be preferable given the data. Because of this, we additionally conducted best, forward, and backward subset selection to evaluate the best variables to keep in the model.

Subset Selection

Best Subset Selection

```
# Best Subset Selection
library(leaps)
set.seed(8)
regfit.full = regsubsets(norm_streams~., spotify_new, nvmax=13)
regfit.sum <- summary(regfit.full)
which.min(regfit.sum$cp)
```

```
## [1] 10
```

```
which.min(regfit.sum$bic)
```

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

```
which.max(regfit.sum$adjr2)
```

```
## [1] 13
```

```
coef(regfit.full, 5) # Check five-variable mod
```

```
##         (Intercept)       released_year   in_spotify_charts             st
reams
##        5.165945e+02       -1.811198e-01       -9.864112e-02        8.49253
1e-08
##     in_apple_charts in_deezer_playlists
##        3.758734e-02       -1.977675e-02
```

Best subset selection indicated that the five variable, three variable, and six variable models had the lowest Cp, lowest BIC, and highest adjusted R-squared value.

```
set.seed(1)
# Validation Set Approach
train = sample(c(TRUE,FALSE), nrow(spotify_new),rep=TRUE)
test =(! train )
```

```
# return all the models, up to a 13-variable model
regfit.best = regsubsets(norm_streams~., spotify_new[train,], nvmax=13)
test.mat = model.matrix(norm_streams~., spotify_new[test,])
val.errors = rep(NA,13)


for (i in 1:13){
        coefi = coef(regfit.best, id=i)
        pred = test.mat[,names(coefi)]%*%coefi
        val.errors[i] = mean((spotify_new$norm_streams[test]-pred)^2)
}
i <- which.min(val.errors)
coef(regfit.best, i)

##         (Intercept)             streams in_deezer_playlists
##        1.505387e+02        8.748909e-08       -1.195396e-02
```

The validation set approach additionally selected the three variable model with artist_count, released_month, and in_spotify_playlists.

Forward Selection

```
#Forward Stepwise Selection
library(leaps)
# Return all the models, up to a 13-variable model
set.seed(8)
forward.mod = regsubsets(norm_streams~., spotify_new, nvmax=13, method="forwa
rd")
#Output the best set of variables for each model size
forward.sum <- summary(forward.mod)
which.min(forward.sum$cp)

## [1] 10

which.min(forward.sum$bic)

## [1] 3

which.max(forward.sum$adjr2)

## [1] 13

coef(forward.mod, 3) # Check 3-variable mod

##         (Intercept)       released_year             streams in_deezer_play
lists
##        5.459580e+02       -1.953042e-01        8.470848e-08       -1.90923
4e-02
```

Forward selection produced similar five variable, three variable, and six variable models, with the first three variables selected being artist_count, released_month, and in_spotify_playlists.

Backward Selection

```
# Backward Stepwise Selection
# return all the models, up to a 13-variable model
set.seed(1)
backward.mod = regsubsets(norm_streams~., spotify_new, nvmax=13, method="back
ward")
backward.sum <- summary(backward.mod)
which.min(backward.sum$cp)

## [1] 10

which.min(backward.sum$bic)

## [1] 3

which.max(backward.sum$adjr2)

## [1] 13

coef(backward.mod, 6) # Check six-variable mod

##         (Intercept)       released_year        released_day   in_spotify_c
harts
##        5.499924e+02       -1.985311e-01        1.309136e-01       -9.98032
1e-02
##             streams     in_apple_charts in_deezer_playlists
##        8.450292e-08        3.796365e-02       -1.907248e-02
```

Backward selection reaffirmed the outcomes of the other selection methods. Because of the similar results, we opted to construct and compare a three variable, five variable, and six variable model as well as a model based upon the LASSO regression.

```
# LASSO Model
lasso_mod <- lm(norm_streams ~ artist_count + released_month + in_spotify_pla
ylists + speechiness_. + bpm + danceability_. + liveness_., data = spotify)
summary(lasso_mod)

##
## Call:
## lm(formula = norm_streams ~ artist_count + released_month + in_spotify_pla
ylists +
##     speechiness_. + bpm + danceability_. + liveness_., data = spotify)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -152.83  -13.67    1.39   15.07   79.00
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.534e+02  6.263e+00  24.494  < 2e-16 ***
## artist_count         -3.068e+00  9.224e-01  -3.326  0.00092 ***
## released_month        7.054e-01  2.313e-01   3.049  0.00237 **
```

```
## in_spotify_playlists  6.005e-03  2.167e-04  27.711  < 2e-16 ***
## speechiness_.        -1.356e-01  8.149e-02  -1.664  0.09651 .
## bpm                   3.536e-02  2.936e-02   1.205  0.22870
## danceability_.        1.113e-01  5.890e-02   1.889  0.05927 .
## liveness_.           -3.530e-02  5.867e-02  -0.602  0.54757
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.3 on 821 degrees of freedom
## Multiple R-squared:  0.499,  Adjusted R-squared:  0.4948
## F-statistic: 116.8 on 7 and 821 DF,  p-value: < 2.2e-16
```

```
# Three variable model
three_var_mod <- lm(norm_streams ~ artist_count + released_month + in_spotify
_playlists, data = spotify)
summary(three_var_mod)
```

```
##
## Call:
## lm(formula = norm_streams ~ artist_count + released_month + in_spotify_pla
ylists,
##     data = spotify)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -152.185  -14.040    1.161   15.311   75.494
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.633e+02  2.301e+00  70.977  < 2e-16 ***
## artist_count         -2.970e+00  8.990e-01  -3.304 0.000996 ***
## released_month        6.648e-01  2.310e-01   2.878 0.004101 **
## in_spotify_playlists  5.998e-03  2.156e-04  27.816  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.34 on 825 degrees of freedom
## Multiple R-squared:  0.4948, Adjusted R-squared:  0.493
## F-statistic: 269.4 on 3 and 825 DF,  p-value: < 2.2e-16
```

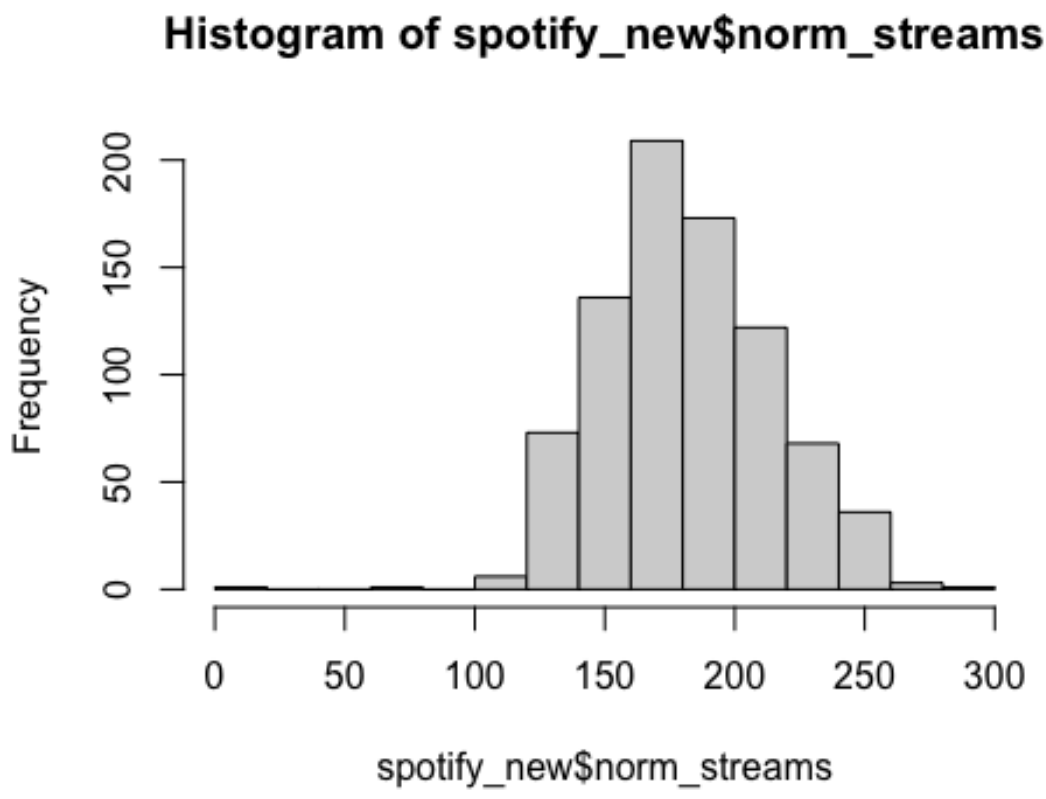
```
# Five variable model
five_var_mod <- lm(norm_streams ~ artist_count + released_month + in_spotify_
playlists + speechiness_. + danceability_., data = spotify)
summary(five_var_mod)
```

```
##
## Call:
## lm(formula = norm_streams ~ artist_count + released_month + in_spotify_pla
ylists +
##     speechiness_. + danceability_., data = spotify)
##
```

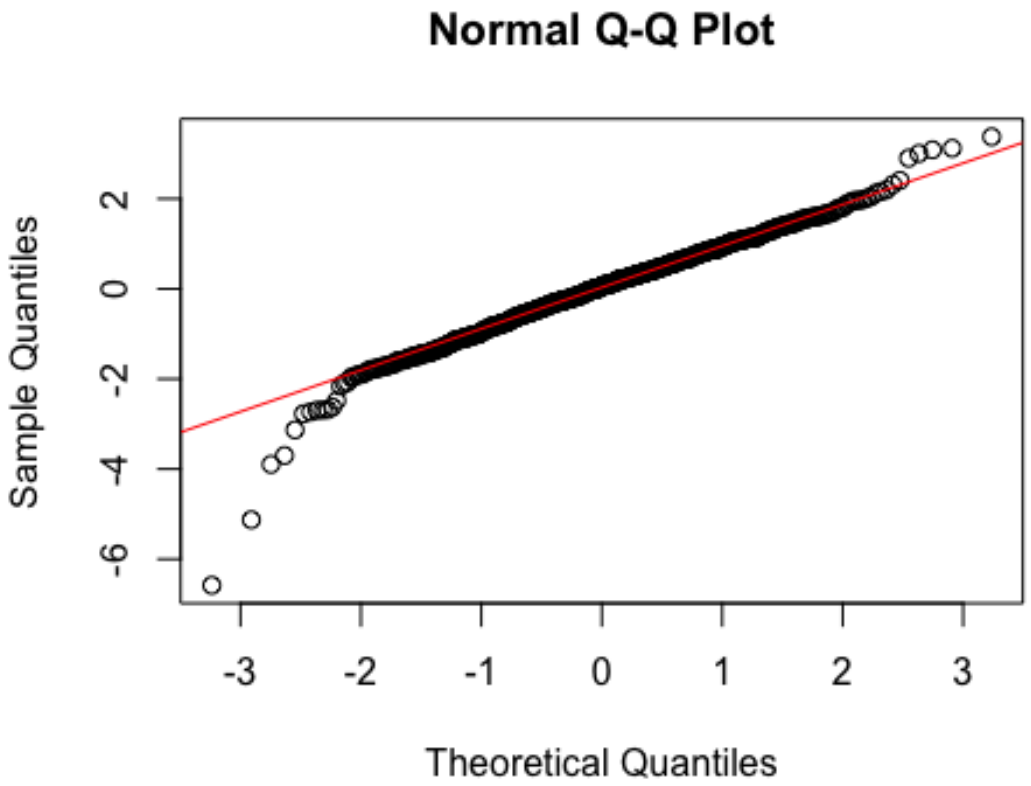
```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -151.124  -13.775    1.403   14.796   76.702
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.576e+02  4.349e+00  36.240  < 2e-16 ***
## artist_count         -3.149e+00  9.197e-01  -3.424 0.000647 ***
## released_month        6.921e-01  2.309e-01   2.998 0.002800 **
## in_spotify_playlists  6.014e-03  2.165e-04  27.781  < 2e-16 ***
## speechiness_.        -1.283e-01  8.129e-02  -1.578 0.115025
## danceability_.        1.051e-01  5.803e-02   1.811 0.070433 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.3 on 823 degrees of freedom
## Multiple R-squared:  0.4979, Adjusted R-squared:  0.4949
## F-statistic: 163.2 on 5 and 823 DF,  p-value: < 2.2e-16

# Six variable model
six_var_mod <- lm(norm_streams ~ artist_count + released_month + in_spotify_p
laylists + speechiness_. + danceability_. + bpm, data = spotify)
summary(six_var_mod)

##
## Call:
## lm(formula = norm_streams ~ artist_count + released_month + in_spotify_pla
ylists +
##     speechiness_. + danceability_. + bpm, data = spotify)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -152.489  -13.658    1.099   15.159   78.214
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           1.525e+02  6.074e+00  25.104  < 2e-16 ***
## artist_count         -3.103e+00  9.202e-01  -3.372 0.000782 ***
## released_month        7.085e-01  2.312e-01   3.065 0.002251 **
## in_spotify_playlists  6.011e-03  2.164e-04  27.774  < 2e-16 ***
## speechiness_.        -1.338e-01  8.140e-02  -1.644 0.100552
## danceability_.        1.149e-01  5.857e-02   1.961 0.050160 .
## bpm                   3.553e-02  2.935e-02   1.211 0.226351
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 23.29 on 822 degrees of freedom
## Multiple R-squared:  0.4988, Adjusted R-squared:  0.4952
## F-statistic: 136.3 on 6 and 822 DF,  p-value: < 2.2e-16
```

After comparing the three variable, five variable, six variable, and LASSO models, the six variable model had the lowest residual standard error and the highest adjusted R-squared value overall. The best model determined via several selection models gave insight into what aspects of songs influence their success. When it comes to audio features, our analysis indicates that speechiness and danceability hold the greatest significance in influencing streams. An increase in speechiness results in a decrease in streams. Ergo, more verbose songs are less likely to accrue more streams than songs with less lyrics. Danceability, on the other hand, has a positive effect on streams. Songs with greater danceability–strong and consistent rhythm and beat, faster tempo, and dynamic shifts within the songs—have a greater number of streams. Additionally, artist count has an unexpected negative impact on streams, indicating that artist collaboration and an increase in features on tracks does not increase streams. Songs present in more playlists also have greater streams. Lastly, songs released later in the year also have greater streams. The adjusted R-square of .4952 shows that the variables included within the best model jointly account for 49.52% of variation in streams; ergo, the variables within the model account for less than half of the variation in streams which does not indicate that they are particularly strong predictors of music success despite their statistically significant effects on streams. This entails that linear modeling is not the optimal course of action for analysis of variables influencing streams.

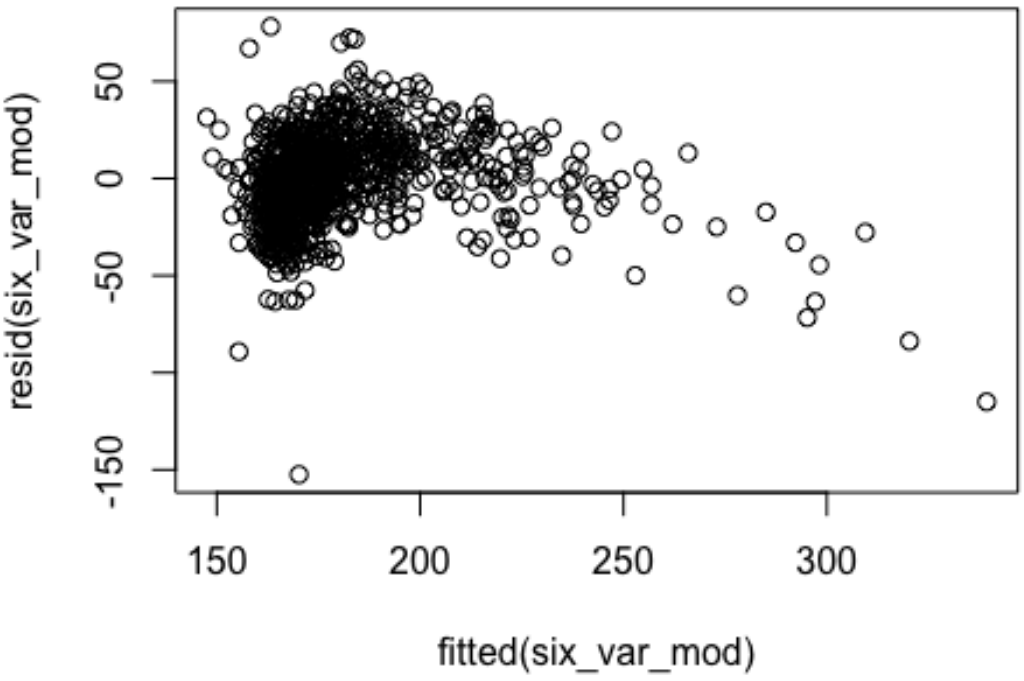
```
hist(spotify_new$norm_streams)
```



```
res = rstandard(six_var_mod)
qqnorm(res)
qqline(res, col="red")
```




```
plot(fitted(six_var_mod), resid(six_var_mod))
```



As seen in the residual plot, the response variable of streams primarily clusters around the horizontal axis but exhibits a downward trend, displaying a pattern where the residuals tend to be larger for observations with higher fitted values of predicted stream counts. This pattern suggests that the model may be systematically overestimating the number of streams for songs with lower predicted values and underestimating for songs with higher predicted values. This likely relates to outliers that draw in abnormally high streams such as top artists in the music industry or low streams from more underground artists. The quantile-quantile plot primarily follows the normal line with the left tail of the plot extending further downwards than the right tail, suggesting that there is an excess of lower values in the data compared to what would be expected in a normal distribution. This indicates that the left tail of the distribution is heavier or longer than that of a normal distribution. Conversely, the right tail going slightly above the line implies that an excess of higher values exist in the data compared to a normal distribution. The histogram of the response variable follows a normal distribution, which is due to the earlier transformation of the variable.

Main findings, recommendations, and conclusions

Due to our response variable, streams, being normalized, we cannot interpret our resulting coefficients as the exact increase or decrease associated with a one-unit change in a predictor variable. Instead, we can generalize the effects of each variable depending on if the coefficient was negative or positive. The six-variable model emerged as the most

effective in predicting stream counts, with speechiness, danceability, artist count, playlist presence, release timing, and bpm being significant predictors. Speechiness and danceability had the most substantial influence, with higher speechiness associated with decreased streams and greater danceability correlated with increased streams. Surprisingly, an increase in the number of artists in a song had a negative impact on stream counts. This suggests that while artist collaboration is common, it does not necessarily translate to increased popularity. Songs featured in more playlists and those released later in the year tended to have higher stream counts. This underscores the that playlist inclusion is important. Despite the significant effects observed, the adjusted R-squared value of 0.4952 indicates that the variables included in the model collectively account for less than half of the variation in stream counts. This suggests that while the model provides valuable insights, other factors not considered in the analysis likely also influence stream counts. Further research is required to explore the impact of additional variables on stream counts. Expanding the scope to a larger dataset with data from more years present may reveal insights into the trends of music consumption and listener preferences over time. We recommend investigating the impacts of genre to gain insights into the preferences of listeners. We also hypothesize that artist popularity has a significant effect on the chart performance of a song after release. Furthermore, interaction effects between variables could uncover how the audio variables interact and influence stream counts. The last topic of interest would be investigating the duration of a song on the charts, and analyzing how long-term success correlates with various audio features.

Relevant references

- Dataset: https://www.kaggle.com/datasets/nelgiriyewithana/top-spotify-songs-2023
- Lee, J., & Lee, J. S. (2018). Music Popularity: Metrics, Characteristics, and Audio-based Prediction. IEEE Transactions on Multimedia, 20(12), 3213-3224. Retrieved from https://arxiv.org/abs/1812.0055
- Parth (2021). Duration of songs: How did the trend change over time and what does it mean today? Medium. Retrieved from https://parthmusic.medium.com/duration-of-songs-how-did-the-trend-change-over-time-and-what-does-it-mean-today-a41258a41b12.
- Pham, J., Kyauk, E., & Park, E. (2015). Predicting Song Popularity. Stanford University. Retrieved from https://cs229.stanford.edu/proj2015/140_report.pdf
- Ordanini, A., Nunes, J. C., & Valsesia, F. (2018). The featuring phenomenon in music: how combining artists of different genres increases a song's popularity. Journal of Marketing Behavior, 3(2-3), 123-141. Retrieved from https://link.springer.com/article/10.1007/s11002-018-9476-3
- AspiringYouths. (n.d.). Advantages and Disadvantages of KNN Algorithm. Retrieved from https://aspiringyouths.com/advantages-and-disadvantages-of-knn-algorithm/
- GeeksforGeeks. (2024, March 12). KNN vs Decision Tree in Machine Learning. Retrieved from https://www.geeksforgeeks.org/knn-vs-decision-tree-in-machine-learning/

- GeeksforGeeks. (n.d.). K-Nearest Neighbor(KNN) Algorithm - GeeksforGeeks. Retrieved from https://www.geeksforgeeks.org/k-nearest-neighbours/
- Spotintelligence. (2023, August 22). K-Nearest Neighbours Explained, Practical Guide & How To Tutorial. Retrieved from https://spotintelligence.com/2023/08/22/k-nearest-neighbours-explained-practical-guide-how-to-tutorial/
- Thomas, N. (2020, May 9). Using k-nearest neighbours to predict the genre of Spotify tracks. Retrieved from https://towardsdatascience.com/using-k-nearest-neighbours-to-predict-the-genre-of-spotify-tracks-5e7eb6f28e3
- Sebastian, N., Jung, F., & Mayer, F. (2024). Beyond Beats: A Recipe to Song Popularity? A machine learning approach. Retrieved from https://arxiv.org/abs/2104.07670
- Arora, S., & Rani, R. (2024). Soundtrack Success: Unveiling Song Popularity Patterns Using Machine Learning Implementation. Retrieved from https://link.springer.com/chapter/10.1007/978-981-15-3380-8_11