Report

—

Team 7
Open IIT Data Analytics

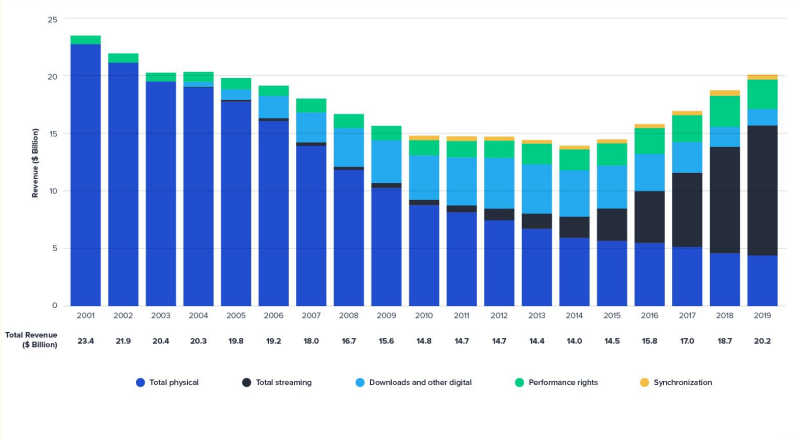
Index

INTRODUCTION

Music is one of the most popular types of online source of entertainment and the importance of the music industry can be expressed in its total revenues. For 2020 they were US$20.1 billion, the industry shows a very promising growth curve too. While the major players in this revenue growth are streams, official rights and physical performances, over the past decade the revenue growth via streams has shown positive growth percentages. Technological advancements have seen the rise of streaming services. Streaming services such as Spotify, YouTube Music etc. have a huge impact on the revenue of the music industry.

This growth also shows why major streaming service providers have moved their attention to machine learning and deep learning deployments so that the revenue can be maximized. Spotify acquired 'music intelligence' company 'The Echo Nest' for €49.7m to develop recommendation system, YouTube is reliant on it's deep learning based recommendation system and viewers impression for the same. While one thing to note is the similarity between the features they use and the features we have.

Next to the recommendation value, there is the importance of predicting the popularity of music. With the huge development in the supervised learning algorithms over the time, music industry has emphasised on the importance of popularity prediction as a major challenge. Major producers and independent artists, all are interested in connecting consumers with content they will love and buy. But the question raised is what makes a song popular and what makes it a failure. This has gained limelight since 2017 with Spotify releasing it's dataset for Kaggle. Over the past 3 years it has lead to understanding the importance of various features that gives information about the popularity of music.



Source: International Federation of the Phonographic Industry (IFPI)

The problem statement states to maximize the revenue of the predicted popularity of songs. The dataset provided consists of 12227 data points with 17 columns. The 16 independent variables are useful in the prediction of the dependent target variable, popularity. Based on the prediction of the music, a bid is made along with a specified expected revenue from the same. If the predicted value is lower than the actual popularity then there is no revenue gained on that music. However, if the prediction is higher or equal to the actual popularity of the song then the auction is won and expected revenue is gained. The main focus is to maximize this revenue collected in total from all the songs.

DATA DESCRIPTION

The music dataset appears to be extracted from Spotify. Open Source tools like Spotify library and Spotify's web API are used to scrap audio instances at random with the features needed. The train dataset has 12227 samples. The feature space could be technically interpreted as categorical and numerical variables but a more domain specific feature interpretation is quite helpful. The dataset has musical features like acousticness, energy, danceability, liveness, tempo etc. It has metadata features like release date, year and duration. It also has audio features like key, note, loudness etc.

2.1 Data Visualization

Fig1: Percentage Popularties Over the Months

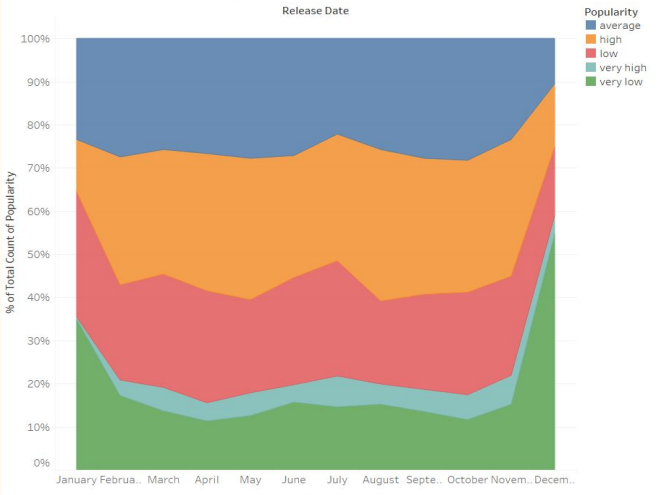
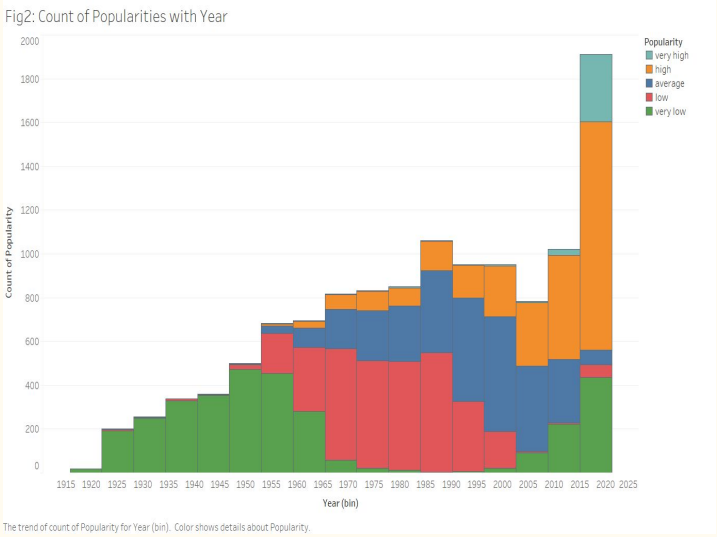
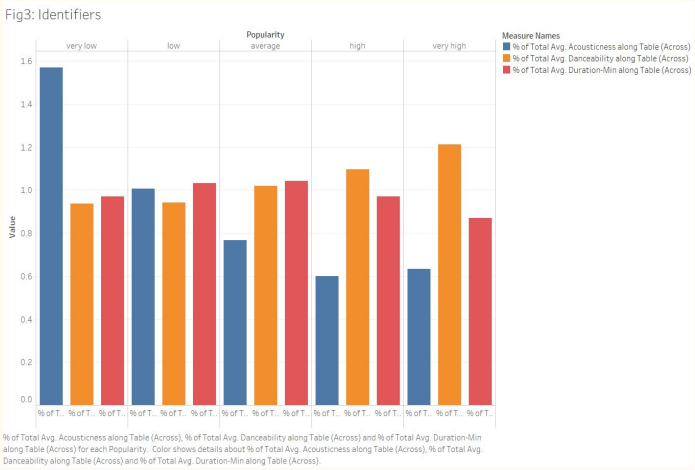


Fig. 1 explains the relationship of popularity with the month it was released in. The plot gives us two important inferences firstly the hike in the percentage of very low popularity in the months on January and December and secondly smaller area of the very high popularity indicator suggesting imbalance in the dataset.

Fig2: Count of Popularities with Year

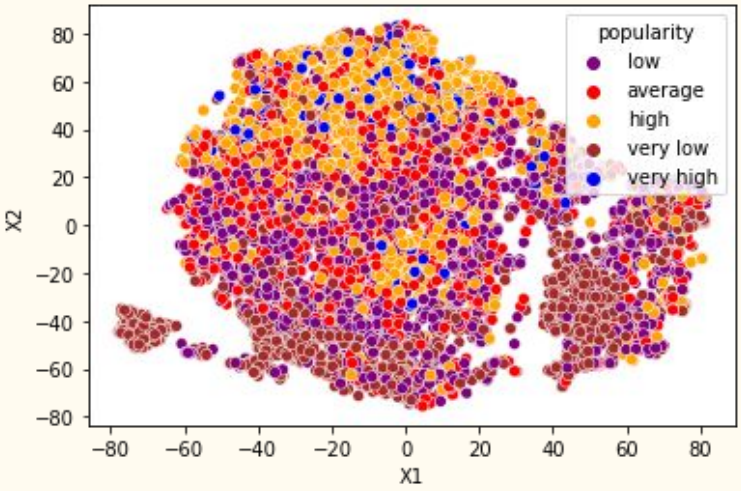


In Fig. 2, year of release has a major correlation with popularity. The production of the songs in the dataset linearly increases with year ranging from as old as 1920 to as recent as 2021. Evidently, It can be observed from the obtained histogram that songs were lesser popular when produced earlier with normal distribution and as time increases the popularity eventually grows to be exponentially increasing.

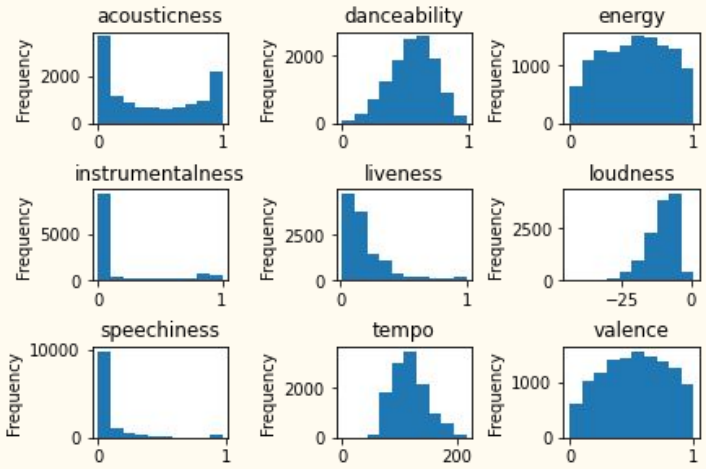


Looking at Fig. 3 we can clearly eliminates duration and danceability but it provides evidence for exigent correlation of accousticness with popularity. While most of the data here gives a visual idea about the dataset, the idea that some features together under some heuristic or functional relationship might be able to explain the problem much better. We'll discuss this and it's consequences in the feature engineering section.

2.2 Data Interpretation



We try to understand the data in higher dimensions using t-Distributed Stochastic Neighbor Embedding (**t-SNE**) method. t-SNE is a more probabilistic method than PCA a unsupervised matrix method for explaining variance. Here X1 and X2 are the dimensions of the embedded space based on the original feature space. We can identify clusters for high, very low and low with certain distinction. This also provides us an intuition to use unsupervised techniques for developing new features.

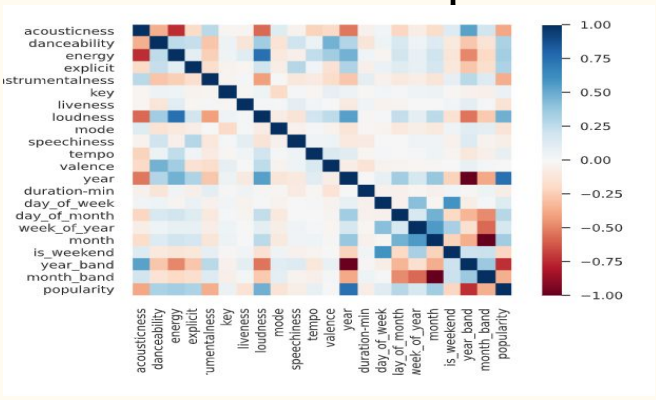


The collaged frequency distribution plot below shows that most of the numeric features aren't normally distributed. . Energy and danceability are however distributed normally.

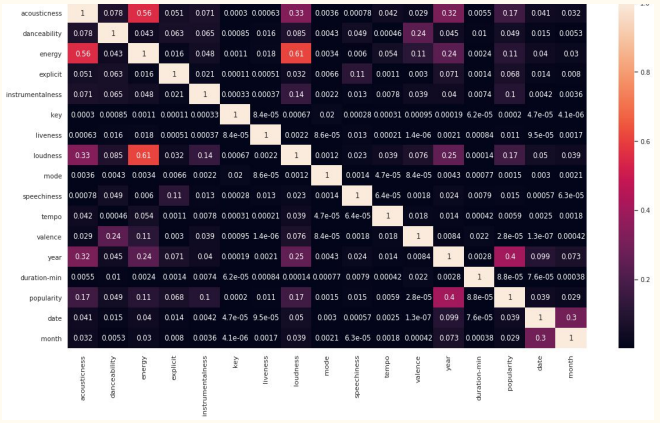
The duration of a song is normally distributed in the dataset excluding the outliers. It is distinctly observed that the high popularity of music is much concise in the interval near 3-4 mins duration.

EXPLORATORY ANALYSIS

3.1 Correlation Inspection



The correlation table provides Pearson's R coefficient. Uni modulus value of Pearson's R coefficient signifies more relation between the two variables. A positive correlation refers to the direct proportionality of the variables whereas a negative correlation refers to the inverse proportionality between the same.



The correlation table helped establish a measure dependency of all the variables. Rather than just looking over the whole of the correlation table, we focused our attention more on the dependency of popularity over other variables. While analysing the R2 coefficient values in the table, we applied 0.04 thresholding to concentrate more on the more correlated variables beneficial for the prediction of popularity, the target variable.

It is evident that there exists a very high positive correlation between the energy and loudness of a song. Variables like date and month, loudness and year, energy and year, danceability and valence also possess a high positive correlation with one another. In specific to popularity variable features like year, loudness, explicit, danceability, energy, date, month possess a positive correlation greater than the threshold. Other features like instrumentalness and acousticness provide a negative correlation to the dependent variable, popularity. Rather than taking all variables as input, selecting an efficient subset from the following provide a better edge on model training.

3.2 Feature Description

The release date information is vital for the prediction of popularity. The majority of the songs are produced in the month of January which consists of about 36.277% of the distribution. While the high popularity songs are uniformly distributed, the lesser ones are centric to the month of January. A much similar trend is seen in date distribution, with the first day of the month being 37.997% probable for producers to release their songs. Other information like date, day, month, week is extracted from the same.

Year of release has a major correlation with popularity. The production of the songs in the dataset linearly increases with year ranging from as old as 1920 to as recent as 2021. Evidently, It can be observed from the obtained visual plots that songs were lesser popular when produced earlier with normal distribution and as time increases the popularity eventually grows to be exponentially increasing.

Loudness is the measure of how loud or sharp the song is, the standard measuring unit is of dB. The distribution of the variable over the dataset is left-skewed. The mean and median of the distribution as -8.49 and -9.82 respectively. A similar shape of the distribution is to be seen within each popularity category, with the median shifting right (increasing) with the popularity of the song.

Acoustic music is one that primarily uses instruments that produce sound through acoustic means, as opposed to electric or electronic means. It has a non-linear distribution with songs majorly ranging at extreme ends forming a concave-shaped curve. It has a high negative correlation with the year. Acoustic music can be much related to songs that are soft, negatively correlated with loudness.

Instrumentalness, similar to acousticness, it varies on the scale of 0-1, representing the primary use of instruments in the song. On visualizing the dataset it is observed that 28.89% of the dataset have 0 instrumentalness. However, the songs with an instrumentalness value greater than 0.5, are mostly categorized with lower popularity. With the increase in popularity, there is a centric behaviour of the instrumentalness to confine with a value approximate to 0.

Explicit is a categorical feature having classes of No and Yes, which are label encoded as 1-2 respectively. By applying the SMOTE method it generated explicit values for the minority category (very high popularity) which varied between 1-2. Thus thresholding of 0.5 was applied to convert the continuous variable to a categorical one. With the varied distribution in the popularity category, it can be statistically seen that with the increase in the popularity of the song more and more songs are been built as Yes explicit from 5.2% to 38.08%. However No explicit category taking much of the distribution of overall as well as categorical popularity subsets.

Danceability, scaled from 0-1, provides the measure to determine if the song is preferred by people to dance and groove on. The dataset consists of the Gaussian distribution of danceability in songs, which can even be seen among the various categories of popularity. In total there is just a slight shift of the distribution towards the right in terms of popularity. That means that songs that are more danceable tend to get more and more popular.

Energy feature has a high correlation with the popularity of the song, scaled from 0-1. There is a generalistic positive trend with the loudness and release time span. Energy itself is distributed in the normal distribution. It is observed that songs with lower energy have statistically been less popular.

Other features containing lesser correlation with popularity can be feature extracted to enhance the relationship between the two and help improve the results as well. Valence on a scale of 0-1, 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). It can be inferred that songs are not popularized due to the positive or negative valence but due to the other factors contributing much more to the same. Features like liveness, speechiness tend to have a skewed right distribution not just in the dataset but in categorical popularity distribution as well. Tempo variable is normally distributed like danceability of the song with outliers near 0. The feature key has hardly any significance in the prediction of the popularity of a song. Lastly, in general, the mode of the song was labelled as 1-2, representing Major, Minor respectively,, where the majority of the songs were distributed in 1.

DATA PREPROCESSING

4.1 Basic Preprocessing Steps

4.1.1 Label Encoding of Categorical Features

The features “popularity”, “mode”, “explicit” were categorical, so we encoded these columns as follows -

- pop_dict = {'very low': 1, 'low': 2, 'average': 3, 'high': 4, 'very high': 5}
- mode_dict = {'Minor': 1, 'Major': 2}
- explicit_dict = {'No': 1, 'Yes': 2}

4.1.1 Adding New Features

To improve the performance of the classifier, we constructed the following features from the given features to provide a stronger signal to the model -

- We encoded the “release_date” features into -
  - “year”
  - “month”
  - “week_of_year”: Integer [1-52] for the location of the week with respect to the year
  - “day_of_week”: Integer [0-6] representing the day of the week
  - “day_of_month”
  - “is_weekend”: Boolean which is True if the day is a weekend

- **Novel Features**
  - “Year_band”: It was globally observed that rather than using individual year we could group a decade into a year band, to provide better correlations.
    Information taken from: Click here
  - “month_band”: Similarly, categorizing dual months in one go like “January-February”, “March-April” and so on, we obtain better results on popularity predictions.
    Information taken from: click here

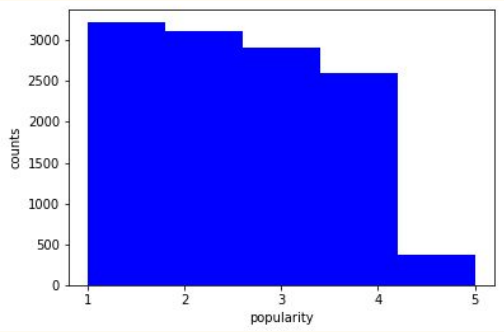
- **Transformations**
  - “loudness”: Applied binning with respect to distance of multiples of standard deviations from mean, followed by logarithmic scaling to normalize the distribution.
  - “acousticness”: Exponential scaling of the feature giving a 5% increase in correlation.
  - “speechiness”: Exponential scaling of the feature giving a 6.45% increase in correlation.
  - “instrumentalness”: Logarithmic scaling there was transformation of skewed to normal distribution in various categories of popularity.
  - “energy”: Logarithmically scaled giving a 6.13% increase in correlation.
  - “valence”: Exponentially scaled which substantially improved the dependency.
  - “danceability”: Exponentially scaled due to the skewed distribution.
  - “duration-min”: Binned into categories depending on various time stamps, followed by a logarithmic scaling to convert the skewed distribution to a normal distribution.

- **Dropped Features**
  - “release_date” - After extracting the above features from the release date, the column is no longer necessary and hence we dropped it
  - “id” - We did not observe any clustering of ids with respect to the popularity, hence we dropped the same

- **SMOTE aftermath**
  - After over sampling, the categorical variables: year, month, day, explicit, mode, key converts to continuous distribution.These were later scaled back to categories to retain the properties of the variables.

4.2 Synthetic Minority Oversampling Technique (SMOTE)

The distribution of values in the "popularity" column is as follows -



Hence it's clearly evident that there is significant imbalance in the dataset, particularly in the "very high" category of popularity. As a result, most machine learning techniques have poor performance on the minority class. To address the same, we used SMOTE.

The basic idea of SMOTE is to **oversample the minority class** by generating new samples. Since these samples are "synthetic", i.e. not observed data points but rather interpolated points from the n-dimensional feature space, hence the name "Synthetic" Minority Oversampling Technique. Using SMOTE we achieve a massive boost in performance, with the F1 score increasing from **0.62** to **0.71** with the Random Forests based classifier.

We used a number of variants of the basic SMOTE algorithm -

- **Borderline-SMOTE ->** A variant of SMOTE that involves selecting those instances of the minority class that are misclassified with a k-nearest neighbor classification model. The points on a **decision boundary** in the feature space are likely to be ambiguous (i.e. can belong to either region sharing the boundary) and are thus more important for classification. Emphasizing the generation here would enable the generation of samples that allow the models to learn the decision boundary better.
- **Borderline-SMOTE SVM ->** an alternative of Borderline-SMOTE where an SVM algorithm is used instead of a KNN to identify misclassified examples on the decision boundary. This model enjoys the advantages of Borderline SMOTE and also adds a superior Support Vector Machine for the detection of the decision boundary. Due to this, this variant of SMOTE gives the best augmented dataset that is able to achieve an F1 score of **0.72** (which is increased further using other techniques).

4.3 Feature Selection

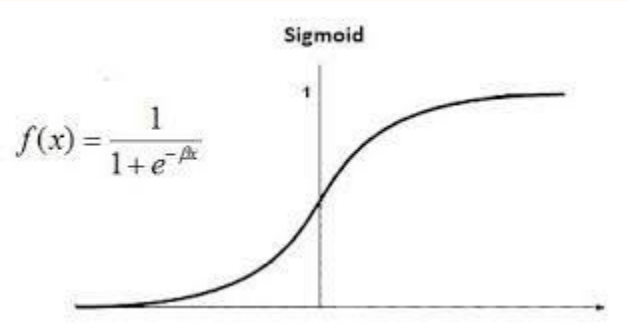
While the features given in the dataset are ample to make a predictive pipeline but what could be noted from literature present for music analysis over the year that genre and the artist's identity itself could give a strong predictive pipeline. Here's where we use the existing features to develop better ones.

Generating artist's name based on other features is quite difficult because most artist's perform different genre. Hence we apply **K-means algorithm** on numerical features like acousticness, energy, loudness, speechiness, danceability etc. to make clusters which could very well be identified as music genre thereby enhancing classification power. Plotting SSE vs number of clusters indicate an elbow at around K=25. Hence we get 25 genre of music for the dataset. The genre is then added as a feature to the dataset. Refer to annexure **Pg. 22** for the maths and plot.

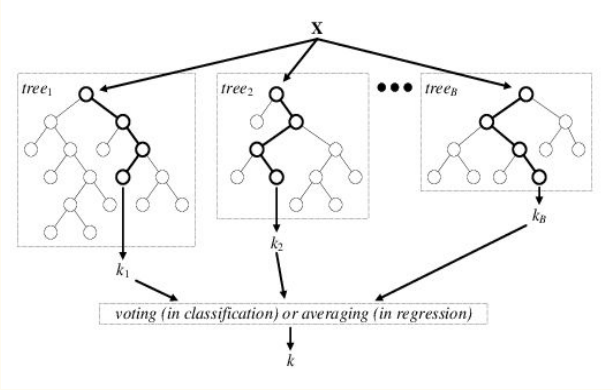
MODEL ARCHITECTURES

5.1 Machine Machine Models Applied

1)Logistic Regression is one of the simplest machine learning algorithms and is easy to implement.We have used as **a benchmark model to measure performance.** As the training data has sufficient number of training examples, logistic regression is **less prone to overfitting**.



2)Random Forest is a **bagging algorithm** and hence it **reduces overfitting** problem by taking in the decisions of many trees **which also reduces the variance and therefore improves the accuracy.**
The model is usually **robust to outliers** and can handle them well and hence overall this model can be trained on dataset to give desired results.



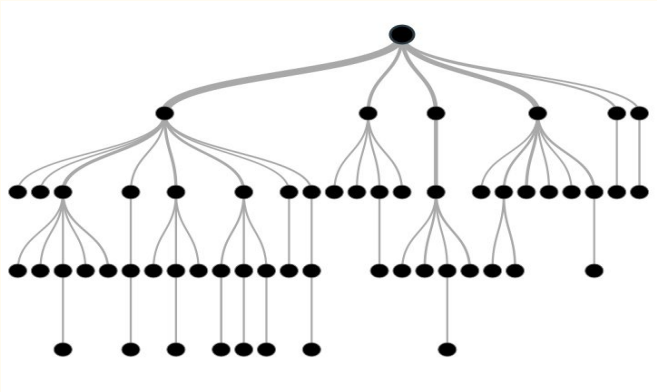
3)XG Boost is an efficient and easy to use algorithm which delivers high performance and accuracy. It has **in-built L1 (Lasso Regression)** and **L2 (Ridge Regression) regularization** which prevents the model from overfitting. XGBoost also utilizes the power of **parallel processing** and making it **much faster than many algorithms** and a popularly chosen algorithm for training datasets.

4)Light GBM is a fast, distributed, **high-performance gradient boosting framework.** LightGBM uses **histogram based algorithm** i.e it buckets continuous feature values into discrete bins which fasten the training procedure also results in lower memory usage.Its also supports parallel computing and has great **compatibility with large datasets.**

5)Gradient Boosting is also a **tree - based algorithm**. One of the main advantages of gradient boosting is that it **allows to optimise a user specified cost function**.It is well suited for the problem statement provided where we have created our own cost function to optimize revenue generated. It also **performs well when you have unbalanced data** such as in real time risk assessment.

6)CatBoost provides state of the art results and it is competitive with any leading machine learning algorithm on the performance front. It **reduces the need for extensive hyper-parameter tuning** and **lower the chances of overfitting** also which leads to more generalized models. Its a good model to use for many diverse type of datasets.

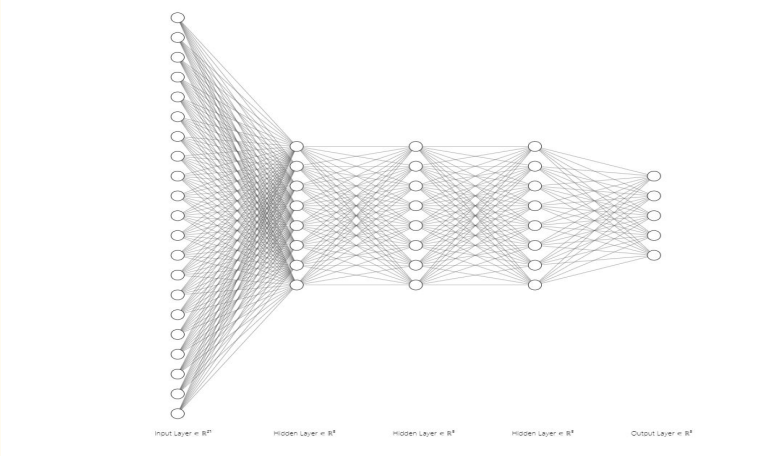
7)AdaBoost is easier to use with **less need for tweaking parameters** unlike other algorithms. AdaBoost is **not prone to overfitting** because of the reason that parameters are not jointly optimized — stage-wise estimation slows down the learning process. It **can be used to improve the accuracy of your weak classifiers** hence making it flexible to use with other models as well.



8)Stacking Ensemble Classifier is an ensemble machine learning algorithm.The benefit of stacking is that it can **harness the capabilities of a range of well-performing models** on a classification or regression task and **make predictions that have better performance** than any single model in the ensemble. In our model we have stacked the 3 models giving good results namely: **Gradient Boosting, AdaBoost** and **Light GBM**.

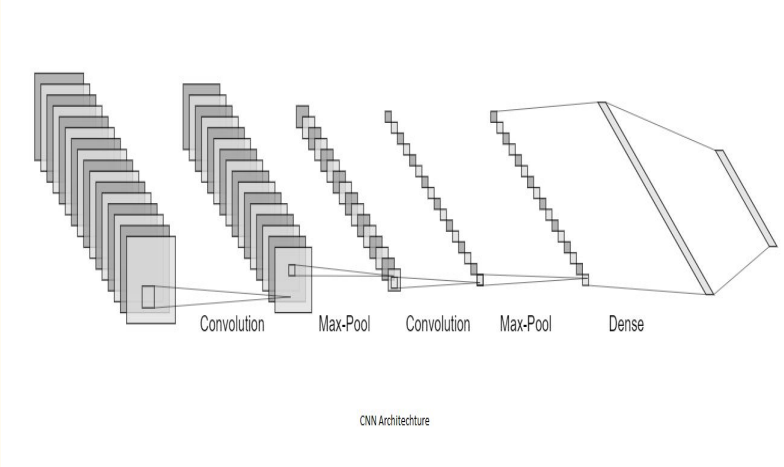
5.1 Neural Networks

Artificial Neural Network
ANN is a very good deep learning models to capture complex relation between the features. We trained a deep neural network to check how it performs on our dataset.

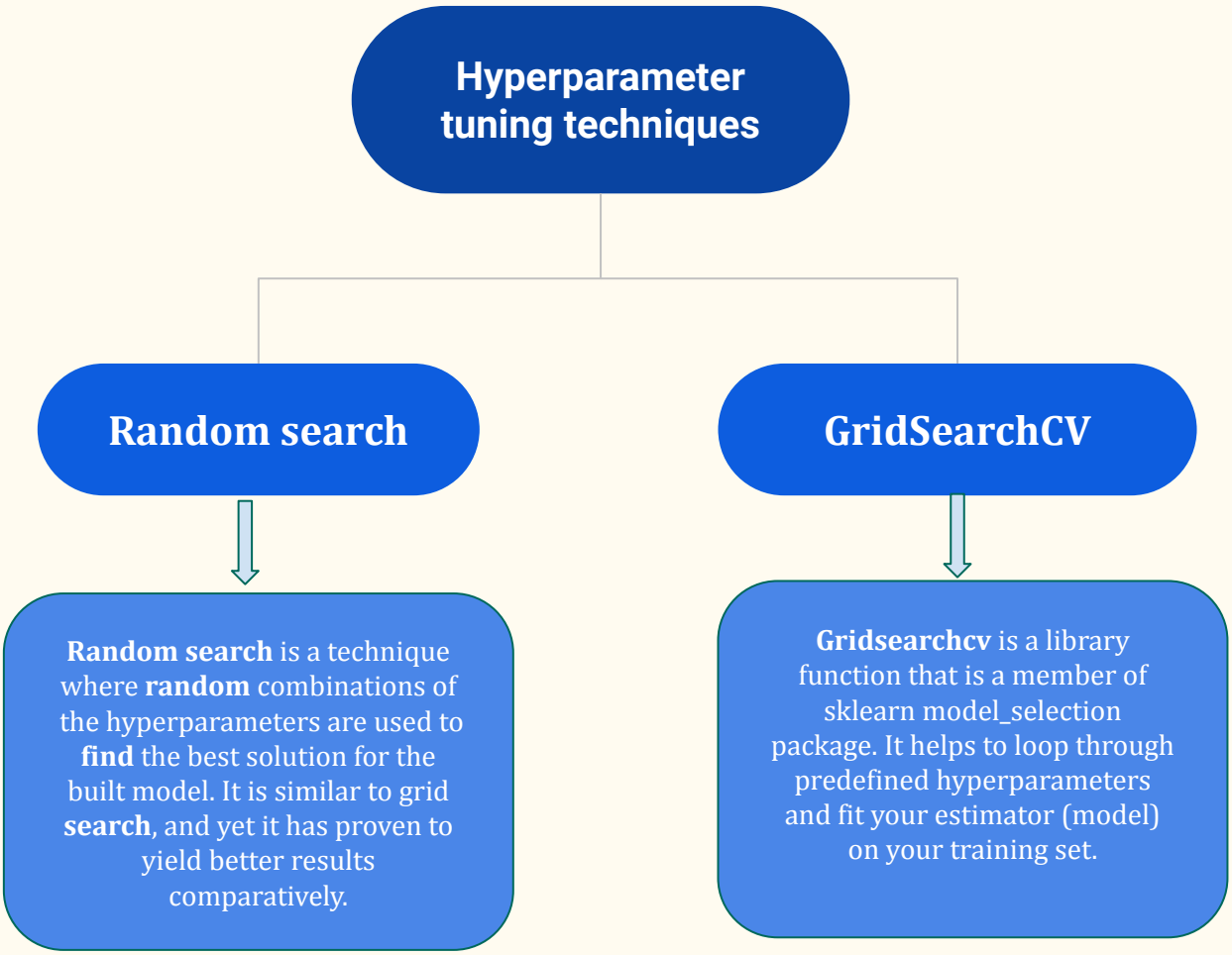


1-D Convolution Neural Network

CNNs are typically used for Vision tasks, but 1D CNNs have been shown to be effective at reducing the dimensionality of large datasets.



5.2 Hyperparameters & Tuning



Optimization with Orthogonal Array Tuning

Deep learning algorithms have achieved excellent performance lately in a wide range of fields (e.g., computer version). However, a severe challenge faced by deep learning is the high dependency on hyper-parameters. The algorithm results may fluctuate dramatically under the different configuration of hyper-parameters. (Recurrent Neural Networks and Convolutional Neural Networks). So we use the Orthogonal Array tuning method to get the best hyperparameters for the dataset.

Random Forest Classifier Parameters :

- **n_estimators**:We may think that using many trees to fit a model will help us to get a more generalized result, but this is not always the case.
- **max_depth**: It governs the maximum height upto which the trees inside the forest can grow.
- **max_features**: Random forest takes random subsets of features and tries to find the best split. It can take four values “auto“, “sqrt“, “log2” and None.

USING GRID SEARCH AND RANDOM CV, BEST PARAMETERS ARE:
max_features='auto', n_estimators= 7500, n_jobs=-1

LGBM Classifier Parameters

- With LightGBM you can run different types of Gradient Boosting methods. You have: GBDT, DART, and GOSS which can be specified with the boosting parameter.
- **num_leaves:** Surely num_leaves is one of the most important parameters that controls the complexity of the model.
- **max_bin:** Binning is a technique for representing data in a discrete view (histogram). LightGBM uses a histogram based algorithm to find the optimal split point while creating a weak learner.

USING GRID SEARCH AND RANDOM CV, BEST PARAMETERS ARE:
max_depth= 4, learning_rate=0.2, n_estimators=200,objective='binary', min_child_samples=20, colsample_bytree=0.5, reg_alpha=1, reg_lambda=0,random_state=0, n_jobs=- 1, importance_type='split'

XGB & AdaBoost Parameters :

- **estimators:** The list of classifiers provided to be fit into the model.
- **classes:** The class labels.
- **estimator_weights_:** The weight assigned to each base estimator.
- **estimator_errors_:** Classification error for each estimator in the boosted ensemble.
- **feature_importance_:** Shows us which column has more importance than the other.

USING GRID SEARCH AND RANDOM CV, BEST PARAMETERS ARE:
n_estimators=4, random_state=0, algorithm='SAMME'
XGBClassifier(objective ='reg:logistic', colsample_bytree = 0.5, learning_rate = 0.5,max_depth = 2, alpha = 0, n_estimators = 40)

Artificial Neural Network Parameters :

- **Number of hidden layers** : We use hidden layers to capture more complex relation between the features. It also governs the size of our model.
- **Nodes** : Number of neurons in each hidden layer, more number of nodes will make a strong network but we also have to keep overfitting in mind.
- **Activation** : We use different methods to find relation between the data some of them being sigmoid, tanh, relu.
- **Optimizer** : We have different algorithms to improve our model few of the most powerful ones are Adam, exp weighted average etc
- **Batch Size** : Number of data we input at a time in the bureau network is governed by this parameter.
- **Epochs** - This parameter governs the number of iteration we run our model for better learning.

USING GRID SEARCH AND RANDOM CV, BEST PARAMETERS ARE:
Number of hidden layer=3, nodes=8, batch_size=32, epochs=100, optimizer="Adam"

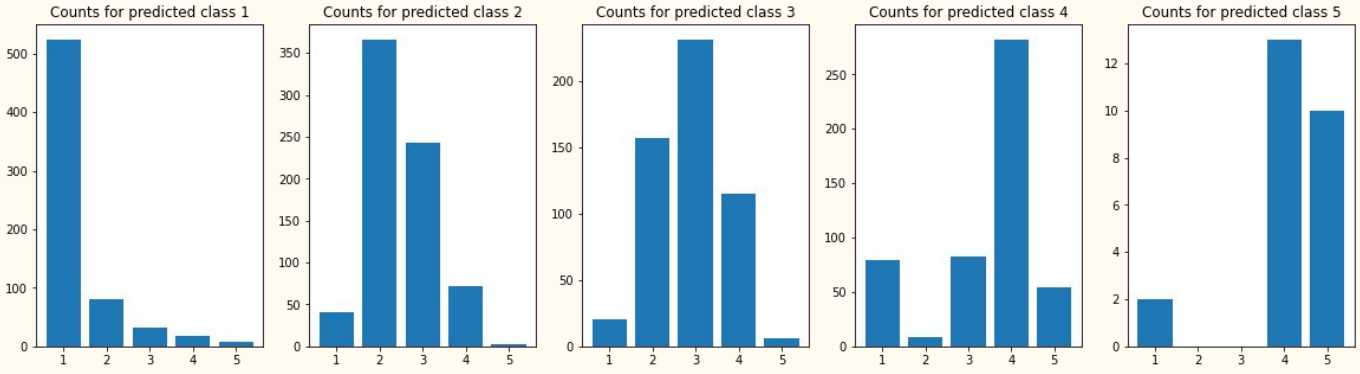
Convolutional Neural Network Parameters:

- **Number of hidden layers** : We use hidden layers to capture more complex relation between the features. It also governs the size of our model.
- **Nodes** : Number of neurons in each hidden layer, more number of nodes will make a strong network but we also have to keep overfitting in mind.
- **Activation** : We use different methods to find relation between the data some of them being sigmoid, tanh, relu, softmax(for classification model).
- **Optimizer** : We have different algorithms to improve our model few of the most powerful ones are Adam, exp weighted average etc
- **Batch Size** : Number of data we input at a time in the bureau network is governed by this parameter.
- **Epochs** : This parameter governs the number of iteration we run our model for better learning.
- **Max Pool Layer Shape :** Since it is 1d **CNN** the shape of the moving window will affect the result.

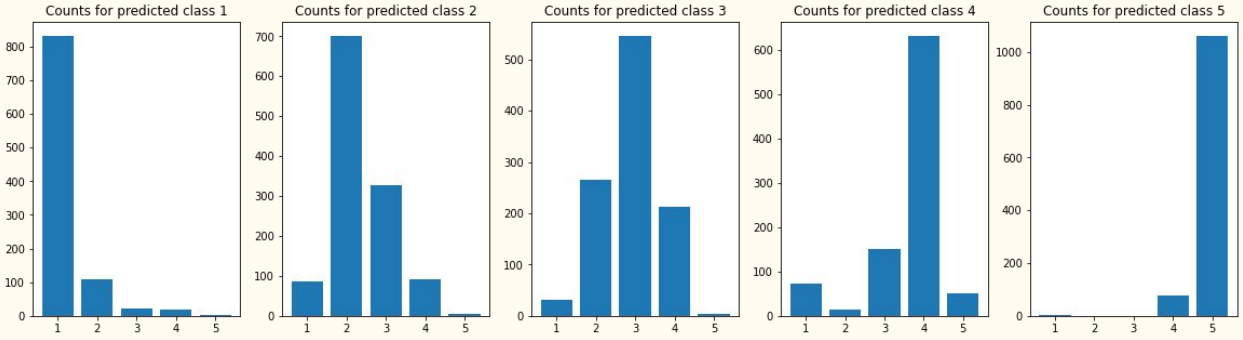
USING GRID SEARCH AND RANDOM CV, BEST PARAMETERS ARE:
Number of hidden layer=6(2+2+1+1), nodes=8, batch_size=32, epochs=100, optimizer="Adam"

ERROR ANALYSIS

We carried out the error analysis both before and after preprocessing the data. Before preprocessing, the 'release_date' column was split into two separate columns ('day' and 'month') and the categorical variables were given corresponding integer labels. AdaBoostClassifier was employed to model the train data and the predictions were checked on the validation data. In order to carry out the error analysis, we designed a function that takes in the predicted and the actual labels and then shows a bar plot for each of the predicted labels. To be specific, suppose pick the predicted label as 1 i.e. 'very low', then the counts of all the true labels whose predicted label was 1 was computed. This gave us 5 count values for the 5 classes and thus resulted in the bar plot as can be seen in the 'baseline histogram' notebook. This gave us the idea about where the model is going wrong. It was observed that the model was committing errors for labels '2', '3' and '4' mainly.



After the preprocessing steps, RandomForestClassifier was again employed on the new data. This improved the predictions by the model and the plots can be seen below. Since we used the custom loss function and feature engineering we learn the parameters better and this can be seen in the error plot below:



It can be inferred that the error % as observed for the predicted labels of '2', '3' and '4' have decreased substantially.

CUSTOM LOSS FUNCTION AND CUSTOM METRIC

The custom loss has been designed to capture various different scenarios in model training or tuning hyperparameters. These scenarios are derived with respect to the relationship between the predicted and the actual bid. Following are the scenarios and a detailed description of implementation of custom loss:

- Scenario A : The bid predicted is same as actual bid Loss value stays as 0
- Scenario B : The bid predicted is less than actual bid
  - We have a no_bid situation i.e., we can not bid
  - Consider a case when the actual bid was 5, and expected revenue was 10. Now, our model predicts the bid to be anything less than 5, then we are unable to gain (10 - 5) = 5. That's a huge loss. **We should penalize severely**
  - Also consider when the actual bid was 2, and our model predicted to bid 1. But, this is in stark contrast with above point. In the above case, there was a gain of 5, but here we had a gain of only (4-2) = 2, thus not a large loss. Hence **should be penalized less severely**.
- Scenario C : The bid predicted is higher than actual bid
  - We are incurring certain loss and thus accordingly penalizing. Two examples are provided below to clarify how the loss is being evaluated.
  - Case a: Say the actual bid was 1. Our model predicts 5. Let us calculate the loss. Had we bid 1, the gain we would have got is (2 - 1) = 1. Now our model predicted bid value as 5. Thus, the gain we have is, (2 - 5) = -3. Thus the total loss = 1 - (-3) = 4
  - Case b: Say the actual bid was 3. Our model predicts 4. Let us calculate the loss. Had we bid 3, the gain we would have got is (6 - 3) = 3. Now our model predicted bid value as 4. Thus, the gain we have is, (6 - 4) = 2. Thus the total loss = 3 - 2 = 1

So what's the deal with all this loss evaluation? The total loss is sum of all the loss values obtained. We just take a base and exponentiate. This means, say the base = 1.1. Then if a loss is 3, the contribution to the total loss will be **(1.1)^3**

For each no_bid scenario, if we have the actual bid to be x then we are incurring a loss of **2x − x = x** i.e. equal to the actual bid.

RESULTS

Model name	F1 Score	Revenue
baseline random forest dataset_2	0.7229430105	19352
baseline AdaBoost	0.4150970674	15933
baseline CatBoost	0.6765829873	18819
baseline Gradient Boost	0.6705473489	18728
baseline LightGBM	0.6977605291	19029
baseline XG Boost	0.6697120434	18695
baseline Logistic Regression	0.4101189649	15832
baseline Stacking Classifier	0.6917760043	18994
Baseline ANN	0.583	14326
Baseline CNN-1D	0.6379	15321

We should realize that even though the model might perform well in the F1 or the Accuracy metric that does not guarantee that the revenue is maximized. So in order to correctly formulate the custom metric, we used the revenue generated after investing the a total sum of 10000$ on 4000 validation music stocks. We purposely made a train-val split such the validation dataset has exactly 4000 music albums to bid for. All the predictions of the model for the validation set is used to calculate the revenue generated by the model. We also use a constant seed number so that the test split remains the same on every run. This is required when we calculate the maximum possible revenue the model can generate. We calculated that if the test set has 4000 music albums to be generated then based on the popularity of the albums the maximum possible revenue is **22013$.** This means that if our model has 100% accuracy then the revenue generated by the model is 22013$. Since it is difficult to maximize the accuracy we instead maximize the revenue directly by adjusting the loss function. In this way we train the model to atleast give a positive profit rather than zero or negative ones. After using the revenue as the main metric function and the custom loss function instead of the normally used loss function. Our revenue before any preprocessing and custom loss was around 15753$. After doing the preprocessing and using custom loss function we got a boost in revenue to **19352$** and the F1 score is around **0.7229**. So although our accuracy of prediction on the validation set is around 73% the percentage of revenue generated is close to **88 %.**

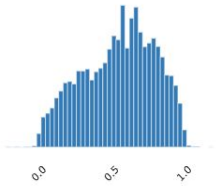
ANNEXURE

Descriptive Statistics

energy
Real number (ℝ)

Distinct	5244
Distinct (%)	32.6%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	0.5462043374
Minimum	-0.2309064825
Maximum	1.287282868
Zeros	0
Zeros (%)	0.0%
Memory size	126.0 KiB



Statistics | Histogram | Common values | Extreme values

Quantile statistics

Minimum	-0.2309064825
5-th percentile	0.112
Q1	0.355
median	0.573
Q3	0.742
95-th percentile	0.919
Maximum	1.287282868
Range	1.51818935
Interquartile range (IQR)	0.387

Descriptive statistics

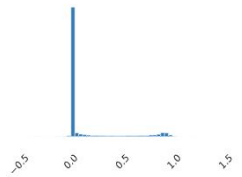
Standard deviation	0.2484581454
Coefficient of variation (CV)	0.4548813116
Kurtosis	-0.8126947651
Mean	0.5462043374
Median Absolute Deviation (MAD)	0.189
Skewness	-0.2715017335
Sum	8799.351875
Variance	0.06173145
Monotocity	Not monotonic

instrumentalness
Real number (ℝ)

ZEROS

Distinct	6421
Distinct (%)	39.9%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	0.118383469
Minimum	-0.4547805407
Maximum	1.433386109
Zeros	4689
Zeros (%)	29.1%
Memory size	126.0 KiB



Toggle details

Statistics | Histogram | Common values | Extreme values

Quantile statistics

Minimum	-0.4547805407
5-th percentile	0
Q1	0
median	3.668197839×10^5
Q3	0.0142
95-th percentile	0.878
Maximum	1.433386109
Range	1.88816665
Interquartile range (IQR)	0.0142

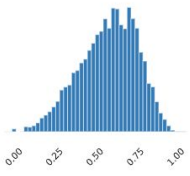
Descriptive statistics

Standard deviation	0.2723564093
Coefficient of variation (CV)	2.300628724
Kurtosis	3.253766619
Mean	0.118383469
Median Absolute Deviation (MAD)	3.841531744×10^5
Skewness	2.186814763
Sum	1907.157686
Variance	0.07417801369
Monotocity	Not monotonic

danceability
Real number (ℝ)

Distinct	4695
Distinct (%)	29.1%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	0.5802932018
Minimum	-0.06233258651
Maximum	1.161840125
Zeros	15
Zeros (%)	0.1%
Memory size	126.0 KiB



Toggle details

Statistics | Histogram | Common values | Extreme values

Quantile statistics

Minimum	-0.06233258651
5-th percentile	0.264
Q1	0.459
median	0.598
Q3	0.716
95-th percentile	0.844461731
Maximum	1.161840125
Range	1.224172711
Interquartile range (IQR)	0.257

Descriptive statistics

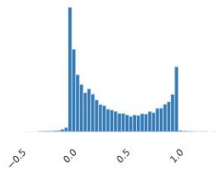
Standard deviation	0.1780829808
Coefficient of variation (CV)	0.3068844857
Kurtosis	-0.3097005416
Mean	0.5802932018
Median Absolute Deviation (MAD)	0.126
Skewness	-0.3884625954
Sum	9348.52348
Variance	0.03171354805
Monotocity	Not monotonic

acousticness

Real number (ℝ)

Distinct	6566
Distinct (%)	40.8%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	0.3946110588
Minimum	-0.4132191078
Maximum	1.367773668
Zeros	0
Zeros (%)	0.0%
Memory size	126.0 KiB



Toggle details

Statistics | Histogram | Common values | Extreme values

Quantile statistics

Minimum	-0.4132191078
5-th percentile	0.0005548
Q1	0.0658
median	0.286
Q3	0.737
95-th percentile	0.986
Maximum	1.367773668
Range	1.780992776
Interquartile range (IQR)	0.6712

Descriptive statistics

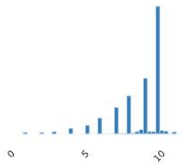
Standard deviation	0.3546008814
Coefficient of variation (CV)	0.8986085754
Kurtosis	-1.263793401
Mean	0.3946110588
Median Absolute Deviation (MAD)	0.2658057483
Skewness	0.4493346825
Sum	6357.184158
Variance	0.1257417851
Monotocity	Not monotonic

loudness

Real number (ℝ)

Distinct	1497
Distinct (%)	9.3%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	8.664623087
Minimum	-1.727015516
Maximum	12.24272828
Zeros	0
Zeros (%)	0.0%
Memory size	126.0 KiB



Toggle details

Statistics | Histogram | Common values | Extreme values

Quantile statistics

Minimum	-1.727015516
5-th percentile	5
Q1	8
median	9
Q3	10
95-th percentile	10
Maximum	12.24272828
Range	13.96974379
Interquartile range (IQR)	2

Descriptive statistics

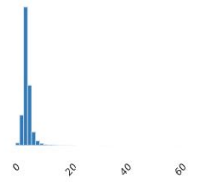
Standard deviation	1.759509488
Coefficient of variation (CV)	0.2030682086
Kurtosis	2.763255415
Mean	8.664623087
Median Absolute Deviation (MAD)	1
Skewness	-1.605067292
Sum	139587.0779
Variance	3.09587364
Monotocity	Not monotonic

duration-min

Real number (ℝ)

Distinct	3784
Distinct (%)	23.5%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%

Mean	3.802159476
Minimum	-0.391670827
Maximum	72.8
Zeros	0
Zeros (%)	0.0%
Memory size	126.0 KiB



Toggle details

Statistics | Histogram | Common values | Extreme values

Quantile statistics

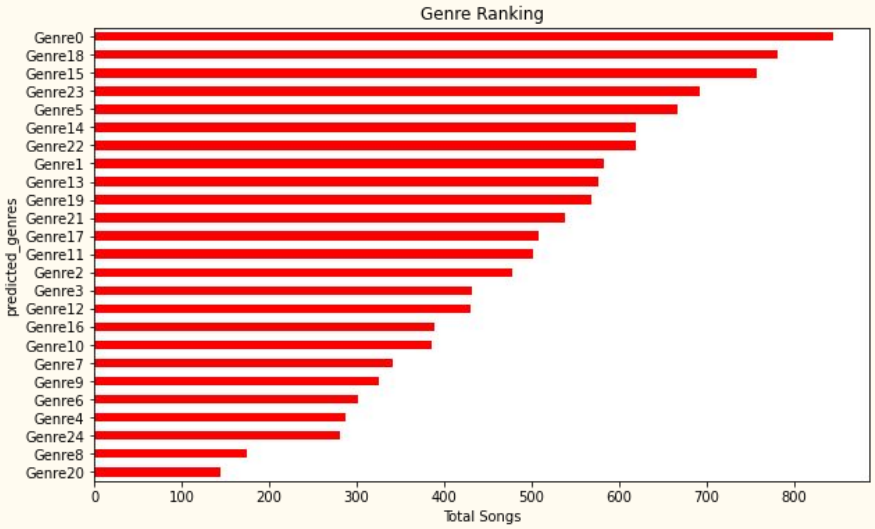
Minimum	-0.391670827
5-th percentile	2
Q1	2.996736751
median	3.5
Q3	4.2
95-th percentile	6.3
Maximum	72.8
Range	73.19167083
Interquartile range (IQR)	1.203263249

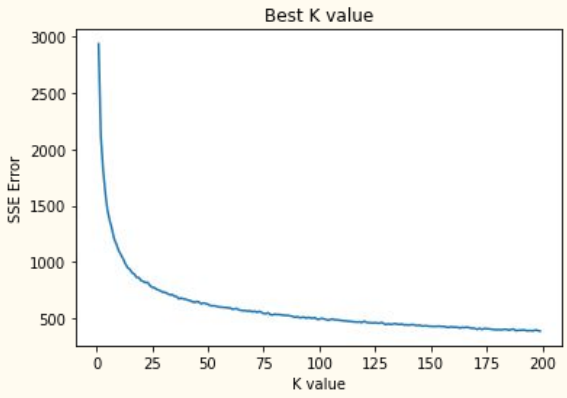
Descriptive statistics

Standard deviation	2.117198276
Coefficient of variation (CV)	0.5568409977
Kurtosis	347.7854059
Mean	3.802159476
Median Absolute Deviation (MAD)	0.6
Skewness	13.61588358
Sum	61252.78916
Variance	4.482528541
Monotocity	Not monotonic

Feature Generation

We have used K-means for generating additional features in the dataset namely the genre of the music. The plots here validate the mathematical, graphical inferences for the choice of K and it's distribution in the training data. The plot of right is the SSE vs K plot, we see an elbow formation in the range of 23-27, so we use K=25. The count of the music ids belonging to these 25 genre is represented in the figure on the left.





The table below gives us the percentage variation explained by each eigenvalue using PCA when mode, explicit, key, id, popularity, release date were dropped from the dataframe. A clear indication for strong feature engineering

Eigenvalue	Percentage Variance Explained
EV-1	30.1%
EV-2	14.4%
EV-3	10.3%
EV-4	9.4%
EV-5	8.4%
EV-6	8.1%
EV-7	7.2%
EV-8	5.1%
EV-9	3.3%
EV-10	2.7%
EV-11	1.1%

Model Code

Preprocessing function

```
def preprocess(data):

  # Label Encode Features
  if "popularity" in data:
    data["popularity"] = data["popularity"].apply(lambda x: pop_dict[x])

  data["mode"] = data["mode"].apply(lambda x: mode_dict[x])
  data["explicit"] = data["explicit"].apply(lambda x: explicit_dict[x])

  # Add new features
  data["day_of_week"] = data["release_date"].apply(lambda x: datetime.strptime(x, "%d-%m-%Y").weekday())
  data["day_of_month"] = data["release_date"].apply(lambda x: datetime.strptime(x, "%d-%m-%Y").day)
  data["week_of_year"] = data["release_date"].apply(lambda x: datetime.strptime(x, "%d-%m-%Y").isocalendar()[1])
  data["month"] = data["release_date"].apply(lambda x: datetime.strptime(x, "%d-%m-%Y").month)
  data["year"] = data["release_date"].apply(lambda x: datetime.strptime(x, "%d-%m-%Y").year)
  data["is_weekend"] = data["day_of_week"].apply(lambda x: 1 if x>=5 else 0)
  data["year_band"] = get_year_band(data)
  data["month_band"] = get_month_band(data)
  data["loudness"] = transform_loudness(data)

  data = data.drop(["release_date", "id"], axis=1)

  return data
```

Custom Metric function (revenue)

```
def revenue(Y_test, Y_pred):
  Y_test = Y_test.to_numpy()
  revenue = 10000
  i = 0
  while i < len(Y_test) and revenue
>= 0:
    if Y_test[i] <= Y_pred[i]:
      revenue += Y_test[i]*2 -
Y_pred[i]
    i += 1
  return revenue
```

Custom loss function for hyperparameter tuning

```
popularity_mapper = {
                        'Very high': 5,
                        'high': 4,
                        'average': 3,
                        'low': 2,
                        'very low': 1
                    }
def custom_loss_function(y_pred, y_true, **kwargs):
    # kwargs contain additional parameters
    # kwargs should ideally contain -
        # 1. base: an integer
            # on which base you want to take the loss
        # 2. no_bid: an array (one entry for each class) of penalty for no bid
            # the amount of penalty you want to add to the loss
            # this is applicable in scenarios such as - when actual is average but you bid less than average (say low)
            # each class should be penalised differently - say actual is very high and your prediction was unable to bid - you loss straight 10K $
            # but say it was low and you predicted very low - it was mere 4K $ that you lost

    # base is close to 1 - so that the total_loss does not explode :(
    base, no_bid = 1.1, None
    no_bid_updated = False

    # stuffs with kwarg dictionary
    for key, value in list(kwargs.items()):
        if key == 'base':
            # update the base
            base = value
        elif key == 'no_bid':
            # update the no_bid array
            no_bid = value
            no_bid_updated = True

    # update no_bid finally if the no_bid has not been updated
    if not no_bid_updated:
        no_bid = [base**(i+1) for i in range(5)]

    # now convert the popularity predictions into numbers
    converted_pred = list(y_pred) # [popularity_mapper[x] for x in y_pred]
    converted_true = list(y_true) # [popularity_mapper[x] for x in y_true]

    # find the loss that you have incurred
    # there is loss incurred only if we have a different prediction than the actual popularity
    converted_pred_len = len(converted_pred)
    loss_array = [0] * converted_pred_len

    max_loss = sum([(1.1)**(i) for i in converted_true])

    for i in range(converted_pred_len):
        if converted_pred[i] != converted_true[i]:
            # first check if this is a no_bid situation
            if converted_true[i] > converted_pred[i]:
                loss_array[i] = no_bid[converted_true[i] - 1]

            else:
                actual_gain = converted_true[i]
                our_gain = (2 * actual_gain - converted_pred[i])
                loss_array[i] = base**(actual_gain - our_gain)

    total_loss = sum(loss_array)
    return total_loss/max_loss
```

Models score on non preprocessed dataset

Model name	F1 Score	Accuracy	custom metric score
baseline random forest	0.6085487645	0.615	15753
baseline AdaBoost	0.5335260524	0.56125	14586
baseline CatBoost	0.6033313936	0.6065	15717
baseline Gradient Boost	0.6047292825	0.6095	15683
baseline LightGBM	0.5966604148	0.5995	15694
baseline XG Boost	0.607197798	0.61325	15666
baseline Stacking Classifier	0.6057084171	0.61025	15759
baseline Logistic Regression	0.3156620616	0.34025	13230

Models score on preprocessed on SVM SMOTE() dataset

Model name	F1 Score	Accuracy	custom metric score
baseline LightGBM	0.6936372937	0.69275	19018
baseline random forest	0.7068120187	0.70675	19223
baseline AdaBoost	0.4368050079	0.52375	17531
baseline Gradient Boost	0.6685740247	0.6695	18701
baseline CatBoost	0.6877739644	0.68875	[18911]
baseline Logistic Regression	0.3427510927	0.35275	15083
baseline Stacking Classifier	0.690944176	0.68925	18974
baseline XG Boost	0.6672602407	0.6705	18722