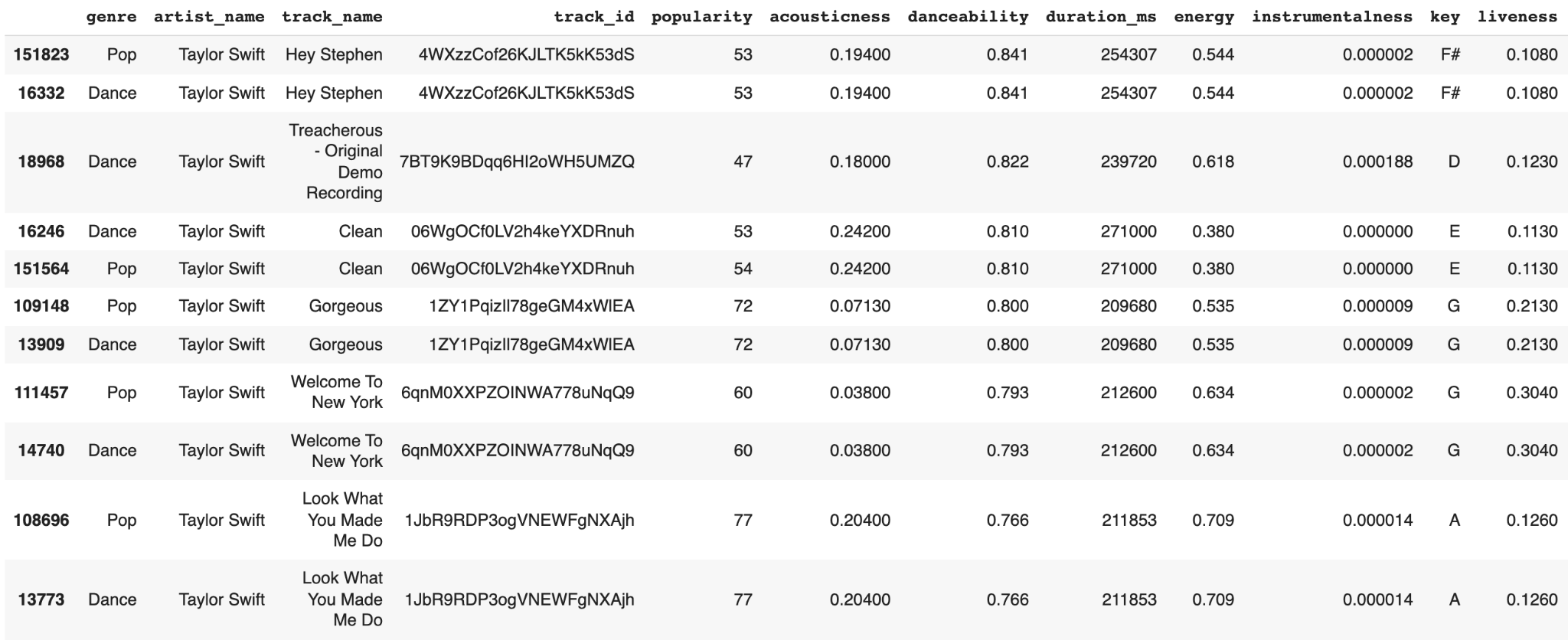
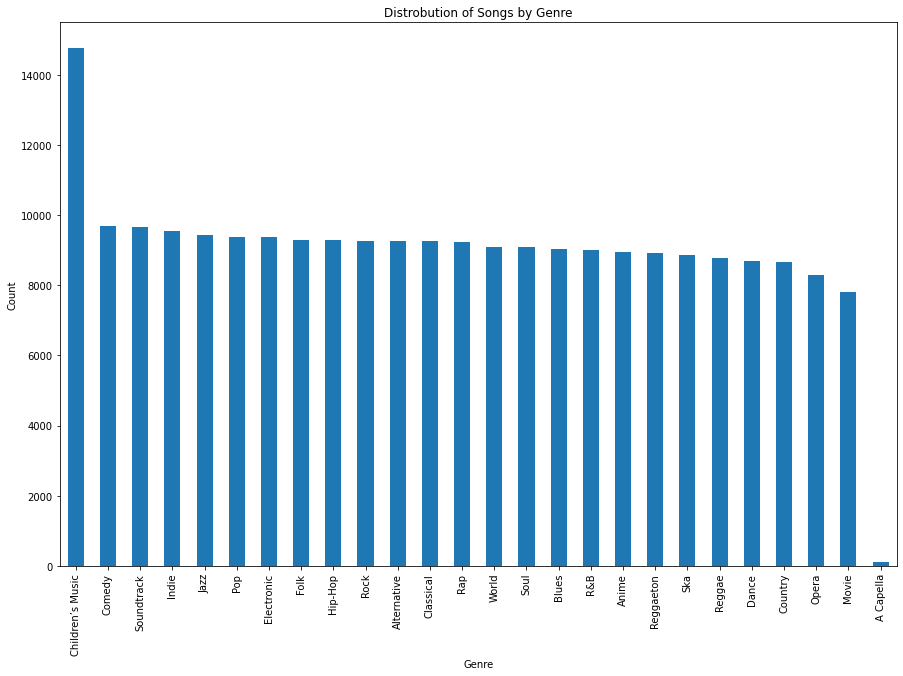
## Predicting Song Danceability

Beyond maintaining a database of songs, genres, tempos, and time signatures, Spotify also uses a plethora of calculated metrics to label their songs, the most interesting of which is danceability. Their [API](https://developer.spotify.com/documentation/web-api/reference/#/operations/get-several-audio-features) describes danceability as “how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity.” These metrics are often used by Spotify for song recommendations. It’s important to note that danceability, along with many other metrics like energy, acousticness, and instrumentalness, is based on an automated analysis of the track. Therefore, other features are likely to have some correlation with danceability, meaning predictions are likely to be pretty accurate.

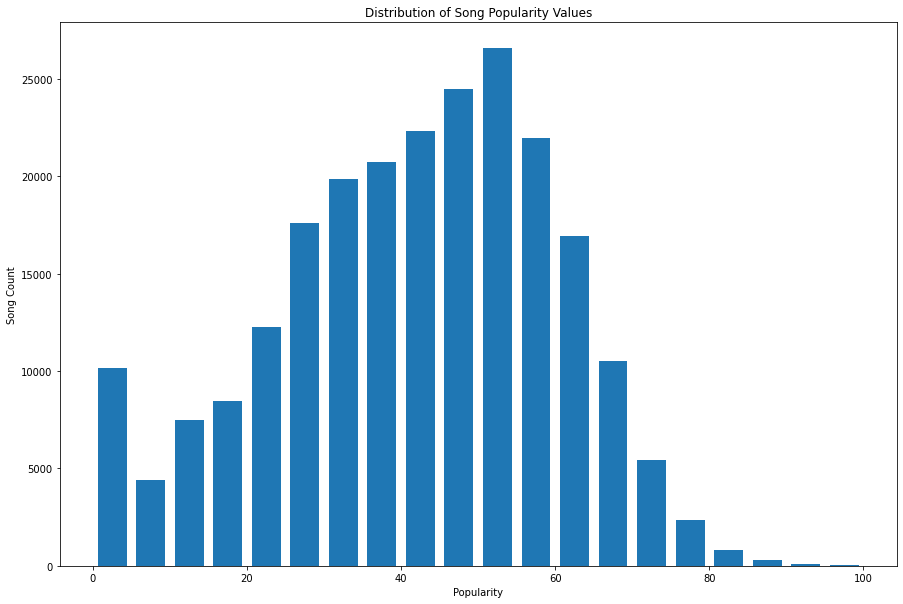
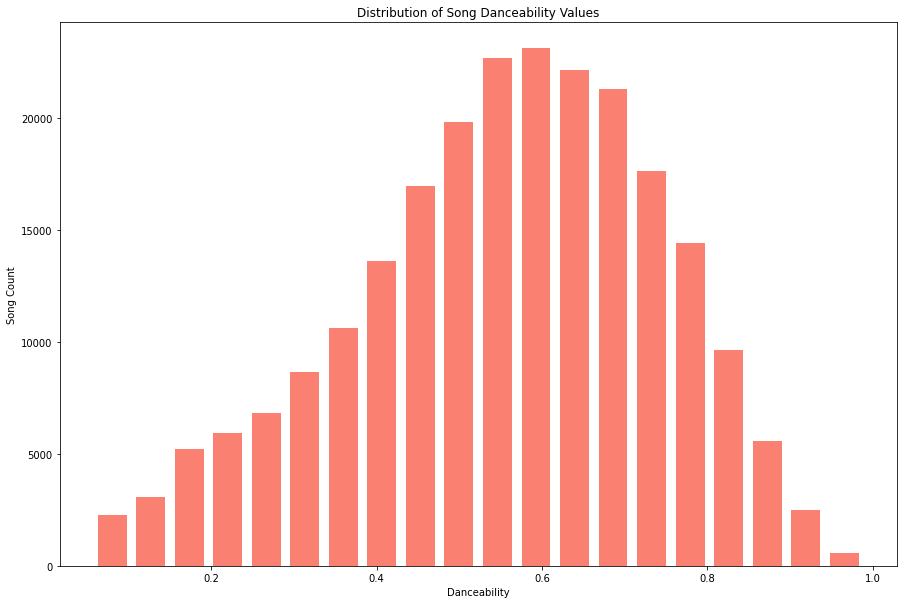
Our goal was to see if we can use the other metrics from a [Kaggle Spotify dataset](https://www.kaggle.com/zaheenhamidani/ultimate-spotify-tracks-db) to predict the danceability of songs. Below is a snapshot of the dataset, filtered for songs by Taylor Swift.



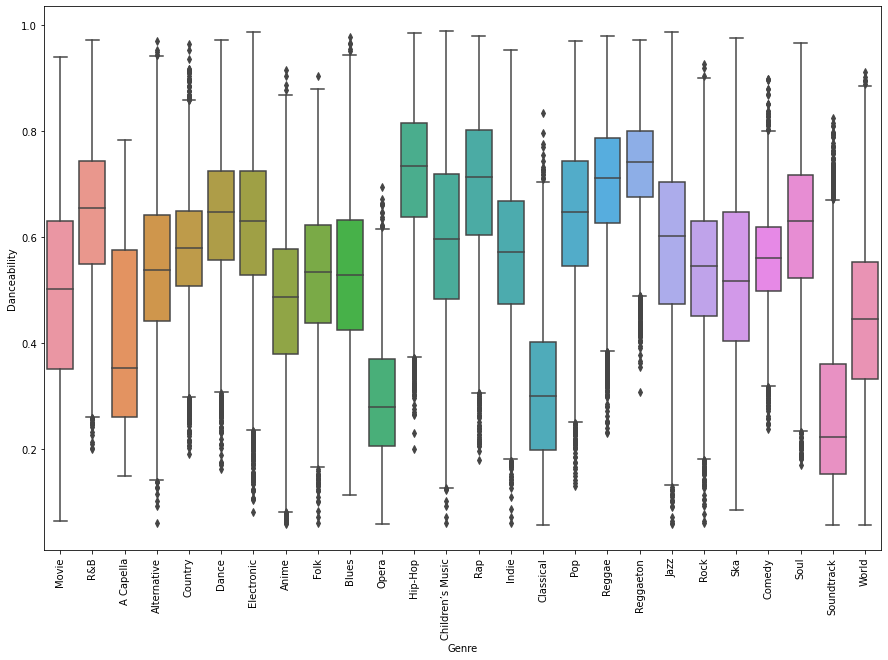
We began by first doing some basic EDA (exploratory data analysis), which helps us understand the structure and distribution of the data. It also gives us insight into what parts of the data needs to be cleaned. For instance, we plotted a bar graph of the genres in the dataset, and we noticed that there were two genres titled “Children’s Music” even though there should only be one. We made a note to fix that issue when we cleaned the data. Moreover, looking at the above snapshot, many songs occur twice, once for each genre. This was the second major issue with the dataset.



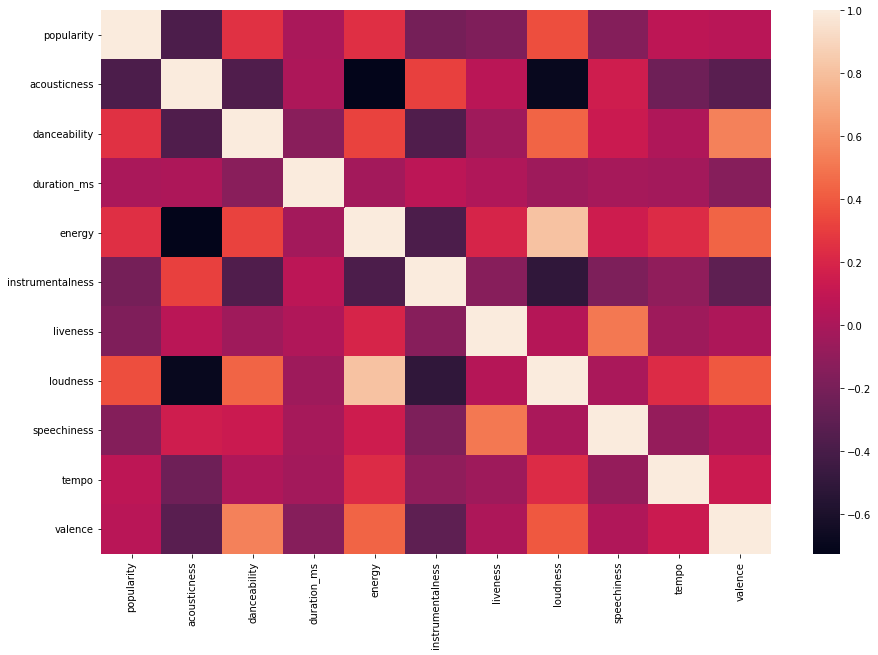
For our EDA, we also made some other plots to more clearly visualize the data. First, we plotted the distribution of danceability and popularity values. In both cases, the distribution was not normal. This was a very informative step; rather than predicting the range of the danceability (e.g. does *Shake It Off* have danceability in the range 0.8-1.0 or 0.6-0.8?) , it would make more sense to predict the quintile in which the song lies, in a range from 1 to 5.



Another plot was made to visualize the distribution of danceabilities for a certain genre. The most appropriate visualization was a box and whisker plot for each genre; this helped illuminate which genres had more danceable songs on average. As the plot below demonstrates, genres like Soundtrack and Classical are not very danceable while Reggaeton and Hip-Hop are very danceable on average, a preliminary finding that makes sense.



The last plot we made was to examine how correlated different features are. Since danceability was a calculated metric, we hypothesized that there should be some strong correlation between other metrics and danceability. Indeed, that was the case. We plotted a heat map to represent how strongly any two features are correlated; in this graph, very dark and very light colors are indicative of a strong correlation. We found danceability to be inversely correlated to acousticness and instrumentalness while it is correlated to valence (a measure of how positive a track sounds). Again, these findings make perfect sense; most songs we consider dance songs tend to use electronic rather than acoustic sounds, and with the exception of EDM, most dance songs feature vocals.



In preparation for building our models, we had to clean our data first. The two different Children’s Music genres turned out to differ because one employed a single quote while the other used an apostrophe. We simply relabeled them to a consistent label. The second problem was duplicate entries for the same track for different genres; this was slightly more difficult to solve. First, we had to one-hot encode the genre, time signature, key, and mode because they were categorical variables and not floating point numbers. Afterward, we grouped the dataset by the track\_id and aggregated each group, which effectively combined the different entries into a single entry. For example, Taylor Swift’s masterpiece, *Love Story*, is both Dance and Pop music, so it initially had two entries. After the cleaning, there was only one entry, which had a 1 in the one-hot columns for Dance and Pop.

In choosing the models that we wanted to use for predicting the dacenablity of the songs we compared the results of Logistic Regression, XGBoost, and Random Forest models. Logistic Regression provides a quick and straightforward way to get a sense of how a regression model will do on the dataset and help identify any issues early. XGBoost is an efficient implementation of a gradient boosted decision tree algorithm which gives some advantages like regularization, parallelization, and sparse awareness that give the model boosted performance compared to other boosting models. Finally, the Random Forest approach allows us to apply the bagging technique and give a random element. Between these three approaches we are hoping to find a successful model for predicting danacablity.

After training and predicting on the Logistic Regression, XGBoost, and Random Forest models we are able to predict the danceability of songs with a test accuracy of 99.9% using the XGBoostRegression model. After using GridSearchCV for hyperparameter tuning, we ran both models on the train and tested data which we created using a 20% test split. We chose to look at three metrics, train accuracy, test accuracy, and root mean square error in particular to see how well each model was learning, predicting, and how close the errors were to the actual result. The root mean square error (RMSE) is a good measure of the third criteria because the squaring of the error increases the effect it has on the score, or in other words the more inaccurate we are, the RMSE increases. Comparing the train and test accuracy is also a good check to make sure that they are similar values because otherwise the model may be overfitting to the training set. From the results in the table below, XGBoost is the clear best performer with the highest train and test accuracies and the lowest RMSE.

| Model | Train Accuracy | Test Accuracy | RMSE |
| --- | --- | --- | --- |
| Logistic Regression | 91.1% | 91.1% | .427 |
| XGBoost | 99.99% | 99.9% | .113 |
| Random Forest | 95.6% | 95.7% | .298 |

The biggest challenge we faced in our project was figuring out what data needed to be cleaned. Since the dataset came from Kaggle, it was already relatively clean; from our initial EDA, there seemed to be no null/NaN values in the dataset. Typically, handling null values and inconsistent data are the most time-consuming and common data cleaning tasks, but we encountered none. Moreover, in our typical homework assignments, the instructions inform us which parts of the dataset require cleaning. In this case, we had to examine the data closely on our own.

It was only after some more analysis after we had built the models that we encountered the flaws in the data. The plots obviously showed that there was a duplicate label for Children’s Music. But the issue with duplicate tracks was unknown to us at first. A cursory glance at the data by examining the first several rows did not reveal any duplicates, so we assumed that this problem was nonexistent. It was only after we filtered by a specific artist (you guessed it, Taylor Swift) that we realized there were many duplicates in the dataset. Because this issue was discovered later in our project timeline, we had to re-train the models with the cleaned data. Had we not discovered the duplicate values, the models would not have been correct.

Overall, we found the experience of working with and modeling data to be an engaging but challenging task. Splitting the project into separate stages (EDA, cleaning, modeling, etc.) turned out to be crucial to our success.