STAT 471 Final Project

"Hot or Not?": Predicting Billboard Hot 100 Rankings

Lucy Wu
Andrew Zheng
Duong Nguyen

May 5, 2019

Contents

1 Executive Summary

Most people are unable to pinpoint what sets their favorite song apart from the other millions of songs in the world. Despite this uncertainty, however, artists like Drake and Taylor Swift are able to produce numerous top-ten hits – as if there is some special musical formula that guarantees that a song will become extremely popular. In this project, we attempt to uncover this formula by investigating which factors influence a song's ranking within the Billboard Hot 100.

The dataset we use in our analysis contains every song that appeared in the Billboard Hot 100 from 2000 through 2018 along with a set of 37 possible predictive factors. These factors range from metrics quantifying the song's musical characteristics to the popularity of the artist that created the song.

To model the song ranking based on these factors, we fit both a regression model (to predict a song's rank) and classification models (to predict whether or not a song would be a top 40 hit). For the regression model, we used ordinary least squares with LASSO; for the classification models, we used logistic regression with LASSO and Random Forests with boosting. Ultimately, the best classification model resulted in a testing misclassification error of 0.15, while the best regression model resulted in a testing mean squared error of 450.

2 Background & Motivation

From the pan flutes of the ancient Greeks to the synthesizer-heavy dance beats of the modern day, humans have been fascinated by music for centuries. Today, music is not only an artistic pursuit, but a major industry: in 2018, the music industry brought in global revenues of over 50 billion USD[1]. The United States in particular plays a major role in the industry, accounting for over 40% (20 billion USD) of the global revenue.

Clearly, making music has become big business. As such, music professionals – singers, songwriters, producers, agents, and more – spend much of their time searching for new talent and/or figuring out how to make a song popular.

One might argue that there isn't one set way to make a song popular; that popularity depends entirely on the fickle ears of the public. While this is certainly true to some extent, there does appear to be some way(s) to reliably produce major hits: the artists Taylor Swift and Drake topped the charts with six top-ten hits each from 2000 through 2018, and Justin Bieber and Lady Gaga

[1] Statista (2018). https://www.statista.com/topics/1639/music/

came in at a close second with four top-ten hits apiece. Meanwhile, big-name producers like Pharell and Kanye West have had a hand in creating even more.

By studying the factors that influence a song's popularity, we hope to (1) gain insight into characteristics shared among popular songs, and (2) build a model that can predict a newly-released song's popularity.

3 Data Overview

Our dataset includes all songs that appeared in the Billboard Hot 100 from 2000 through 2018.

In total, we have 3,320 observations of 41 variables in our dataset. Each observation represents one song. The 41 variables can be grouped into five broad categories as follows:

Non-musical characteristics: Non-musical characteristics of the song.

- `song`: The title of the song. (Not included as a predictive factor because it essentially functions as an ID for the song.)
- `artist`: The artist that created the song. (Not included as a predictive factor because it functions as song ID for many of the songs.)
- `release_date`: The date the song was released.
- `release_season`: The season (fall/winter/summer/spring) the song was released.
- `release_year`: The year the song was released.
- `artist.pop`: The popularity of the artist, as measured by the number of Billboard Hot 100 hits they had in the three years before the song was released.
- `lyrics`: The lyrics of the song.

Musical Characteristics: Simple musical characteristics of the song. These characteristics are `tempo`, `mode`, `key`, `time_signature`, and `duration_ms`; each is self-explanatory.

Spotify Audio Analysis Data: Complex musical characteristics of the song, as computed by Spotify[2].

- `acousticness`: How acoustic the track is (as opposed to electrically amplified).
- `danceability`: How suitable the track is for dancing based on measures including its tempo, rhythm stability, beat strength.
- `energy`: How intense/active the track is. For example, a Bach prelude has low energy, while Green Day's "American Idiot" has high energy.

[2]See the Spotify Web API.

- `instrumentalness`: How likely the track is to be instrumental (have no words). For example, a symphony would generally have high instrumentalness since it has no words, while a rap song would generally have low instrumentalness.
- `liveness`: How likely the track is to be a recording of a live performance.
- `loudness`: Average loudness of the track in decibels.
- `speechiness`: How speech-like the track is. For example, a podcast would have high speechiness, while an Adele song would have low speechiness.
- `valence`: How cheerful the track sounds. For example, the Weeknd's "Often" has low valence, while One Direction's "Live While We're Young" has high valence.

Genre: Genre(s) associated with the song's artist[3] according to Spotify. Since the genres provided by Spotify were extremely granular (with only a few artists in the dataset associated with each genre), we picked a few broader genre categories and categorized each song accordingly. Our broader genre categories are trap, hip-hop, indie, punk, rap, jazz, pop, metal, country, folk, bluegrass, house, rock, classical, and funk. For example, a song associated with "Philly rap" would be categorized as "rap".

Popularity (target variables): The ultimate popularity of the song.

- `peak.position`: Peak position that the song reached on the Billboard Hot 100 (with 1 being most popular, 100 being less popular).
- `weeks.on.chart`: Total weeks that the song spent on the Billboard Hot 100 chart.

Most of the data was sourced from components.one; lyrics were scraped from Genius and genre was scraped via the Spotify Web API.

4 Exploratory Data Analysis (EDA)

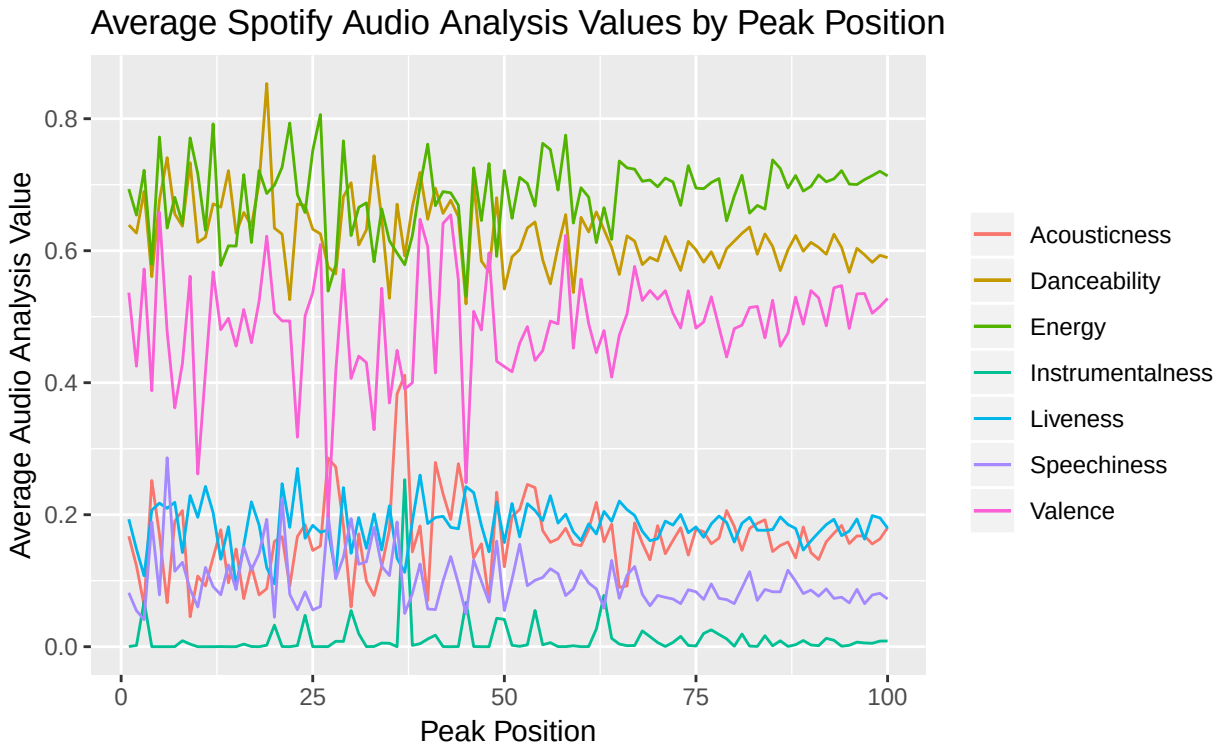
First, let us examine our target variable, `Peak.Position`. The plot is provided in the Appendix. It looks like relatively few songs in the dataset reach the top 50 compared to the total number of songs that reach top 100. In other words, our dataset is unbalanced with regards to peak position – we will need to keep this in mind going forward.

Given that we're predicting a song's popularity in terms of its peak position on the Billboard chart, we might wonder how various factors are correlated with a song's peak position.

All Spotify audio analysis variables are scaled from 0 to 1 with the exception of loudness, so we plot

[3] Ideally, we would have collected genre for each song. Unfortunately, the Spotify API only provides genre by artist, not by song.

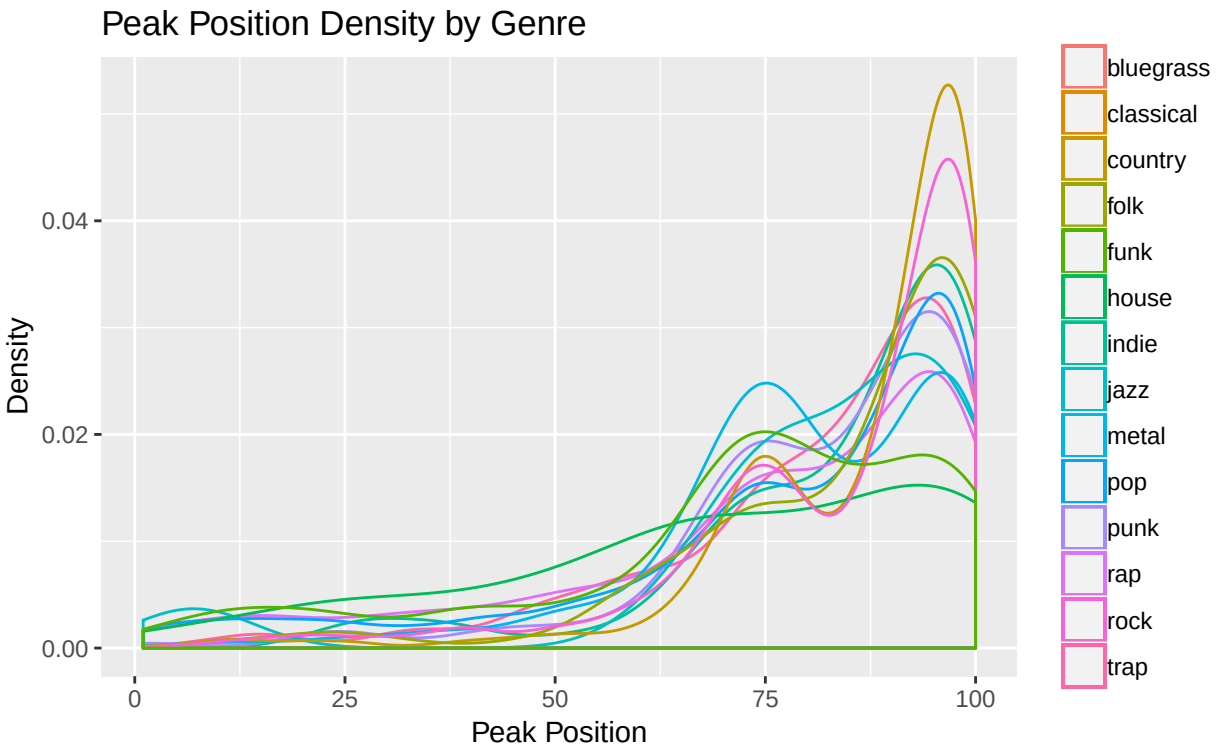
them together below:



As shown in the plot above, there do not appear to be strong trends relating any Spotify audio analysis to the peak position of a song; each audio analysis variable seems to take roughly constant average values for all peak positions.

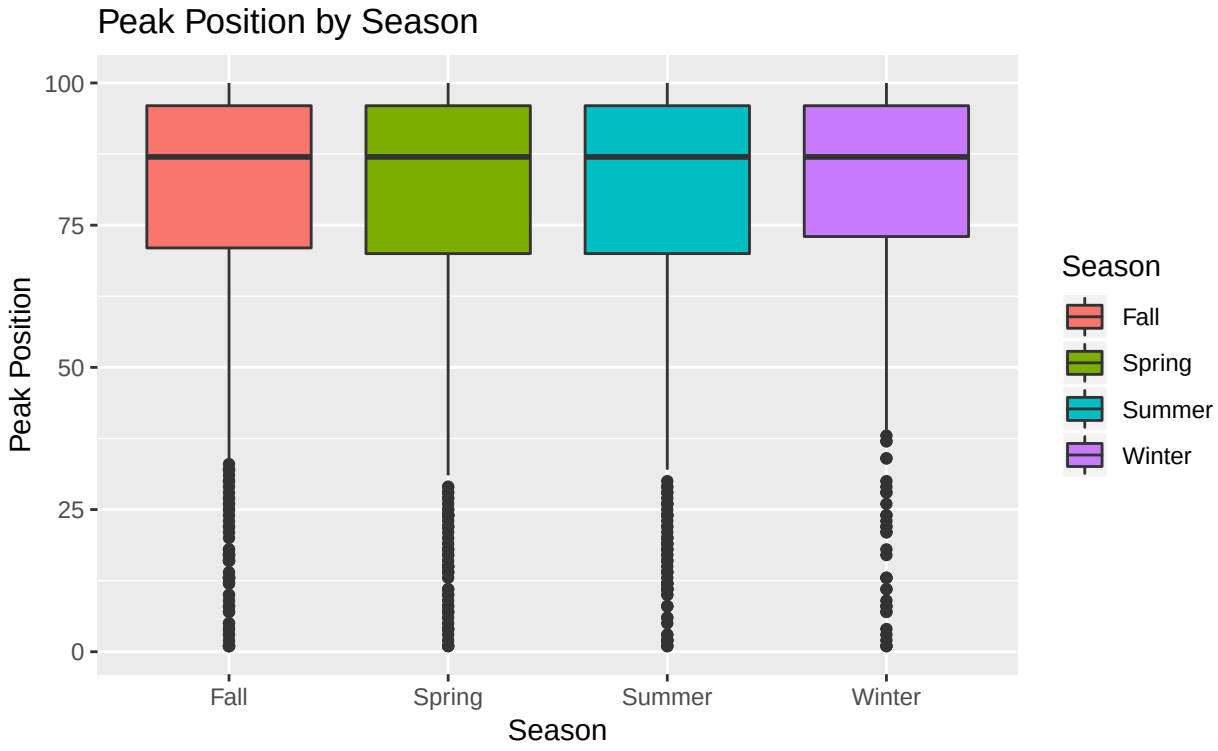
We can also plot the one remaining audio analysis variable, loudness. We do not include the plot here, but the plot can be found in the Appendix. Again, there does not appear to be a clear trend relating peak position and song loudness. There is clearly additional variance in loudness among higher peak positions, but this is likely due to the fewer number of songs with high peak positions.

We might also wonder how genre plays into song popularity.



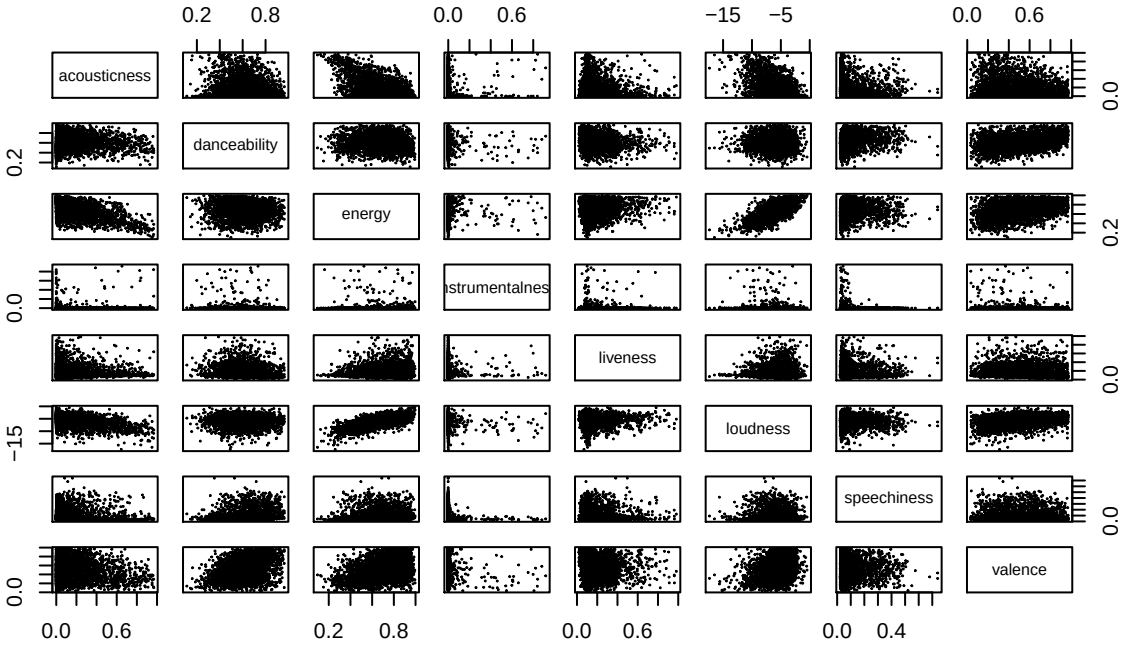
It looks like most genres are alike in that most songs within the genre peak at lower positions. However, it seems like songs associated with the rock and country genres tend to take lower peak positions compared to jazz and house music.

We can also examine how peak position changes with season.



It looks like the distribution of peak positions is roughly the same among all four seasons.

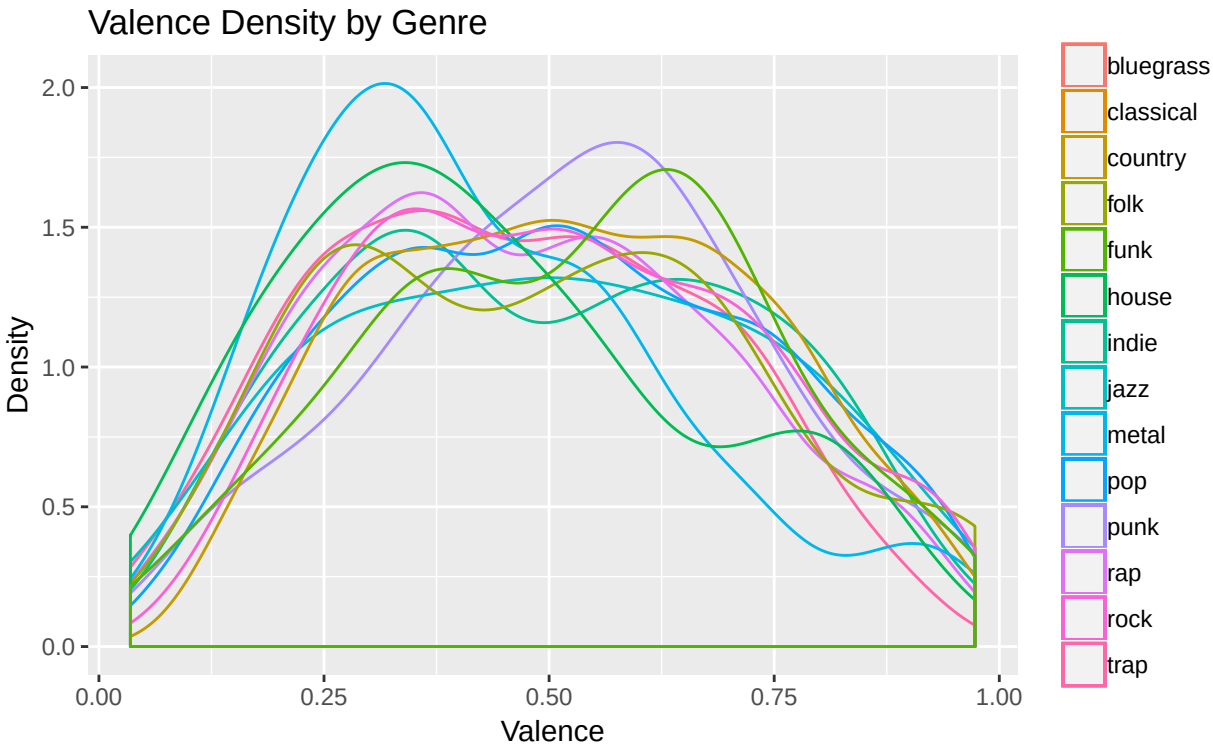
Finally, we will investigate the correlations between the variables in our dataset.



It looks like most pairs of Spotify audio analysis variables are not too strongly correlated, with the exception being energy and loudness. In particular, the correlation between energy and loudness is 0.72. Given that the correlation is relatively high, we will remove energy from the model and keep

only loudness in order to preserve model interpretability.

We might also expect the Spotify audio analysis variables to be somewhat correlated with the genre.



Above, we see that metal tends to have low valence compared to punk, funk, and country. To conserve space, the plots relating other Spotify audio analysis variables to genre can be found in the appendix. In summary, most audio analysis variables do not seem to take notably different values among different genres. The exceptions are energy, where the punk and pop genres have much higher energy compared to indie music; and danceability, where metal takes far lower values compared to trap.

This might seem surprising, but it is likely due to the fact that the "genre" of a song in our dataset is any genre that the song's artist is associated with. Since some artists create music across multiple genres, the "genre" variable is imprecise.

5 Analysis

5.1 Regression Analysis

5.1.1 Linear Model Analysis (OLS)

We first employed a linear model to approach our goal of predicting a song's peak chart position. A quick glance at a histogram of peak chart position showed that the response variable's distribution exhibited a strong negative skew and would not lend itself well for a linear regression analysis in its current form. To address this issue, we considered a variety of transformations including taking the natural logarithm, squaring the data, and reversing the data such that highest charted songs were represented by 100 instead of 1 and then taking the natural logarithm. As the histogram and Q-Q plot indicate, the last transformation left us with the most normal distribution of our response variable, and we proceeded accordingly.

To generate our model, we used the *regsubsets* package and ran backwards selection on all the predictive variables in the dataset excluding `Year` for interpretability reasons. The model that contained 11 variables minimized C_p of the 26 that we generated. From there we manually performed backwards selection to remove variables until all those remaining were significant at the 0.10 level. After two iterations we were left with a model containing 9 variables: `danceability`, `duration_ms`, `valence`, `artist.pop`, `trap`, `indie`, `country`, `folk` and `funk`. Three of the variables came from our table of Spotify audio qualities while the rest either represented the song's tagged genre or information about the artist.

As we can see from the summary output, our model produced an R^2 of 0.10, indicating that our linear representation explained very little of the variation in the response variable.

It quickly became apparent that our linear model would be an inadequate representation of our response variable, particularly when attempting to predict which songs would be the most popular. Our predicted values on the training dataset ranged from 10.48 to 95.32. Although over 50 entries in the dataset had peak positions of at least top 10, our fitted model was not able to detect the differences between the best songs and the rest of the dataset. Unsurprisingly, our mean squared error generated from the training data was much higher than desired at 524, suggesting that the model's estimates missed the true positions by around 23 places on average. The performance on the testing dataset was only slight worse with an MSE of 543.

5.1.2 Random Forest (Regression)

Random Forest was the last method we employed from the family of linear models. To tune our parameters, we analyzed how OOB testing error changed with the number of trees from 0 to 500 with a fixed mtry of 10. As the plot indicates, the error measurements stabilized around ntree = 200. We then analyzed the MSEs of Random Forests generated with mtry values ranging from 1 to 15, ultimately choosing mtry = 10 as it minimized the OOB error. At approximately 234.

When applied to the training data our model generated an MSE of 42, but it quickly became apparent that our Random Forest was considerably overfitting in light of the testing MSE of 450. Although the performance on the testing data was still underwhelming, it was still an improvement over our other linear methods.

5.1.3 Decision Tree

Decision trees provided another approach to modeling a song's predicted chart position. We fit regression tree to predict the untransformed Peak.Position variable on our training data, employing all the variables in the dataset. For this segment of analysis, we included the Year variable as a factor so that we could capture the interaction between the year a song was released and the other audio and genre measures. The tree using all available predictors chose 14 splits on the following nine factors: `country`, `Year`, `artist.pop`, `Season`, `key`, `duration_ms`, `instrumentalness`, `valence`, and `loudness`. The predicted values from the model ranged from 33 to 91 and generated an MSE of 378 on the training data. When applied to the testing data, the MSE rose to 488, a considerable improvement over the least squares' regression model.

5.2 Classification Model Analysis

The obvious inadequacies of the linear regression models necessitated that we consider new approaches to predict a song's popularity and chart performance. The category "Top-40" is a commonly used grouping to differentiate the most popular songs from others, even within the Billboard Top 100. Songs that are charted in this echelon are commonly included in popular playlists and garner considerable radio playtime. As such, we saw practical application value in capturing the differences between Top-40 entries and songs that were on the brink of widespread recognition but just fell short.

One concern with this approach was that there was how to address the severe imbalance between the songs in our dataset that were Top-40 and those that were not. Only 8% of the entries in our training data fell into our target category, which we knew would drastically hamper our models'

classification abilities. To combat this issue, we oversampled the Top-40 songs from our training dataset such that the distribution for our response variable was evenly split between positive and negative indicator values, resulting in 2137 entries for both Top-40 and non-Top-40 songs. The testing dataset was left untouched.

5.2.1 Logistic Regression

To begin we fit a logistic regression to predict our dummy variable top40. We then removed the coefficients with the highest p-values until there were only 15 remaining, the maximum number of variables accepted by the bestglm package. The best model found by the exhaustive search contained nine variables, all of which were significant at the 0.05 level. To formulate classification predictions, we employed a threshold of 0.5. Given an even split between positive and negative values in our response variable, we achieved a training MCE of 0.32 from our logistic regression model. Unfortunately, the model performed much worse on our testing dataset, reporting a testing MCE of 0.35 when only 8.8% of the data points were actually in the Top-40. While we could have trivially achieved a much better MCE by predicting negative values for all our data points, our confusion matrix showed that we had a sensitivity 0.70, indicating that were able to classify a considerable portion of the positive values. The false positive rate of 0.82, however, left much to be desired form our model.

5.2.2 Decision Tree

The classification tree fit on all variables generated 16 splits on a subset of 11 variables from 28 made available. Applying our threshold of 0.5 once again, our tree gave us a MCE of 0.22 on the training data and 0.38 on the testing data. Overall, the testing results were much worse for our decision tree when compared to the logistic regression. As can be concluded from the confusion matrix, sensitivity decreased to 0.68 while the false positive rate climbed to 0.86.

5.2.3 Random Forest

The last classification model employed on the audio, genre, artist and release date variables was a Random Forest fit on all variables. The model output indicates that our OOB estimate of error rate was approximately 0.26%, but those results did not translate to the testing data. Our MCE was approximately 0.08. Given that 8.8% of the underlying test data had positive response variables, we would have only slightly beat the MCE from a model that predicted all negative values. Furthermore, the model only correctly identified 12 Top-40 songs, making for a sensitivity rate of 0.14. Needless

to say, the Random Forest fit did not provide a robust model of our dataset, but it had the lowest false positive rate of the three models at 0.40.

6 Text Mining

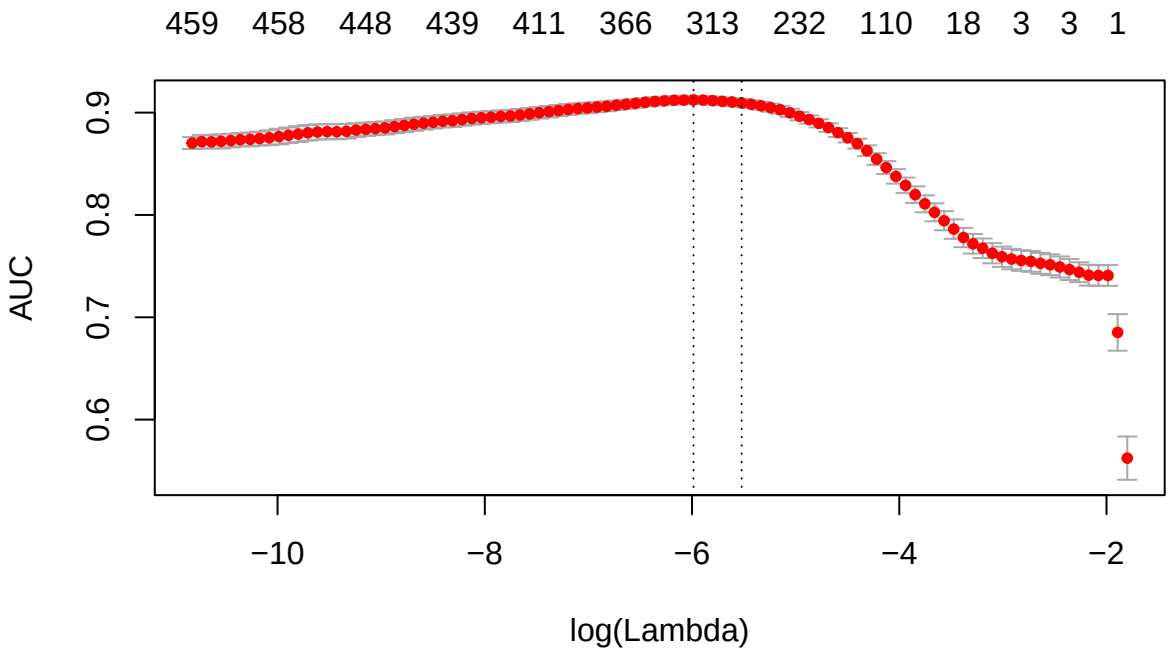
During our process of building the Random Forest Model, we realized that we can significantly enhance our dataset by combining each song with its respective lyrics. With the lyrics data, we hypothesize that there are certain words/topics that would make a song popular enough for the Billboard Top 40.

6.1 Data Collection

First, we used Python and the Genius API (genius.com) to search for song lyrics and match them against our dataset. The Python code for this can be found in Appendix 1. There were a couple of songs that the Genius API cannot search data for so we did that manually by hand and input the lyrics in. Next, we merged this lyrics data with the original `df2000` dataset and we are ready to perform text mining.

6.2 LASSO

The first step in doing data analysis is to clean up the text of the lyrics. We performed the standard corpus transformations such as removing stop words, punctuations and stemming words. Then we decided that we want to focus on words that appear on at least 5% of all documents. There are roughly 435 words that were chosen depending on the seed that we set it to. We also faced the same problem as before of having too many rows that are not in the Top 40 versus rows that are in the Top 40. So like before, we sample with replacement from the training dataset so that the dataset that we use for analysis has roughly equal number of Top 40 verus non-Top 40 songs. With this, we ran a LASSO model with $\alpha = 1$ and error criterion being AUC. Here is the plot of the LASSO model:



The reported minimum Lambda from this model is 0.002513537. Using the untouched testing set, we calculate the testing error for the LASSO model to be 0.246988. Next, we wanted to use the reported betas from the LASSO model to pick out the non-zero ones and refit them into a glm model. The misclassification error for the testing dataset is 0.2188755. For a full list of words that were chosen by the glm model, please see Appendix 2. The full confusion matrix for the glm model can be seen below:

```
##
## class.glm >= 40 Top 40
##    >= 40    749     70
##    Top 40   148     29
```

From the confusion matrix above, we can see that our sensitivity statistic is $\frac{29}{99} = 0.29$ and our specificity statistic is $\frac{749}{897} = 0.84$. The model does not perform so well on determining if a true Top 40 song is going to be in the Top 40. However, it seems to perform better on predicting songs that are not in the Top 40. Although the testing error for the glm model did improve from the LASSO model, we want to run boosting on this dataset specifically to increase the strength of the weak predictors. We have many predictors but none of them are particularly strong based on the results from the previous models, so boosting could potentially make these predictors stronger at predicting the true positives.

6.3 Boosting

For boosting, we used Adaboosting with tree depth of 3 and 100 rounds. This produces a training error of 0.008196721 and a testing error of 0.1556225. Below is the full tree after boosting 100 rounds.

```
n= 5124
node), split, n, loss, yval, (yprob)
      * denotes terminal node
 1) root 5124 0.471875000 1
   2) mean< 1.5 4877 0.437412500 1
     4) everi>=3.5 53 0.000000000 1 *
     5) everi< 3.5 4824 0.437412500 1
      10) leav>=1.5 201 0.007429064 1 *
      11) leav< 1.5 4623 0.429983500 1 *
   3) mean>=1.5 247 0.017020600 2
     6) danceability< 0.5725 32 0.000000000 1 *
     7) danceability>=0.5725 215 0.008417066 2
      14) speechiness< 0.0456 15 0.000000000 1 *
      15) speechiness>=0.0456 200 0.004193916 2 *
```

From the tree above we can see that the words that start with “mean”, “everi”, and “leav” are significant in classifying whether a song is in the Top 40 or not, while other characeristics such as danceability and speechiness are also significant.

7 Conclusion

In conclusion, we performed various numbers of models on the Billboard Top 100 songs dataset from 2000 and after. We combined the dataset with the Spotify characteristics and Genius’s lyrics data to enhance the performance of the dataset. Further, we found that there are a few main features such as “danceability” and “speechiness”, combined with lyrics that include words such as “everi” and “leav” that would create a Top 40 song. Despite the MCE being relatively low, the dataset did suffer from the lack of songs that were in the Top 40 relative to songs that are not in the Top 40. Because of this, our models produced different subsets of significant predictors.

Although our models fell short of achieving the predictive power for which we had hoped, we hope that the analysis of the significant variables within the models can provide some insight into what makes songs popular.

7.1 Limitations & Further Studies

Due to time, length, and knowledge constraints, our project faced several limitations that we believe could be improved in further iterations. Below, we discuss the limitations of our study and suggest ways to eliminate these limitations in future studies.

- **Improve genre classification.** In our study, we handpicked fifteen broader genres in which to categorize each subgenre provided by Spotify. Editing these fifteen genres or adding additional genres could improve the predictive power of the `genre` variable. In addition, it would be nice to obtain genre by song rather than by artist.
- **Include additional factors relating to the artist.** From a qualitative perspective, much of a song's popularity seems to be related to the fame of the artist. We attempted to capture some of this relationship via the artist popularity variable (which turned out to be significant), but we think it could be useful to include other metrics of artist fame such as number Google mentions or net worth.
- **Try other techniques to improve the imbalanced dataset.** As mentioned in previous sections, we faced significant difficulty building our classifier because our target variable was so imbalanced. We attempted to use bagging, boosting, and resampling methods to address this imbalance; in future studies, we would like to try out additional resampling methods and/or try models that adapt to given prior probabilities.
- **Try applying similar models to predict whether a song will be on the Billboard Hot 100 or not.** In this study, we attempted to predict a song's ranking *within* the Hot 100. However, in reality, all of the Billboard Hot 100 songs are quite popular, so there might not actually be a significant difference between a song in the bottom 60 of the Hot 100 versus the top 40; therefore, it might be difficult or impossible to create a model that accurately predicts a song's ranking within the Hot 100. Using a similar approach to determine whether a song will make it into the Hot 100 or not might yield better results, since there is likely a larger difference between an extremely unpopular song and a song in the Hot 100.

8 Appendix

8.1 Python code for pulling lyrics data**

```
import lyricsgenius
import csv

genius = lyricsgenius.Genius("***")
with open('output.csv','w') as file:
    with open('df2000_grouped_morefactors.csv', 'r') as csvfile:
        readCSV = csv.reader(csvfile, delimiter=',')
        next(readCSV, None)
        for row in readCSV:
            song = row[1]
            artist = row[2]
            curr = row[22]
            if len(curr) == 0:
                lyric = genius.search_song(song, artist)
                if lyric is not None:
                    edited = lyric.lyrics.replace('\n', '')
                    file.write(edited.replace(',', ''))
                    file.write('\n')
                else:
                    lyrics.append(" ")
        csvfile.close()
    file.close()
```

8.2 Coefficients produced by the glm model**

```
##      (Intercept)     acousticness     danceability      duration_ms
##    -6.328368e+02    -3.467048e+00     6.303548e+00     1.513296e-05
##           energy instrumentalness         liveness            tempo
##    -2.318463e+00     1.473740e+00    -2.454591e-01    -1.708527e-02
##          valence       artist.pop             trap            indie
##    -6.815927e-01     3.910715e-01    -5.314248e+00     1.713212e+00
##             punk              rap             jazz            metal
```

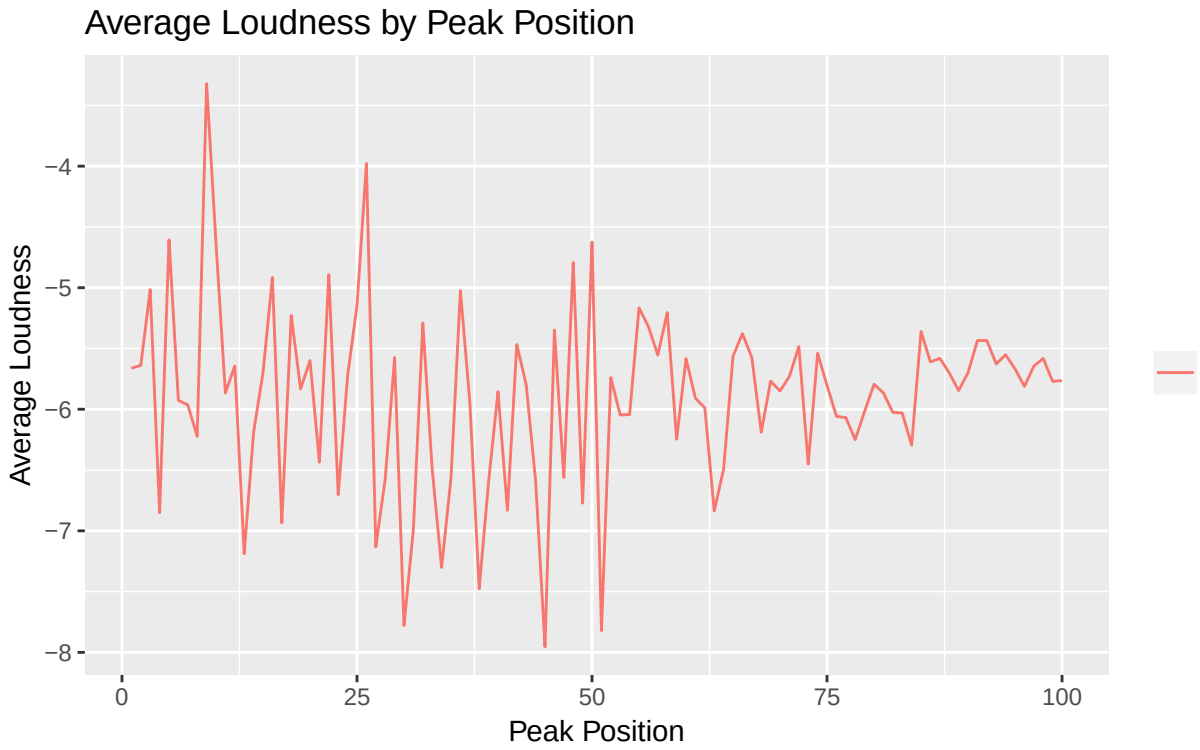
```
##    -4.132947e-01     8.527187e-01     5.799968e+00    -2.156025e+00
##         country             folk            house        classical
##    -6.100864e+00    -6.719019e+00    -2.940715e+00    -1.066406e+01
##            funk             Year              act           afraid
##     2.980729e+00     3.143532e-01     6.419113e-01    -6.689852e-01
##            aint             alon          alright            alway
##    -5.987339e-01    -8.996464e-01    -3.338672e-01    -2.543705e-01
##           anoth            anyth             ask             ass
##    -3.308488e-01    -7.463286e-01    -4.158638e-01     1.356251e-01
##            away             babi             bad            beat
##    -5.024460e-02    -2.060658e-01    -3.978885e-01     8.254368e-01
##          beauti              bed          behind           bitch
##     4.859752e-01     3.809838e-01     1.239701e+00     2.602304e-01
##           black             blow            blue            bodi
##    -4.901743e-01    -3.935843e-01    -6.267070e-01    -7.159591e-01
##            boom           break.          breath            burn
##    -1.587443e+01     4.341093e-01     5.200493e-01     1.213132e-01
##             buy             call           can.t             car
##    -3.926199e+00     2.515081e-01     1.144442e+00    -1.346383e+00
##            care            catch           caught            caus
##     3.048613e-01     5.491207e-01    -1.654185e-02    -3.185379e-01
##           chain            chanc            chang           chase
##    -3.724317e-01     1.465335e-01     1.657689e-01    -1.515308e+00
##           check           chorus             citi            club
##    -5.562071e-01    -2.520367e-01    -7.019690e-01    -5.766107e-01
##            cold             come             cool           count
##     6.504812e-01    -1.475766e-03    -1.803848e+00     9.986950e-03
##         countri              cri              cut            damn
##    -2.939408e-01     2.350085e-01    -3.692069e+00    -3.819618e-01
##            danc             dark             deep         diamond
##    -6.699291e-01     7.831965e-01    -2.426571e+00    -9.089308e-01
##             die             door            dream           drink
##    -1.008925e+00    -1.104132e+00    -7.069920e-01    -3.138382e-01
##            easi              end           enough            even
##     5.739483e-01     5.914413e-01    -8.333779e-01     6.246590e-01
##            face              far             fast            feel
```

```
##     6.599601e-01     4.195250e-01     2.020120e+00    -3.248601e-01
##           feelin             feet             felt            fight
##     1.932885e-01     4.437444e-01     9.791183e-01     5.673326e-01
##             fill             find             fire            first
##     2.432676e-02     3.263846e-01     1.224177e-01     9.993700e-01
##              fli           follow            forev           forget
##     2.914583e-01     1.089145e+00     5.789886e-01    -5.655674e-01
##            found             fuck           fuckin             full
##     6.981237e-01     5.046088e-01     1.243609e+00    -1.811050e+00
##             game             gave              get             give
##     7.607572e-01    -1.604483e+00    -1.241876e-01     1.715121e-01
##              god             goin             gone             good
##     2.681411e-01     3.283394e-01    -8.174350e-02    -3.282588e-01
##              got             grow              gun             hair
##     1.343915e-01    -4.681949e-01     2.117144e-02     5.482272e-01
##             hand             hang           happen             hard
##     4.070493e-01     2.586540e-01    -1.293325e+00    -4.923923e-01
##             hear             hell             help             high
##    -5.558581e-01     2.587166e-02    -1.941146e+00     1.877569e-01
##              hoe             hold             home             hook
##    -1.693206e+00    -1.841443e-02    -1.056008e+00    -1.313586e+00
##             hope             hous            hundr             hurt
##     4.400841e-01    -7.262599e-01    -6.270377e-01    -1.853443e-01
##            insid             just             keep              kid
##     1.111116e+00    -4.339585e-02     1.104354e-01    -3.859392e-01
##             kind             kiss            knock             last
##     6.765148e-01    -2.373422e-01     1.202419e-01    -2.138676e-01
##             late             leav             left              let
##     2.661527e-01    -8.753982e-01    -9.789417e-01    -1.010503e-01
##              lie            light              lil              lip
##    -9.210520e-02    -2.457469e-01     7.821901e-01    -1.406728e+00
##             lone             look             lose              lot
##    -1.115867e+00     7.509195e-02     3.418052e-02    -2.834401e-01
##             love              mad             made             make
##    -1.467377e-01    -6.608823e-01    -2.056050e-01     1.900934e-01
##              man             mani              may             mean
```

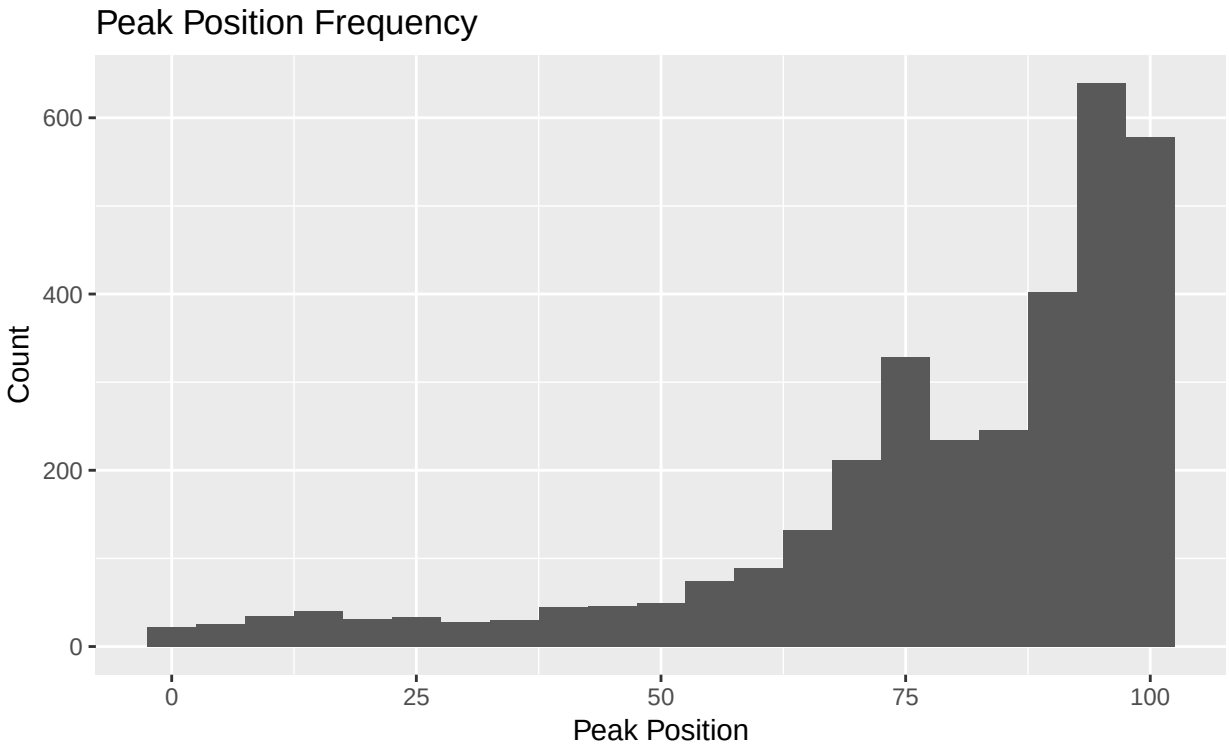
```
##    -3.711478e-01    -1.435767e+00    -1.162721e-01     3.177738e-01
##              mei              met            might             mind
##    -2.277085e-01     1.306346e+00     4.453394e-01    -4.906111e-01
##             mine             miss           moment            money
##     5.366314e-01     3.884793e-01     4.416587e-01    -2.032748e-01
##             morn             move             much             music
##    -1.422929e+00    -3.434230e-01    -3.336950e-01    -1.715308e+00
##             need            never              new            nigga
##    -7.033649e-01     2.366235e-01    -4.653226e-01    -3.145436e-01
##           nobodi           nothin              now             ohoh
##    -3.168796e-02    -9.450081e-01    -2.826592e-01    -1.879654e+00
##              one              ooh             open            outro
##    -2.655724e-01    -3.579682e-01     3.797530e-01     2.990696e+00
##            outta             pain            parti             pass
##    -5.973428e-01    -3.787453e-01     1.341981e+00    -6.005800e+00
##              pay            peopl          perfect             play
##     9.385208e-01    -3.966999e-01    -6.488988e-01     4.192458e-01
##            pop.1             pray         prechorus           promis
##     1.614370e+00    -3.172461e-01    -1.307070e+00    -1.562952e+00
##             pull             push              put              que
##     1.275398e+00     2.932697e-01    -5.887458e-01    -9.077688e+00
##             rain             read            readi             real
##    -7.947479e-01     1.981061e-01    -5.351643e-01     5.611911e-02
##           reason           rememb             ride            right
##    -1.754724e+00    -8.078867e-01    -6.777954e-01    -2.894486e-01
##             ring             road           rock.1             roll
##     7.989166e-01     3.405633e-02     6.214394e-01    -1.088763e+00
##             room             said              say            scare
##    -3.387541e+00     4.840123e-01     7.153878e-02    -4.887159e-01
##           second              see             seem             seen
##     1.132613e-01     3.542530e-01    -1.524393e+00     1.046478e+00
##              set            shine             shit             show
##     1.141643e+00    -4.996429e-01    -4.084823e-01     2.121511e-01
##             sing              sit            sleep             slow
##     5.382618e-01     4.067981e-01    -1.414495e+00    -1.856532e-01
##            smile         somebodi           someon           someth
```

```
##     4.420554e-02    -1.082509e+00     2.899400e-01    -5.699237e-01
##          somethin          sometim          song.1           sound
##     2.148959e+00     1.224452e+00    -4.297206e-01    -8.229585e-01
##           spend            stand             star           start
##    -1.210667e-01    -2.764452e-01     8.305560e-01    -5.801795e-04
##            stay             step           street           summer
##     8.148362e-02    -4.477379e-01    -1.207152e+00    -1.718443e+00
##           sweet           talkin             tast             tear
##    -6.525185e-01    -1.172683e-01    -1.162411e+00     7.342795e-01
##            tell            think            three            throw
##    -1.679958e-01     2.490609e-01     1.545246e+00    -4.677626e-01
##             til             till             time             tire
##    -5.283137e-01    -3.386171e-01    -1.138064e-01     7.743237e-01
##          togeth             told             took              top
##     6.792870e-01     4.197895e-01    -2.665249e+00     1.354436e+00
##           touch            treat              tri             true
##    -6.480814e-02    -1.022433e+00    -8.744132e-02    -1.188388e-01
##           trust            tryin            tryna             turn
##    -8.844425e-01     8.552458e-01    -7.222782e-01    -5.378840e-01
##             two              use             vers             wait
##    -1.334163e+00    -1.365302e-01    -1.009498e+00    -3.998885e-01
##            wake             wall            wanna             wast
##    -1.191256e+00     1.597536e+00     1.194948e-01    -2.327225e+00
##           water              way             well             went
##    -6.066849e-01     2.273656e-01     4.187741e-01    -7.090158e-01
##           white             will             wish              wit
##     9.164909e-01     4.436913e-02    -9.621534e-01    -3.003829e+00
##            woah             word             work            world
##    -4.387387e+00     2.327178e-01    -4.116376e-01    -6.152397e-01
##           worth            wrong             yeah             year
##    -2.925971e-01    -1.398455e-01     1.076196e-01    -1.438524e+00
##             yes           youand             youi            young
##    -9.817144e-02    -4.304239e-01     2.229494e-01     9.215671e-01
```

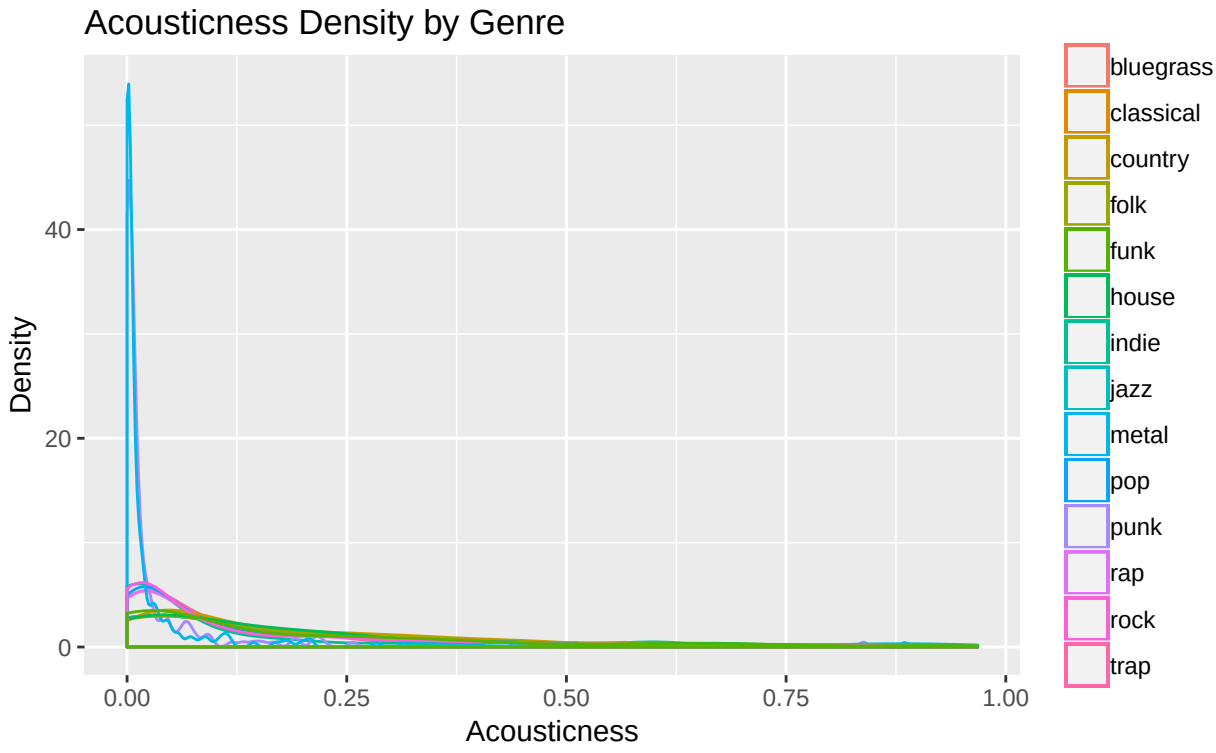
8.3 Plot Loudness

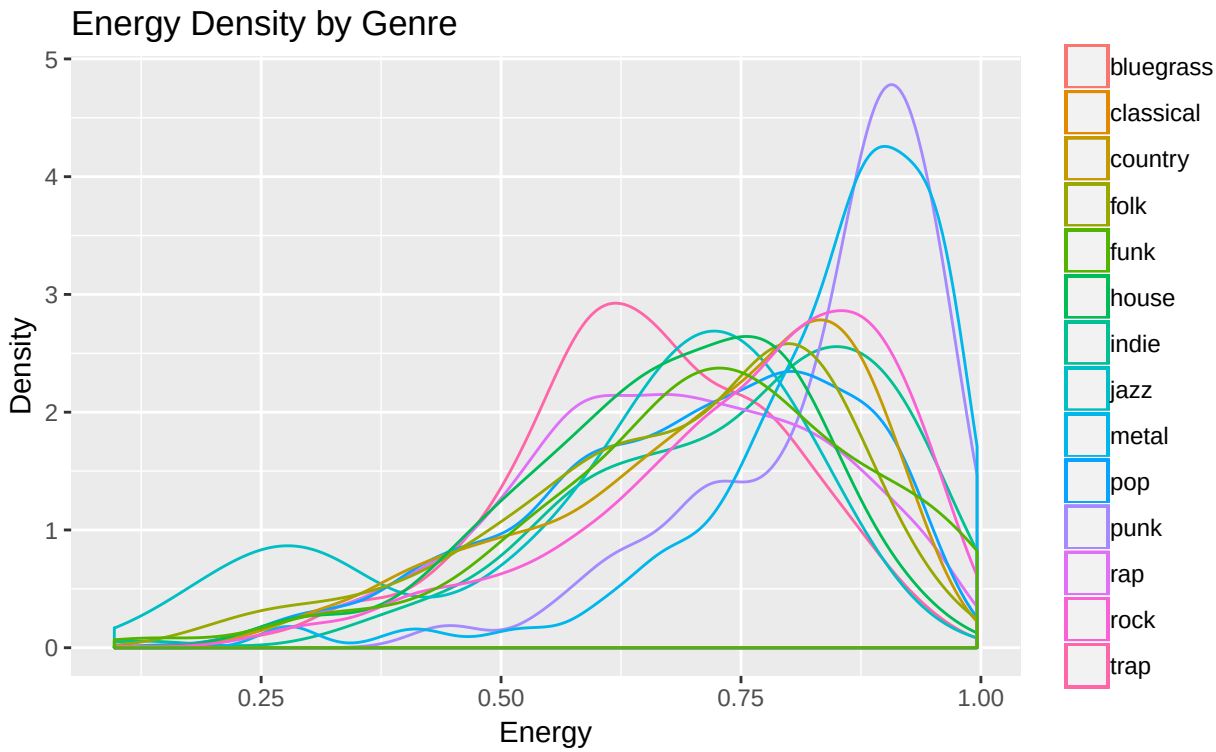


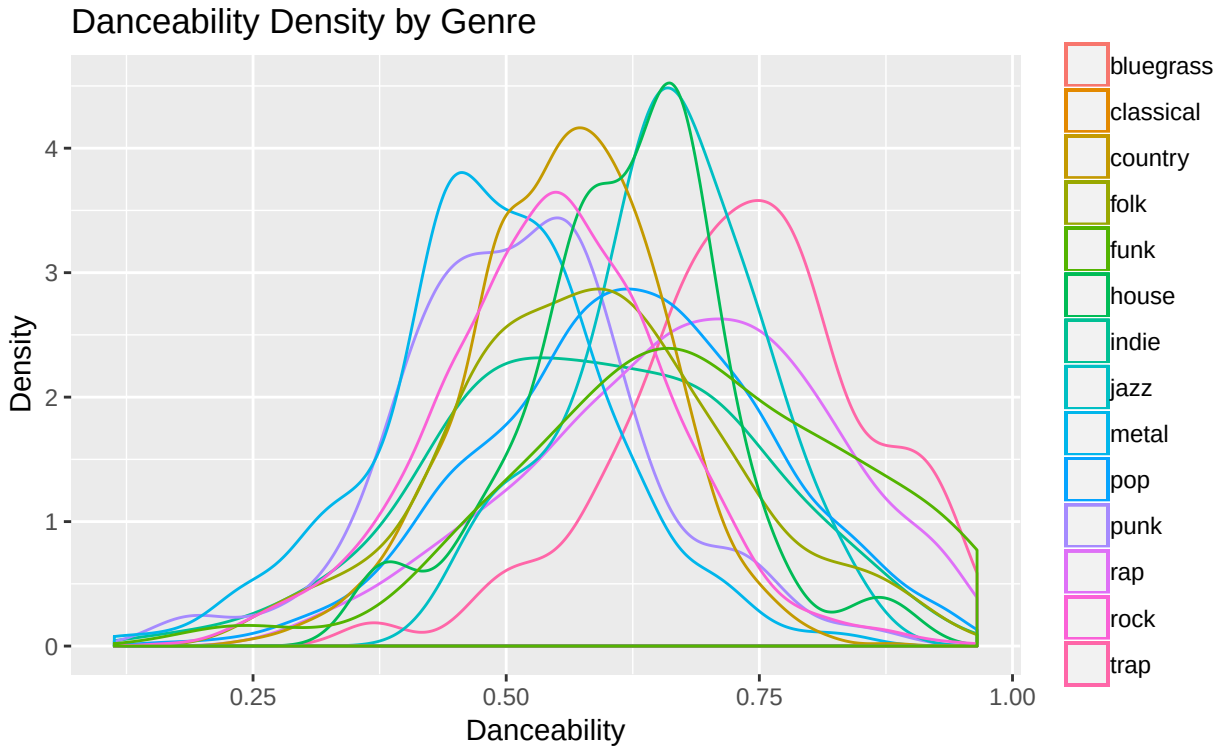
8.4 Peak Position Frequency Plot

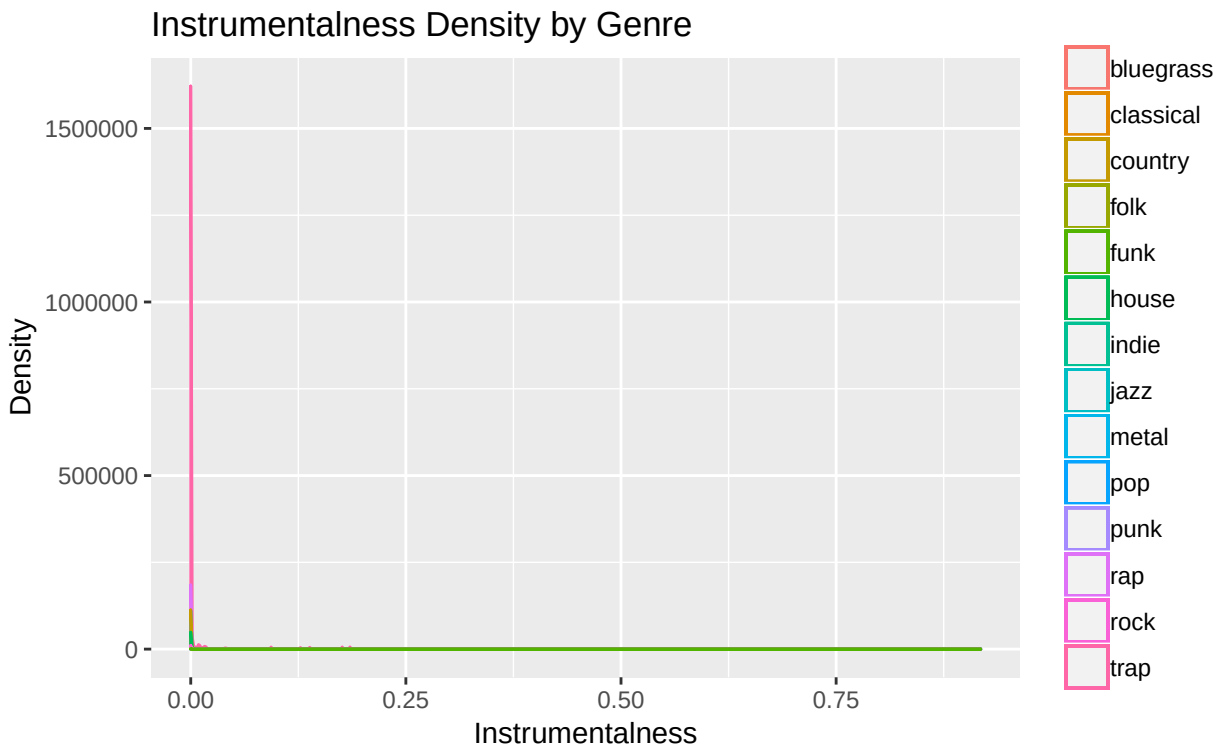


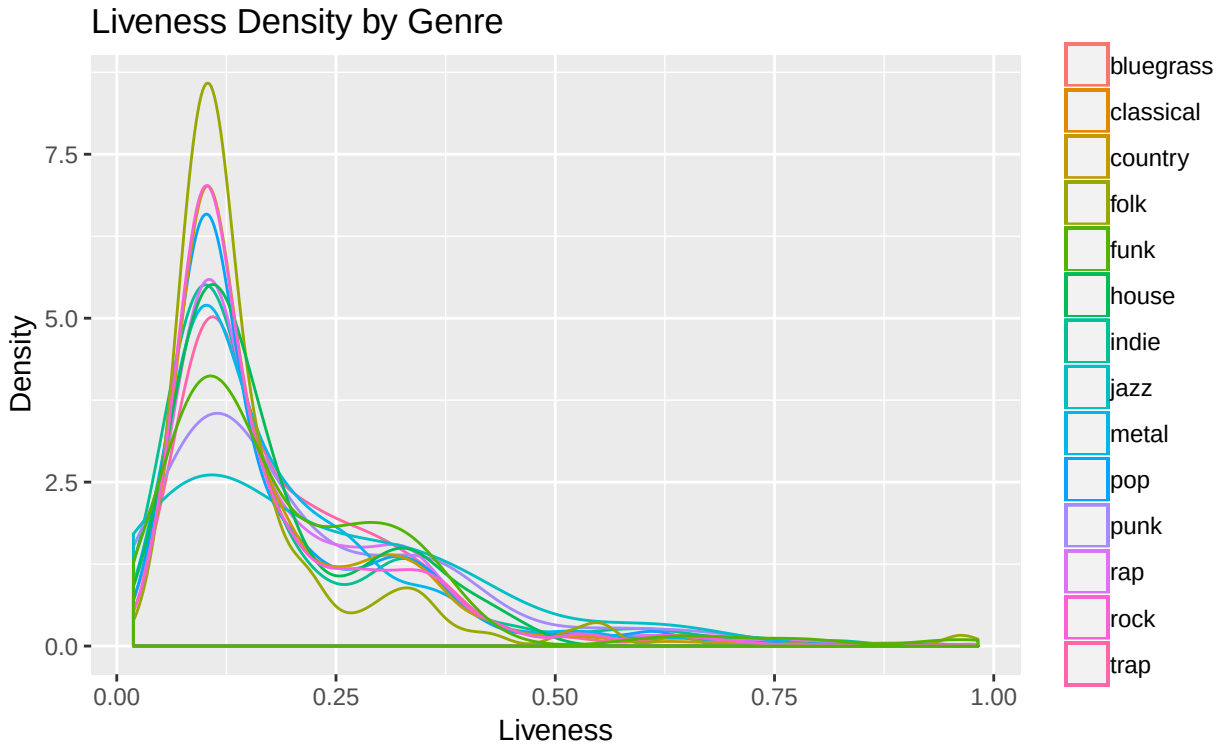
8.5 Spotify Audio Analysis Variables Density by Peak Position

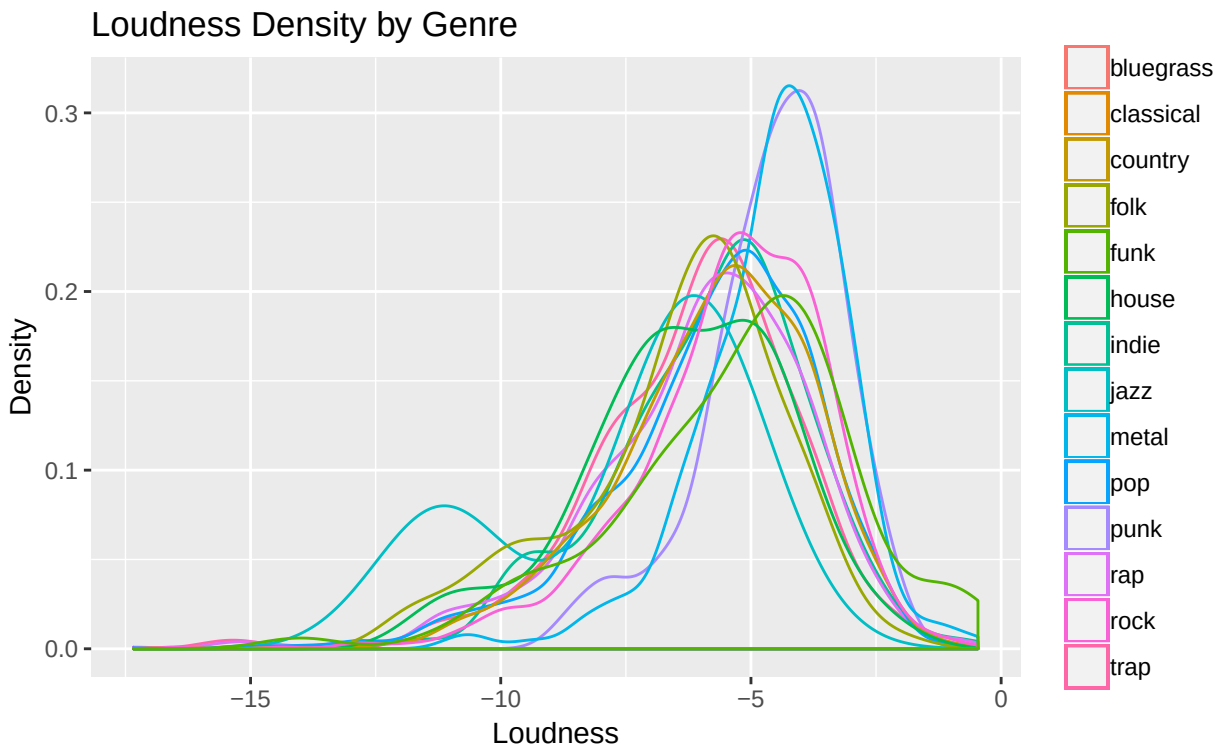


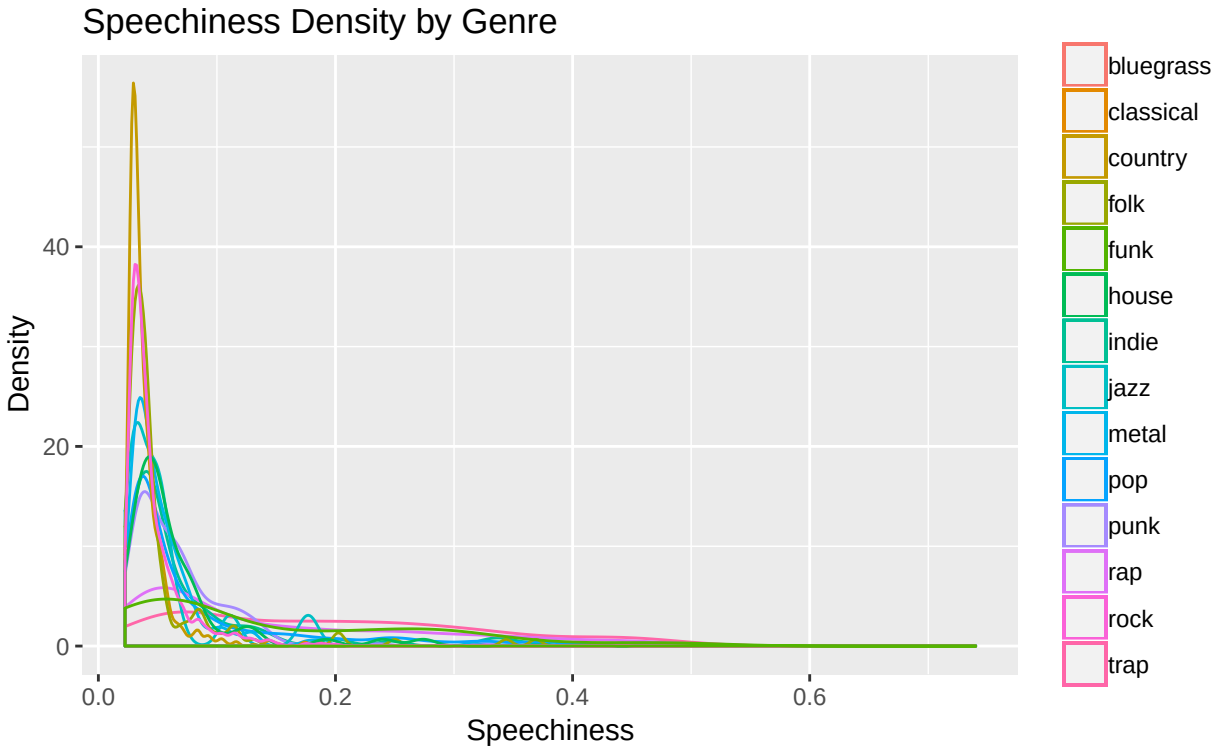



Danceability Density by Genre
Density
Danceability
0
1
2
3
4
0.25
0.50
0.75
1.00
bluegrass
classical
country
folk
funk
house
indie
jazz
metal
pop
punk
rap
rock
trap

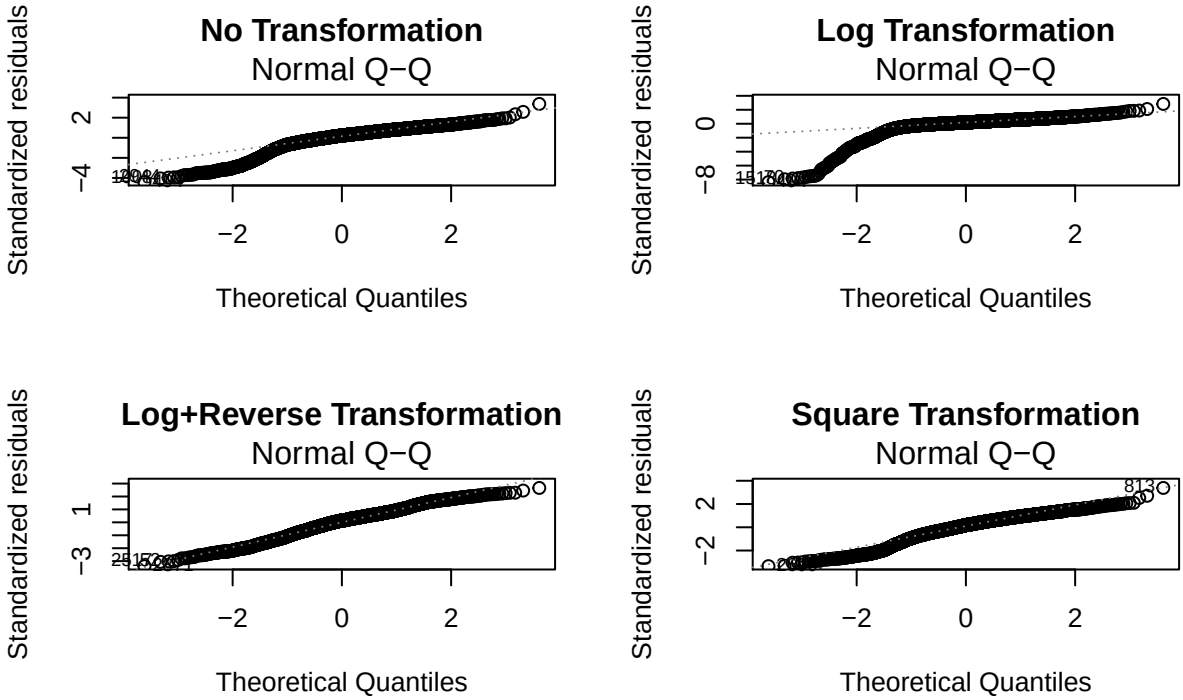

Instrumentalness Density by Genre
Density
Instrumentalness
0
500000
1000000
1500000
0.00
0.25
0.50
0.75
bluegrass
classical
country
folk
funk
house
indie
jazz
metal
pop
punk
rap
rock
trap


Liveness Density by Genre
Density
7.5
5.0
2.5
0.0
0.00
0.25
0.50
0.75
1.00
Liveness
bluegrass
classical
country
folk
funk
house
indie
jazz
metal
pop
punk
rap
rock
trap


Loudness Density by Genre
Density
0.3
0.2
0.1
0.0
−15
−10
−5
0
Loudness
bluegrass
classical
country
folk
funk
house
indie
jazz
metal
pop
punk
rap
rock
trap

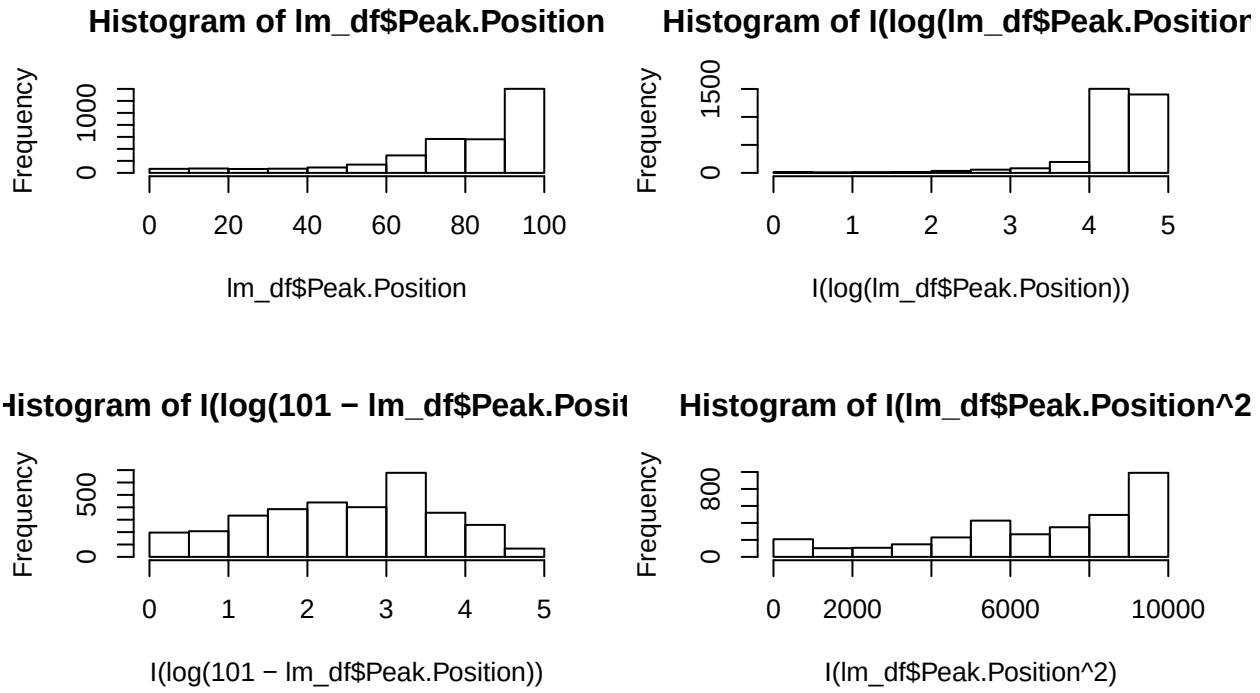


8.6 Linear Model Diagnostics




```
# Histograms of transformed distribution
par(mfrow = c(2,2))
```

```r
hist(lm_df$Peak.Position)
hist(I(log(lm_df$Peak.Position)))
hist(I(log(101 - lm_df$Peak.Position)))
hist(I(lm_df$Peak.Position^2))
```

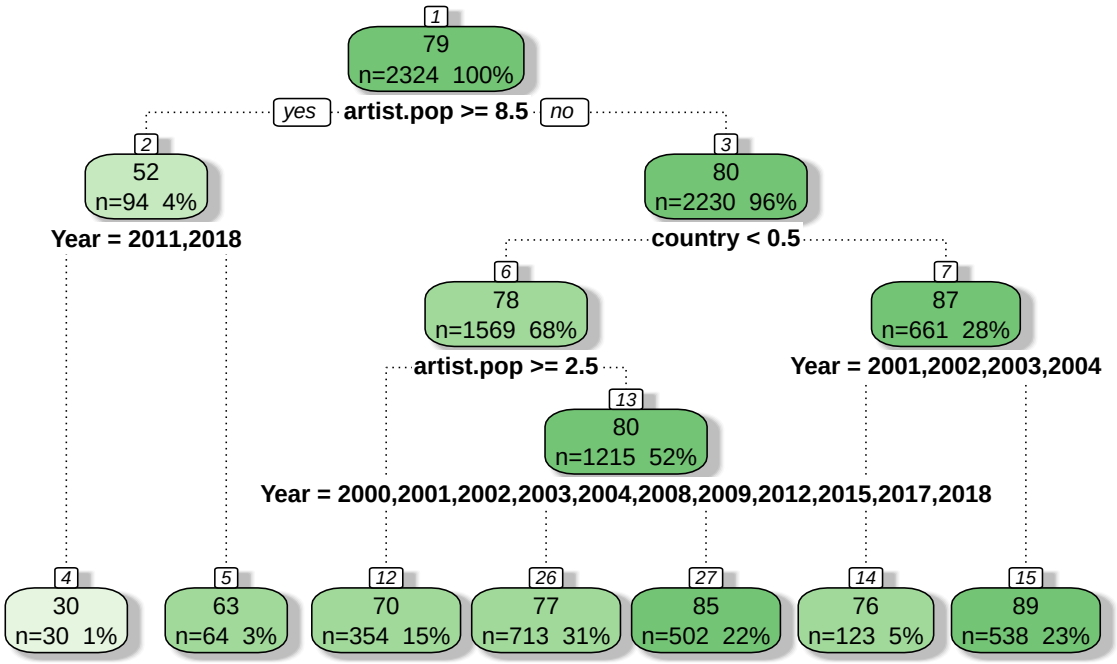


8.7 Linear Model (OLS) Backwards Selection

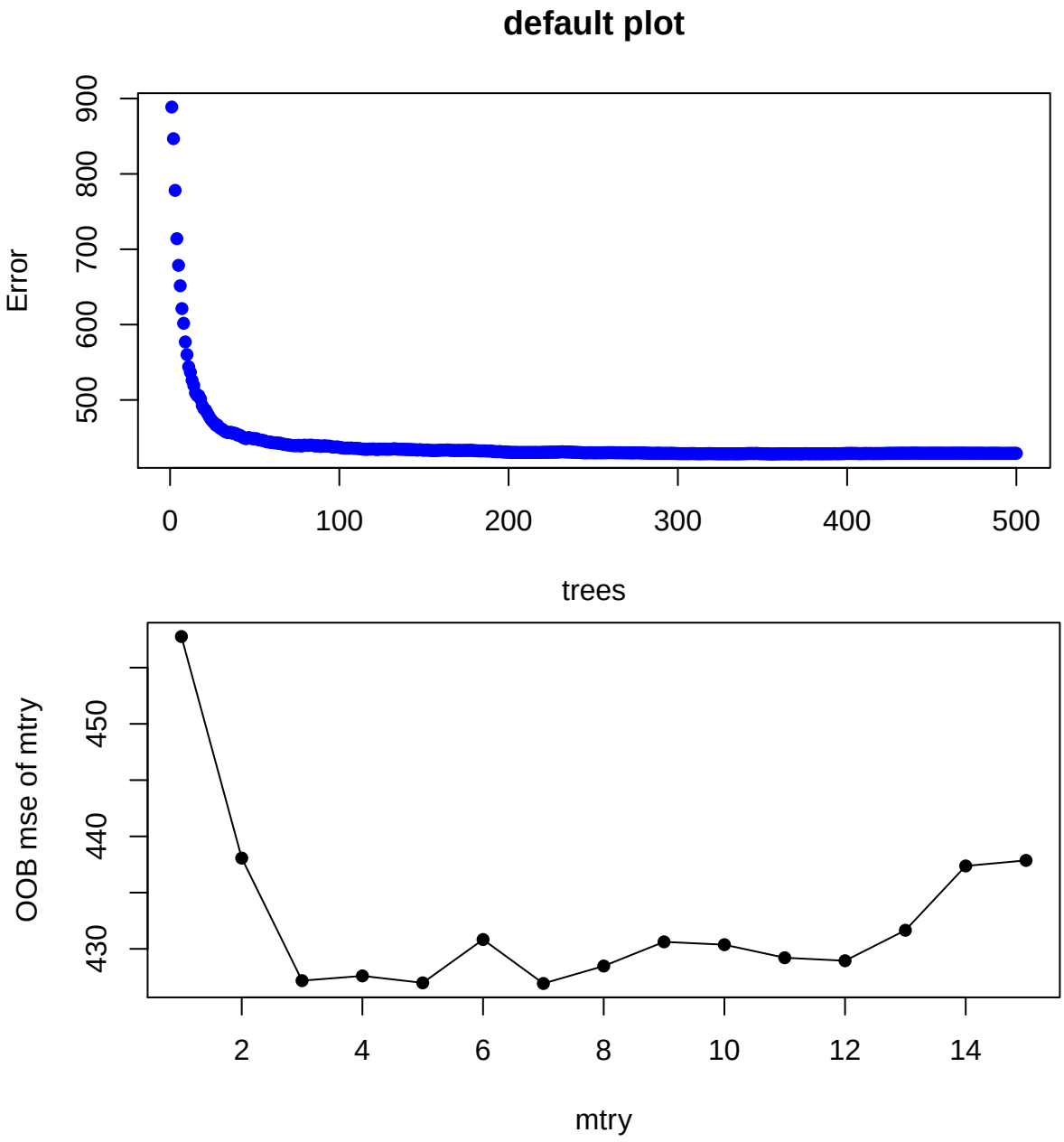
```
##
## Call:
## lm(formula = Peak.Transformed ~ danceability + duration_ms +
##     valence + artist.pop + trap + country + folk + rock + funk,
##     data = lm_df_1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.6158 -0.7705  0.1312  0.8848  2.5888
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.237e+00  1.692e-01  13.226  < 2e-16 ***
## danceability  3.499e-01  1.732e-01   2.020  0.04344 *
```

```
## duration_ms   1.123e-06  4.925e-07   2.279  0.02271 *
## valence      -1.839e-01  1.007e-01  -1.826  0.06800 .
## artist.pop    5.431e-02  5.470e-03   9.928  < 2e-16 ***
## trap         -1.927e-01  7.972e-02  -2.417  0.01571 *
## country      -5.257e-01  4.745e-02 -11.078  < 2e-16 ***
## folk         -3.343e-01  1.099e-01  -3.043  0.00236 **
## rock         -2.048e-01  4.682e-02  -4.374 1.26e-05 ***
## funk          2.213e-01  1.291e-01   1.715  0.08653 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.14 on 3310 degrees of freedom
## Multiple R-squared:  0.0992, Adjusted R-squared:  0.09675
## F-statistic:  40.5 on 9 and 3310 DF,  p-value: < 2.2e-16
```

8.8 Decision Tree (Regression)



8.9 Random Forest (Regression)

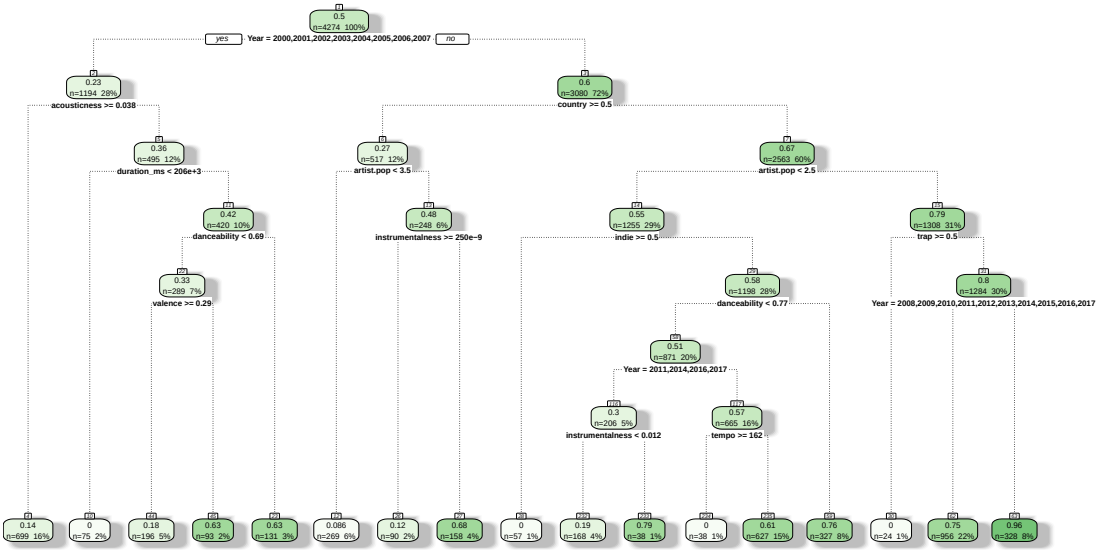


8.10 Logistic Regression

```
##
## Call:
## glm(formula = top40 ~ acousticness + danceability + energy +
##     rock + liveness + loudness + artist.pop + trap + indie +
##     metal + country + folk + funk, family = "binomial", data = train_class_os)
```

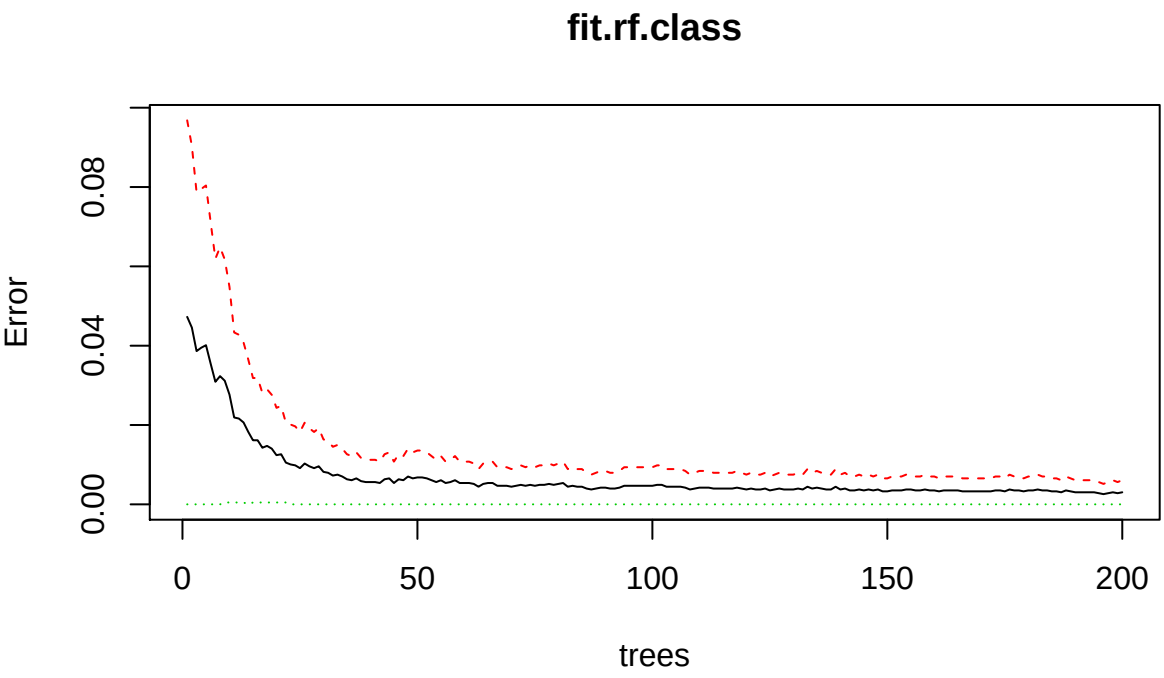
```
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -3.06814  -0.99108  -0.05553   0.99582   1.91950
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   1.48758    0.36463   4.080 4.51e-05 ***
## acousticness -0.86530    0.21686  -3.990 6.60e-05 ***
## danceability  0.93862    0.26203   3.582 0.000341 ***
## energy       -1.40096    0.32680  -4.287 1.81e-05 ***
## rock         -0.27663    0.09294  -2.976 0.002916 **
## liveness     -1.57818    0.29214  -5.402 6.59e-08 ***
## loudness      0.11709    0.02351   4.981 6.32e-07 ***
## artist.pop    0.14241    0.01091  13.059  < 2e-16 ***
## trap         -0.62370    0.13513  -4.616 3.92e-06 ***
## indie        -0.80866    0.22984  -3.518 0.000434 ***
## metal        -0.77102    0.17772  -4.338 1.44e-05 ***
## country      -1.70799    0.10348 -16.506  < 2e-16 ***
## folk         -1.23487    0.27560  -4.481 7.44e-06 ***
## funk          0.82436    0.20751   3.973 7.11e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 5925.0  on 4273  degrees of freedom
## Residual deviance: 4891.2  on 4260  degrees of freedom
## AIC: 4919.2
##
## Number of Fisher Scoring iterations: 5

##
## pred.glm.test   0   1
##             0 589  31
##             1 319  57
```

8.11 Classification Tree



Rattle 2019−May−06 00:10:46 lucy

8.12 Random Forest (Classification)




```
##
## fit.rf.pred.test   0   1
##                0 897  77
```

```
##                  1  11  11
```