Marques Chacon
Sung Min Park

Acoustic Analysis of Spotify Music Using Machine Learning

Summary of Research questions:

1. Are there any correlations between a song's popularity and its acoustic features (danceability, energy, duration, etc.)? Can we use this to predict a song's popularity based off the acoustic features? If we group by genre, are there any differences in primary factors that influence popularity?
    - This would be useful for music creators because it would indicate what kinds of features people generally prefer when listening to music, and would guide artists who are concerned about how well their music will likely be received. For artists who specialize in a particular genre, it could indicate the keys to a successful song of that particular genre, which would be more useful than with the general dataset.
    - **Answer:** No, we couldn't find any meaningful correlations between a song's popularity and its acoustic features. This conclusion was drawn specifically from pop music genre.
2. Can we train a model to classify a song's genre solely on its acoustic features?
    - An accurate model that classifies genres without the need for human input would eliminate the need to assign tags to songs, which are more dependent on user-input, which is variable from person to person. Having a standard that applies across everyone would be useful in identifying songs that users may potentially like.
    - **Answer:** Our model trained with KNN successfully detects comedy genre, but our model did not perform very well on classifying most of the genres.
3. Can we identify correlations in acoustic features in similar sounding songs recommended in a Spotify playlist?
    - This would let us know how Spotify suggests certain songs, and it could also be used for "similar" songs that they don't suggest. Since similarity is very subjective, we can learn more about the criteria that Spotify uses to recommend songs to users--For example, do they put more emphasis on popularity and artist as opposed to danceability and energy (i.e. do they put more emphasis on labels as opposed to acoustic features)?
    - **Answer:** Yes, we can identify correlations in the acoustic features of the songs that Spotify recommends.
4. How would the playlists outputted by our algorithms compare to the list of recommended songs that Spotify generates?
    - Our algorithm involves taking in a specified song parameter, calculating the Z-score for each acoustic attribute of the given song. Then, for every other song, we calculate the Z-scores across all the acoustic features. Next we will calculate

the “similarity” between two songs using two different metrics. First we will calculate the Mean Absolute Error between two songs across each of their acoustic attributes, and generate a “similarity score” (closer to 0 means more similar). The other metric will be the Root Mean Squared Error. For each metric, we will generate a playlist of songs ordered by their similarity scores, and compare to playlists that Spotify generates.
    - This is important because it will give us some insight into how Spotify recommends songs. Furthermore, our algorithm could be used as an additional, and perhaps better way of recommending songs to users (this is subjective, depending on the user, but it would provide a different metric for which people can explore music).
    - **Answer:** The songs outputted by our algorithms are based more off acoustic features than they are with the Spotify recommendations.

Motivation and background:

These research questions are important to answer because they would allow us to understand how people interpret the music that they listen to. What kinds of features do people care about? Knowing this would allow services like Spotify or other music services to cater to their users in a better way. It could also give individual people a better idea of the music that they like. Since we can export people’s playlists into csv files which can then be used for acoustic analysis, we could help people find music that they like in an easier way, based on the sounds of the music that they already have. This goes beyond just music produced by similar artists or even the same artist. Our method, if optimized, would allow people to find the most acoustically similar songs according to their individual tastes, which can transcend artist, time period, and even genre.

Dataset: We will be using two datasets: *Spotify Tracks DB* and *Spotify Playlists Dataset**.

1. **Spotify Tracks DB**: The dataset was collected by a kaggle user via Spotify API. Dataset has basic information about the song such as the song’s title, genre, artist name, track id (for spotify), and some useful metrics like the song’s popularity. Most importantly, it contains acoustic features such as acousticness, danceability, energy, instrumentalness, key, liveliness, tempo, and valence. We want to use these features to predict genres, generate playlists, or even predict the popularity of the song.
    - The dataset has 228,159 songs (rows) and 18 columns.
    - Genres: there are 26 genres. There is an approximately even distribution among different genres, except for A Capella, which has significantly small number of songs (see below)
    - The csv file is downloadable from this URL: https://www.kaggle.com/zaheenhamidani/ultimate-spotify-tracks-db/downloads/ultimate-spotify-tracks-db.zip/2

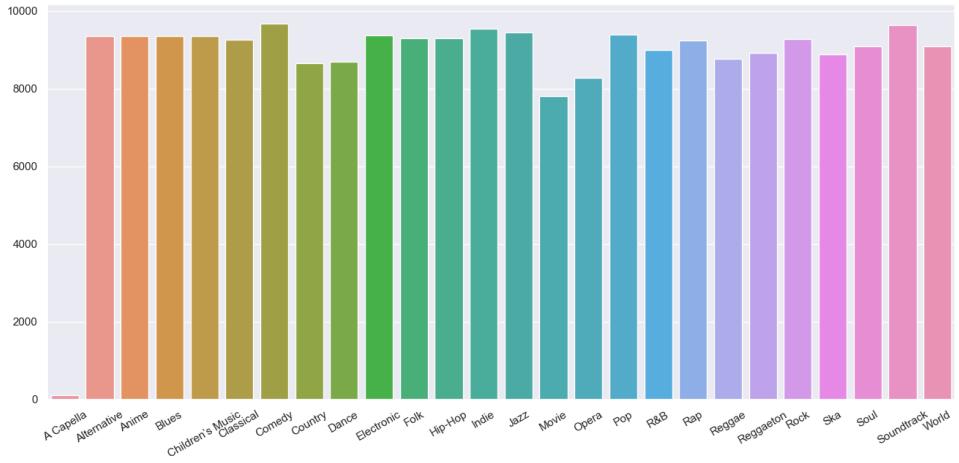

Figure 1) Count of songs for Different Genres in a Dataset

2. **Spotify Playlists Dataset** * : This dataset was unused. Instead, we created our own set of playlist data using ***Exportify***.
3. **Exportify**: This program allows us to create Spotify playlists and export the data into a .csv file. It doesn't include acoustic information, but we will merge it with our Spotify Tracks dataset, where there is information about acoustic features. This will be helpful for questions 3 and 4, where we will need to analyze Spotify's recommended songs playlist and user playlists.

Methodology (algorithm or analysis):

1. For our first research question, finding correlations between a song's popularity and its acoustic features, we will use simple correlation function (.corr) over our dataframe. This will give us a basic idea of what features correlate to popularity of the song. Then we will train two different machine learning models based on some acoustic features (that has numeric values) of the song to predict a song's popularity. In order to do so, we are going to split up the train and test set using sklearn's train test split functions. We will approach this as a regression problem to predict the popularity (numeric) and we will try out models such as Lasso Regressor (sklearn.linear_model.Lasso) or SVR (Support Vector Regressor, sklearn.svm.svr). We will evaluate our model with a test set with RMSE (root mean squared error).
2. In our next question, we will train a model to classify a genre solely on its acoustic features. First we will split the data into test and train set using sklearn's train test split function. We will use K-Nearest Neighbor Classifier (KNeighborsClassifer from sklearn.neighbors) and evaluate the model's accuracy with a test set.
3. Our third research question will center around using Exportify to export a Spotify playlist. First, we will choose a song from our dataset (since we want to be able to access acoustic information about it). Then on Spotify, we will play the song, and then look at the Recommended songs playlist. We will add all of the songs from that playlist into Exportify, and then merge with our dataset. Ideally there should be at least 5 or 6

matches in our dataset, depending on the song we initially chose. However, we may have to try several different songs to get a sizable sample. Once we have our playlist of music, we will create a function to calculate the z-scores for each of a given song's acoustic attributes and then we will store the results for each song into a dataframe. The dataframe will contain a row for each song, and the z-scores for each acoustic attribute for the given song. In another function, using this dataframe, we will create a bar chart for each attribute to compare the distributions among each of the songs. The given song will be shaded a different color to distinguish where all of the other songs are in relation to it. In yet another function, we will compare the average z-scores among all of the recommended songs, for each of the attributes, and compare those to the z-scores for each attribute of the given song. Then we will plot another bar chart that will show two bars for each acoustic attribute--the z-score for the given song, and the average of the z-scores of the other songs. For all of the categorical data (artist name, key, etc.), we will create a function that takes in the given song's attribute (artist, key, etc.) calculates the percentage of the playlist which shares the same value. We will also plot this onto another bar chart, which compares the percentages for each categorical attribute. By looking at the bar charts, we can determine which attributes have the most effect on the Spotify recommendations playlist. For each attribute, values which are closer to the given song's attribute will mean there is a higher emphasis on that attribute when Spotify creates the recommended songs playlist.

4. For our fourth research question, we will purely use our dataset. For a given song from our dataset, we will create two functions that calculate a "similarity score". Each will take in two parameters--the given song, and the recommended song. Both will use the z-score function from earlier to calculate the z-scores for each of the acoustic attributes for each song. Then the functions will calculate a similarity score among the differences in the z-scores of each attribute. The first function will do this using the Mean Absolute Error, and the second function will do this using the Root Mean Squared Error. Each function will return a one-row series, which includes the recommended song name as the index and the similarity score as the column. For each song in our dataset, we will run these functions, using the same given song. We will also create a function will that returns a series that includes the similarity scores for each song. For each series, we will sort by descending order. This will be how we determine our recommended songs. Finally, we will compare the playlist from question 3, to our two song series that we just created.

Results

1. Before training a model to predict the popularity of the songs, we ran some functions to analyze the data we have, and we obtained the correlation of acoustic features and produced a heat map of different musical features with the popularity of the song.

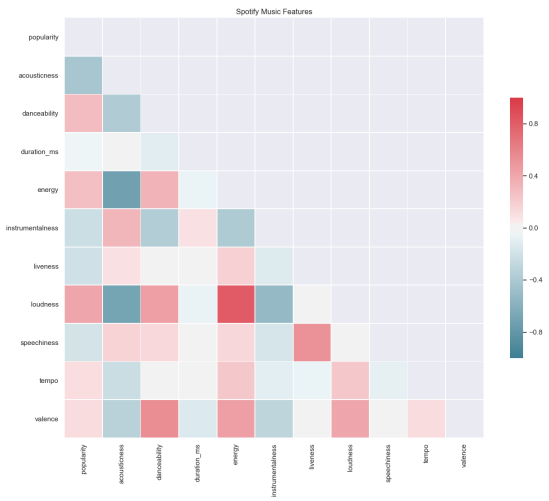

Figure 2) Features associated with popularity (|corr| > 0.25) : Acousticness, Danceability, energy, loudness

Then, we tried to train a support vector regressor by using 60% of the entire dataset. However, running SVR with 8 features, 228159 rows of data took too long, so we decided to narrow the dataset down to one genre, which was pop. We further use this set of data for lasso regression as well. Even after reducing down to a single genre, there were 9386 songs, which we thought would be enough for the popularity analysis.

- **R^2 Score:** 0.0054757892180431 47 (Score - this is from the built in function for the class svr, that returns the coefficient of determination R^2 of the prediction.The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse -- from sklearn documentation)
- **RMSE:** 52.1448795387254
- **Interpretations:** This is a very low R-Squared value, and a very poor RMSE score. Considering the popularity scores range from 3 to 100, RMSE of 52 means that the model is not doing very well. We inspected the predicted scores and the actual scores of the test set, to see if the model is constantly predicting the same values over and over, but the model seemed to predict with varying values.

Next we trained a Lasso regression model. When we tried to use alpha value of 1.0 (baseline) model, we also got a very poor result.

- R^2 Score: -0.00011718077796452064
- RMSE: 51.88058019863151
- Interpretations: both the score and RMSE value we got from the lasso regression were not meaningful.

Finally, we tried to find out if tuning the parameter would help to improve the model in any way. We selected some different alpha values ranging from 0.1 to 1 and tested out all the values and the results is as follows. In order to test this, we held out 20% of our data, but because there was no single model that outperformed all others, we did not use this set of data.

Alpha values	R^2 score	RMSE
0.1	0.006743764969113775	49.434648910388844
0.5	-0.0009776968143411224	49.81894828740529
0.7	-0.0013559758156633792	49.83777534225882
0.9	-0.0013559758156633792	49.83777534225882
0.95	-0.0013559758156633792	49.83777534225882
0.99	-0.0013559758156633792	49.83777534225882
1	-0.0013559758156633792	49.83777534225882

table 1) Different alpha values and corresponding R^2 and RMSE scores. There seems to be no model that performs significantly well than others.

Overall, it seems like musical features such as danceability, acousticness, etc. are not really useful in predicting a song's popularity.

2. For our next question on training a model to classify a genre, we used a K-Nearest Neighbor classifier to predict different genres. Overall, our classifier did not perform very well. To find best parameters for our classifier, we used gridsearch CV and pipeline. For number of neighbors, we tested arbitrary values between 1 and 20, and for two weights were tested: 'uniform' and 'distance'. Accuracy for each genres were low for all genres, except for comedy, opera and soundtracks. This is most likely due to the fact that comedy music has a distinct feature, which is extremely high level of speechiness. Some of the songs in this dataset are from videos are simply audio files from stand-up comedy shows or youtube videos, resulting in extremely high value of speachiness compared to other genres.

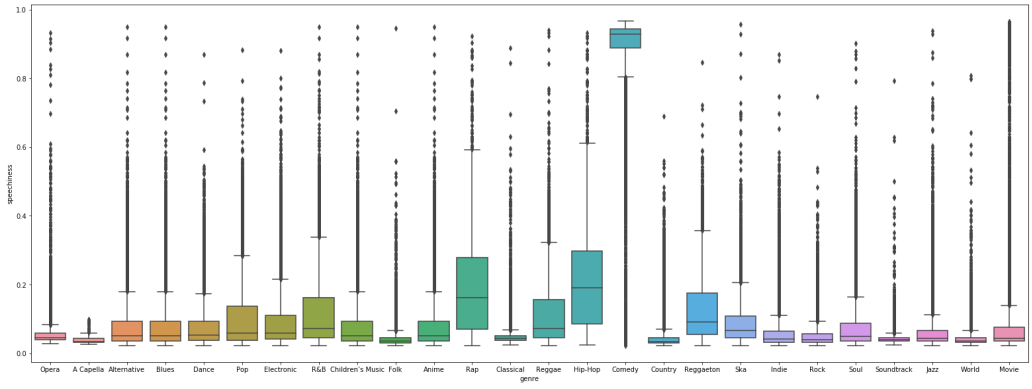

Figure 3) Boxplot: different genres vs. speechiness of the audio file

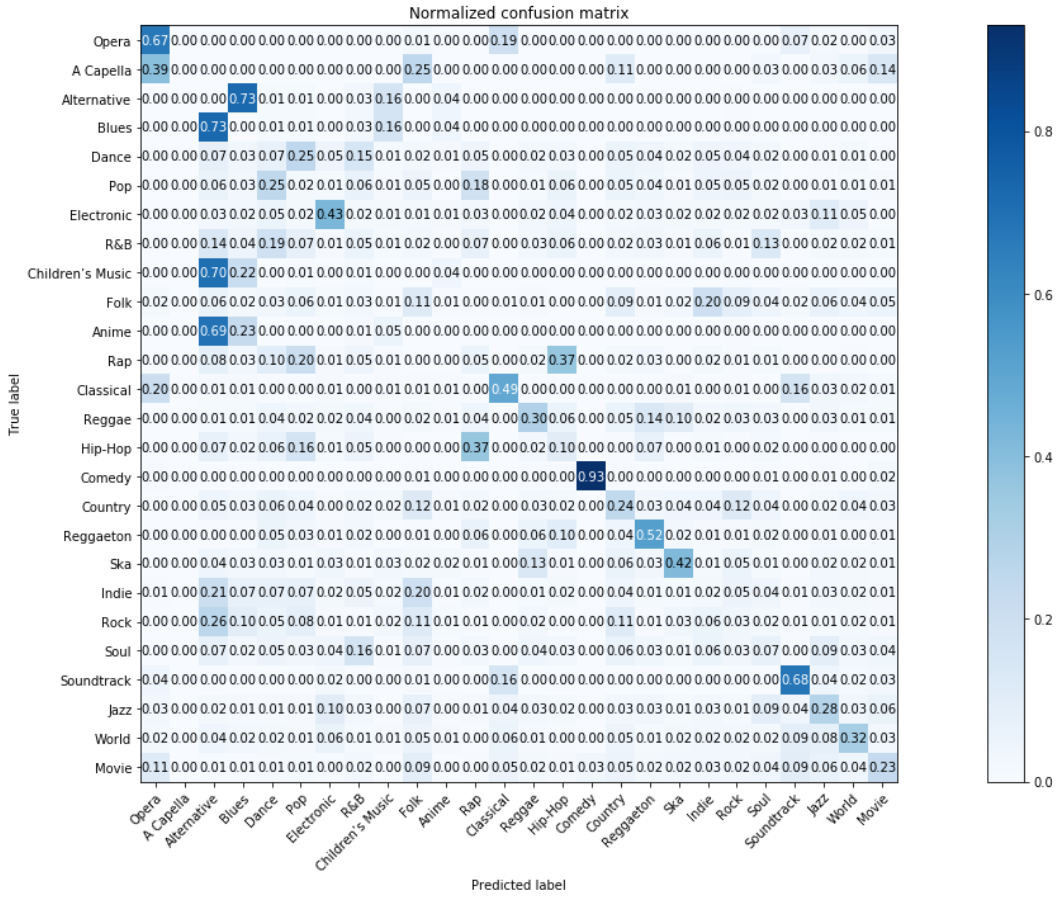

Figure 4) Confusion Matrix of K-Nearest Neighbor Genre Classifier

3. Before I worked on the code to answer this question, I had to merge the Spotify music dataset with a recommended songs playlist that Spotify recommends when you listen to a song. However, I did not realize how Spotify actually recommends music to you. After listening to one song, Spotify recommended a "Daily Mix" based off what I had already listened to. The issue I had with this was that if you listened to any music unrelated to the song you wanted to compare, then your recommended playlist would be diluted with what you listened to before. Instead, I created new playlists, added the given song to it, and then I looked at recommendations that Spotify shows to you while searching for more music to add to the playlist. This is based off what is in your playlist, which means it's more "pure" in the sense that it is only affected by the given song. Thus, I was able to add music in an entirely objective way and avoid biasing my data. When answering this question, I spent a lot of time cleaning the data, dropping duplicates, since songs would often be tagged under multiple genres. Furthermore, I separated the data into quantitative and categorical data, where I would perform different plotting functions for each. For example, I plotted how the acoustic features of each song in the playlist compared with the acoustic features of the given song. The plot below shows how these

attributes are distributed for each song in the playlist. The given song is “Iron & Steel” by Quinn XCII, and the given file that encodes the playlist data is “Iron&Steel.csv”.

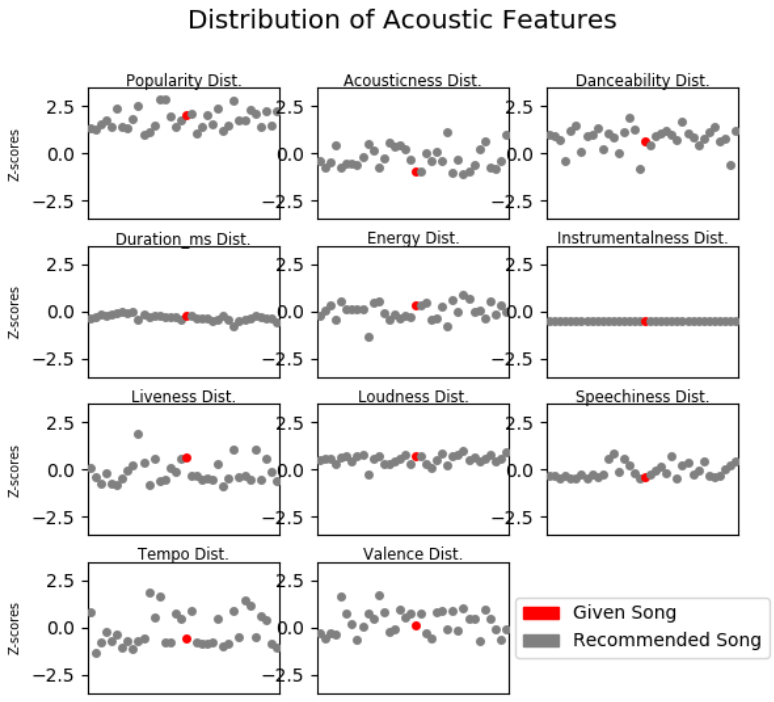

Figure)

This data is generally what I expected, although the “Instrumentalness” scores seem to be the same value. This must either mean that either I came across a very non-diverse set for instrumentalness, or they default to a certain value in the dataset (e.g. 0). Since I expected at least some variability, I ruled this to be due to default values. This plot showed to me that Spotify emphasized certain traits more than others. For instance, it appears that loudness is mostly uniform, along with duration, so these must factor heavily into the songs that Spotify recommends. For the categorical data, we plotted a bar chart. However, our original methodology said that we would calculate the percentage of the playlist that the given attribute occurs in. The issue was that for categories like “Mode”, there are only two options, and for categories like “artist name”, there are countless options, so obviously the “mode” value would be overrepresented in the dataset, and we would falsely come to the conclusion that it had the biggest effect. The new idea was to compute the proportion that a value in one of the given song’s attributes occurs in the dataset, and divide the proportion that it occurs in the playlist by said proportion. For instance, the artist of the given song is Quinn XCII. If Quinn XCII occurs in about 5% the playlist, and about 0.003% of the dataset, you would divide .05 / .00003. As can be expected, this results in a high number, as you can see from the plot that our code constructed below.

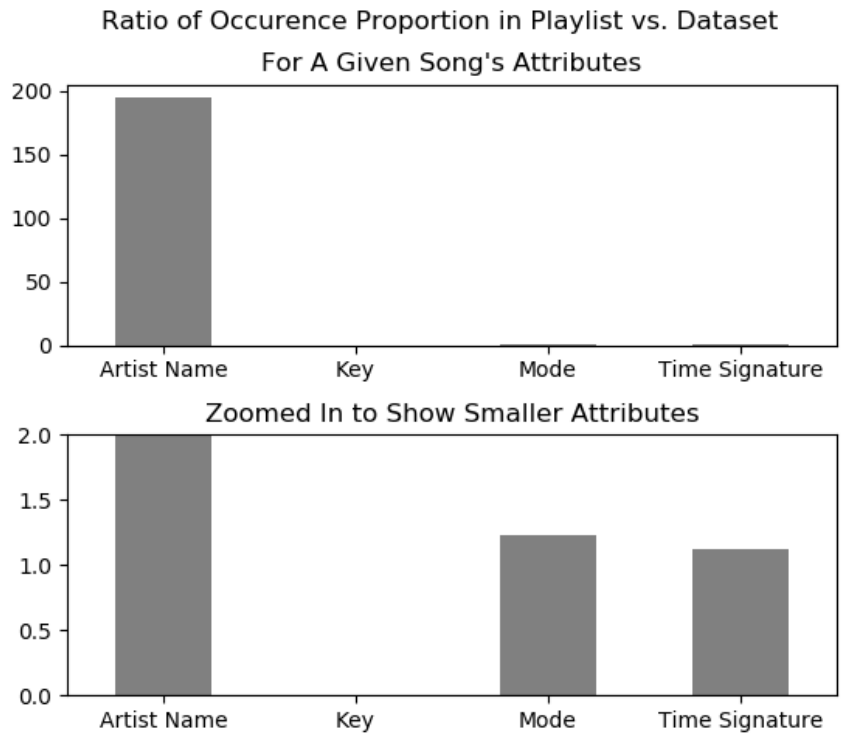

Figure : different genres vs. speechiness of the audio file

This is strong evidence that Spotify predominantly puts emphasis on the name of an artist, which makes sense if you are looking to explore music, since an artist's repertoire is likely to contain very similar-sounding songs. For this particular example, no songs in the playlist matched the same key as the original song, which is why there is no bar for it. In general, we can see that Spotify also puts slightly more emphasis on Mode (Major or Minor) and Time Signature when recommending songs.

4. One of the biggest things I had to consider when making the playlists was "What is considered and acoustic feature?". This was important in answering the question because features like "duration" don't necessarily affect the sound of the song. Features like "popularity" just represent how well the song was received, but it does not pertain to the actual music itself. I decided to cut the two aforementioned features out, and left all the other quantitative features in, since they had a relevant effect to the actual sound of the song. When creating the playlists, I realized that it was unnecessary to create functions to calculate the mean absolute error and the root mean squared error since both of those are calculations provided by sklearn.metrics. There were a few minor adjustments I had to make, such as converting each of the series I constructed for each of the playlists to numpy arrays, since that's what the functions from sklearn.metrics accepted as parameters. When creating the playlists, I did not anticipate them to be relatively similar. Each playlist that I created contained 51 songs--the original song, and 50 other recommended songs. The examples below are shortened to 20 songs per playlist to save space, and they show the playlists created using "Iron & Steel" by Quinn XCII.

Table 2) Using Mean Absolute Error:

Quinn XCII	"Iron & Steel"	0.000000
Tabi Bonney	"Chasing (feat. Matt Beilis)"	0.151657
88GLAM	"Cake"	0.153015
R3HAB	"Rumors (With Sofia Carson)"	0.175173
Mishka	"In A Serious Way"	0.187685
B.o.B	"Creme De La Creme"	0.198456
A.CHAL	"Matrix"	0.209767
Ryan Leslie	"Gibberish"	0.211286
Sammie	"I Left... Because I Love You"	0.211815
Backstreet Boys	"OK"	0.215391
Sticky Fingers	"Happy Endings"	0.215753
Iya Terra	"Humble Yourself"	0.219928
Gucci Mane	"Confused (feat. Future)"	0.220056
Pacific Dub	"Best of You"	0.223390
Shwayze	"Travelin'"	0.226187
Stephen Lynch	"Not That Kind of Thing"	0.226591
Lil Uzi Vert	"Pretty Mami"	0.227094
Halsey	"Without Me (with Juice	0.229355
Julia Michaels	"Anxiety (with Selena	0.234110
Fuego	"Lewinsky"	0.236147

Table 3) Using Root Mean Squared Error:

Quinn XCII	"Iron & Steel"	0.000000
88GLAM	"Cake"	0.127390
Mishka	"In A Serious Way"	0.127573
R3HAB	"Rumors (With Sofia Carson)"	0.128727
Tabi Bonney	"Chasing (feat. Matt Beilis)"	0.129494
Sammie	"I Left... Because I Love You"	0.147164
Ryan Leslie	"Gibberish"	0.148603
Backstreet Boys	"OK"	0.152232
B.o.B	"Creme De La Creme"	0.164101
Syd	"Over"	0.166870
Fuego	"Lewinsky"	0.167093
Cher Lloyd	"None Of My Business"	0.171464
Gucci Mane	"Confused (feat. Future)"	0.171609
NSTASIA	"Parachute"	0.172347
Stephen Lynch	"Not That Kind of Thing"	0.176849
Lil Uzi Vert	"Pretty Mami"	0.177129
Shakira	"Did It Again"	0.178336
Mike Love	"I Love You"	0.178377
Halsey	"Without Me (with Juice WRLD)"	0.180317
Iya Terra	"Humble Yourself"	0.181148

As you can see, both of the playlists that we created are very similar, with only four songs differing. In other words, the playlists have an 80% match rate (although it should decrease with larger playlists). Furthermore, the first song of each playlist is the input song itself, which is expected since the input is closest to itself. Additionally, we were originally going to use these playlists and compare them with the Spotify playlist by using a playlists dataset that users upload, and we wanted to test if any of the the playlists we created or the spotify playlist was more accurate in matching the playlists that others have created. However, this was omitted from our project due to time constraints. However, what we can say about our playlists is that they are not influenced by artist name, which seems to be a big factor when Spotify creates their playlist, as shown in our results for question 3.

Reproducing Our Results:

1. Download “project_code.py” and all the files included in the directory. In the code file, functions `plot_genre_distribution(df)`, `plot_popularity_corr(popularity_features)`, `svr_4_features_pop(df)`, `lasso_pop(df)`, `plot_speechiness(df)` are used to answer question 1 (Including figures 1 and 2) .

2. In the same code file, `genre_classifier_knn(genre_features, gen)` is used to answer question 2. Running grid search takes a lot of time for KNN classifier. In the function, there is a section that has comments with disclaimer-- comment out `k` and `weights` and uncomment the codes where these two variables are given a list of different k and weights. Functions `plot_speechiness(df)` and `genre_classifier_knn(genre_features, gen)` automatically generates and saves figures 3 and 4.
3. Download "project_code.py" as well as all the different input playlists (represented as csv files). To answer only question 3, comment out the following lines: `dataset_zscores = set_up_zscores(whole_data, whole_data)...print(rmse_playlist.iloc[:51]`. That is, ignore every line in the main function starting with "dataset_zscores..." and up to "print(rmse...)". This is inclusive, so you would comment out both, and everything in between. You should get 3 different plots that use "Iron & Steel" as the input song. If you want to change the input song, change the parameter in the second line in main, "playlist = ..." to the name of a different csv file in your directory. The names of the files are denoted by the input songs that the program will use when creating the plots.
4. To answer question 4, keep all files that you needed for question 3. "Un-comment" the lines you commented in question 3, and comment out `playlist_zscores = set_up_zscores(whole_data, merged)... plot_categorical_data(whole_data, merged)`. That is, comment out those two lines and everything in between them. You should generate two series which contain the two playlists of similar songs with the given input song. Be patient, as it may take about 5-10 minutes to completely run, since the code is calculating z-scores for the whole dataset.

Work Plan:

1. **Cleaning up data and Creating Spotify playlist with Exportify**:
    - We will work on cleaning the downloaded dataset together, creating and merging the data frames because it is important to have a same set of data to work with. We plan to do this within a day.
2. **Building algorithms and Implementing**: We will each work on two different part of the research questions, but sharing codes will be done through Git.
    a. Sungmin- Training the model and predicting (questions 1 and 2)
    b. Marques- Implementing z-score song similarity metrics and generating song playlists (questions 3 and 4)
    - Completion of the algorithms and implementing the code will take about a week, so we should finish this part by Monday, June 3.
3. **Evaluate our work and write reports**:
    - We will spend all of week 10 writing the reports. We should finish writing the answers to our research questions within a day (by Tuesday, June 4). After we are

done with our parts, we are going to present our results to each other and work on the evaluation together.
- Within another 2 days, we should be able to complete our *Results* and *Reproducing the Results* sections (by Thursday, June 6).
- Finally, we should finish our *Work Plan Evaluation*, *Testing*, and *Presentation* sections within a day (by Friday, June 7) and we will work on this together.

Work Plan Evaluation:

1. Cleaning up the data and merging the spotify dataset with the playlist created via Exportify took only a few hours. Most of the time spent was actually creating the playlists on Spotify, since I had to manually add songs from the recommended list. Regardless, it took about 3 hours to create twelve different playlists, which is a moderate amount of time.
2. The code implementation took more than a week and took the majority of our time with the project. We did not completely finish the coding aspect until Friday, June 7, which was 4 days later than we intended to finish that section. Most of the time spent was on trying to resolve issues with our regression models. Some major issues were that SVR would take forever to run with too many features we had some unexpected results such as extremely high RMSE or extremely low R^2 scores, which made it harder to interpret the model. Furthermore, when plotting our graphs in answering question 3, there was a lot of stumbling blocks in trying to plot the graphs and using matplotlib functions which would allow us to customize the layouts of our graphs. In fact, that took three days to resolve by itself. It took another whole day to create the playlists for question 4. The time delay was further exacerbated by the fact that the code would take about 20 minutes to run each time I ran it since it had to calculate the z-scores for the whole dataset to create the playlists.
3. This part took us about 4 days, which is also the amount of time we expected to spend on it. Since the coding took forever to finish, we did not start this part until Thursday, June 6. As the length of our paper was way longer than we anticipated, since we had a lot of results to present, it took us a while to finish each of the sections. The Results section in particular took about 2 days because we had so much to present with each other. However, once we finished the Results section, the other sections were quicker to be finished than we anticipated, so it all levelled out and it took us the amount of time we expected, but we did finish 2 days later.

Testing:

- Most of the testing codes for Machine Learning and pre-analysis codes were written in Jupyter Notebook because it was easy to visualize graphs and interpret results. Results were tested multiple times, and all produced similar results for machine learning models.

- Most of the testing for analyzing and creating playlists was done on the code file. We tested out different csv files to make sure our code ran properly. In particular, we were looking for the given song in each of the files to be shifted to the middle in the graphs, which occurred for each of the files we tested. Furthermore, our playlists were tested by testing out different files. The code returned different playlists depending on the input file parameter, and each of the playlists returned playlists that made sense based on the input song (e.g. an opera song generates a playlist of other opera songs, etc)

Presentation Method:

- We are planning to present with a video.

Collaboration:

- There are no collaborators other than course TAs (Erika Wolfe).
- Code credits: Function that was used to produce Confusion matrix for question 2 was downloaded from sklearn's confusion matrix examples and documentation. (https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py)