Predicting Song Popularity Using Acoustic Features

Overview:

Music prediction has been a popular subject in machine learning for a long time, with early work in the field dating back to the early 2000s. Prediction models are extremely useful for the music industry, as both artists and labels need to understand the factors that determine a song's popularity in order to make decisions.

In the past, music popularity was determined by looking at charts such as the Billboard Hot 100, but more recent research has been directed at data from streaming platforms, given that they have become the dominant method for listening to music. In my research, I used two different popularity metrics provided by my datasets. One of them uses the number of recent plays (which I assume is based on Spotify plays) to calculate popularity. The other uses data from past weekly Billboard Hot-100 lists.

Researchers have considered both internal factors, such as a song's acoustic features, and external factors, such as an artist's existing popularity in building their models. However, while it is possible to predict a song's popularity based on factors like its initial debut success and an artist's social media presence, it is more useful to predict a song's success independent of any existing popularity, before it is even released. Therefore, my research question is whether it's possible to predict a song's popularity based on acoustic features, genre, and release year alone. Initially, I planned to focus specifically on one genre of music to reduce the variability of acoustic features in my data; however, I soon found that this approach was not producing the results I had hoped for.

I began answering my research question by starting with a simple linear regression model using my first dataset before transitioning to classification models using my second dataset. I found that using classification on my second dataset lent better results than

regression on my first dataset. I also found that an optimized random forest model has the best performance out of all of my classification models. Finally, I found that the instrumentalness feature was the most indicative of popularity across all of my models and datasets.

Related Work:

There has been a lot of research aimed at predicting the popularity of a song using machine learning. The different approaches vary in their choices for popularity metrics, song features, and predictive models.

Popularity Metrics: In [2], Lee and Lee develop eight different metrics to measure song popularity, all using the Billboard Hot 100 chart. This is by far the most extensive definition of popularity out of all the works discussed. This is done to capture both the static and dynamic nature of music popularity; for instance, while a song's mean rank on the charts is important, the amount of time it spends on the charts reveals another equally important aspect of its popularity. The eight popularity metrics the authors developed were debut (initial) popularity, max popularity, mean popularity, the standard deviation of popularity, chart appearance length, the sum of rankings, skewness of rankings, and kurtosis of rankings [2]. While extensive, there are some limitations to using rankings from the Billboard Hot 100 chart in the first place. First, music streaming platforms have become more popular in recent years, and therefore may contain more relevant data about the popularity of songs. In addition, the Billboard Hot 100 chart is popular in the US, but not globally. The other work discussed below addresses these issues.

While Lee and Lee [2] use the Billboard Hot 100 chart to develop their popularity metrics, Araujo et al. [1] rely on streaming platforms, specifically Spotify, for their definition of popularity. The authors use the "Most Popular" and "Virals" Spotify lists in their analysis; the former ranks songs according to their total number of streams in a recent time frame, while the latter ranks

songs according to their growth in the number of streams [3]. Yang et al. [4] take a similar approach, collecting user listening data from KKBOX Inc., a streaming service provider in East Asia. The authors were able to obtain a set of user listening records over the course of a year, which included the playcounts of 30K users for 125K songs. The popularity metric developed by Yang et al. [4] multiplies a song's playcount (in log scale) by the number of users who have played the song, to account for any songs that may have a heavy "cult following". However, this approach is also a bit limited given that KKBOX Inc. is mostly popular in East Asia. Using Spotify would allow for a more global view of popularity. Finally, Pham et al. [3] have the most primitive measure of popularity, which simply uses the "song hotness" score generated by The Echo Nest as part of The Million Song Dataset.

Song Features: While most works incorporate acoustic data about each song in some form in their prediction models, few rely solely on this information to predict the popularity of a song. The goal of Araujo et al. [1] was to predict whether an artist that is already very popular could experience sudden popularity growth through one of their songs and, conversely, whether a viral song from an unpopular artist could result in lasting popularity. They did this by building a model that would use data from the aforementioned "Most-Popular" Spotify list to predict an appearance in the "Virals" Spotify list and vice-versa. They also aimed to only use "historical data of popularity" rather than acoustic data. The authors built four models, one of which combined historical popularity data with acoustic data and another which used only acoustic data. The acoustic data was gathered from 30-second clips of each song and included the Mel-Frequency Cepstral Coefficients, Spectral Centroids, Spectral Flatness, Zero Crossings, and Tempo [1]. On the other hand, Lee and Lee [2] also take "musical complexity" into account, which measures the structural changes of a song's harmonic, rhythmic, and timbral components. Like [1], they extract their songs' MFCCs and use MPEG-7 Audio tools to extract audio features. Similarly, Yang et al. in [4] use MFCCs for their audio features but take things a

step further by also using a music auto-tagging system called JYNet, which computes scores for 50 music tags including “genres, instruments, and other performing related tags such as male vocal, female vocal, fast and slow” [4]. These tags are closer to the types of acoustic features that I am planning to use in my work and are like those in The Million Song Dataset used by Pham et al. in [3]. The authors of [3] point out that the multitude of features extracted from songs in other studies can make models prone to overfitting. While they still rely on the acoustic features (duration, key, release year, etc.) and metadata (danceability, energy, etc.) of a song, they also utilize three different feature selection algorithms (forward/ backward stepwise selection and regularization) to minimize the number of audio features that will be considered by the model.

Predictive Models: Both Araujo et al. [1] and Lee and Lee [2] utilize Support Vector Machines (SVMs) to classify songs as popular vs. not popular. This model projects the data into a space of higher dimensionality during training. On the other hand, Yang et al. [4] use deep learning, specifically convolutional neural networks (CNNs), which learn audio features directly from the data and pass them through multiple layers of computation. Finally, Pham et al. use a multitude of models in [3], including SVMs, neural networks, logistic regression, Gaussian discriminant analysis, and linear regression.

One of the biggest limitations I see with these prior works is their treatment of time in collecting their data and training their models. Because it would be difficult to account for changing music trends over time, most works collect data from recent years. Araujo et al. [1] use song data that spans the course of three months, Lee and Lee [2] five years, and Yang et al. [4] one year. Only Pham et al. [3] use time as one of the features in their model, as The Million Song Dataset spans many decades.

Relevant Data:

My initial dataset was the Million Song Dataset, which is a “collection of audio features and metadata for a million contemporary popular music tracks.” The dataset contains many detailed acoustic features and descriptive tags for each song. Here is an example of some of the features for the song “Never Gonna Give You Up” By Rick Astley:

loudness: -7.75
general loudness of the track
mode: 1
estimation of the mode the song is in by The Echo Nest
mode_confidence: 0.434
confidence of the mode estimation
release: Big Tunes - Back 2 The 80s
album name from which the track was taken, some songs / tracks can come from many albums, we give only one

My first discovery was that the dataset was 30GB, which meant that I would have to use an EBS disk instance attached to an EC2 Amazon virtual machine and that using the disk costs money. Since this is not ideal, I decided to use the randomly selected 10,000-song subset. However, my second discovery was that this dataset was really complicated to use, as all the songs were distributed among a collection of HDF5 files. I soon realized that I was spending too much time trying to get my data in a clean tabular format and needed to find a different option, so I turned to Kaggle.

The two datasets I found on Kaggle were the Spotify Tracks Dataset and The Spotify Hit Predictor Dataset (1960-2019). The column descriptions for the Spotify Tracks Dataset are as follows:

- track_id: The Spotify ID for the track

- artists: The artists' names who performed the track. If there is more than one artist, they are separated by a ;
- album_name: The album name in which the track appears
- track_name: Name of the track
- popularity: The popularity of a track is a value between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are. Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently. Artist and album popularity is derived mathematically from track popularity.
- duration_ms: The track length in milliseconds
- explicit: Whether or not the track has explicit lyrics (true = yes it does; false = no it does not OR unknown)
- danceability: Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable
- energy: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale
- key: The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1
- loudness: The overall loudness of a track in decibels (dB)
- mode: Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0

- speechiness: Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audiobook, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks
- acousticness: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic
- instrumentalness: Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content
- liveness: Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live
- valence: A measure from 0.0 to 1.0 describes the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry)
- tempo: The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration
- time_signature: An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4.
- track_genre: The genre in which the track belongs

The column descriptions for The Spotify Hit Predictor Dataset (1960 - 2019) that are distinct from the previous dataset are as follows:

- uri: The resource identifier for the track.
- target: The target variable for the track. It can be either '0' or '1'. '1' implies that this song has featured in the weekly list (Issued by Billboards) of Hot-100 tracks in that decade at least once and is therefore a 'hit'. '0' Implies that the track is a 'flop'.The author's condition of a track being 'flop' is as follows:
    - The track must not appear in the 'hit' list of that decade.
    - The track's artist must not appear in the 'hit' list of that decade.
    - The track must belong to a genre that could be considered non-mainstream and/or avant-garde.
    - The track's genre must not have a song in the 'hit' list.
    - The track must have 'US' as one of its markets.
- chorus_hit: This is the author's best estimate of when the chorus would start for the track. It's the timestamp of the start of the third section of the track. This feature was extracted from the data received by the API call for Audio Analysis of that particular track.
- sections: The number of sections the particular track has. This feature was extracted from the data received by the API call for Audio Analysis of that particular track.

These datasets fulfilled the requirements for pursuing my research question since they contain quantitative acoustic features derived from each track's audio by Spotify's API, though the features are much less expansive than the ones contained in the Million Song Dataset. There are also some distinct differences between the datasets which affected the methods I used for each one. Firstly, the two datasets have two different definitions of popularity (highlighted in yellow). The first dataset has a numeric value for popularity ranging from 0-100,

which means that it is better suited for a linear regression problem. On the other hand, the second dataset simply classifies each song as popular or unpopular, which makes it better suited for a classification problem. In addition, the first dataset contains a genre feature but only pools songs from a collection of recent years while the second dataset contains a release year feature (spanning several decades) but does not reveal the song's genre.

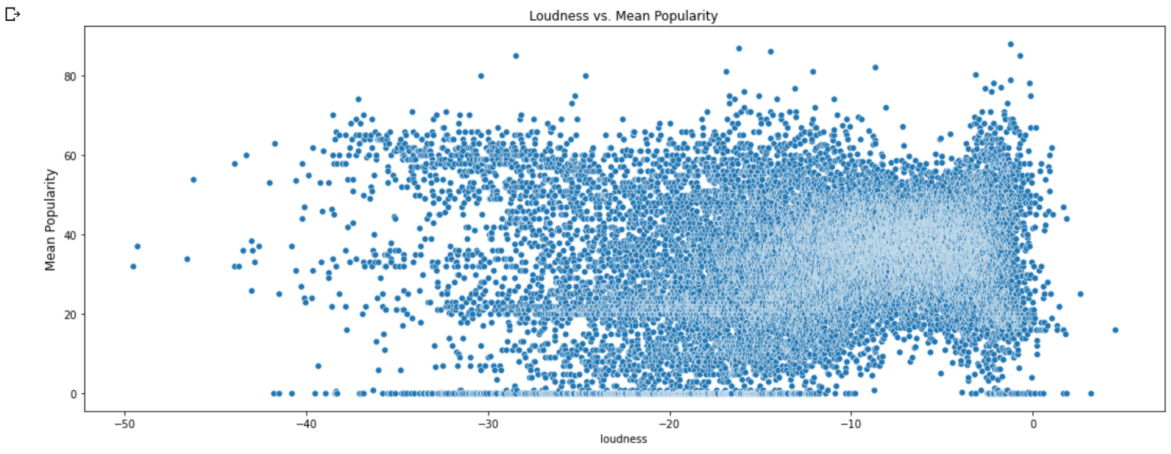
Analysis/ Modeling:

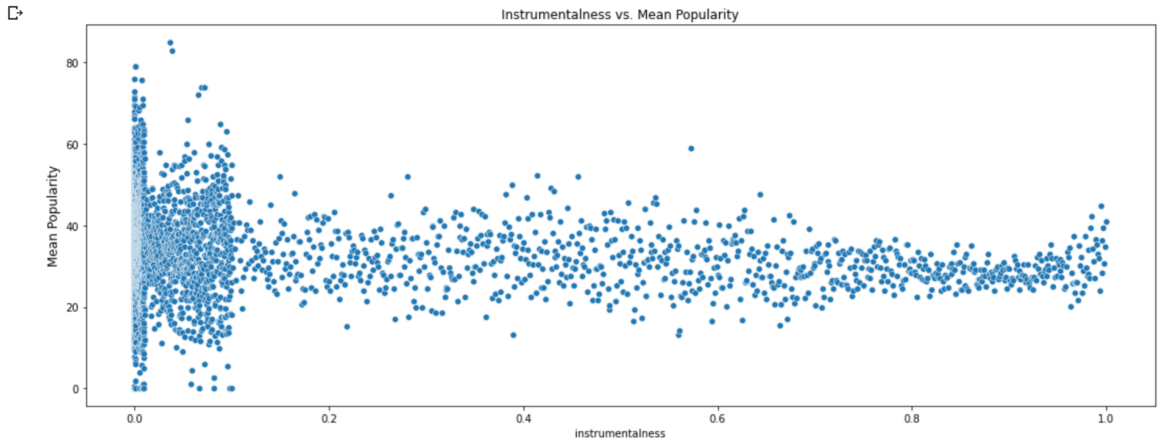
Dataset 1:

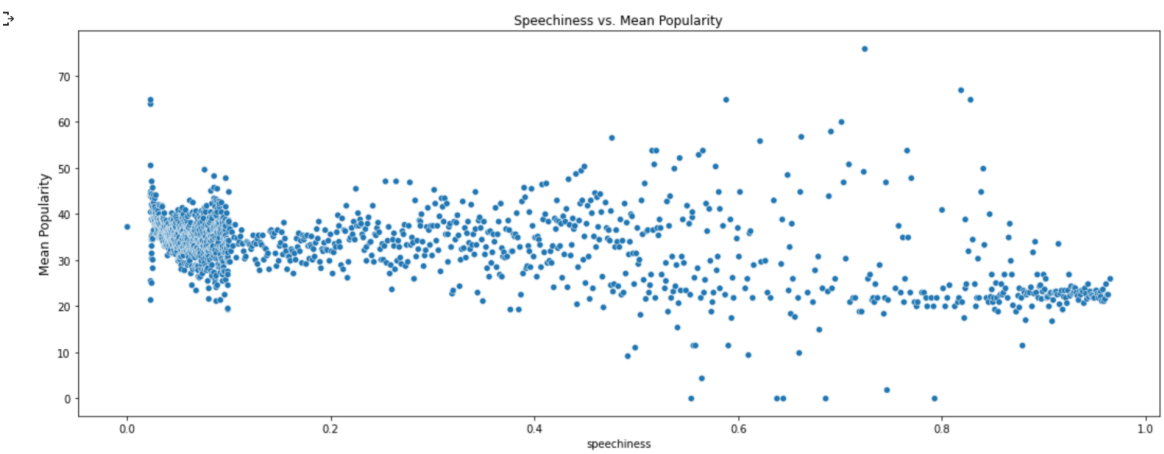
I decided to handle each dataset separately, starting with the Spotify Tracks Dataset, which has a numeric popularity value from 0-100 and a genre feature. I first performed some dataset analysis, including looking into the dataset's shape, columns, and column types. I found that the dataset had 114,000 rows and 21 columns, which meant that I would have to be careful about the types of computation I did since such a large number of data can take a very long time to process. Then, I dropped the 'track_id', 'track_name', 'artists', 'album_name', and 'Unnamed: 0' features from the dataset. These features were not relevant to my research question since I did not want to use any external features like artist or album popularity in my model. I did not want the model to learn, for example, that because the artist Doja Cat has many popular songs, Doja Cat is highly correlated with popularity, and any song she releases will be popular. Instead, I wanted my model to analyze the internal features (quantitative acoustic features) of a song to determine if it will be popular. This would create a model that is much more useful for, say, a small independent artist hoping to create their first big hit. After performing some data cleaning, like dropping duplicates and checking for missing values, I moved on to EDA.

I began my EDA with a basic correlation matrix and was a little dismayed to find that none of my features were strongly correlated with popularity, the highest correlation being 'instrumentalness' with a value of -0.142826. Since the correlation has a negative value, this means that popularity decreases as instrumentalness increases; in other words, generally, tracks with more vocal content rather than instrumentals are popular. After creating a pairplot on a random sample of the data (using all the data would have taken too much time and not have been interpretable in the image), I pick the three features with the highest correlation to zoom in on: 'loudness', 'instrumentalness', and 'speechiness'.

```
popularity          1.000000
duration_ms        -0.037658
explicit            0.052100
danceability        0.056825
energy             -0.007603
key                -0.001787
loudness            0.071638
mode               -0.003725
speechiness        -0.061695
acousticness       -0.023203
instrumentalness   -0.142826
liveness           -0.022800
valence            -0.014620
tempo               0.002464
time_signature      0.033359
Name: popularity, dtype: float64
```







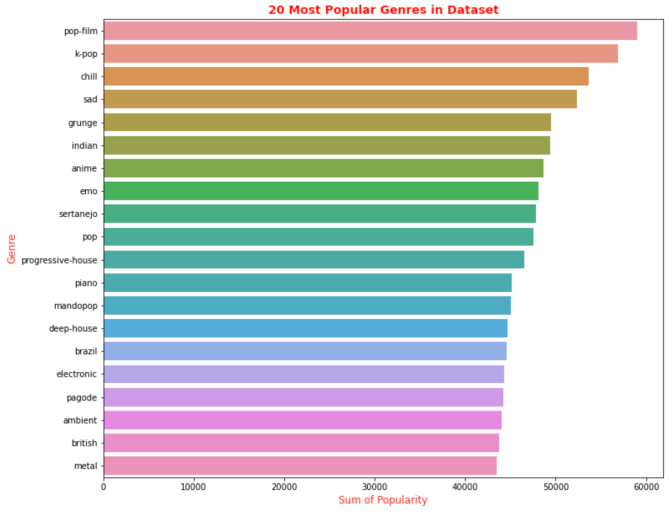
Loudness: Most of the songs are clustered between -15dB and 0dB. There is a wide range of popularity for the entire loudness range, with the mean popularity being around 40. Song with 0dB range from being not at all popular (0) to very popular (100). Songs with 0 popularity range from being very quiet (-60dB) to very loud (0dB).

Speechiness: There are two speechiness clusters in the dataset between 0-0.2 speechiness and 0.8-1 speechiness. There seems to be a very slight visible negative trend between speechiness and popularity, meaning that speechy songs are slightly less popular on average (which is confirmed by the negative correlation value). Songs with a speechiness above 0.5 and between 0-0.1 take on a wider range of popularity values, with the mean popularity otherwise being around 28 and most data points not deviating much from the mean.

Instrumentalness: There is a cluster of songs between 0-0.1 instumentalness, with those songs taking on a wide range of popularity values. Otherwise, the mean instrumentalness is around 30 with most data points not deviating much from the mean. There is also a slightly visible negative trend between instrumentalness and popularity, which I established earlier based on the negative correlation value.

As a whole, none of the three features show a clear positive or negative linear correlation with popularity; the points instead form a straight line across the mean, which is reflected by the correlation matrix values.

In addition to numerical features, I also examined the categorical ‘track_genre’ feature and learned that the three most popular genres were pop-film, k-pop, and chill. Given that two of these genres have “pop” in their names, I decided to create a baseline regression model without first encoding my genres, as I wanted to see whether it's possible to predict a song as popular not because it is from a popular genre, but based on its acoustic features alone.



My baseline model is a simple linear regression model with X being all the features besides ‘popularity’ and ‘track_genre’, and Y being ‘popularity’. I performed an 80-20 train-test split and used 5-fold cross-validation. I chose R^2 and RMSE as my evaluation metrics as they do the best job of explaining how a linear regression model is performing, including how much of the variability in the target variable can be explained by the model and how accurate the model is in fitting the data. This model performed exceptionally low, with a cross-validation R^2 score of 0.040935, a testing R^2 score of 0.041562, and an RMSE of 20.734094. This means that my baseline model explains only about 4% of the variability in popularity (R^2) and the predicted values in my model vary from the actual model by an average of about 20 “popularity units” (RMSE). In other words, there is a very weak linear relationship between popularity and the acoustic features of each song. This makes sense since from my EDA I saw that none of the features in my dataset were linearly correlated with popularity.

Given the poor performance of my baseline, I speculated that I may be able to fit my data better if I created polynomial features, which became my next approach. I used a sklearn Pipeline to standardize my data, create Polynomial features of degrees 2, 3, and 4 (the highest degree I could reach before I ran out of RAM), and perform another linear regression. Using this method, I was able to increase my training score to 0.088723 and testing score to 0.091359 for polynomial features of degree 3. However, by degree 4 both the training and testing scores fell into the negatives (-0.478863 training and -0.274130 testing). This was a bit surprising since I assumed that increasing polynomial features would always increase training performance, perhaps to the point of overfitting; but in this case, my training score was even worse than my testing. This tells me that perhaps there exists a degree of polynomial features that optimally fits a set of data that can be surpassed.

Despite the fact that my data wasn’t overfitting, I still try to improve it with Lasso regularization in the hopes that it will place less weight on insignificant features and improve my

performance. I try out different hyperparameters (0.1, 0.5, and 1 for Lasso) and found that with an alpha of 0.1 I am able to achieve a training score of 0.092067 and a testing score of 0.090542, which resulted in a drop in testing performance. Overall, these changes were very minute and my model was still not generating an acceptable training or testing score. Eventually, I decided to include the 'track_genre' feature in my model with the hopes that this would improve my performance. My reasoning was that without genres in my model, there was a lot of variation in acoustic features and no clear algorithm of which combinations of them lead to popularity. However, when coupled with the other acoustic features in my dataset, 'track_genre' may provide some helpful information to minimize this gap in knowledge; for instance, the model may learn that "a k-pop song with high danceability and medium energy will be more popular than a k-pop song with low danceability and high energy." The next step in my process was deciding how to encode my genres.

The first approach I tried was one hot encoding the genres so that each one was its own column. I transformed my data using sklearn's OneHotEncoder() and added it to my dataset. However, I realized right away that this change would bump my number of features up to 128, which may cause some issues further on. I perform some basic EDA by printing a correlation matrix for the top 10 most correlated features.

```
popularity          1.000000
iranian             0.148934
instrumentalness    0.142826
romance             0.136824
detroit-techno      0.109734
pop-film            0.108635
chicago-house       0.104606
k-pop               0.103401
latin               0.096645
grindcore           0.094308
Name: popularity, dtype: float64
```

While I saw that the correlation values have improved, I also noticed that the most correlated features were all genres (using absolute value), which made me concerned that the model would focus too much on genre and not enough on acoustic features.

After performing another train-test split and creating a basic linear regression model, I get a training score of 0.286793, a testing score of 0.280804, and an RMSE of 18.230213. This is the biggest jump in improvement I had yet to achieve in terms of R^2, but still not very good. I also realize that because I have so many features, using polynomial regression is out of the question as it would instantly crash my RAM. After trying Lasso regularization I find that Lasso regularization again only decreased my performance, which makes sense since the model wasn't overfitting much to begin with.

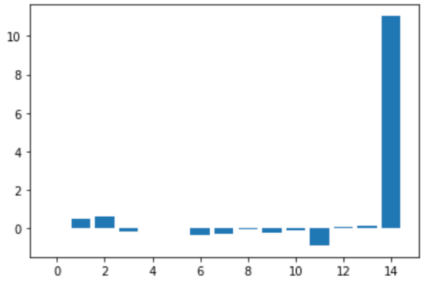
The next step in my process was figuring out a way to encode the genre variable without using so many features, in the hopes that polynomial regression would improve my performance even more. For this task, I chose target encoding, which replaced each instance of 'track_genre' with the mean popularity for that genre. To accomplish this, I install the category_encoders package. After creating an 'encoded-genre' column to replace my 'tack_genre' feature, I perform some basic EDA again via correlation matrix and heat map.

```
Unnamed: 0          0.032142
popularity          1.000000
duration_ms        -0.007101
explicit            0.044082
danceability        0.035448
energy              0.001056
key                -0.003853
loudness            0.050423
mode               -0.013931
speechiness        -0.044927
acousticness       -0.025472
instrumentalness   -0.095139
liveness           -0.005387
valence            -0.040534
tempo               0.013205
time_signature      0.031073
encoded_genre       0.504154
Name: popularity, dtype: float64
```

Not surprisingly, ‘encoded_genre’ had the highest correlation of 0.5 as the column’s value is the mean popularity for each genre. After performing another train-test split and retrying a basic linear regression model, I get a 0.257520 training score, a 0.261960 testing score, and an RMSE of 19.247089, which is slightly worse than the performance of my previous one-hot encoded data. Applying Lasso regularization again did not have much effect on the performance, which makes sense since the regularizer is best suited for overfitted data.

At this point I analyzed the feature importance of my model; I decided to do this before trying out polynomial regression because I didn’t want a large number of polynomial features in my analysis as those can be difficult to interpret and unintuitive. After standardizing the data, I looked at the Lasso regression model’s feature coefficients and found that, not surprisingly, ‘encoded-genre’ was the most important feature by a high margin, with a weight of 11.03635 (the second most important feature had a weight of -0.89626). In summary, the most important features in my model, which were determined by setting a cutoff of 0.1, were ['explicit' 'danceability' 'energy' 'mode' 'speechiness' 'instrumentalness' 'valence' 'time_signature' 'encoded_genre']. Both instrumentalness and speechiness were in the list of the top three most correlated features with a negative correlation, which explains why they’re also on the list of highly weighted features and have a negative weight. One way to analyze the feature importances would be to say that increasing ‘encoded_genre’ by one standard deviation would

```
Feature: 0, Score: 0.00103
Feature: 1, Score: 0.51706
Feature: 2, Score: 0.58031
Feature: 3, Score: -0.19732
Feature: 4, Score: -0.00151
Feature: 5, Score: 0.00000
Feature: 6, Score: -0.32916
Feature: 7, Score: -0.27289
Feature: 8, Score: -0.06509
Feature: 9, Score: -0.25611
Feature: 10, Score: -0.11524
Feature: 11, Score: -0.89626
Feature: 12, Score: 0.06241
Feature: 13, Score: 0.13495
Feature: 14, Score: 11.03635
Top Features:
['explicit' 'danceability' 'energy' 'mode' 'speechiness'
 'instrumentalness' 'liveness' 'valence' 'time_signature' 'encoded_genre']
```



increase the popularity score by about 11 units. Also, increasing speechiness and instrumentalness by one standard deviation would decrease popularity by about 0.3 units.

After performing polynomial regression again with features of degrees 2 and 3 (degree 4 was crashing my RAM), and then performing Lasso regularization on the degree 3 model, the best training and testing scores I got were 0.268476 and 0.273900 respectively for the Lasso model, which is still slightly worse than the one-hot encoded model performed.

At this point, I was still not satisfied with my model, as the performance was very poor and all the techniques I was trying were creating very minute changes in the outcome.

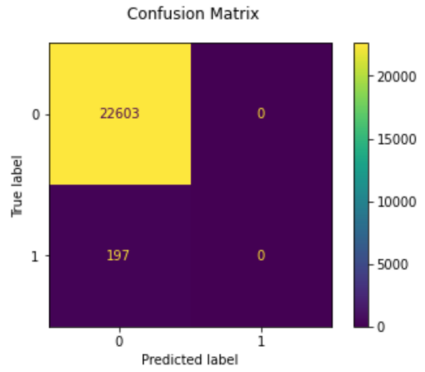
I decided to take a subset of the data for one genre: indie/ alternative. My thinking was that perhaps there was too much variation in acoustic features in the entire dataset, which was affecting performance. Since one genre would have similar combinations of acoustic features across all the songs, it might be easier for the model to learn which patterns lead to popularity. After looking at all the genres, I determined that ‘alternative,’ ‘indie,’ ‘alt-rock,’ and ‘indie-pop’ could be combined to create one large indie/ alternative dataset. After looking at the absolute value of this new dataset’s correlation matrix, I found that the features were slightly more correlated with popularity than my first pass at the dataset, which was promising. However, after running a basic linear regression model I got a training score of 0.064555, a testing score of 0.027475, and an RMSE of 31.022603. This told me that the model was beginning to overfit, as the training score is a bit higher than the testing score. Also, the testing performance and RMSE

```
popularity          1.000000
encoded_genre       0.201545
valence             0.130109
energy              0.119774
loudness            0.105450
acousticness        0.090135
instrumentalness    0.078549
danceability        0.038110
time_signature      0.034552
duration_ms         0.028185
Name: popularity, dtype: float64
```

were even worse than my first baseline model. This told me that I was taking my research in the wrong direction. Rather than pursue the idea further and try to improve the model a bit using past techniques, I switched gears and decided to reframe the question as a classification problem.

In order to accomplish this, I create another column in my dataset called ‘popular’ which has a 1 for each row where a track’s popularity is above 80 (an arbitrary number I picked) and a 0 for each row where a track’s popularity is less than or equal to 80. After creating another train-test split, I use a Support Vector Classification model with an ‘rbf’ kernel as my baseline model. Given the observed lack of linearity in this dataset, I thought it would be best to avoid the ‘linear’ kernel as it seems like my data is not easily linearly separable. I used a metrics.accuracy_score() as my metric, which simply gives a percentage of correct classifications (y_pred = y_true).

```
Confusion matrix:
[[22603     0]
 [  197     0]]
```




```
0.99170 training accuracy with a standard deviation of 0.00003
testing accuracy: 0.99136
```

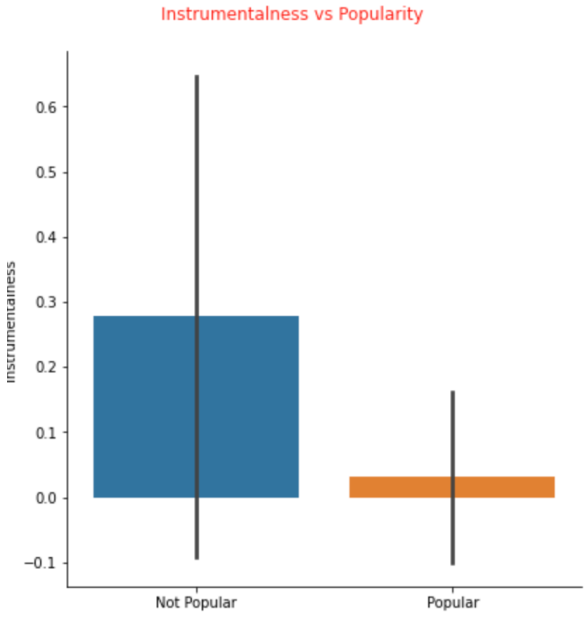
Amazingly enough, I achieved a training score of 0.99170 and a testing score of 0.99136. However, after looking at the confusion matrix, I quickly realized that my model was predicting only 0s and there was simply a small number of 1s in my data. Nevertheless, I decided to continue pursuing my question as a classification problem since classifying a song as “popular” or “ not popular” is a lot more intuitive and straightforward for users than predicting a popularity number. For instance, it may be unclear to a music artist how to interpret a

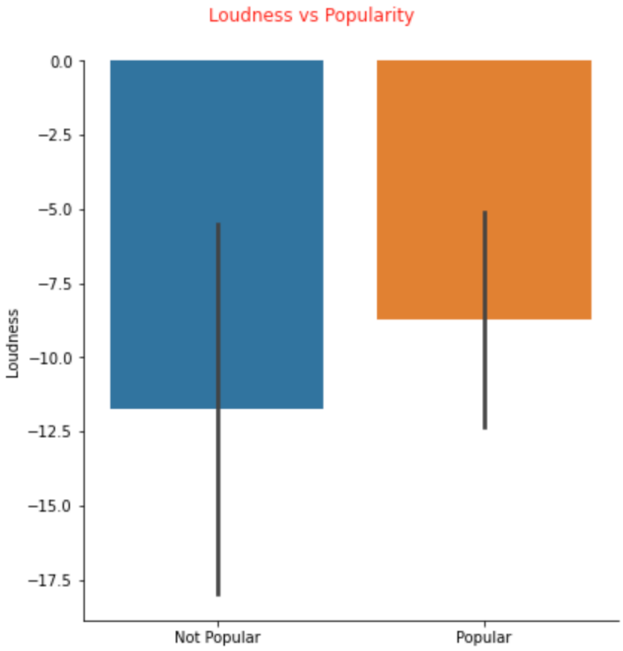
popularity score of “50”. In addition, from my EDA I saw there were some interesting clusters in the data which likely can’t be captured by a linear model, but could be handled by a classifier such as an SVC with an rbf kernel. However, instead of arbitrarily picking 80 as my cutoff number, I decided to transition to the second dataset, since its creator had a more precise/ non-arbitrary set of requirements for determining whether a song was a hit or a flop.

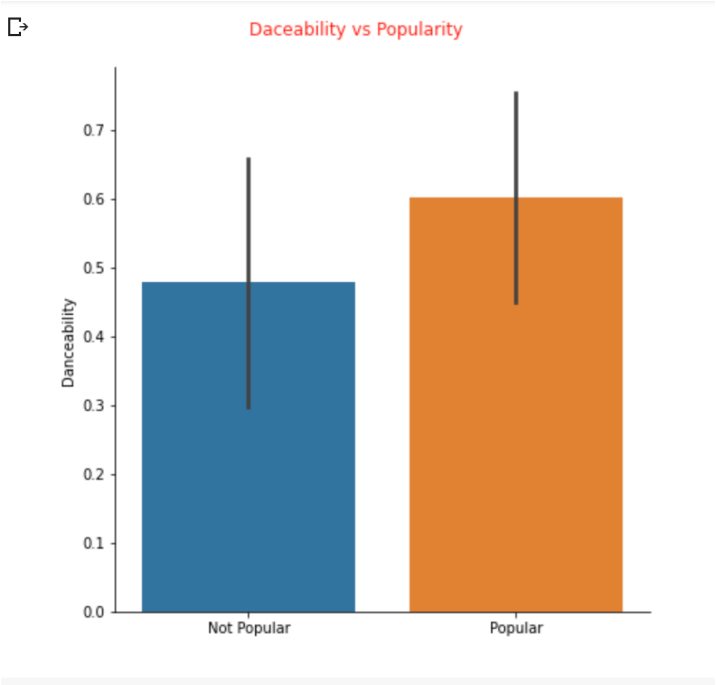
Dataset 2:

The second dataset was The Spotify Hit Predictor Dataset 1960 - 2019 which contains a release year feature and a binary target column that classifies each song as a hit or a flop. It was given to me as 6 separate datasets (one for each decade) so I decided to combine all of these datasets into one, which produced a dataset with 41106 rows and 19 columns. I dropped ‘track,’ ‘uri,’ and ‘artist’ as features for similar reasons as the last dataset. I also performed data cleaning by dropping duplicates and checking for missing values. My EDA followed a similar process as last time: creating a correlation matrix/ heat map, plotting a row of pair plots, and zooming in on the three most correlated features, which were danceability, loudness, and instumentalness.

```
danceability          0.345984
energy                0.177292
key                   0.010814
loudness              0.285327
mode                  0.079093
speechiness          -0.039944
acousticness         -0.245931
instrumentalness     -0.405726
liveness             -0.051373
valence               0.249269
tempo                 0.032849
duration_ms          -0.073781
time_signature        0.104248
chorus_hit           -0.044969
sections             -0.060129
target                1.000000
Name: target, dtype: float64
```







Instrumentalness: In this dataset, instrumentalness ranges from -0.1 to 0.6. For unpopular songs, 50% of songs are between 0 and 0.275 instrumentalness. For popular songs, 50% of the songs are between 0 and 0.05 instrumentalness.

Loudness: In this dataset, loudness ranges from 0 to -17.5dB. For unpopular songs, 50% of the songs are between 0 and -11dB. For popular songs, 50% of the songs are between 0 and 9dB.

Danceability: In this dataset, danceability ranges from 0 to 0.8. For unpopular songs, 50% of the songs are between 0 and 0.475 danceability. For popular songs, 50% of the songs are between 0 and 0.6 danceability.

While there are differences between the distribution of data for popular and unpopular songs, there's also a lot of overlap for the 50% ranges between the two categories, which is evident from their weak linear correlations.

After my EDA, I get started on modeling, starting with an SVC model with its default parameters and an 'rbf' kernel. Like the last dataset, the low correlation values told me that my data was not linearly separable which made me hesitant to use a 'linear' kernel. I performed an

80-20 train-test split and used 5-fold cross-validation. My model performed much better than the baseline model for my first dataset, with a training accuracy of 0.599026, a testing accuracy of 0.601464, and an F1 score of 0.67. If I had to choose, I think precision, the model's ability to not label an unpopular song as popular is a bit more relevant to my problem, as it would prevent artists and labels from wasting their time and money on a flop. However, recall, the ability of my model to correctly classify all the popular songs as popular is also important as a poor recall could lead to missed opportunities and wasted potential among music artists. Therefore, I chose F1 as my evaluation metric as it balances precision and recall, giving me the best of both worlds.

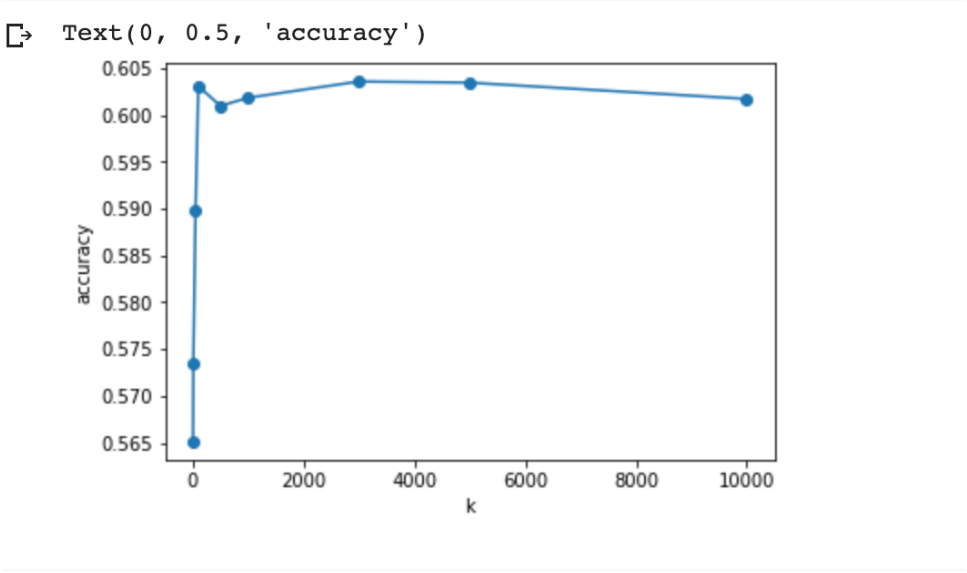
I decided to try to make my model better by using grid search to tune my hyperparameters. The parameters I chose were C (with values of 1, 10, 100) and gamma (with values of 1e-2, 1e-3, 1e-4). After about an hour, my grid search finished and produced the following as the best parameters.

```
[CV 1/5] END ................C=10, gamma=0.0001;, score=0.575 total time= 2.7min
[CV 2/5] END ................C=10, gamma=0.0001;, score=0.566 total time= 2.7min
[CV 3/5] END ................C=10, gamma=0.0001;, score=0.569 total time= 3.0min
[CV 4/5] END ................C=10, gamma=0.0001;, score=0.571 total time= 2.8min
[CV 5/5] END ................C=10, gamma=0.0001;, score=0.566 total time= 2.7min
[CV 1/5] END ..................C=100, gamma=0.01;, score=0.541 total time= 2.2min
[CV 2/5] END ..................C=100, gamma=0.01;, score=0.541 total time= 2.2min
[CV 3/5] END ..................C=100, gamma=0.01;, score=0.544 total time= 2.3min
[CV 4/5] END ..................C=100, gamma=0.01;, score=0.551 total time= 2.3min
[CV 5/5] END ..................C=100, gamma=0.01;, score=0.549 total time= 2.2min
[CV 1/5] END ................C=100, gamma=0.001;, score=0.563 total time= 3.1min
[CV 2/5] END ................C=100, gamma=0.001;, score=0.550 total time= 3.1min
[CV 3/5] END ................C=100, gamma=0.001;, score=0.556 total time= 3.1min
[CV 4/5] END ................C=100, gamma=0.001;, score=0.570 total time= 3.2min
[CV 5/5] END ................C=100, gamma=0.001;, score=0.569 total time= 3.1min
[CV 1/5] END ...............C=100, gamma=0.0001;, score=0.565 total time= 7.0min
[CV 2/5] END ...............C=100, gamma=0.0001;, score=0.556 total time= 7.1min
[CV 3/5] END ...............C=100, gamma=0.0001;, score=0.567 total time= 6.9min
[CV 4/5] END ...............C=100, gamma=0.0001;, score=0.569 total time= 7.1min
[CV 5/5] END ...............C=100, gamma=0.0001;, score=0.571 total time= 7.2min
{'C': 1, 'gamma': 0.0001}
SVC(C=1, gamma=0.0001)
```

However, after rebuilding my model with those parameters, I found that the performance was worse than the original, which really confused me at first. I then realized that this could

happen if the parameters I checked in my grid search performed worse than the default parameters of the model. This was confirmed when I checked the default gamma for my first model and found that it was 1.527463e-11, while the smallest gamma I checked in my grid search was 1e-4.

Next, I moved on to a KNN model, testing it first on the default parameters and receiving a training score of 0.566059, a testing score of 0.565120, and an F1 score of 0.58, which is a slightly worse performance than the SVC.

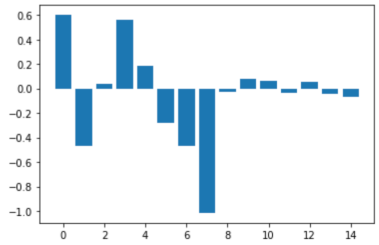


I then performed hyperparameter tuning for the number of nearest neighbors, trying the values 5, 10, 50, 100, 500, 1000, 3000, 5000, and 10000. I found that 3000 neighbors produced the best results, bumping the training accuracy up to 0.595801, the testing accuracy up to 0.601836, and the F1 score up to 0.66.

Next, I moved on to a logistical regression model, standardizing my data through a pipeline and trying the default parameters for the first pass. There was a drastic improvement in the performance, with a training score of 0.728135, a testing score of 0.722277, and an F1 score of 0.74. I analyze the features by observing the model's coefficient weights and found that the most important features are ['danceability' 'energy' 'loudness' 'acousticness'

'instrumentalness'] after setting a cutoff of 0.3, with the most important feature being instrumentalness with a weight of -1.01535. This makes sense since it was also on the list of most correlated features, as well as danceability and loudness. One way to analyze the feature importance is to say that increasing instrumentalness by 1 standard deviation is associated with multiplying the odds of the song being popular by e^(-1.01) or about 0.36 (lower probability). Also, increasing danceability and loudness by 1 standard deviation is associated with multiplying the odds of the song being popular by e^(0.6) or 1.82 (higher probability).

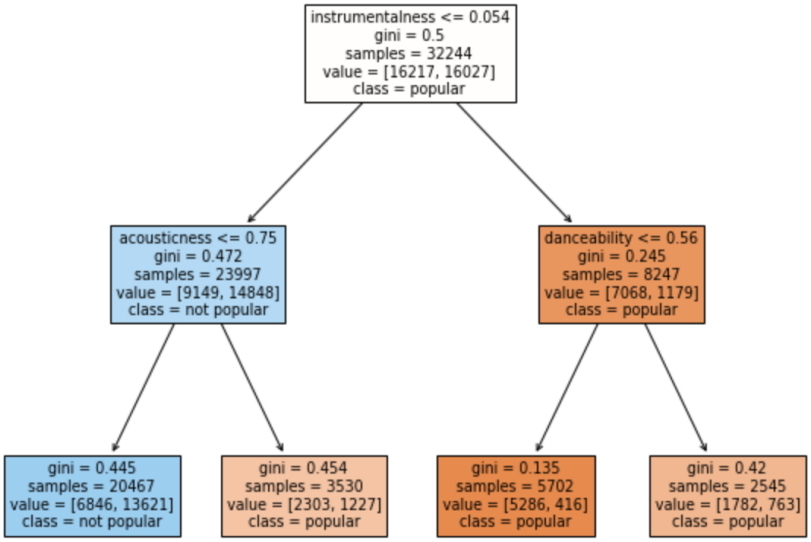
```
Feature: 0, Score: 0.59978
Feature: 1, Score: -0.46979
Feature: 2, Score: 0.03506
Feature: 3, Score: 0.55976
Feature: 4, Score: 0.18867
Feature: 5, Score: -0.28083
Feature: 6, Score: -0.46760
Feature: 7, Score: -1.01535
Feature: 8, Score: -0.02609
Feature: 9, Score: 0.07902
Feature: 10, Score: 0.06541
Feature: 11, Score: -0.03566
Feature: 12, Score: 0.05787
Feature: 13, Score: -0.04613
Feature: 14, Score: -0.06613
Top Features:
['danceability' 'energy' 'loudness' 'acousticness' 'instrumentalness']
```



Next, I moved on to a Gaussian NB model, using the default parameters. For this model, I got a training score of 0.624953, a testing score of 0.623419, and an F1 score of 0.71, which is a decrease in performance compared to the logistic regression.

After that, I started on decision tree and random forest modeling. My baseline decision tree used the default parameters and achieved a training score of 0.696098, a testing score of 0.682833, and an F1 score of 0.68. I performed hyperparameter tuning by adjusting the max_depth parameter, inputting the values [5, 6, 7, 8, 9, 10, 50, 100], and found that the best

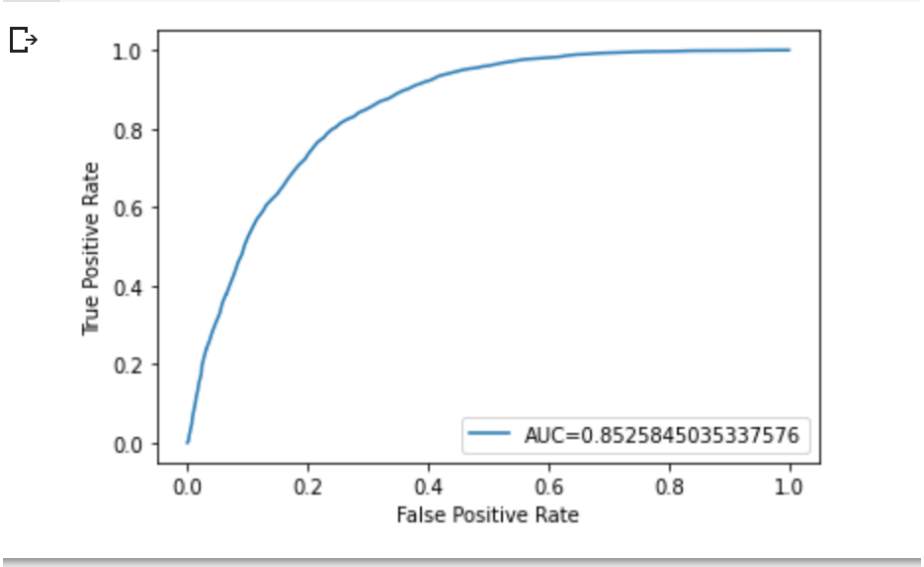
max_depth was 8 levels. This bumped my training score to 0.749070, my testing score to 0.742992, and my F1 to 0.77. For visualization purposes, I also plotted a diagram for a decision tree of max_depth 2, as a max_depth of 8 made it difficult to see the features that were splitting each branch. I also analyzed the feature importances for my best decision tree model by using the .feature_importances_ attribute. I found that instrumentalness was the most important feature, with an importance of 0.46417. I isolated the most important features by choosing a cutoff of 0.1 and found that overall, ['danceability' 'acousticness' 'instrumentalness'] were the most important features. This makes sense to me since instrumentalness and danceability were both highly correlated with popularity. Since all of these values are positive they have a high contribution towards making positive classifications.



Finally, I created a random forest model with the default parameters and got a training score of 0.694982, a testing score of 0.681097, and an F1 score of 0.68. After performing hyperparameter tuning for max_depth with the values [5, 10, 50, 100, 400, 500, 600, 1000], I found that the optimal max_depth was 600 levels. This bumped the performance of my model up to a training score of 0.780734, a testing score of 0.778715, and an F1 score of 0.79, my

highest performance yet! Analyzing the feature importances produced very similar results to the decision tree, with ['danceability' 'acousticness' 'instrumentalness'] having the highest importance when using a cutoff of 0.1 and instrumentalness having the highest overall importance of 0.16409. Since all of these values are positive they have a high contribution towards making positive classifications.

The final step in my modeling process was plotting an ROC curve for my best-performing model, the optimized random forest model. I found that the weight under the curve was 0.85. Since the AUC provides a collective measurement of performance for all decision thresholds, an area of 0.85 tells me that the model does a good job of making correct classifications. Moreover, the model hugs the upper left-hand corner, which means that there is a decision threshold for which the model will have a high true positive rate and a low false positive rate simultaneously. From the AUC, I can conclude that my model has an 85% chance of correctly classifying a sample.



Results:

The first discovery I made was that my method of narrowing my data down to one genre in order to reduce variability in acoustic features did not improve my linear regression model as I thought it would. This could be because indie/ alternative is a more general genre/ label than I originally thought, so it may not have reduced the variability in features enough. An idea for future work would be to choose an even more specific genre, such as 'chicago-house' or 'bluegrass' to see if that could allow me to create a successful linear regression model.

I also learned that my question is much more suited for classification than linear regression. This is likely because both of my datasets are not linearly separable and need more complicated algorithms to determine trends and patterns in the data. In terms of classification, my second dataset performed better than the first. This could be because data spanning multiple decades with the release year provided as a feature gives the models more useful knowledge to make predictions. However, given that release year was never labeled as an important feature this is not likely. Perhaps, the second dataset just had “better” data or the author defined a popularity metric that was more correlated with acoustic features.

Across all my datasets and models, instrumentalness always appeared in my lists of highly correlated features during EDA and most important features during my modeling. I can therefore conclude that instumentalness is a very high indicator of popularity. It seems that in general, people prefer listening to songs with a lower instumentalness level, with completely instrumental tracks rarely being popular. Some other features that were commonly assigned a high correlation/ weight were loudness, danceability, and speechiness. It seems that in general, tracks that are loud and danceable are more popular and tracks that are very speechy are less popular. This makes sense since tracks that are completely spoken word are not likely to be played on the radio as a top hit, but loud songs that people can dance to are.

The best linear regression model I developed was the model that included one-hot encoded genres, which had a testing score of 0.280804. The best classification model I developed was the optimized random forest model with a max depth of 600 levels and a testing

score of 0.778715. I believe my random forest model performed the best because it aggregates multiple decision trees with random samples of data, meaning those trees will all consider different acoustic features. This makes it a very expansive model that considers a lot of different features and performs efficient feature reduction on those that are not significant.

If I were to continue my research, I would likely try more advanced models such as a convolutional neural network that could learn the relevant acoustic features directly from audio samples. I would also look into developing my own popularity metric using Spotify charts. One topic I'm interested in is micro-trends, which is when trends stay popular for a very short period of time such as a month or even a week. It would be interesting to see if I could fine-tune my model to capture musical micro-trends. This may require using more time-specific Spotify charts (such as weekly or monthly charts) to develop my popularity metric.

Works Cited

[1] Araujo, Carlos, Marco Cristo, and Rafael Giusti. "Predicting Music Popularity on Streaming Platforms." *Anais do XVII Simpósio Brasileiro de Computação Musical, São João del-Rei, 2019*. SBC, 2019, pp.141-148.

[2] J. Lee and J. -S. Lee, "Music Popularity: Metrics, Characteristics, and Audio-Based Prediction," in IEEE Transactions on Multimedia, vol. 20, no. 11, pp. 3173-3182, Nov. 2018, doi: 10.1109/TMM.2018.2820903.

[3] Pham, James, Edric Kyauk, and Edwin Park. "Predicting song popularity." *Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Tech. Rep* 26 (2016).

[4] L. -C. Yang, S. -Y. Chou, J. -Y. Liu, Y. -H. Yang and Y. -A. Chen, "Revisiting the problem of audio-based hit song prediction using convolutional neural networks," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2017, pp. 621-625, doi: 10.1109/ICASSP.2017.7952230.