Project L: Final Submission

Ilona Demler, Daniela Garcia, Kayla Manning, and Saul Soto
December 10, 2021

This report will include the following: a description of the data, EDA, our research question, a description of the model selection process, and an analysis of our results. The materials used to generate this report can be found on our GitHub Repository.

1 DATA DESCRIPTION

1.1 RAW DATA

This project will utilize the Million Playlist Dataset, which consists of a sample of 1 million public playlists from over 4 billion public playlists on Spotify. Consisting of over 2 million unique tracks from nearly 300,000 artists, the dataset contains public playlists created by Spotify users between January 2010 and October 2017.

The dataset, sourced from AIcrowd, contains the following columns:

- `pid` - integer - playlist id - the MPD ID of this playlist (integer between 0 and 999,999)
- `name` - string - the playlist name
- `description` - optional string - the description given to the playlist (relatively new feature of Spotify, so most playlists do not have descriptions)
- `modified_at` - seconds - timestamp (in seconds since the epoch) when this playlist was last updated. Times are rounded to midnight GMT of when the playlist was last updated
- `num_artists` - the total number of unique artists for the tracks in the playlist
- `num_albums` - the number of unique albums for the tracks in the playlist
- `num_tracks` - the number of tracks in the playlist
- `num_followers` - the number of followers this playlist had at the time the MPD was created (does not include the playlist creator)

- num_edits - the number of separate editing sessions. Tracks added in a two-hour window count as a single editing session
- duration_ms - the total duration of all the tracks in the playlist (in milliseconds)
- collaborative - boolean - if true, the playlist is collaborative and allows multiple users to contribute tracks
- tracks - an array of information about each track in the playlist. Each element in the array is a dictionary with the following fields:
- track_name - the name of the track
- track_uri - the Spotify URI of the track
- album_name - the name of the track's album
- album_uri - the Spotify URI of the album
- artist_name - the name of the track's primary artist
- artist_uri - the Spotify URI of the track's primary artist
- duration_ms - the duration of the track in milliseconds
- pos - the position of the track in the playlist (zero-based)

Playlists are sampled with randomization and anonymized to protect user privacy. The dataset is manually filtered to maximize quality and to remove offensive content, and some playlists have fictitious tracks added as noise. Therefore, the playlists in this dataset are not representative of the overall population of playlists on Spotify.

- Original Data Site and Data Download
- Downloaded Sliced and Joined Raw Data.

1.2 Processed Data

To examine the data and build our model, we first joined the 1000 data slices into one aggregate data frame. From there, we sampled 100,000 observations from the 1,000,000. Next, we cleaned the data and generated new predictors.

We created the binary variables collaborative and has_description, which indicate whether the playlist is collaborative and/or has a description, respectively.

Next, we created the binary variable popular_name, which indicates whether each playlist name contains a word listed as one of the most popular in the stats.txt file provided in the data download. We chose to generate this variable to generalize the playlist names variable, as there are too many unique playlist names to reveal any significant trends. This allows us to use the names as a more effective predictor.

Then, we iterated through each playlist and counted the total number of popular songs and artists (according to the stats.txt file) and saved the counts in variables total_popular_tracks and total_popular_artists.

In the end, our cleaned dataset contains the following variables, which we can now use to conduct EDA: popular_name, has_description, num_artists, num_albums, num_tracks, duration_ms, collaborative, total_popular_tracks, total_popular_artists, and num_followers (outcome variable).

2 Exploratory Data Analysis

2.1 Outcome Variable

First, we need to determine whether the response variable should be continuous or categorical. To do this, we first examine the distribution of playlist followers in the training dataset (after filtering out playlists with less than 3 followers, to reduce some of the skewing in the data):

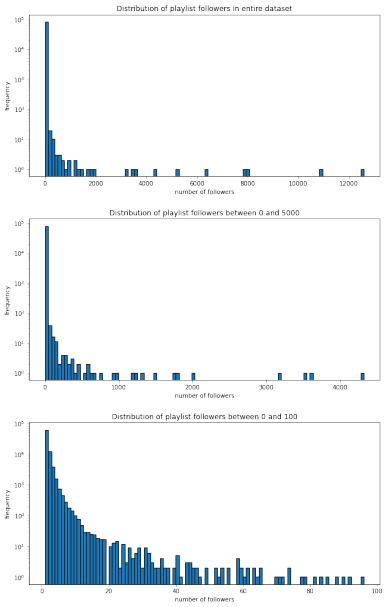

Figure 2.1: Distribution of playlist followers in training dataset over: all ranges, 0-5000 followers, and 0-100 followers. Y-axis is log-scaled. The majority of playlists have under 20 followers, but there are peaks at higher ranges as well, which suggests that we could use a categorical outcome variable.

Based on these histograms, the data appears to follow a very right-skewed distribution, with most playlists having fewer than 20 followers.

We considered translating `num_followers` into a binary categorical variable with a threshold of 20, with playlists with 20+ followers being considered "popular". However, there were very few playlists with greater than 20 followers, which raises concerns about overfitting the "popular" category. For our baseline model, we decided to stick with regression.

2.2 Outcome Variable vs Each Predictor

Before constructing our model, we explored the distributions of our different variables. First, we generated visualizations that treated our response variable as categorical, classifying a playlist as popular or unpopular based on if it fell above or below 20 followers. The below plots reveal that popular and unpopular playlists follow the same general distributional shape for

each of the predictors. However, the distributions of the popular playlists tend to have much greater variance. On the other hand, unpopular playlists tend to fall on the extremely low end of the spectrum. Most of these values are clustered near 0 for continuous predictors or in the 0 category for binary predictors. This likely results from people creating playlists but not fully developing them. These incomplete, small playlists have few albums, few tracks, few artists, and therefore few followers.

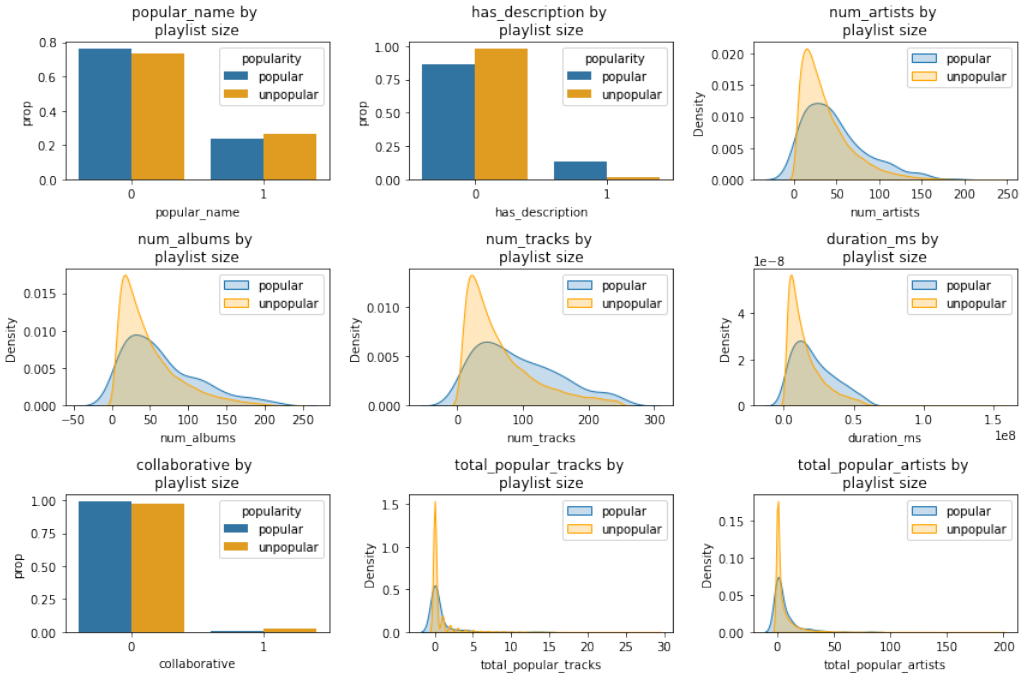

Figure 2.2: Comparing the predictors to a categorical outcome variable.

We then generated a second set of plots, treating the number of followers as a continuous variable. These plots map the number of followers along the y-axis, rather than treating them as two separate distributions. We glean similar results from these plots. Overall, there are very few observations with large numbers of followers. However, the pattern emerges that collaborative playlists and those that have descriptions or popular names tend to have a greater number of followers on average.

Based on the scatterplots, we see a negative trend between the follower count and the number of artists, playlist duration, total popular tracks, and total popular artists.

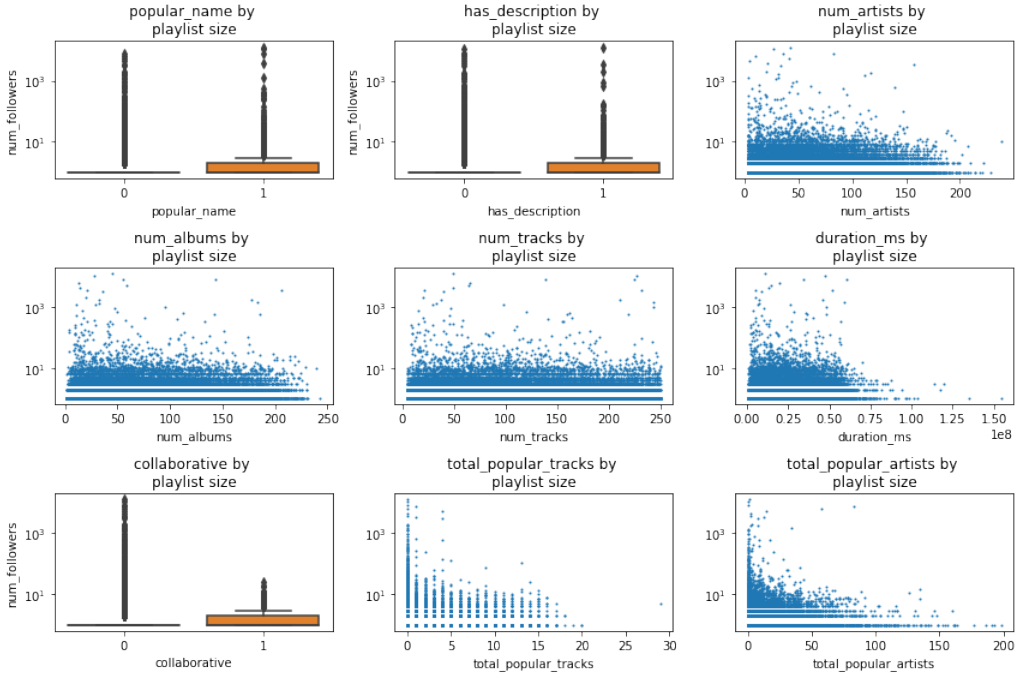

Figure 2.3: Comparing the predictors to a continuous outcome variable.

2.3 Collinearity between Predictors

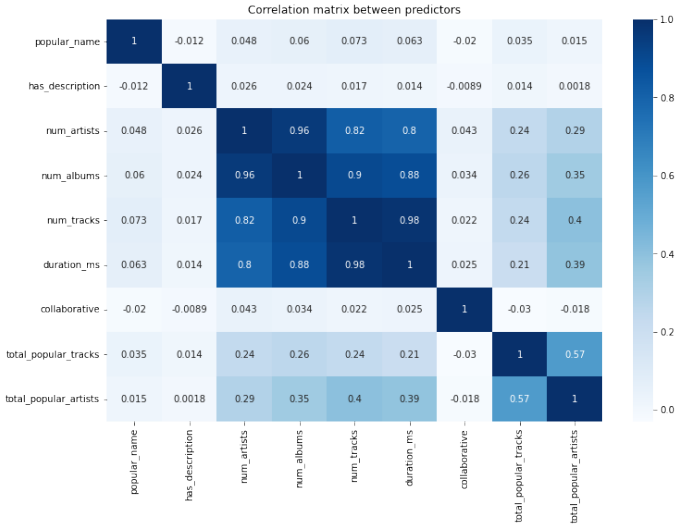

Figure 2.4: Correlation matrix of predictors in train dataset. Values close to 1 indicate high collinearity.

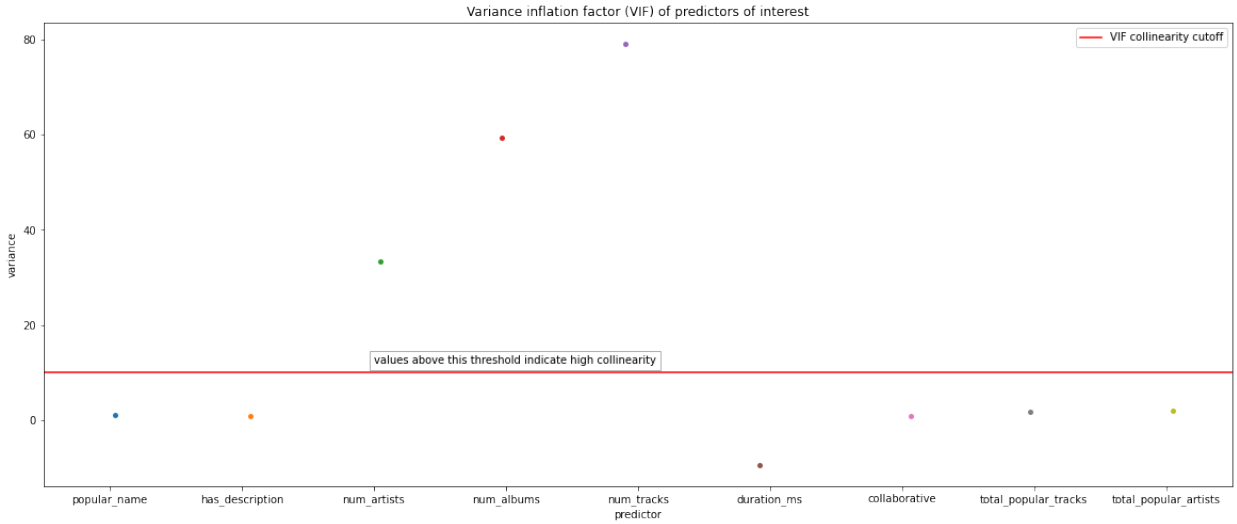

Figure 2.5: Variance inflation factor (VIF) plot showing the amount of variance on the regression coefficients due to collinearity. VIF values above 10 are considered highly collinear.

.

After conducting some baseline plots, we assessed the potential collinearity between predictors. Our plots reveal a strong correlation between several of the predictors.

In the correlation matrix, we see 6 strong correlations: `num_tracks` and `duration_ms` (0.98), `num_albums` and `num_artists` (0.95), `num_albums` and `num_tracks` (0.90), `duration_ms` and `num_albums` (0.88), `num_tracks` and `num_artists` (0.80), `duration_ms` and `num_artists` (0.77).

In the VIF plot, four predictors lay above the threshold, and those with the highest correlation between other predictors (in ascending order) are `num_artists`, `num_albums`, `duration_ms`, `num_tracks`.

The correlated predictors tend to make sense; for example, longer playlists will tend to have more tracks. Furthermore, since the number of albums is not highly correlated with the number of artists, we should expect to see multiple albums from the same artist (or vice versa) in some of the playlists. When we go on to perform our regression, we will have to consider the relationship between these four predictors as they can skew our results to the extreme.

2.4 Missing Data

Since this dataset is processed in advance there is no missingness in any of our tables. However, the playlist description fields (denoted by the `has_description` variable) are sometimes empty, because user-provided description fields are a relatively new feature on Spotify. We don't anticipate this variable will contribute significantly to our models.

3 Research Question

What drives popularity (quantified by the number of followers) for a Spotify playlist?

Our predictors of interest include an indicator for whether playlist name includes commonly used words; an indicator for whether the playlist has a description; the number of artists, albums, and tracks included on the playlist; the duration of the playlist; an indicator for whether the playlist is collaborative; and the number of commonly played songs and artists featured in the playlist.

Our ultimate goal is to build an effective regression model using some, if not all, of these predictors, to predict the number of followers of a playlist. We can then discuss what number of followers would deem a playlist to be "popular".

4 Initial Baseline Model

Since we are working with a continuous outcome (`num_followers`), we decided to build a regression model. For our baseline model, we chose an interpretable, singular `DecisionTreeRegressor` with a `max_depth` of 3. This value was chosen fairly naively, and we plan to perform cross-validation on future iterations of the model to tune this parameter. We used all predictors in the dataset for this model. Our goal with this baseline model was to see which predictors initially drive the variance in the data (the splits), which ended up being `total_popular_tracks` and `num_tracks`. As a result, our `DecisionTreeRegressor` model is as follows, with a training MSE and R2 of 60104.36 and 0.15, respectively, and a test MSE and R2 of 92953.20 and 0, respectively. We will then refine this baseline model with parameter tuning, bootstrapping, bagging, and boosting approaches with the aim to improve predictive quality.

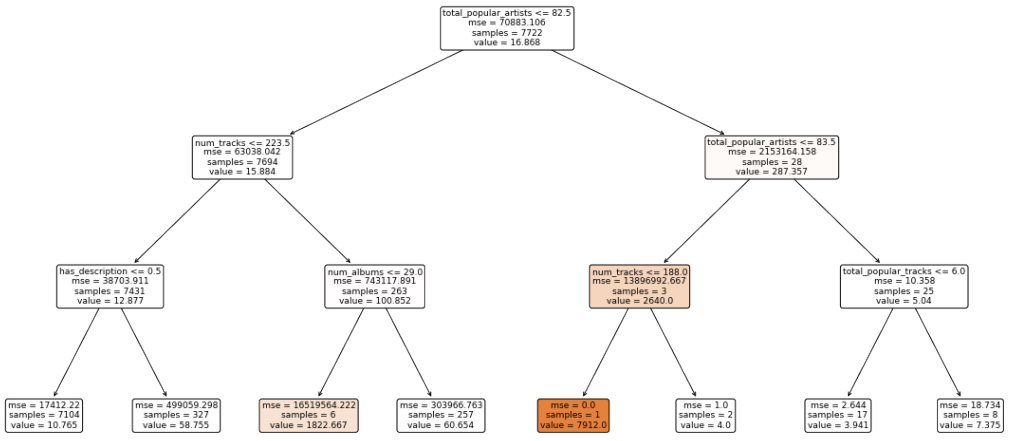

Figure 4.1: `DecisionTreeRegressor` plot showing the decision boundaries for the baseline model.

5 Model Fitting

We decided to explore several different models to answer our research question of determining what makes a playlist popular, namely linear regressions, bagging, and gradient boosting. We used parameter tuning, bagging, and boosting approaches to determine if we could improve the predictive quality of our model beyond the simplistic baseline approach. Our general approach was to split our data into train, test, and (if applicable) validation sets, and then train our models. If we were hyper-parameter tuning, we iterated through various values and then used the parameters that gave the best MSE score on the training dataset, and then evaluated on the testing dataset.

5.1 Linear Regression

A simple linear regression will tell us whether there is a relationship between our predictor variables and the outcome variable, and if there is, what the strength of this relationship is. We immediately notice that the `num_tracks` variable has approximately no impact on the outcome variable, whereas the `has_description` variable has a very high impact on it (Figure 5.2). Only two variables with negative coefficients: whether the playlist `is_collaborative` and the `total_popular_tracks`.

```
popular_name: 9.32
has_description: 54.81
num_artists: 0.53
num_albums: -1.19
num_tracks: 0.54
duration_ms: 0.00
collaborative: -9.40
total_popular_tracks: -3.19
total_popular_artists: 0.64
```

Figure 5.1: Linear regression coefficients

5.2 Linear Regression with CV

We next tested a linear regressor with cross validation. By performing ten-fold cross validation, we address the potential problem of overfitting on the training set that we could have encountered when simply training a linear regression model on one training set and evaluating on one test set. Here, we noticed that the cross-validation scores for both the training and test sets are roughly the same as they were above, for the simple linear regression.

5.3 Bagging

A bagging regressor fits many relatively complex decision trees and returns the average of each tree's outputs. This approach allows for high expressiveness in that each of the deep trees can approximate complex functions and decision boundaries. However, it also maintains a

low variance since averaging the predictions across all models will reduce the variance in the final prediction. In the context of this dataset, each of our deep trees will consider the many variables used as predictors. However, averaging will help to reduce the impact of overfitting our training set.

While bagging has benefits for prediction, it does lose much of the interpretability offered by our linear regression. The previous linear regression allowed us to examine which terms have the largest and most significant association with the follower count of a playlist. Bagging, on the other hand, aggregates many models and no longer allows us to trace the logic of the output through a single tree.

Despite its inability to isolate the variables that drive playlist popularity, a bagging model can produce a strong predictive model for a playlist's number of followers. To accomplish this, we first fine-tuned the maximum depth and number of bootstraps with a validation set. After determining which combination of depths and bootstraps minimized the validation MSE, we fit a bagging regressor on the entire train set and recorded its MSE and R^2 in the relevant dictionaries. We see that our test MSE is slightly higher for this bagging regressor than it was for our linear regression with cross-validation.

5.4 Gradient Boosting

Boosting improves decision trees in a way different from a bagging approach; while bagging decreases the variance of complex trees, boosting decreases the bias of simple trees. Essentially, additively combines an ensemble of shallow trees into a single, more complex model. More specifically, a gradient boosting regressor will iteratively add simple models that compensate for the weaknesses of the current ensemble.

Once again, we fine-tuned our hyperparameters on a validation set. Once determining the proper parameters, we fit our gradient boosting regressor on the entire training dataset. Our gradient boosting regressor has the lowest test MSE of all of the above models, indicating that our dataset benefited from its expressive abilities.

6 Conclusion

6.1 Model Selection and Implications of Results

Based on the two dataframes above, the `BoostingRegressor` reported the lowest MSE on the test set. The R2 values were all negative on the test set, indicating that our model fitting was no more successful than predicting the mean `num_followers` as the true `num_followers` for each playlist.

As a result, out of the options above, the `BoostingRegressor` is the "best" model because it resulted in the lowest MSE.

The reason why our models could have performed so poorly is due to the disparity between what made a playlist popular or not. Out of the one million playlists in the dataset, only 0.003% (< 3000) had 20 or more followers (our threshold for 'popular'). This number might have been even smaller during our training and testing of our models, leading to our models not being able to find a trend to discern playlist features that affect popularity.

Mean Squared Error Results:

	train	test
DecisionTreeRegressor	60104.356338	92953.198955
LinearRegression	70424.983482	92976.603174
LinearRegressionCV	71070.471543	93065.692295
BaggingRegression	70409.712791	93023.635883
BoostingRegressor	70796.534384	92724.305320

R2 Results:

	train	test
DecisionTreeRegressor	0.152064	-0.002685
LinearRegression	0.006463	-0.002938
LinearRegressionCV	-0.194040	-0.665072
BaggingRegression	0.006679	-0.003445
BoostingRegressor	0.001221	-0.000216

Figure 6.1: `Final MSE and R2 Values` Table comparing the MSE and R2 values for all of the models.

6.2 Strengths and Limitations

Our project approach is strong in that our dataset is clean and easily understandable. We also have a lot of observations to work with, which should theoretically allow for us to uncover trends. In addition, since we did not perform PCA or any transformations on the data, our results are highly interpretable.

The main limitation of our project approach is that our model predictions are not strong. The MSE values are still relatively high, and the R2 values indicate that our predictions are no better than predicting the mean `num_followers` for each observation. This, of course, is a significant drawback, as our goal is to be able to effectively predict the number of followers for each playlist. The dataset also did not have a normal distribution of popular and not popular playlists, with less than 1% of all one million playlists being popular. For this reason, our model likely lacked sufficient information at the outset to accurately predict the number of followers. Lastly, the predictors we used might not even influence a playlist's popularity, so we would need more information to build better models.

6.3 Areas for Future Exploration

One way in which we can improve our approach is to successfully query the Spotify API and add more interesting variables (such as song/playlist sentiment) to our data frame. The Spotify API is a powerful tool that allows us to access all the information that Spotify has. This includes audio features, audio analysis (more specific information about the track like its tempo and

when it starts), and more. By using the `Spotipy` module in python, we can provide it with our credentials to get access to even more predictors that can help us improve our model's performance and accuracy. For our project, the audio features we were looking at potentially related to playlist popularity were:

- `danceability`: Suitability of a track for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. 0.0 is least danceable, 1.0 is very danceable

- `energy`: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Energetic tracks feel fast and loud/noisy

- `valence`: A measure from 0.0 to 1.0 describing 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)

We did not end up using the Spotify API due to it taking too long to render (about 10,000 playlists / 20 hours) and we wanted to use as large of a sample size as possible. However, future projects that are not stuck up on time constraints should explore these features. With more and more playlists being themed, it is hard to ignore that the types of songs in a playlist can also affect popularity. Would playlists with mostly fast, happy, danceable songs be more popular than playlists with slow, sad, chill songs? The inclusion of these predictors may result in more meaningful predictions if these new variables help explain the variance in the outcome. We provide a brief demo of using the API to visualize audio feature data in our Jupyter Notebook.