Spotify Song Popularity Prediction

Tiantong Li, Yong Zhang, Bailey Bai, Alison Yao
TF Contact: Håkon Grini

1 Introduction

Using the observed data to estimate the unseen data has been one of the key tasks in data science. This also applies to the music industry predicting the commercial value of a song or an album. Song popularity is a crucial metric that helps artists evaluate themselves and significantly impacts the recommendation system for Spotify users. Therefore, being able to accurately predict the song's popularity is an interesting and meaningful challenge for avid Spotify listeners such as ourselves. We are interested in how the audio features of a song could be used to estimate its potential popularity before getting released. With this motivation, we have chosen to sample the data from Spotify, one of the most popular music applications in the world.

Within the Spotify API, a song can be decomposed into multiple features that cover but are not limited to audio-related data. These audio features could be represented as numeric values or categorical values. Since a piece of music can be successfully parametrized, it is possible to make quantitative sense of Spotify tracks and employ predictions to forecast the popularity of each track. During our exploration of the Spotify API, we realized that song popularity is a more suitable response than the number of Spotify streams proposed in the project spreadsheet. Song popularity describes how popular a song is based on Spotify algorithms that consider the total number of plays. Song popularity captures the quantity information of a track. Therefore, our modified project question is how to predict the song popularity based on the aforementioned predictors.

2 Data Description

In this project, we aim to analyze the song popularity on Spotify based on the musical characteristics of the song such as *accounticsness* and *tempo*. The data mining process begins with getting credentials for Spotify API and using Python to retrieve information about the tracks. This allows us to form a client and have access to up-to-date data in Spotify.

As specified in the project proposal, we started with the songs that are classified as electronic music by Spotify and pulled 22,000 unique songs from 2000 to 2021. Specifically, we pulled 1,000 songs per year to form the whole dataset. Each song contains multiple features that might be useful to infer the song's popularity. In particular, we have selected the following 18 predictors during data mining: *artist_name, track_name, track_id, type_release, collaboration, danceability, valence, energy, tempo, key, mode, loudness, time_signature, speechiness, duration_ms, acousticness, instrumentalness, release_date*. Note that we are not using all the collected features when training the model. A brief overview of the whole dataset is illustrated in Section 2 EDA part of our jupyter notebook.

3 Visualizations & EDA

Before diving into specific visualizations and data relationships, we start with an overview of the dataset. According to the info table, the dataset consists of 22,000 observations, 5 categorical features (*artist_name, track_name, track_id, type_release, collaboration*), and 14 numerical features. Noticeably, the dataset does not contain any null values. In addition, summary statistics show that the average popularity of the 22,000 songs is only 27.65 out of 100, which is lower than we previously thought. Also, features such as duration_ms and speechiness seem to have a skewed distribution, which we fully explore in the following section.

3.1 Univariate Exploration

In this section, we explore each predictor and response variable separately to understand their properties better. We first study the distributions of 12 numerical features in the dataset. According to Figure 1, *speechiness*, *duration*_ms, and *acousticness* have an extremely right-skewed distribution, indicating that the majority of the 22,000 songs are electronic music, and they are relatively little spoken and short in duration. In *addition*, *danceability*, *tempo*, and *loudness* are approximately normally distributed. Last, we notice the distribution of *time_signature* is highly imbalanced. As *time_signature* specifies how many beats are in each bar, it is reasonable that most songs in the dataset have time signature “4/4”.

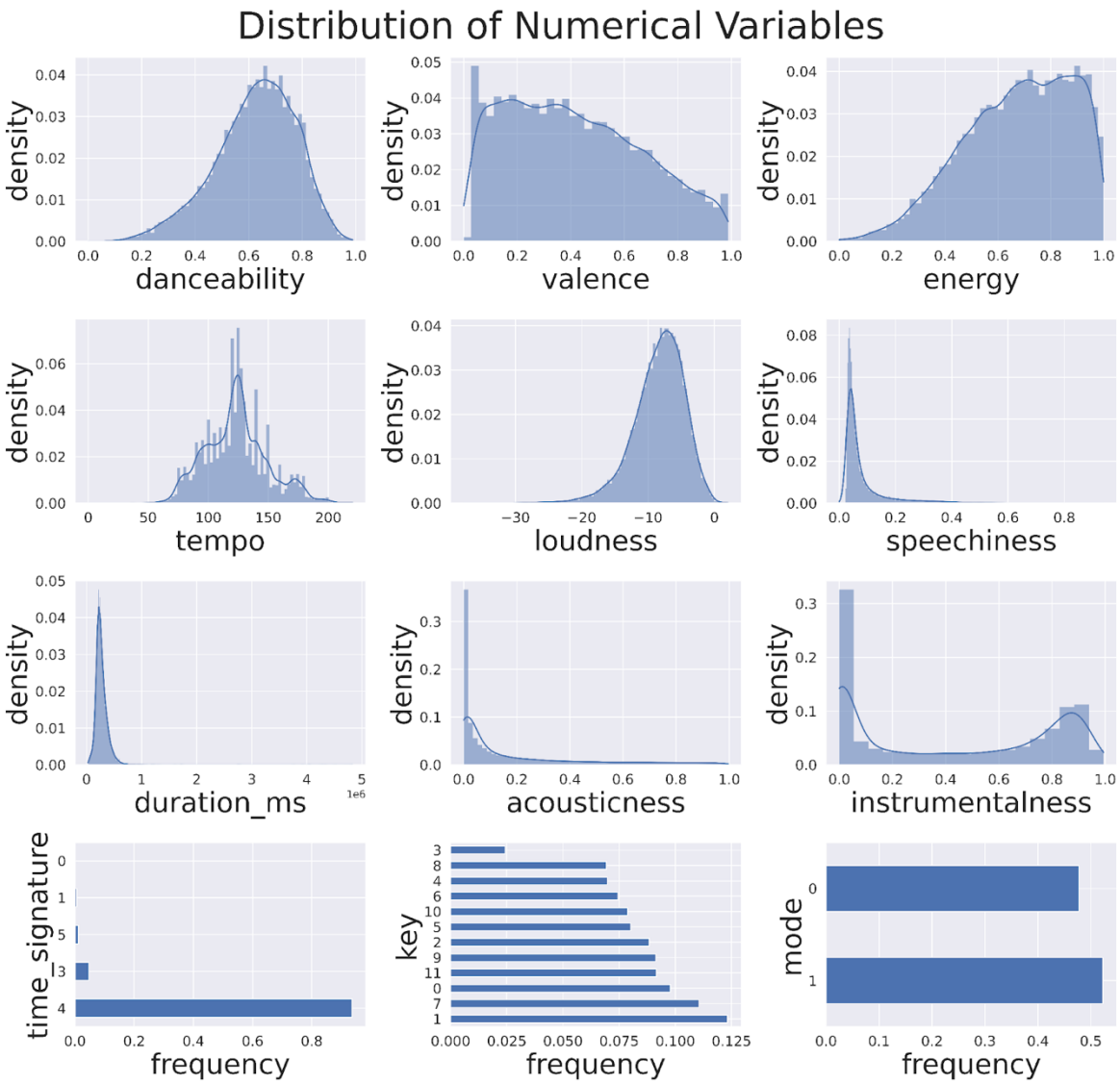


Figure 1: Distribution of numerical variables

Finally, we plot the distribution of the response variable, *popularity* in Figure 2, which is approximately normally distributed with a right-skewness.

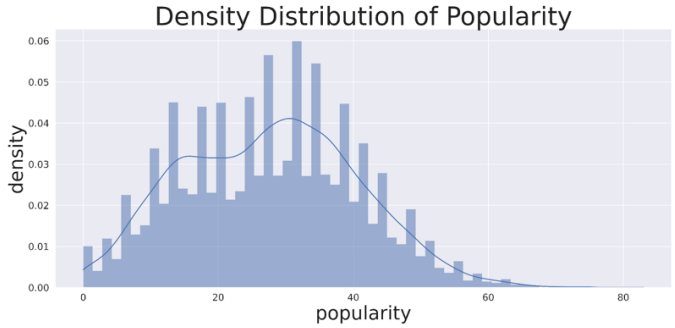


Figure 2: Density distribution of popularity

3.2 Categorical Feature Encoding

In addition to the features studied above, two categorical features, *type_release,* and *collaboration*, can be encoded into discrete values for future analysis. Since *collaboration* takes two values, Y or N, we encode it with the binary number 1 or 0. For *type*_release, as it takes multiple values (“single”, “album”, and “compilation”), we encode it using one-hot encoding.

3.3 Bivariate Exploration

In this section, we study the relationship between every two variables. An intuitive question to ask is whether a song's popularity is credited to its producer. Since producers who produce more are likely to have more popular songs, we investigate whether the song's popularity is subject to the annual production of its

producers. According to the scatterplot in Figure 3, we see that the expected popularity of a song is likely to be higher when a producer produces around 4 songs annually. However, we do not see a clear trend between the song popularity and the producer's annual production.

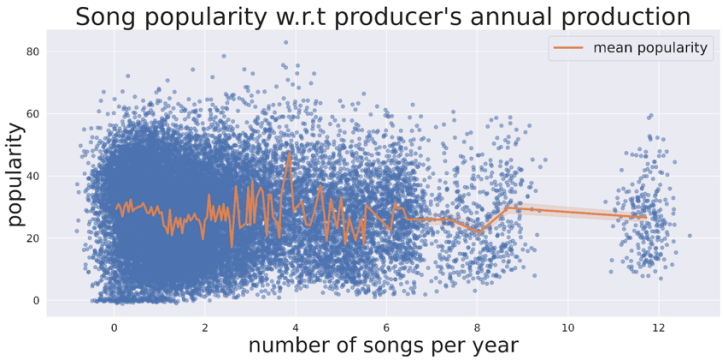
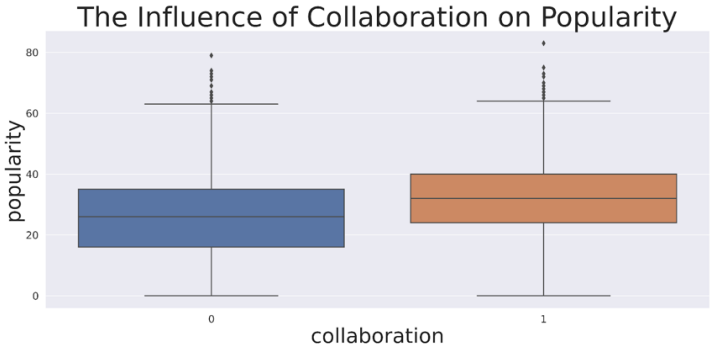
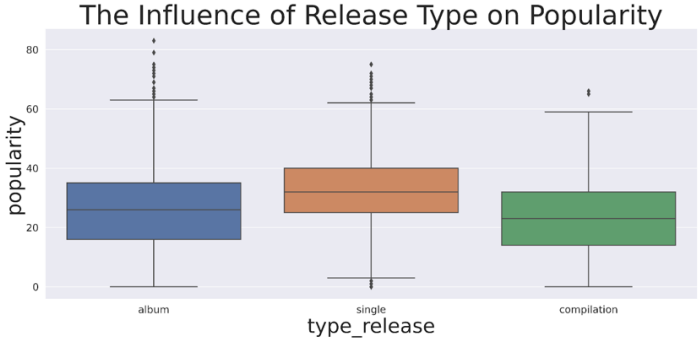


Figure 3: Song popularity with respect to producer's annual production

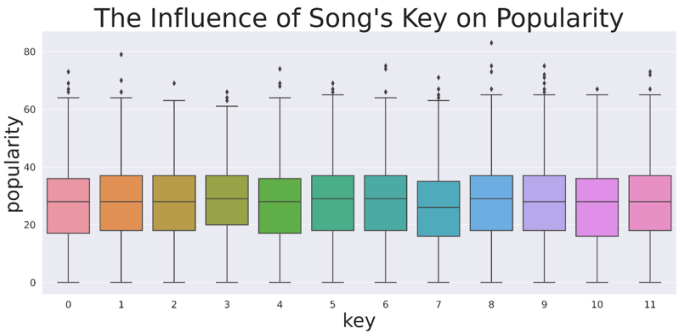
Next, we study how the categorical features affect the song's popularity. It is worth noticing that songs with collaboration have a higher interquartile range (IQR) popularity than the songs without collaboration; meanwhile, single songs have a higher IQR popularity than the other two types. In addition, songs in minor keys have a higher average than songs in major keys. The related plots are shown in Figure 4.



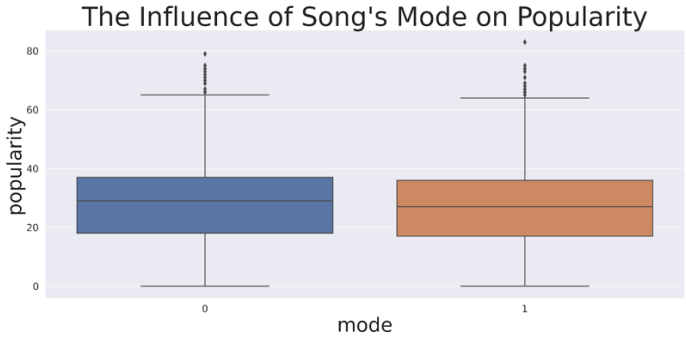
(a)



(b)



(c)



(d)

Figure 4: The influence of categorical features on popularity; (a) shows the influence of collaboration on popularity; (b) shows the influence of release types on popularity; (c) shows the influence of song's key on popularity; (d) the influence of song's mode on popularity

Finally, we investigate how the numerical features influence popularity. Given that the dataset includes songs from 22 years, we study how popularity differs by years as shown in Figure 5. The plot below shows that songs in more recent years likely have higher average popularity, indicating the year of publication is likely an important feature in predicting popularity.

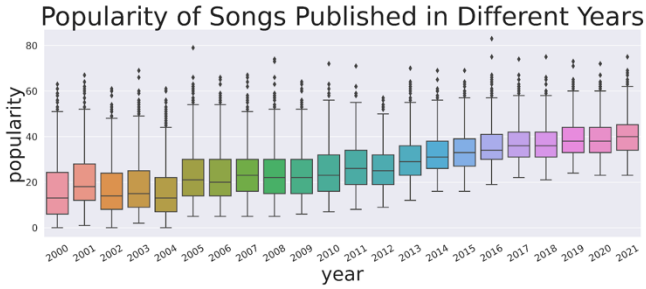


Figure 5: Popularity of songs published in different years

3.4 Multicollinearity

In addition to studying how each predictor affects the response variable, it is also important to understand how predictors relate to each other. Here we study the

correlation between each pair of predictors shown in Figure 6. According to the heatmap, several pairs of predictors are highly correlated. For instance, *loudness* is highly positively correlated with *energy* (corr=0.74). Meanwhile *acousticness* and *energy* are highly negatively correlated (corr=-0.54). This reminds us to be aware of the potential confounding bias when including all the correlated features into our predictive models.

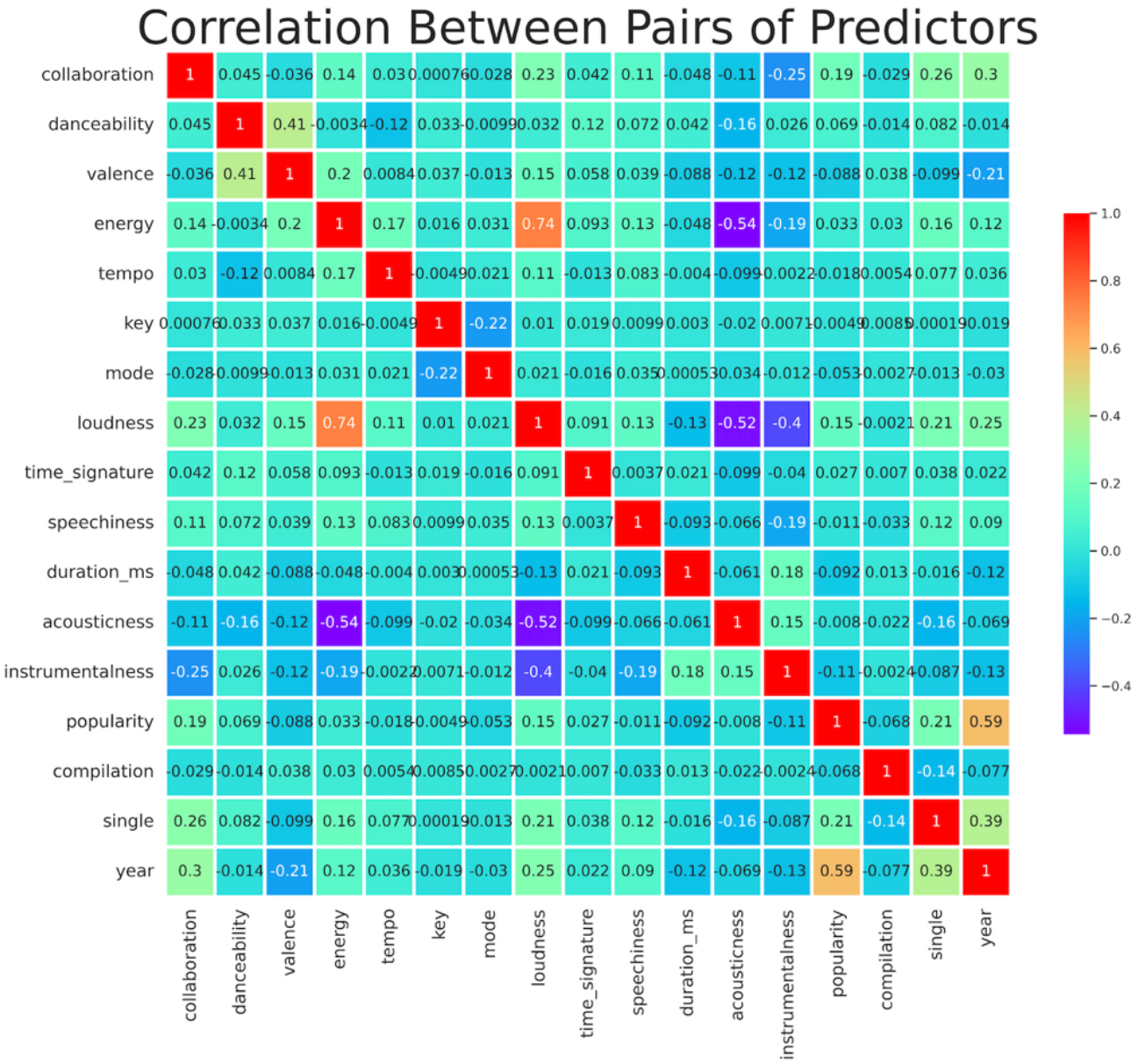


Figure 6: Correlation between pairs of predictors

4 Modeling

Before fitting any machine learning models, we preprocess the categorical predictors and continuous variables in different ways. As mentioned in section 3.2, categorical features are transformed using one-hot encoding. The continuous variables are standardized before being fed into the following models.

4.1 Baseline - Linear Regression

Since the response is a numerical value, we formulate our prediction problem as a regression problem. The baseline model we chose is vanilla linear regression. Using the *LinearRegression* model from *Sklearn*, we fit the linear regression model on 80% of the original dataset as training data and the rest as test data with random_state equals 109. The predictors are *collaboration, danceability, valence, energy, tempo, key, mode, loudness, time_signature, speechiness, duration_ms, acousticness, instrumentalness, compilation*, and *single*. The model has a training MSE of 148.3425 and an R-squared of 0.1174. For testing, the baseline model has a test MSE of 156.4662 and a test R-squared of 0.0841. We can see that there is plenty of room for improvement in both training and testing.

4.2 Ridge/Lasso Regression

A natural improvement of the vanilla linear regression model is to add regularization, so we tried Ridge and Lasso regressions. However, adding regularization does not improve MSE or R-squared in our case, so we turned to a possible solution of adding more relevant features. Recall in section 3 we found that year seems to be an influential factor of popularity, we include year as another feature in our regression models.

After adding the *year* predictor, the training MSEs of both Ridge and Lasso regressions drop to around 107, and the training R-squared increase to 0.3640. The test MSEs increase to around 111 and the test R-squared increase to around 0.353. This makes sense if we look at the coefficients of both regression models in Figure 7.

Baseline		best_ridge		best_lasso	
intercept	26.2785	intercept	28.1455	intercept	28.1035
collaboration	3.4154	collaboration	0.4448	collaboration	0.3881
danceability	1.5685	danceability	1.1965	danceability	1.1762
valence	-1.7073	valence	-0.0302	valence	-0.0103
energy	-0.9576	energy	-0.5182	energy	-0.4996
tempo	-0.1362	tempo	-0.1984	tempo	-0.1956
key	-0.1462	key	0.0316	key	0.0244
mode	-1.1978	mode	-0.6653	mode	-0.6307
loudness	2.5342	loudness	0.6226	loudness	0.5876
time_signature	0.0928	time_signature	0.0915	time_signature	0.0824
speechiness	-0.8567	speechiness	-0.9720	speechiness	-0.9600
duration_ms	-0.8208	duration_ms	-0.2256	duration_ms	-0.2189
acousticness	1.1062	acousticness	0.5653	acousticness	0.5519
instrumentalness	-0.3955	instrumentalness	-0.4489	instrumentalness	-0.4459
compilation	-2.5433	compilation	-1.7009	compilation	-1.4640
single	4.2088	single	-0.6495	single	-0.5525
		year	7.5925	year	7.5871

Figure 7: Coefficients of features in baseline/ridge/lasso regression

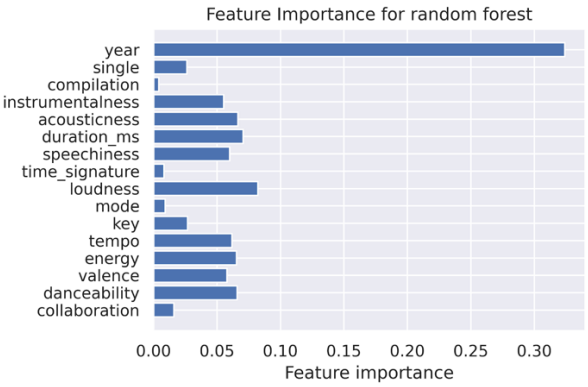
Although none of the coefficients is zero, their absolute values do decrease when we add the year to the list of predictors. In comparison, the coefficient of the year predictor is very large, with a magnitude of around 7.59. This means that one year increase is associated with around 7.59 increase in popularity, holding all other predictors constant. Because of the significant performance boost, we will keep *year* in the following models as well.

4.3 Generalized Linear Model

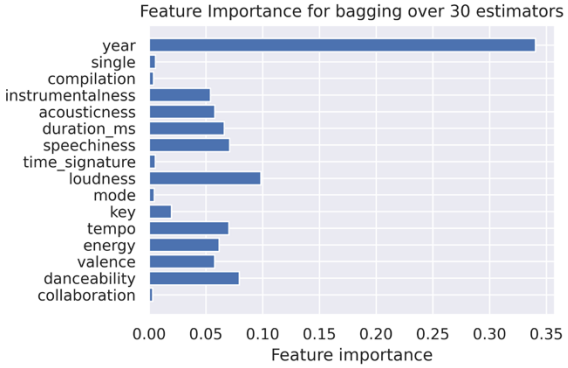
During our EDA process in section 3, we noticed that a right-skewness presents in the *popularity* distribution. To continue to work within the framework of linear regression, we decide to use a generalized linear model (GLM), which directly relaxes the normality assumption on the data. We choose a GLM with a Poisson distribution, which is a regression model that assumes the response variable follows a Poisson distribution. The Poisson regressor here uses the 'log' link function. The GLM has similar performance on training and test sets compared to the regularized regression models. This indicates that the normality assumption on our data with slight right-skewness is not necessarily violated so the linear model cannot predict the results well. By adding regularization terms, we can achieve the same result as using GLM.

4.4 Bagging/Random Forests

Instead of using the regression models, we have also tried trees such as the decision tree and its variations. We start by trying different tree depths to see the bias and variance trade-off. In particular, we found that for a single decision tree, the depth that yields the minimum test MSE is around 7. When we continue to increase the tree depth, we observe the trend of overfitting as the test MSE starts to increase. Based on the results for the single decision tree, we explore bagging and random forest to see whether we can reduce the test MSE by cutting the variance. However, as shown in the results in section 5, from the mean squared error, the performance of a single decision tree is better than all other competitors of the tree family. The R square in general is worse than the baseline. We deduce that if we average all the weak or inaccurate models might not improve the overall performance. This has been confirmed by the test MSE of bagging and random forest. Further, when selecting a feature for the root node, we constantly find that year would be the feature that gives the best quality of split as shown in Figure 8. Since it delivers the highest importance magnitude among all the features, it does not make much difference if we aggregate the trees but with the same features. In general, the tree family performs worse than the regression models for this dataset.



(a)



(b)

Figure 8: Feature importance of bagging/random forest. (a) shows the feature importance for random forest; (b) shows the feature importance for bagging

4.5 Boosting

We experimented with two boosting models, gradient boosting and XGBoost. For each of the models, we used RandomizedSearchCV to find the approximately best combination of its hyperparameters. As the result, gradient boosting with "*n_estimators*"=300, "*max_depth*"=3, "*learning_rate*"=0.05, and XGBoost with "*n_estimators*"=100, *"max_depth"*=4, "*learning_rate*"=0.05, "*reg_alpha*=0.9" gave comparable results, and their R squared on the test data is around 0.4. Noticing that boosting models perform much better than bagging/random forests, we infer that the bias of the model dominates its expected error on the test data. Therefore, boosting methods achieved a higher test accuracy by decreasing the bias iteratively.

4.6 Stacking

Ensemble methods are a type of machine learning algorithms that combine the predictions of multiple individual models to produce a more accurate and stable prediction. Stacking is a type of ensemble methods that combine the predictions of multiple models using a meta-learning algorithm. Here we chose 5 first-stage models: *LinearRegression*, *LassoCV*, *RidgeCV*, *PoissonRegressor*, and *GradientBoostingRegressor* from previous sections. In a stacking ensemble, the individual models are trained on the same dataset and make predictions on new data. These predictions are then combined using a meta-learning algorithm to produce the final ensemble prediction. We simply choose a LinearRegression as the meta-learning model to make final predictions.

From the result table in section 5, the stacking model has the lowest training MSE and explains the variance in the response variable the most. At the same time, it also performs the best on the test set. This is because it ensembles previous models for low variance. However, we also observed that the stacking model is only slightly better than Gradient Boosting. Because Gradient Boosting has better performances than linear regression models, it explains most of the variance, and ensembling other linear regression does not help boost the performance significantly.

5 Results

Compared to the baseline model, all models except single decision trees and bagging trees make improvements due to the addition of the *year* predictor. However, the degree of improvement varies. Table 1 summarizes the training and test outcomes of all the models presented above.

	Train MSE	**Train R^2**	**Test MSE**	**Test R^2**
Baseline	148.3425	0.1174	156.4662	0.0841
Ridge Regression	106.8963	0.3640	110.5702	0.3528
Lasso Regression	106.9021	0.3640	110.5771	0.3527
Poisson Regression	106.8377	0.3321	110.2366	0.3223
Single Decision Tree	136.6184	0.0821	141.5188	0.0461
Bagging	136.4756	0.1698	148.9134	0.0506
Random Forest	135.4114	0.1888	150.4384	0.0400
XGBoost	95.2443	0.4333	103.0746	0.3967
Gradient Boosting	93.7936	0.4420	102.7697	0.3984
Stacking	85.7978	0.4895	102.4279	0.4004

Table 1: Train and test statistics using 80% of the observations from 2000-2021 as the training set and the rest 20% as the test set. The data is shuffled before splitting.

The linear regression family has similar performance and improves upon the baseline model by 0.20+ in train R-squared and 0.24+ in test R-squared. However, fine-tuning the distribution of the response variable in GLM seems to be unnecessary since the Poisson regression is performing slightly worse than regularized linear regressions with the assumption of normal distribution. In comparison, the single-decision tree, bagging trees and random forest have relatively disappointing results. The reason is that single trees do not perform well and ensembling learning of parallelization of weak learners does not help as much. Boosting trees, on the other hand, are performing much better than bagging trees. Gradient boosting and XGBoost both achieve R-squared of 0.43+ and 0.39+ in training and testing. Sequential learning seems to be more suitable for our case. Finally, we use stacking to aggregate the outcomes of the linear regressor, Lasso CV, Ridge CV, Poisson regressor, and Gradient Boosting regressor and increase the test and train R-squared once more to ~0.49 and 0.4+.

	Train MSE	**Train R^2**	**Test MSE**	**Test R^2**
Linear Regression	148.342	0.117	266.010	-1.374
Ridge Regression	106.896	0.364	114.302	-0.020
Lasso Regression	106.902	0.364	114.405	-0.021
Poisson Regression	106.837	0.332	119.735	-0.045
Single Decision Tree	136.618	0.082	131.162	0.042
Bagging	136.475	0.169	162.823	0.031
Random Forest	135.411	0.188	165.897	0.028
Gradient Boosting	93.793	0.442	116.983	-0.044
XGBoost	95.244	0.433	119.137	-0.063
Stacking	85.797	0.489	112.867	-0.007

Table 2: Train statistics of our models from section 4 and test statistics on out-of-distribution data from 2022.

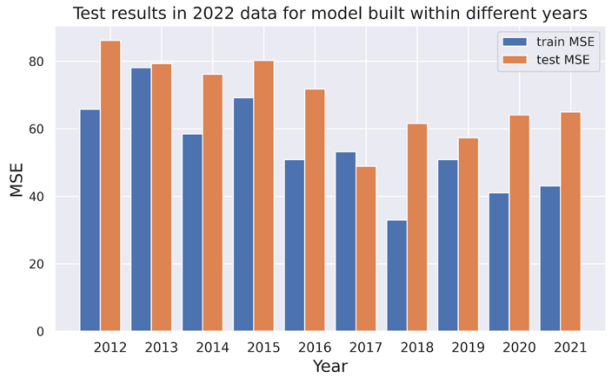
6 Extension: Forecast the Song Popularity in 2022

In this section, we would like to evaluate the forecast capability of each model we trained in section 4. To do so, we pulled data of songs released in 2022 from Spotify API, preprocessed the data in the same manner as before, and tested the models using the data.

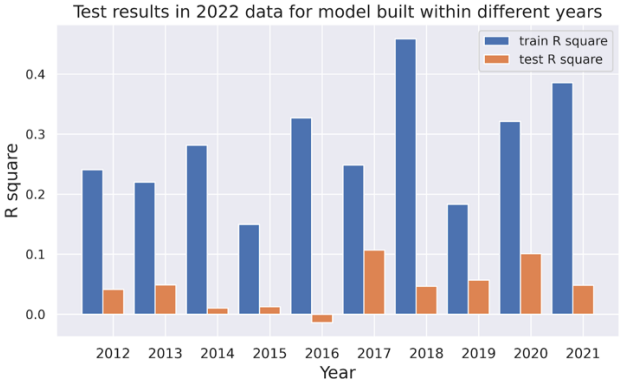
In the Table 2, we show the test accuracies of each model we trained in section 4 when predicting the data in 2022. For each model, the test R-squared is relatively low compared to the train R-squared, showing that the models likely overfit. Specifically, the poor forecasting capability of the models is likely caused by two aspects. First, as we see in the EDA section that each year the song popularity has a different distribution, it is likely that the popularity distribution in 2022 is rather different from the popularity from 2000 to 2021. Second, the data in 2022 behaves as an extrapolation on our origin dataset, and it is likely that our models cannot generate sound predictions for the extrapolated data. This motivates us to build time series models that could have better forecasting capabilities in the future.

Noticing the song popularity in 2022 is likely to follow a different distribution compared to the previous years, we would like to investigate in which single year between 2012 and 2021 the song popularity has a closer distribution to the popularity in 2022. To do this, we train a stacking model, in the same manner in section 4.6, to aggregate the outcomes of a linear regressor, Lasso CV, Ridge CV, Poisson regressor, and Gradient Boosting regressor. However, currently we train the stacking model on a single year between 2012 and 2021 and test its accuracies in predicting its popularity in 2022. Below, the figure 9 shows the accuracies of the stacking model trained each year.

According to the table, the stacking model trained on the data in 2017/2020 achieves the test R-squared around 0.10, which is relatively higher than the test R-squared of other models. This result suggests the song's popularity in 2017/2020 likely has a closer distribution to the song's popularity in 2022. Also, we notice the test accuracies of the stacking models trained on data in a single year are generally higher than the models trained on multiple years. This further suggests that the song's popularity each year shows a grouping pattern.



(a)



(b)

Figure 9: The test results of 2022 data for stacking models trained on each year in 2012-2021. The evaluation metrics are mean squared error and R-squared for figure (a) and (b), respectively.

7 Future Work

Year agnostic model

From our observations in section 6, the popularity of each year has a slightly different distribution with a different mean. Acknowledging the trend of *popularity* in *year*, we might want to incorporate time series components in our model to achieve more accurate predictions for different years.

Incorporating other genres

Due to time constraints, we only investigated our training data of electronic music among all 6,006 genres on Spotify. In the future, we want to build models that are predictive for other top genres, like Hip Hop, Rock, Country, C-Pop, and so on, as well. For future work in this direction, we could try the transfer of learning by testing our existing models on a different dataset. Additionally, we can also explore how to build a genre-agnostic model that can give an accurate popularity prediction across genres.

Adding more features

As a platform with more than 400 million users, we believe that there is definitely a further exploration of the API. We are currently only utilizing 18 predictors and there are more features we can incorporate from the API. If computational resources are etiquette, we want to use natural language processing techniques on the song track name as well as categorize the artist. Meanwhile, instead of focusing on the audio-specific features, we could also extend the feature space to artistical characteristics such as the creativity of the music in its generation.

References

[1] L. Colley et al., "Elucidation of the Relationship Between a Song's Spotify Descriptive Metrics and its Popularity on Various Platforms," *2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC)*, 2022, pp. 241-249, doi: 10.1109/COMPSAC54236.2022.00042.

[2] M. Tingle, "Getting Started with Spotify's API & Spotipy," *medium.com*, Oct. 3, 2019. [Online]. Available: https://medium.com/@maxtingle/getting-started-with-spotifys-api-spotipy-197c3dc6353b.

[3] R. Sangani, "Dealing With Features That Have High Cardinality," *towardsdatascience.com*, Aug. 4, 2021. [Online]. Available: https://towardsdatascience.com/dealing-with-features-that-have-high-cardinality-1c9212d7ff1b.