Predicting the Popularity of Newly Released Songs on Spotify

Asuka Li, Ameer Husary, Samir Chowdhury

ABSTRACT

As the 2022 Spotify Wraps –a summary of users' music taste that all Spotify users look forward to the most every year– have just come out, many people have been talking about their summary and finding commonalities between their list with their friends' list. Music has been prevalent in people's lives for decades and it's something many people can find entertainment and enjoyment in. This project aims to look at the potential relationships between the musical elements of each song and evaluate the best model to conduct the prediction of the popularity of newly released songs. The features we evaluated include elements like danceability, energy, loudness, and tempo, along with the y variable of popularity scores in 5 categories with 20 points ranging from 0 to 100. After building the dataset and conducting preprocessing, we conducted feature selection to reduce data size to simplify the necessary models we'll be building for better results and higher efficiency in computation. We conducted Pearson Correlation, Principal Component Analysis, and feature expansion to determine the best feature engineering model. Although the Pearson correlation didn't give us much information, principal component analysis gave us useful information on feature reduction. We then conducted various machine learning models to evaluate if there is a relationship between the musical attributes and popularity scores, and if there is, what the best model for estimating the popularity score of each song is. We conducted K-Nearest Neighbors, Decision Tree Classifier, and Random Forest with the full dataset of 13 quantitative attributes since there were no strongly and similarly correlated attributes shown from Pearson correlation. After many attempts using multiple other regression models like Lasso, Ridge, Elastic Net, Logistic, and Multilayer Perceptron, most models gave us similar predictions with accuracy scores of approximately 90% with undesirably low f1-scores.

INTRODUCTION

In recent years, audio streaming services have allowed for the globalization of music. People from different countries can now share their similar perspectives about instruments, artists, and so on. Spotify, the world's leading audio streaming service, has 433 million users from around the world. In 2021, some of the most popular songs on Spotify were: "As It Was," by Harry Styles, "Heat Waves," by Glass Animals, and "STAY," by The Kid Laroi and Justin Bieber. We believe it would be an interesting objective to discover what are the factors that lead to songs' popularity in the world. Throughout this paper, we will try to answer the question: "can we train and test a machine learning model to correctly predict the popularity of songs using only their Spotify audibility features such as danceability, energy, and so on?". We believe it would be important and interesting to find out why some songs end up being more popular than others. We will start by obtaining the Spotify track data via web scraping, preparing the data via preprocessing, training and testing different machine learning models such as (K-Nearest Neighbors, Decision Tree Classifier, Pearson correlation, PCA, Random Forest, Linear and logistic regression, and so on, and determine which machine learning model would correctly predict the popularity category of songs.

BACKGROUND

There is a significant body of work in the tasks of Music Information Retrieval and Popularity Prediction. The first task consists of finding ways to reduce the highly dimensional audio waveforms. The second task consists of using various features either directly from the audio waveforms or derived from to predict a defined 'popularity' metric. Spotify handles the first task for us, by providing a set of audio features for each song listed on their platform discussed below. Other such methods include the use of lyrics, meta-data, low level audio features (i.e. spectral and fourier analysis), preview audio files, and artist information (Martin-Guiterrez, 2006) (Votter, 2021). People have tried a variety of approaches, including SVMs (Support Vector Machines) and LSTMs (Long Short Term Memory Transformers) (Yu, 2019), and temporal popularity analysis (Lee, 2015), regression and classification techniques (Votter, 2021), and even the user's previously played songs (Vall, 2019).

EXPERIMENTS / RESULTS

Data Description, Preprocessing, Data Cleaning

We initially downloaded a dataset with 1.2 million unique Spotify songs' track IDs from the Kaggle competition platform that came from a personal scraping project of Spotify music. Due to the calling limit of Spotify API, we randomly selected 0.5 million rows, and using the track IDs, we then scraped for the musical and instrumental elements of each song using Spotify API. After the combination of the two datasets, our final dataset contains 495,250 rows (≈ 0.5 million) with 17 attributes ('id', 'name', 'artist_ids', 'danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'duration_ms', 'time_signature', 'popularity'), 13 of which are quantitative attributes that we applied machine learning algorithms on. These quantitative attributes consist of the target variable popularity scores, along with musical elements danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo, duration_ms, and time_signature. Finally, we removed all the null values from the dataset.

Exploratory Data Analysis

After that, we wanted to take a look at our data to better familiarize ourselves with it. We found out that the data was already clean and there were no null values that we needed to take into account. Since many of the machine learning models are classifiers we thought it would be better to factor the data from a purely quantitative variable (0-100) into a categorical variable of 5 categories with 20 points ranging from 0 to 100. Category 0 "Not Popular" has about 457165 rows (92.3% of the dataset), Category 1 "Slightly Known" has about 31230 rows (6.3% of the dataset), Category 2 "Known" has about 5947 rows (1.2% of the dataset), Category 3 "Popular" has about 887 rows (0.18% of the dataset), Category 4 "Very Popular" has about 21 rows (0.004% of the dataset), as we can see in figure 1 below.

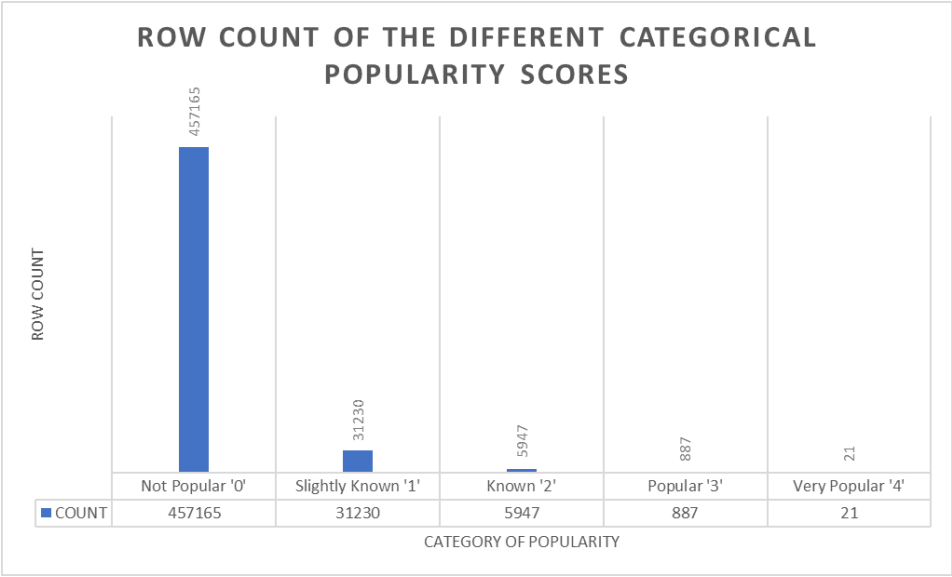


Figure 1: Row Count of the Different Categorical Popularity Scores

We also wanted to see if we could draw boundary lines between the different categorical popularity scores and the variables in the data set. However, as we can see in figure 2, the categorical popularity scores seem to be scattered across and not forming any specific clusters.

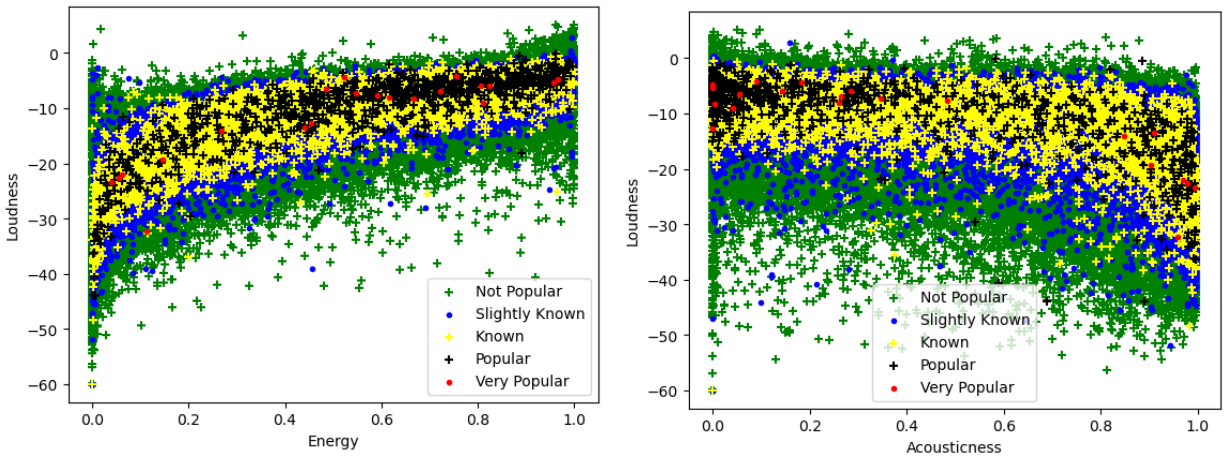


Figure 2: Distribution of the Popularity Categories on the Most Highly Correlated Variables

Feature Selection

Pearson Correlation

For the initial Pearson Correlation we have conducted, there seem to be almost no attributes with a high correlation to the popularity variable. There is not much correlation between each attribute, as seen by the distribution of the relationship between them in figure 4. The strongest relationships we found were between Energy and loudness (as seen in figure 2), and loudness and acousticness (as seen in figure 3). However, most of the data seemed to be relatively weakly correlated to popularity from the Pearson correlation matrix (seen in figures 5 and 6). With the weakly correlated attributes, we decided to not use any information we derived from computing Pearson correlation.

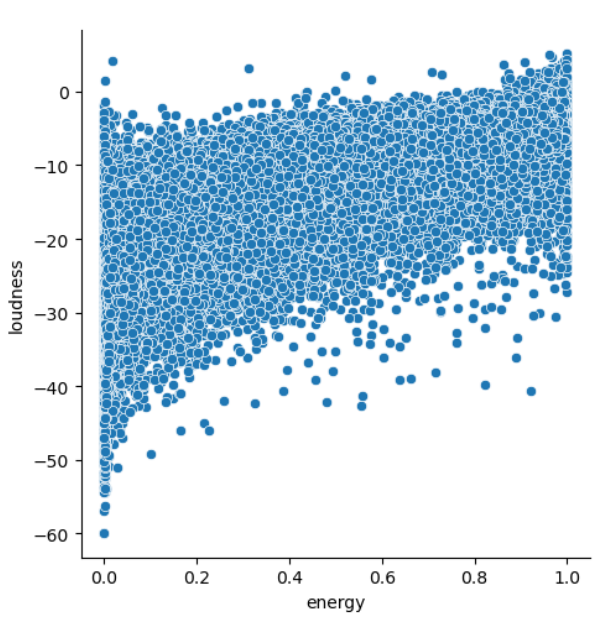
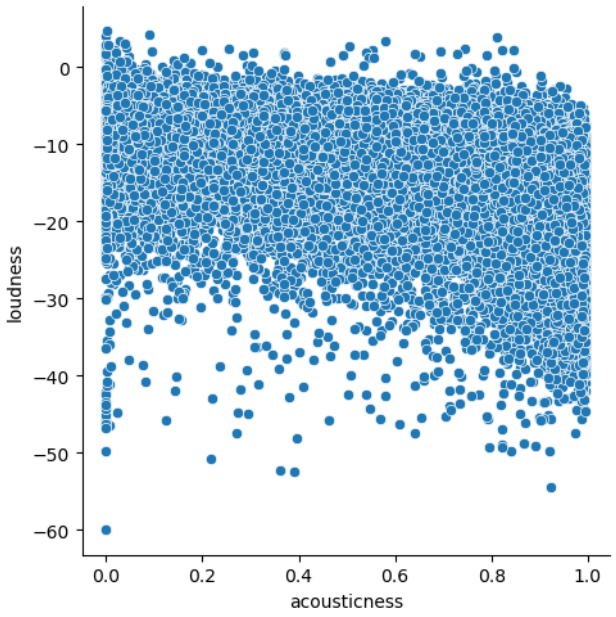


Figure 3: Energy vs Loudness Distribution



Acousticness vs Loudness Distribution

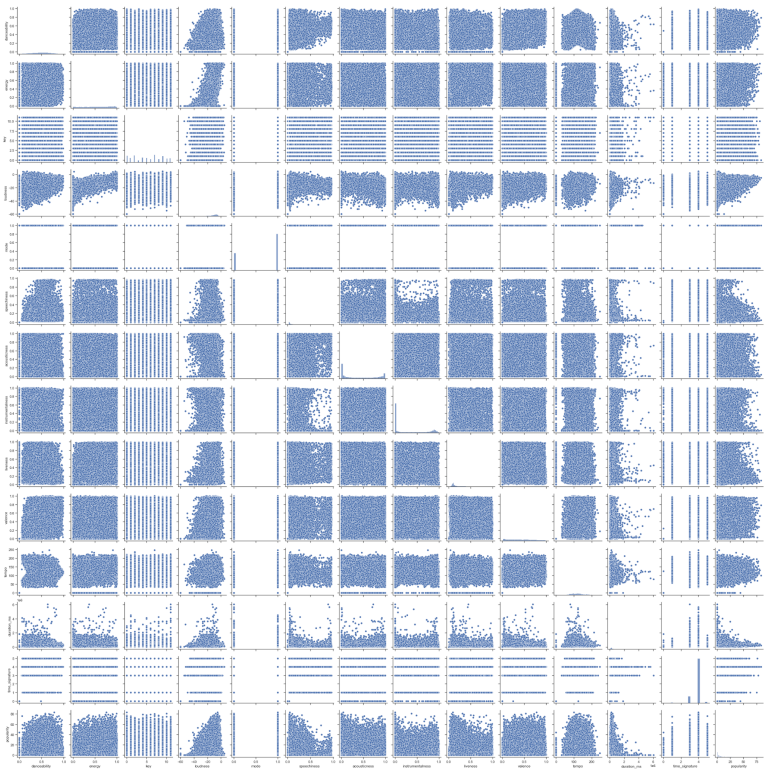


Figure 4: Distribution of all the Quantitative Variables

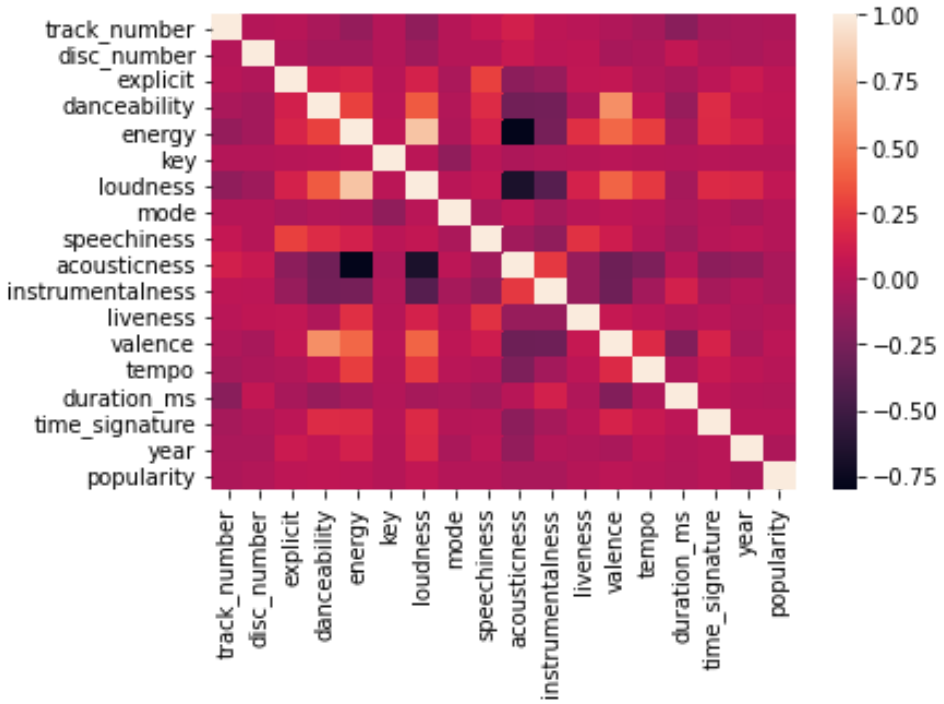


Figure 5: Pearson Correlation Matrix

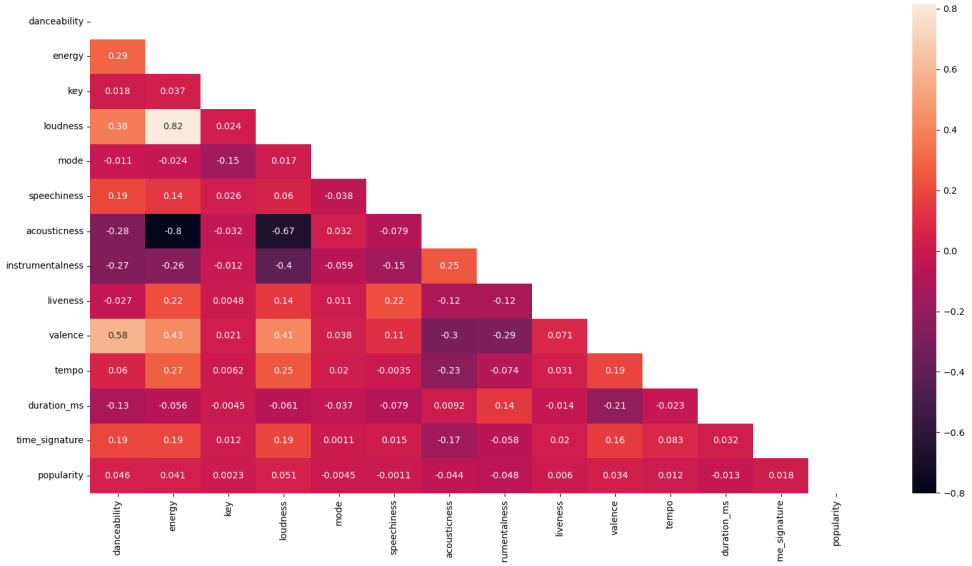


Figure 6: Lower Triangular Pearson Correlation Matrix

Principal Component Analysis

For principal component analysis, we wanted to reduce our highly dimensional data and capture the most important sources of variability. In order to choose the number of components, we opted to use the elbow method to determine the number of components. In the figure below, you can see the variance explained by each component significantly drops at n=2, so we opted for n=3 to capture the most variance in the least amount of components while also having the added benefit of nice visualizations available to us. Another potential choice could have been 10 components as we see a significant drop-off in explained variance but approach relatively close to the number of original features. We then plotted the PCA data (Figure 10) with the color of each point determined by its popularity.

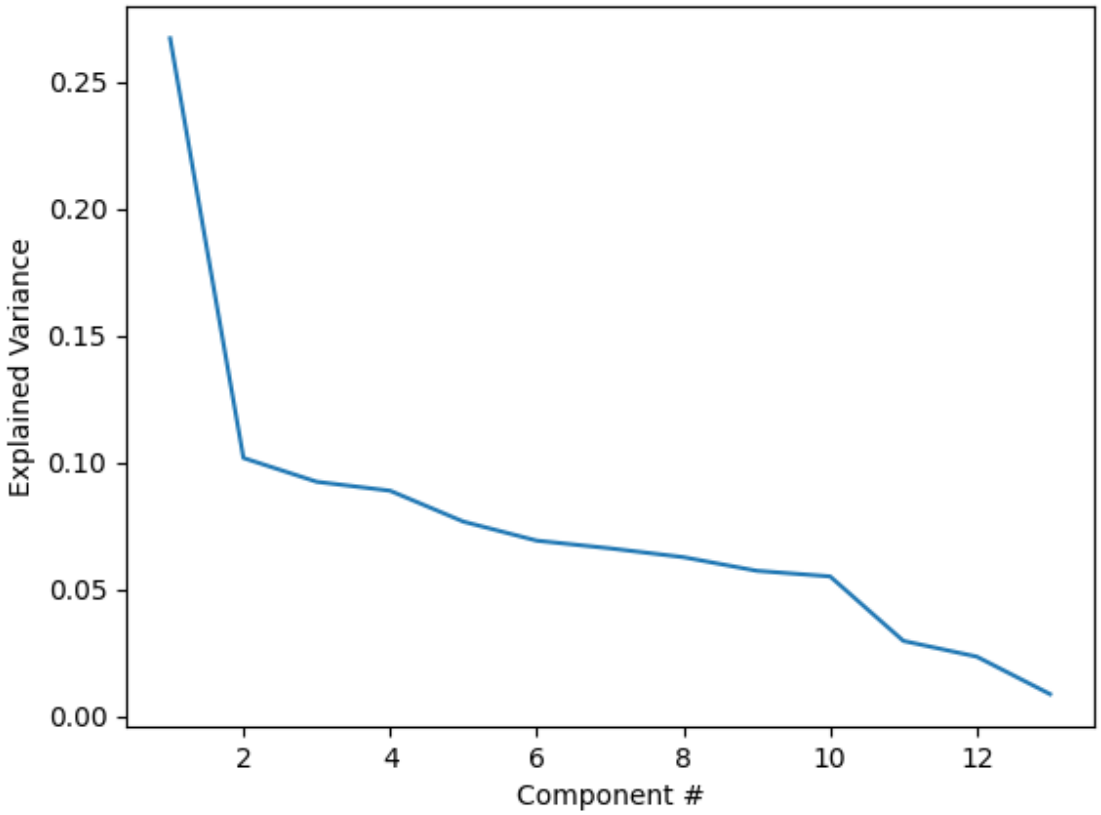


Figure 9: PCA Elbow Plot

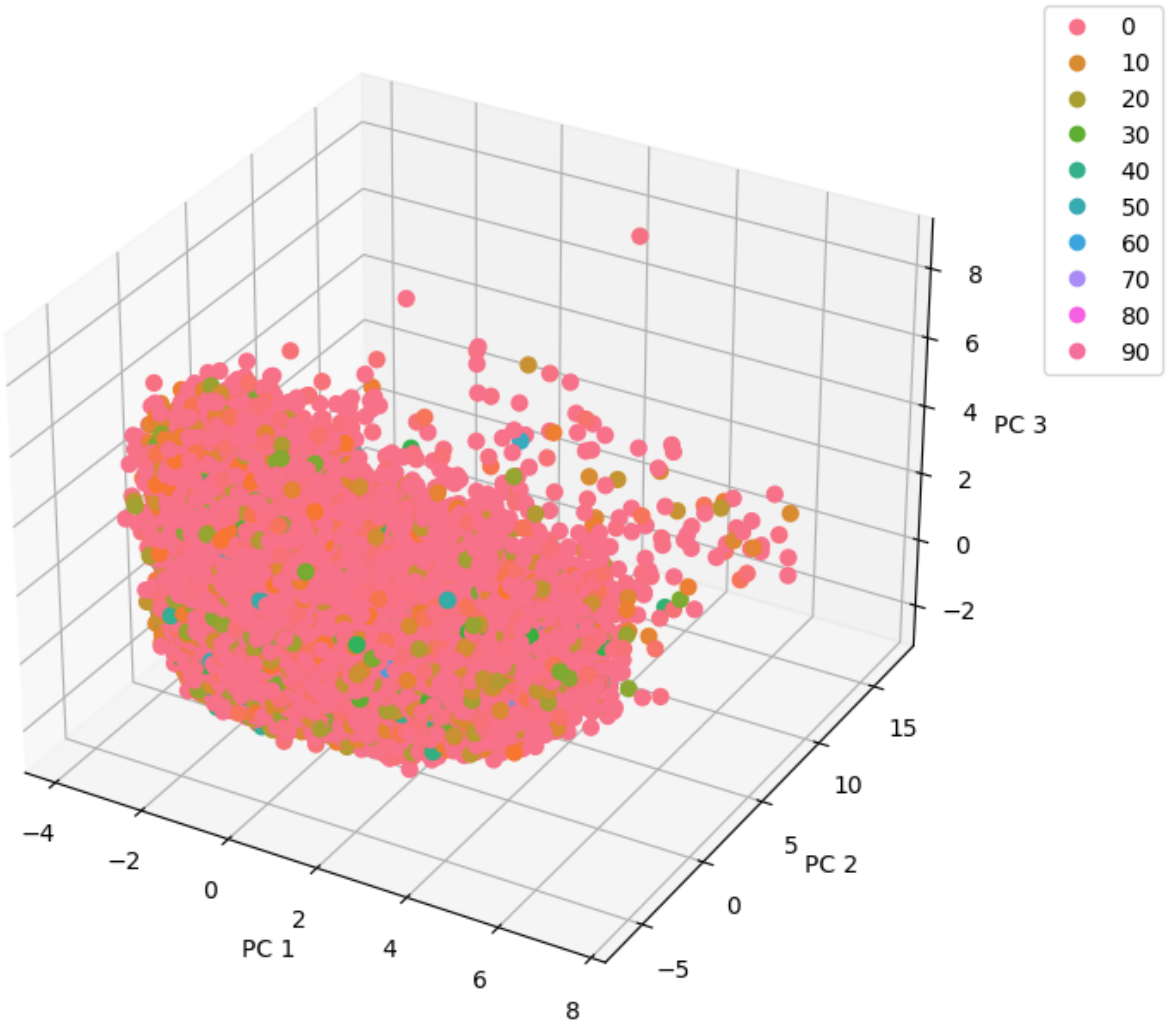


Figure 10: PCA Data by Popularity

Modeling Choices

K-Nearest Neighbors: As one of the simplest models in machine learning, we wanted to look at whether there are specific neighborhoods for a specific popularity class to appear. Through parameter tuning, we determined the number of neighbors that produce the best result is 11.

Decision Tree Classifier: This model gives us interpretable results because we can look at the resulting tree structure after fitting the model to see which features and values are best for minimizing the MSE and maximizing the accuracy. Through hyperparameter tuning, we determined the best max depth, min-leaf sample, and criteria to use for the model to minimize MSE and maximize accuracy.

Random Forest: The decision tree classifier gave us results similar to that of K-Nearest Neighbor, we wanted to see if a combination of multiple trees (random forest) would give us a better result. Random Forest regressor can also model non-linear boundaries, which would have a better chance of Through hyperparameter tuning, we found the number of trees and max depth, and min leaf sample to minimize MSE and maximize accuracy.

Regression: Regression models give easily interpretable results because, after hypothesis testing, the values of the significant coefficients capture the strength of each feature's relationship to the popularity score. We conducted Linear regression, Regularized linear regression such as Lasso and Ridge, Elastic Net Regression, and Logistic Regression with categorical variables. In order to explore if any of the models would give us relatively more accurate estimations compared to the above three models we've had, we decided to try various regression methods.

MLP: Multi-Layer Perceptron models allow for deep models to capture an arbitrary amount of dimensionality in the data. Especially since there can be more parameters than there input features, the model can learn subtle traits of the data. Given the complex nature of music and popularity, it could be a potential strength. One issue may be convergence to local minima. Due to the large number of parameters, it might be difficult to have the model converge quickly to a good minima and offer good performance.

Bootstrapping: As in the Exploratory Data Analysis process, we've realized the unbalances of the dataset, bootstrapping would be helpful with resampling the data to balance out the discrepancy between different classes within the labeled categories. In addition to this, the other models mentioned above have all performed similarly, giving the accuracy and f1 score in the proximate range of percentage of class 0 (popularity score of 0-20), which is the most frequently appeared category within our dataset. In order to utilize the machine learning models to accurately train and give predictions for our model, we wanted to compensate for the imbalances of the data and attempt bootstrap sampling. We applied the decision tree model to the bootstrap sampled data to determine if there was an improvement in the performance of the model.

Bagging: also known as bootstrap aggregating; an ensemble learning technique that involves using multiple learners "machine learning models" to solve the same problem independently and then average them. It can reduce bias and variance while simultaneously improving performance and avoiding overfitting. Overfitting occurs when the model cannot generalize and fits too closely to the training dataset instead. We believe our models are overfitting and misclassifying the data into mostly the zero popularity category due to the imbalance in the train data set (where zeros are the majority). Therefore, due to the imbalance, we believe bagging and specifically bootstrapping with decision trees and random forest models could help.

XGBoost: boosting involves building models sequentially to reduce bias. XGBoost which stands for Extreme Gradient Boosting is a scalable, distributed gradient-boosted decision tree (GBDT) machine learning

Empirical Results and Comparisons

KNN

	Train Accuracy	Train f1 score	Test Accuracy	Test f1 score
Original Data	0.9231145885916204	0.19221225506238124	0.9230489651691065	0.19199697602872773
Scaled Feats	0.923107016658	0.192179002845	0.92303886926	0.1919958840

Using GridSearchCV, we were able to determine that the best n for the KNN model was n=11 with a min error of 0.1. We have trained, tested, and fitted the KNN model via Standard Scaler and ended

up with high accuracy and low f1 scores for the train and test sets when compared to the original dataset. As we can see from the table above, for our scaled feature dataset we ended up with a test accuracy score 0f 0.923 and an f1 test score of 0.19. Although it seems like the accuracy is high, if we take a look at the confusion truth matrix we end up realizing that our model is highly skewed in favor of misclassifying the data into category 0 “Not Popular” with a popularity range from 0-20 (seen in figure 7 below) when it should be correctly classifying the popularity categories. If we take a look at the classification report on the KNN model (figure 8) we can further see how our model performed in terms of different metrics such as precision, recall, count, and so on. It seems that the weighted average for precision is 0.85, the weighted average for the recall is 0.92, and the weighted average for the f1-score is 0.89. This confirms that the model is not accurately classifying the data and is actually misclassifying the majority into 0s.

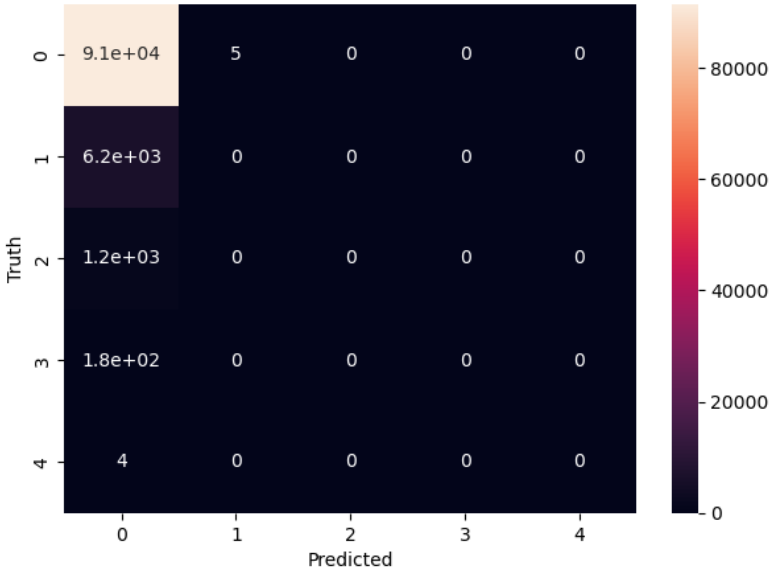


Figure 7: Confusion Matrix on the KNN model

	precision	recall	f1-score	support
0	0.92	1.00	0.96	91433
1	0.00	0.00	0.00	6246
2	0.00	0.00	0.00	1189
3	0.00	0.00	0.00	178
4	0.00	0.00	0.00	4
accuracy			0.92	99050
macro avg	0.18	0.20	0.19	99050
weighted avg	0.85	0.92	0.89	99050

Figure 8: Classification Report on the KNN model

Decision Tree Regressor

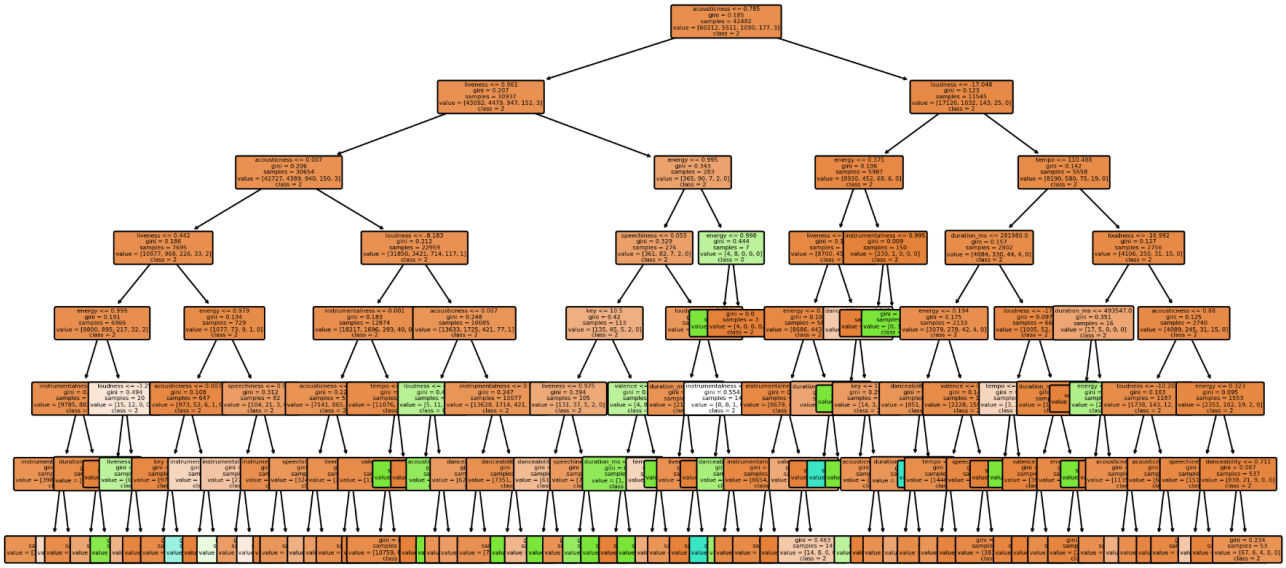
	Train Accuracy	Train f1 score	Test Accuracy	Test f1 score
Original Data	0.923086464	0.193688	**0.923291**	**0.19368806**
Scaled Feats	0.90220272027	0.90220272027	0.90169508475	0.90169508475
Bootstrap Sampling Data	**0.9233889954**	**0.196833465**	0.9223467693	0.1925243

Through hyperparameter tuning using GridCV, we conducted the model training and testing with a max depth of 9, minimum leaf sample of 11, and Gini index as criteria. For bootstrapping, we used n_bootstraps = 8 with out of bag sample as the testing dataset and took the average of all 8 decision trees' results for the statistics. Of all the models, interestingly bootstrap sampled data with the decision tree model gave the best training accuracy and f1 scores, while the original data with categorical popularity score gave the best testing accuracy and f1 score of 0.9233 and 0.194, respectively. Our f1 scores were significantly lower than the accuracy scores due to the unbalances of our data. Even with bootstrapping, due to the drastic difference between the discrepancy in the number of rows for classes 0 and 1 (popularity score 0-20 and 20-40) and the rest of the classes, we were not able to get a balanced resampling data, resulting in the low f1 scores for both training and testing dataset.

Random Forest Regressor

	Train Accuracy	Train f1 score	Test Accuracy	Test f1 score
Original Data	0.922959544241	0.19202376679	**0.923432609793**	**0.192038463961**
Scaled Feats	0.9015651565	0.190514625119	0.902145107	0.2374512873
Bootstrap Sampling Data	**0.92316784452**	**0.1921659231**	0.922977403	0.191997879721

As the decision tree regressor gave similar results to those of K-Nearest Neighbor, we thought Random Forest could compensate for the unbalances in our dataset by a little. Through parameter tuning, we conducted the model with n = 3 trees and a max depth of 7. With bootstrap sampled data, just like for decision tree regressor, we sampled with replacement 8 times to compensate for the unbalances of the dataset. However, the results didn't improve much from the decision tree regressor at all. Again, the bootstrap sampled data gave us the best training accuracy and f1 scores, while the original data gave us the best test accuracy and f1 scores with not much difference across different datasets.
Below is one of the trees in the sample.



Regression:

We performed several types of regressions to see if perhaps a simpler model could learn well on the data. We found that in general, regression performed poorly. In the table below, we can see that across the board, regardless of how we tried to regularize our linear regression, it did not perform well in regards to R^2 and accuracy. With R^2 values hovering around 0 for all forms of linear regression. To further diagnose the low performance, we calculated both Root Mean Square Error (RMSE) and F1-Score for the regression and classification models respectively. The RMSE and the F1-Score in general across all the models hovered around *10* and *85*.

	Linear	Lasso	Ridge	ElasticNet	MLP	Log (cat)	MLP (cat)
Normal	0.01539	0.00985	0.01539	0.01001	-7.85930	0.90160	0.90160
Scaled	0.01539	0	0.01539	0.00903	0.02284	0.90160	0.9015
PCA	0.01226	0.00917	0.01227	0.0107	0.01732	0.90160	0.90160

Figure 11: Regression Results (R^2 & Accuracy)

When examining closer, we found that the maximum predicted values for the regression models were all under 10. Even more enlightening was the confusing matrix for the classification models. As seen in the confusion matrix below, the model predicts zero for the entirety of the dataset. The model is heavily biased to predicting low values which causes the models to be frustratingly stagnant in performing better than guessing the most popular class

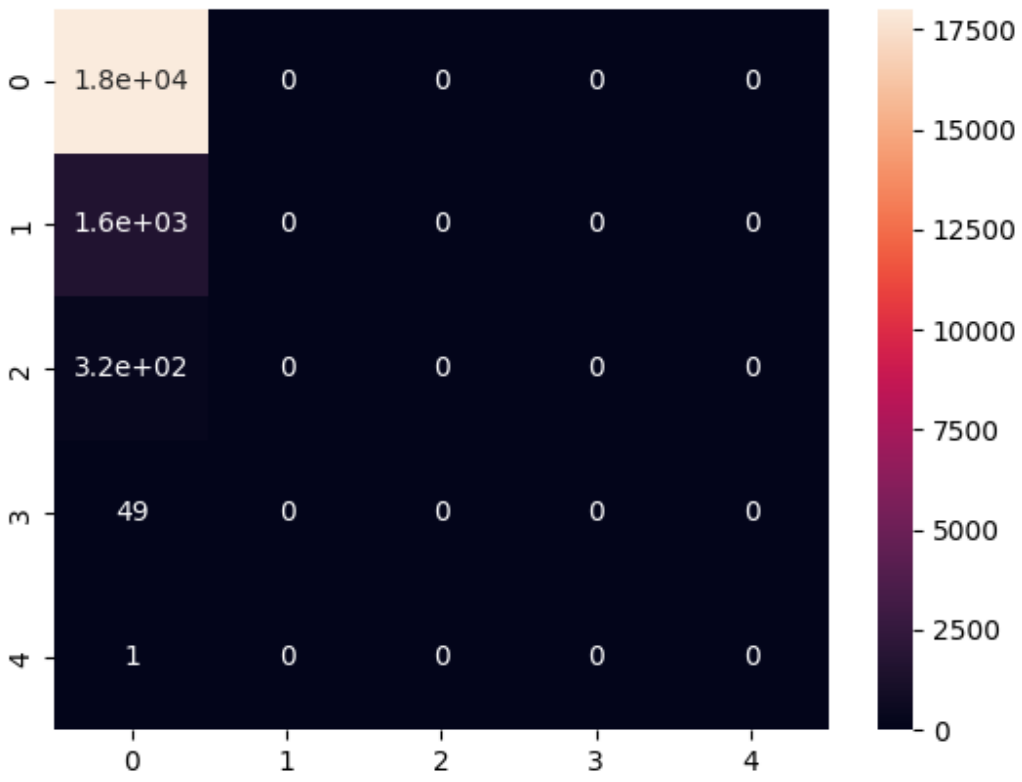


Figure 12: Confusion Matrix of Log. Classifier

Boosting:

We wanted to see if boosting could help increase our test accuracy score of our Decision Tree machine learning model, and we ended up with the following. Our non-parameter-tuned Decision Tree model had a test accuracy score of 0.842. However, when we applied the AdaBoostClassifer on the non-parameter-tuned Decision Tree model with n_estimators = 10 and a learning rate of 1, we ended up with a test accuracy score of 0.9165. Although this is a relatively high increase in accuracy score, it was not higher than the parameter-tuned Decision Tree discussed earlier of 0.92. We also wanted to see if we apply xgboost to our dataset will it increase the accuracy? However, we ended up with a surprisingly low result seen in figure 13 below. Unfortunately, applying xgboost did not increase the test accuracy score, it was actually lower. As we can see the AUC was 0.569 and it took 52 times to achieve this score.

```
[52] validation_0-auc:0.56937 XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None, colsample_bylevel=1, colsample_bynode=1,
colsample_bytree=0.5, early_stopping_rounds=None, enable_categorical=False, eval_metric='auc', gamma=0, gpu_id=-1, grow_policy='depthwise',
importance_type=None, interaction_constraints='', learning_rate=0.1, max_bin=256, max_cat_to_onehot=4, max_delta_step=0, max_depth=5, max_leaves=0,
min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=5000, n_jobs=0, num_parallel_tree=1, objective='multi:softprob', predictor='auto',
random_state=0, reg_alpha=0, ...)
```

Figure 13: XGBoost Results

DISCUSSION:

Key findings

Throughout our analysis, we found that the given features extracted from Spotify are not sufficient enough to predict the given popularity metric. We believe that there are several reasons for this and will examine them below.

The primary reason we believe this to be the case is because of the major class imbalances. When inspecting the poor performance of our models, it is because the models are heavily biased towards predicting zero or low popularity scores. This is in general due to the nature of music as only a few hundred songs can be significantly popular at any given moment compared to the hundred of thousands of songs being released. This significant imbalance hindered our models from accurately being able to differentiate between popular and unpopular songs.

Another limiting factor that contributed towards our weak performance was the source of our targets. We had no control in the design of Spotify's in-house popularity metric, and cannot entirely say for sure whether or not their metric is an accurate indicator of popularity of a song (however one defines popularity). Another concern is the measurement of popularity (in terms of streams). Songs released before the advent of mass music streaming would have a different set of metrics for Spotify to base their popularity on compared to contemporary music. Combine that with the virality of music in contemporary culture due to the slower release cycle and consumption in the pre-internet era, it could result in older songs being in general rated lower, which is a bulk of our data.

Limitations/Further Improvements

We propose three points that could further improve upon our work and are as follows:

The first point being dealing with class imbalances. With the significant amount of unpopular songs, there are three possible remedies to this problem. The first (and least practical) is to curate a significant amount of popular songs from billboard charts across the world across several years, and use those songs to prop up the count of more popular songs. The second is to take those popular songs and to create synthetic data based off that set of popular data. The third is to evaluate the popularity of songs pre-2010, and determine if there needs to be a reevaluation of popularity scores for older songs to see if they're being artificially suppressed.

The second point is the waveform. While the features given were very accessible and usable in terms of computation power, a lot of information is being lost in the waveform. The waveform could yield very useful information in determining popularity, analogous to the real world where one can listen to a song and determine several features about the song, our models could possibly do so as well if given the information. While such datasets with waveforms do exist, they are either too large to work with on our local environments, or not available to the public.

The final point being the artist and album information. While the audio features of the song do convey a lot of information, the artist and album information contains a lot of information about the song's popularity. For example, an extremely popular artist like 'Taylor Swift' will almost regardless have a relatively popular song respective to the entire corpus of music compared to most music. Capturing this information should be critical in determining popularity and finding a good way to vectorize this information is a research question on its own. The same logic applies to albums but has a different set of challenges in its vectorization.

CONTRIBUTIONS:

Asuka Li (2377470)

I actively spearheaded in scheduling meetings and creating to-do lists along with milestones for us to make consistent progress in a timely manner. In addition to this, I was responsible for the work we divided equally in the technical portion with implementations of different machine learning models. Specifically, I took charge in the initial process of data scraping and cleaning, some feature selection, decision tree regressor, random forest regressor, and bootstrapping to predict the popularity scores using attributes of music elements. I wrote the initial stage of code for scraping popularity data from Spotify through Spotify API using secret and client keys. Then, I removed all null data points and created the categorized popularity score for the dataset for easier analysis and split the data into 30/70 for testing and training set. Furthermore, I conducted decision tree and random forest regression to potentially predict the popularity scores, using parameters tuning to find the best parameter. After applying the scaled and unscaled and getting results not very desirable, I utilized bootstrapping in attempting to counter the imbalance of the dataset. Throughout the process, I also actively updated the github repository in keeping the newest version of all notebooks and files I've used for this project.

Ameer Husary (2378674):

I was able to collaborate alongside the rest of the team in dividing the work equally, setting up meeting times, coming up with solutions and ideas, and updating our code repository in GitHub and Google Drive. I was specifically involved in making the Exploratory Data Analysis (code found in EDA.ipynb), which involved filtering and discovering the data shape and the uniqueness of values. In addition to handling null values, creating the correlation matrix, and reporting the most highly correlated variables. I also trained and tested the K-Nearest Neighbors model in KNN.ipynb, found the train and test f1 and accuracy score, and determined the best N to use for the KNN model. Produced the classification report, the confusion truth matrix, and attempted to draw the boundary lines of the popularity categories. I was able to filter the large dataset to end up with a subset of newly released songs (2020) with no popularity score in order to see if our models can accurately predict their popularity category. I also ran XGBoostClassifier and AdaBoostClassifier (seen in boosting.ipynb), as well as helped run the bagging (bootstrap) on Decision trees and Random forest models (seen in bagging.ipynb).

Samir Chowdhury (2393246)

I was able to work with our team with determining what needed to be done, and exactly the best way to go about it as well as generating ideas for both our project and different analysis. In addition to this, I contributed towards scraping the data from the Spotify API in a way that did not exhaust our API limit. I also worked on performing the PCA dimensionality reduction and as well as scaling the data to perform better for our tasks. I also handled all of the regression tasks for our team from Linear Regression, lasso regression, ridge regression, elastic net regression and MLP regression. I also utilized Logistic regression and MLP Classifier for classification. I created visuals for the PCA and the regression with the use of 3D plots and confusion matrix.

REFERENCES:

D. Martín-Gutiérrez, G. Hernández Peñaloza, A. Belmonte-Hernández and F. Álvarez García, "A Multimodal End-to-End Deep Learning Architecture for Music Popularity Prediction," in *IEEE Access*, vol. 8, pp. 39361-39374, 2020, doi: 10.1109/ACCESS.2020.2976033.

M. Vötter, M. Mayerl, G. Specht and E. Zangerle, "Novel Datasets for Evaluating Song Popularity Prediction Tasks," *2021 IEEE International Symposium on Multimedia (ISM)*, 2021, pp. 166-173, doi: 10.1109/ISM52913.2021.00034.

Vall, A., Quadrana, M., Schedl, M. et al. Order, context and popularity bias in next-song recommendations. Int J Multimed Info Retr 8, 101–113 (2019). https://doi.org/10.1007/s13735-019-00169-8

Haiqing Yu, Yanling Li, Shujun Zhang, and Chunyan Liang. 2019. *"*Popularity Prediction for Artists Based on User Songs Datase*t." In Proceedings of the 2019 5th International Conference on Computing and Artificial Intelligence (ICCAI '19). Association for Computing Machinery, New York, NY, USA, 17–24.* https://doi.org/10.1145/3330482.3330493

Junghyuk Lee and Jong-Seok Lee. 2015. Predicting Music Popularity Patterns based on Musical Complexity and Early Stage Popularity. In Proceedings of the Third Edition Workshop on Speech, Language & Audio in Multimedia (SLAM '15). Association for Computing Machinery, New York, NY, USA, 3–6. https://doi.org/10.1145/2802558.2814645

LINKS:

- Google Drive: https://drive.google.com/drive/folders/1JUYedOLSJPTP4eA22m_7rCIeRCetnXZf?usp=sharing
- GitHub Repository: https://github.com/alee248/cs334_spotifyProject