OPEN IIT DATA ANALYTICS

Team Number: 44

Contents

<u>Introduction</u>

With new music trends coming up with the time and competition of music labels to purchase the best music tracks, it's really important that investments are made in the correct places.

<u>Overview</u>

- The following data analysis has been done for a music record label to help them purchase the most appropriate music track.
- A set of features describing the music tracks have been provided in correspondence to the popularity of the music track.
- Based on the predictions, 10000(in 10k $) will be invested to place bids on the 4000 music tracks. This model is expected to generate the highest possible revenue.

The popularities have been classified into 5 types as follows :
- Very High
- High
- Average
- Low
- Very Low

Data Description

The training dataset consists of 12,227 rows and 16 features. The description of each feature is given below :

	Features	Non-Null	Count	Dtype
0	id	12227	non-null	int64
1	acousticness	12227	non-null	float64
2	danceability	12227	non-null	float64
3	energy	12227	non-null	float64
4	explicit	12227	non-null	object
5	instrumentalness	12227	non-null	float64
6	key	12227	non-null	int64
7	liveness	12227	non-null	float64
8	loudness	12227	non-null	float64
9	mode	12227	non-null	object
10	release_date	12227	non-null	object
11	speechiness	12227	non-null	float64
12	tempo	12227	non-null	float64
13	valence	12227	non-null	float64
14	year	12227	non-null	int64
15	duration-min	12227	non-null	float64
16	popularity	12227	non-null	object

- ‘Acousticness’, ‘danceability’, ‘energy’, ‘instrumentalness’, ‘liveness’, ‘loudness’, ‘speechiness’, ‘tempo’, and ‘valence’ are continuous variables

- ‘Explicit’, ‘key’ and ‘mode’ are categorical features.

  Explicit represents whether the music is explicit or not(**Yes/No**).
  Key values range between **0 to 11**.
  Mode indicates the modality (**major or minor**) of a track.

Feature Description

- **Acousticness** - A scale from 0.0 to 1.0 that indicates whether or not the track is acoustic. The number 1.0 denotes a high degree of confidence that the track is acoustic.

- **Danceability** - Danceability is a musical concept that refers to how ideal a track is for dancing depending on different factors such as speed, rhythm stability, beat power, and overall regularity. The least danceable value is 0.0, and the most danceable value is 1.0.

- **Energy** - Energy is a perceptual indicator of stress and movement that ranges from 0.0 to 1.0. Typical energetic tracks have a fast, loud, and noisy feel to them.

- **Instrumentalness** - Determines whether or not a track has no vocals. Tracks like rap or spoken word are simply "vocal." The closest the instrumentalism score gets to 1.0, the more often the track is devoid of voices.

- **Key** - The estimated overall key of the track. Pitch Class notation is used to map integers to pitches.

- **Liveness** - The presence of an audience in the video is observed. Higher liveness values suggest a greater chance of the track being played live. If the value is greater than 0.8, the track is almost certainly live.

- **Loudness** - The decibel level of a track's average volume (dB). Loudness levels are summed over the whole track and can be used to compare the relative loudness of different songs.

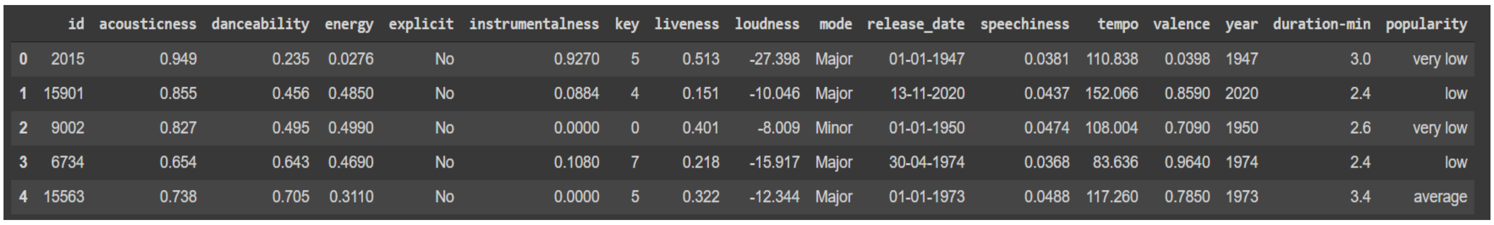
- **Mode** - The form of scale from which a track's melodic material is derived is indicated by the track's modality (major or minor). The number 1 represents the major, while the number 0 represents the minor.

- **Speechiness** - The appearance of spoken words in a track is detected by speechiness. The attribute meaning is similar to 1.0 the more purely speech-like the recording is (e.g. chat show, audiobook, poetry).

- **Tempo** - In beats per minute, a track's average approximate tempo (BPM). The tempo is the speed or rhythm of a piece in musical terms, and it is determined by the average beat length.

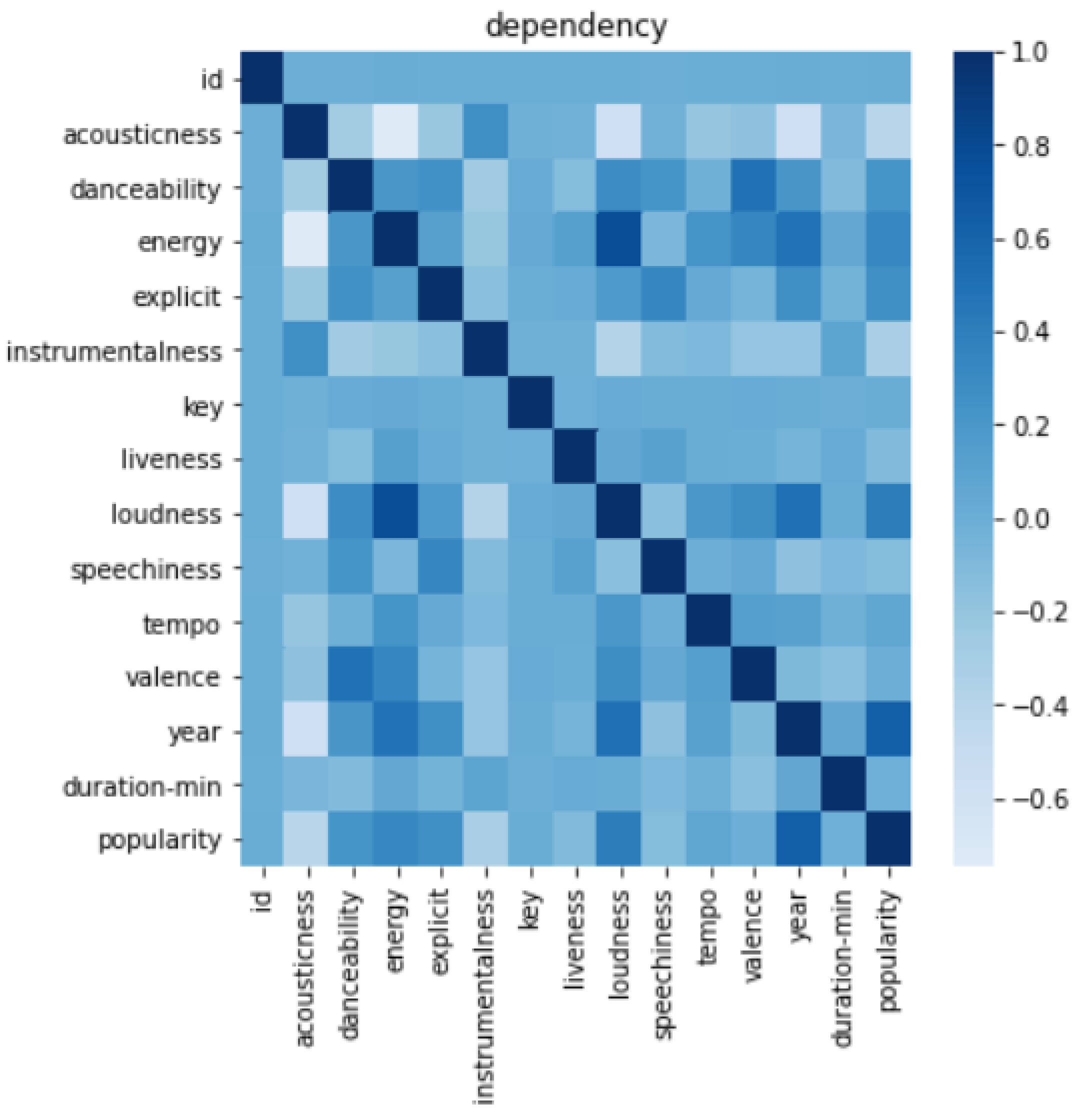
- **Valence** - Tracks with a high valence sound more optimistic, while tracks with a low valence sound more pessimistic.

Top 5 rows of provided train-dataset



	id	acousticness	danceability	energy	explicit	instrumentalness	key	liveness	loudness	mode	release_date	speechiness	tempo	valence	year	duration-min	popularity
0	2015	0.949	0.235	0.0276	No	0.9270	5	0.513	-27.398	Major	01-01-1947	0.0381	110.838	0.0398	1947	3.0	very low
1	15901	0.855	0.456	0.4850	No	0.0884	4	0.151	-10.046	Major	13-11-2020	0.0437	152.066	0.8590	2020	2.4	low
2	9002	0.827	0.495	0.4990	No	0.0000	0	0.401	-8.009	Minor	01-01-1950	0.0474	108.004	0.7090	1950	2.6	very low
3	6734	0.654	0.643	0.4690	No	0.1080	7	0.218	-15.917	Major	30-04-1974	0.0368	83.636	0.9640	1974	2.4	low
4	15563	0.738	0.705	0.3110	No	0.0000	5	0.322	-12.344	Major	01-01-1973	0.0488	117.260	0.7850	1973	3.4	average

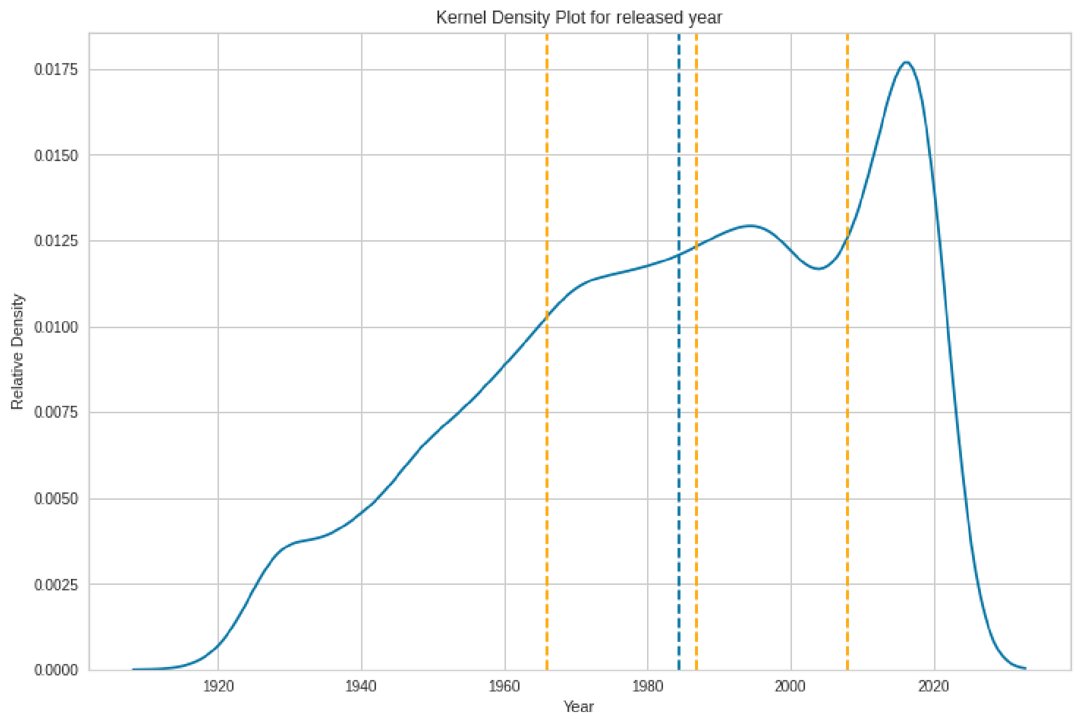
Drawing Correlation between features



Observation: We find that popularity has a highest positive dependency with "**year**" having a correlation coefficient of 0.635 and then with "**loudness**" and "**energy**". It also has a highest negative dependency with "**Acousticness**" having a correlation coefficient of -0.407.

Understanding the distribution over the years

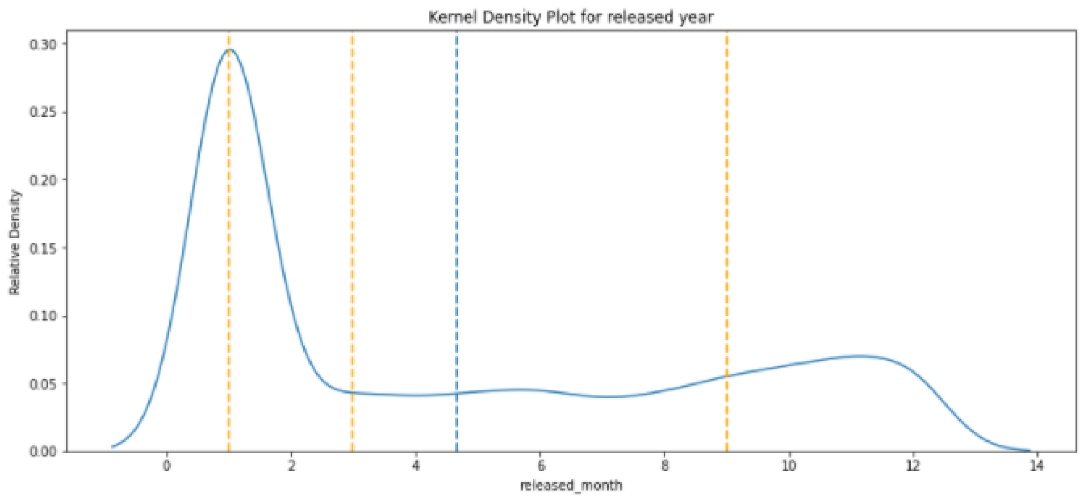
- Density Plot of the distribution of data over a continuous passing year through kernel smoothing.



We observe that with time more and more music tracks are being released. Also to incorporate the effect of release year, we create **10 bins** representing the deciles release years.

Understanding the distribution over a month

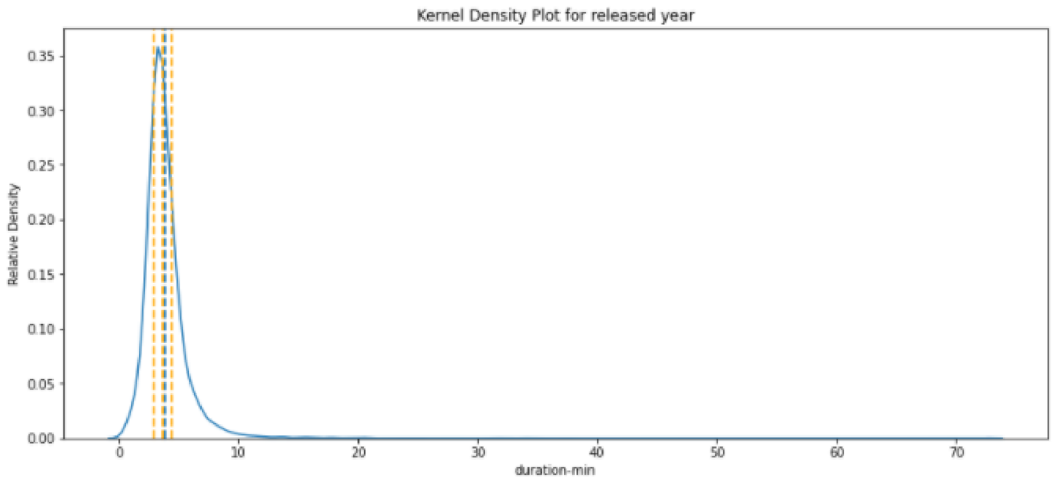
- Density Plot of the distribution of data over a continuous passing month (release month is taken from release date).



Most of the songs are released either during the start or the end of a year. We incorporate the effect of the month by using it as a categorical variable (to check for any festive or vacation or seasonal effect)

Understanding the distribution over the duration

- Density plot of the distribution of data over the duration



We prefer to think that song tracks of certain duration are more popular than others. To incorporate that in our more we create **10 bins** for the “duration” after finding its deciles.

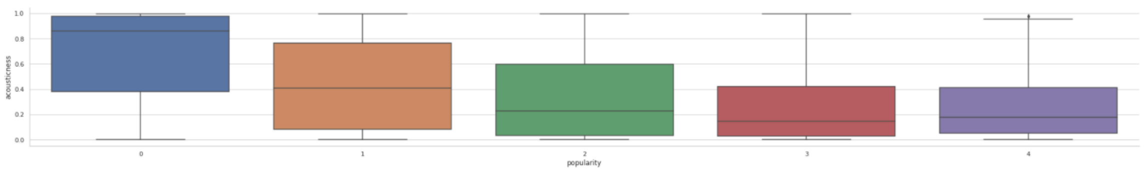
Note-

★ Kernel Density Charts are a variation of a Histogram that uses kernel smoothing to plot values, allowing for smoother distributions by smoothing out the noise.

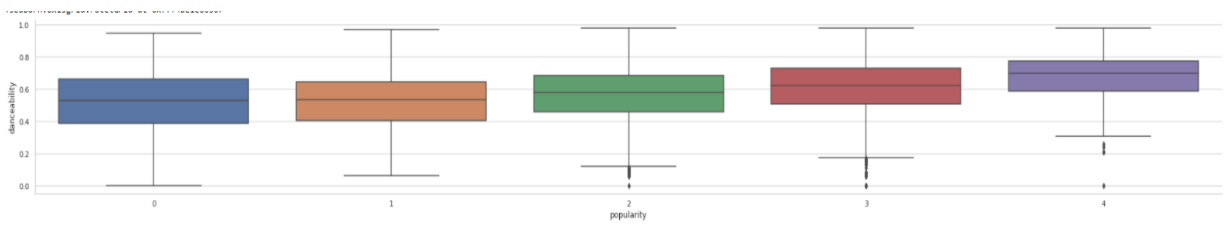
Data Visualisation

Box Plot: Boxplots are a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”).

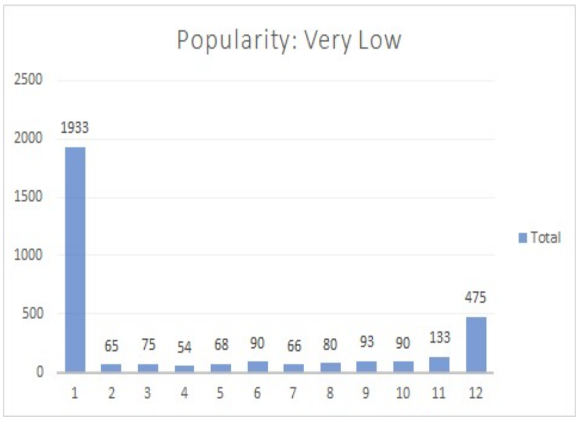
- Acousticness

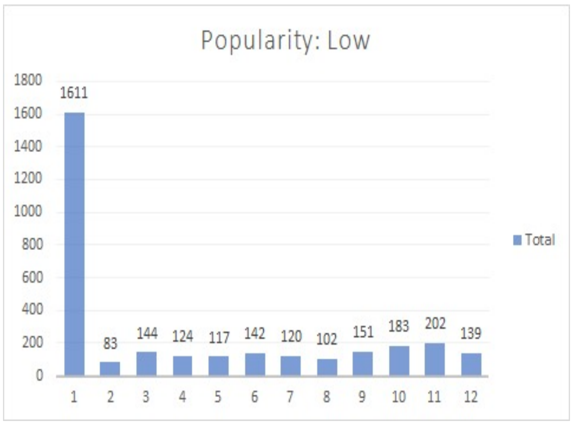


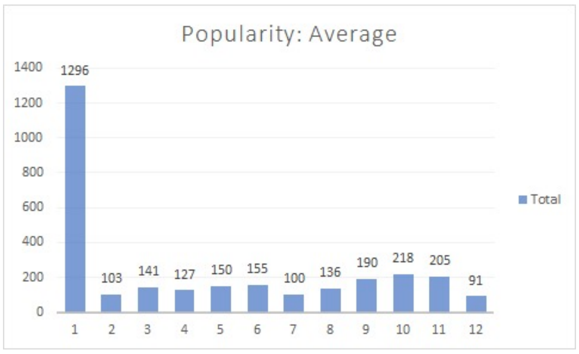
- Danceability

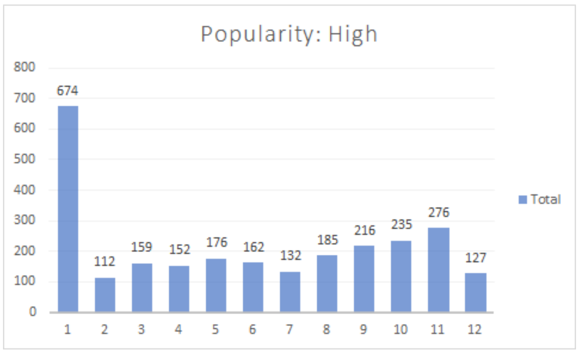


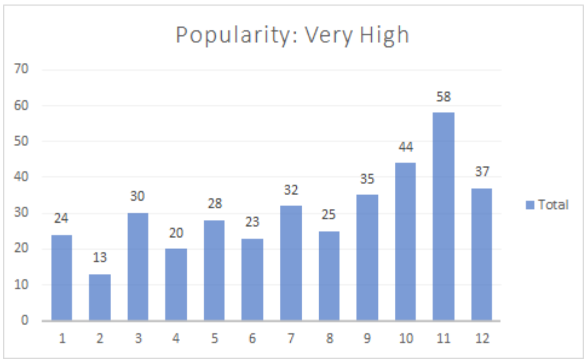
➔ **Histogram distribution based over month**

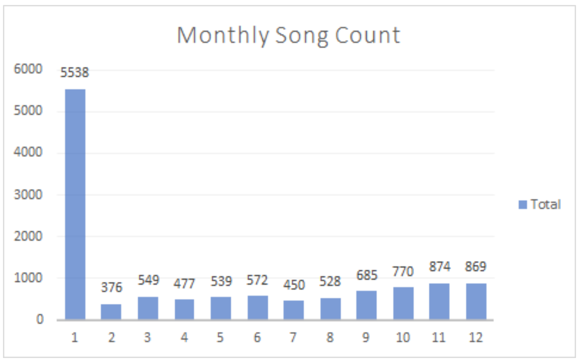






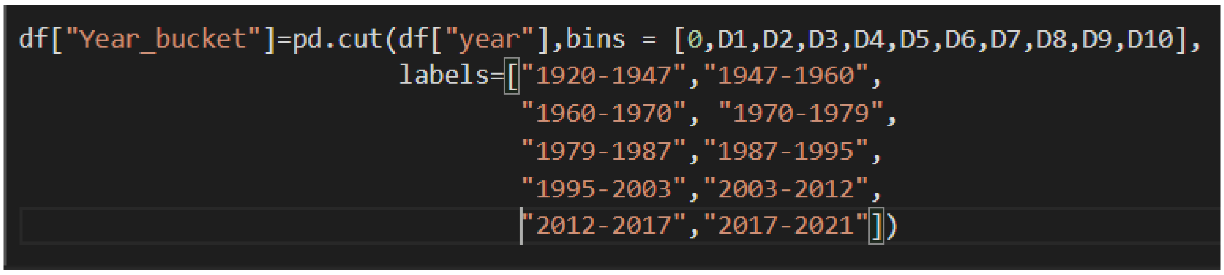






Creation of new variables

- From the kernel density plot, it can be concluded that the frequency of songs has been rising in the recent two decades. To uniformly study the distribution

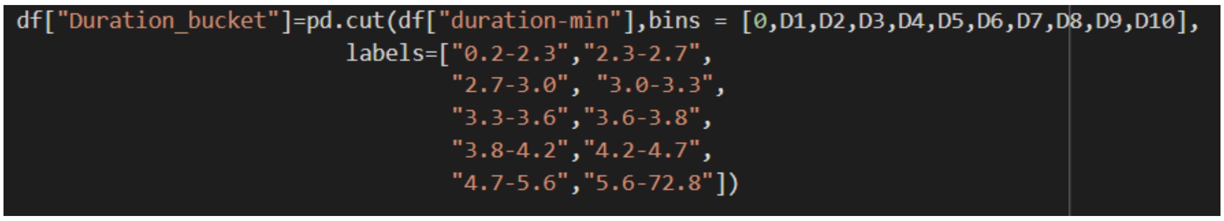


```
df["Year_bucket"]=pd.cut(df["year"],bins = [0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10],
                         labels=["1920-1947","1947-1960",
                                 "1960-1970", "1970-1979",
                                 "1979-1987","1987-1995",
                                 "1995-2003","2003-2012",
                                 "2012-2017","2017-2021"])
```

  and draw appropriate logical conclusions, we split the data into 10 cohorts.

- With music being dependent a lot on festive seasons and occasions around, the month of launch might also act as a parameter to decide how popular a song is. So from the ‘date’ feature we go on to create a new variable ‘month’.

- Similarly we found a lot of uneven distribution in the duration of songs, with it getting denser in the 5-7 minute region. Hence, to uniformly distribute we split the duration into 10 equal cohorts.



```
df["Duration_bucket"]=pd.cut(df["duration-min"],bins = [0,D1,D2,D3,D4,D5,D6,D7,D8,D9,D10],
                             labels=["0.2-2.3","2.3-2.7",
                                     "2.7-3.0", "3.0-3.3",
                                     "3.3-3.6","3.6-3.8",
                                     "3.8-4.2","4.2-4.7",
                                     "4.7-5.6","5.6-72.8"])
```

Dummy Variables

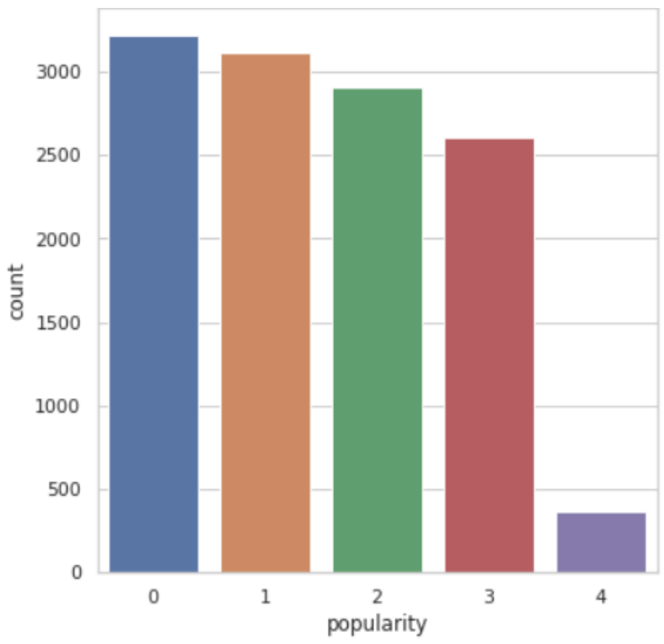
In the list of given features and features newly created, we find many categorical variables, with most of them consisting of more than 2 categories.
So we create a new variable(feature) for each category and represent it with dummy variables. Where, 1 represents the presence of that feature and 0 the absence of that feature. Dummy of following variables (with categorical values) were created:

released_month
Duration_bucket
Year_bucket
key
explicit
mode

Model Specification

Check for balanced Classes

➔ **Histogram**



➔ **Tabular**

Popularity	Numerical Representation	Test data Count
Very Low	0	3222
Low	1	3118
Average	2	2912
High	3	2606
Very High	4	369

Classes are **unbalanced,** especially “Very Popularity” class with only **369 values**

Train - Validation Split :

We’ve split the original training data into two parts, the training set and the validation set. The ratio for training and validation set is **75:25** respectively.

After splitting into training and validation sets, we get the following distribution in the training set.

Historiographical representation of Train data count for different popularity

Value counts of the labels in Train Set:

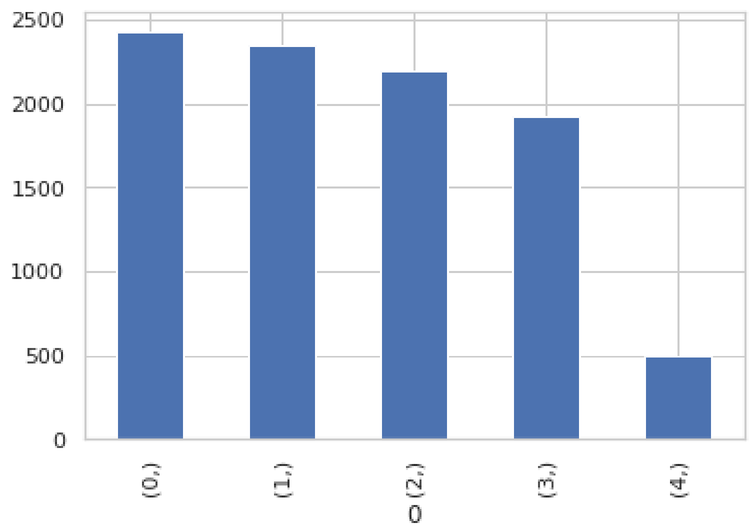
- 0: 2428
- 1: 2348
- 2: 2128
- 3: 1928
- 4: 277

Challenge: the training data counts for the songs in the “very high” category is low, the model needs sufficient data to learn parameters so that it can correctly classify the “very high” category

Used **oversampling** to adjust the class distribution of a data set (i.e. the ratio between the different classes/categories represented) and increased the training data count for “very high” by random sampling from 277 to 500.

After oversampling the labels in the Train Set:

- 0 :2428
- 1 :2348
- 2 :2189
- 3 :1928
- 4 :500



We don’t do oversampling for the validation set to maintain consistency of the model. We wish to create a model which can be used again and again on different real datasets.

Model fitting

We tried following models for our multi-class classification purpose. Every model was reiterated with different hyperparameters and finally the best hyperparameter that gave the highest accuracy and least difference between test and validation accuracy was picked up. The process was repeated for each of the models mentioned below

- **Decision Tree**: A flowchart-like structure in which each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label.

- **Random Forest:** ensemble learning method for classification, regression, and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees.

- **Logistic Regression:** A supervised learning classification algorithm used to predict the probability of a target variable. The nature of the target or dependent variable is dichotomous, which means there would be only two possible classes.

- **SVM:** A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model set of labeled training data for each category, they're able to categorize new text. So you're working on a text classification problem.

- **Naive Bayes:** Naïve Bayes is a simple learning algorithm that utilizes Bayes rule together with a strong assumption that the attributes are conditionally independent, given the class. While this independence assumption is often violated in practice, naïve Bayes nonetheless often delivers competitive classification accuracy.
- **KNN:** The k-nearest neighbors (KNN) algorithm is a simple, supervised machine learning algorithm that can be used to solve both classification and regression problems. It's easy to implement and understand but has a major drawback of becoming significantly slow as the size of that data in use grows.

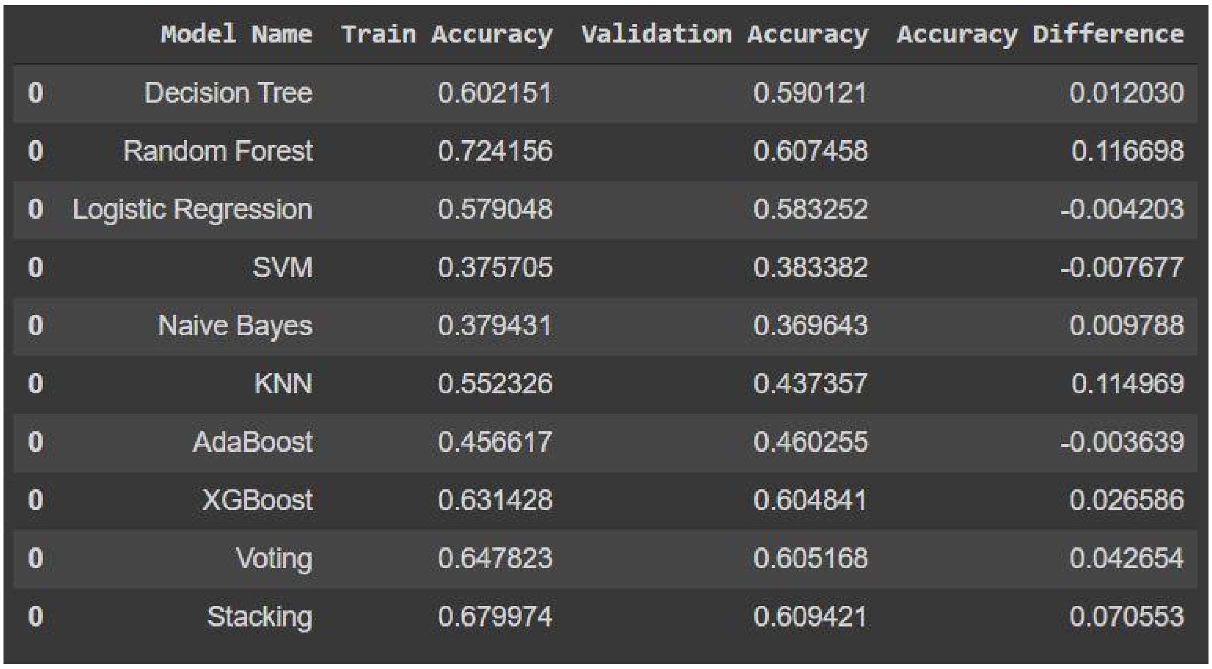
- **AdaBoost:** AdaBoost is best used to boost the performance of decision trees on binary classification problems. The basic concept behind Adaboost is to set the weights of classifiers and training the data sample in each iteration such that it ensures accurate predictions of unusual observations.

- **XGBoost**: In prediction problems involving unstructured data it can be used to solve regression, classification, ranking, and user-defined prediction problems.XGBoost is an ensemble tree method that applies the principle of boosting weak learners using the gradient descent architecture.

- **Voting** is an ensemble machine learning algorithm that involves making a prediction that is the average of multiple other regression models. The voting classifier isn’t an actual classifier but a wrapper for a set of different ones that are trained and evaluated in parallel in order to exploit the different peculiarities of each algorithm.

- **Stacking:** It involves combining the predictions from multiple machine learning models on the same dataset, like bagging and boosting. In stacking, the models are typically different and fit on the same dataset and a single model is used to learn how to best combine the predictions from the contributing models.

Summary of best models (corresponding to best hyperparameter in each category of model)



	Model Name	Train Accuracy	Validation Accuracy	Accuracy Difference
0	Decision Tree	0.602151	0.590121	0.012030
0	Random Forest	0.724156	0.607458	0.116698
0	Logistic Regression	0.579048	0.583252	-0.004203
0	SVM	0.375705	0.383382	-0.007677
0	Naive Bayes	0.379431	0.369643	0.009788
0	KNN	0.552326	0.437357	0.114969
0	AdaBoost	0.456617	0.460255	-0.003639
0	XGBoost	0.631428	0.604841	0.026586
0	Voting	0.647823	0.605168	0.042654
0	Stacking	0.679974	0.609421	0.070553

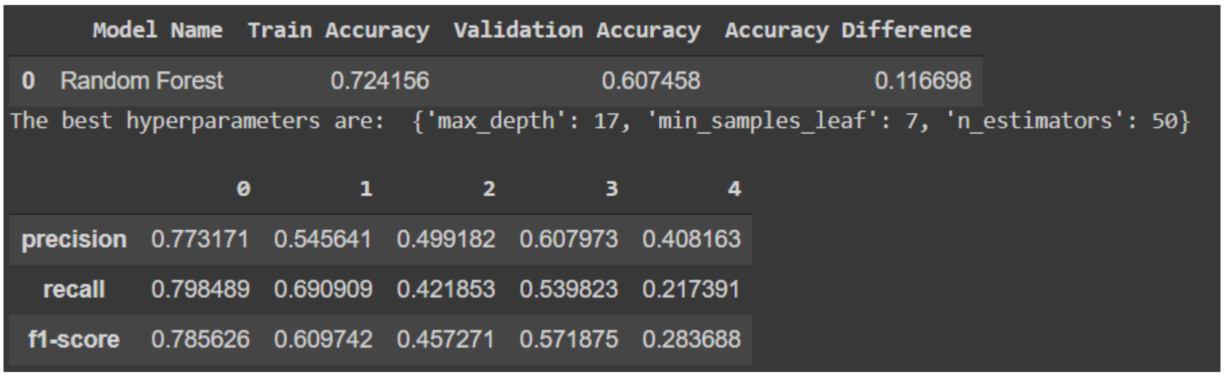
Final Model

Random Forest was finally picked with below mentioned hyperparameters. This is so because compared to others it is giving **highest Validation accuracy and lowest accuracy difference.** We have also considered **confusion matrices** of different models (it is discussed below).

```
For Random Forest :
 The best hyperparameters are:
{'max_depth': 17,
 'min_samples_leaf': 7,
'n_estimators': 50}
```

The results above are **after oversampling**

Train Accuracy - **0.72**
Validation accuracy - **0.60**



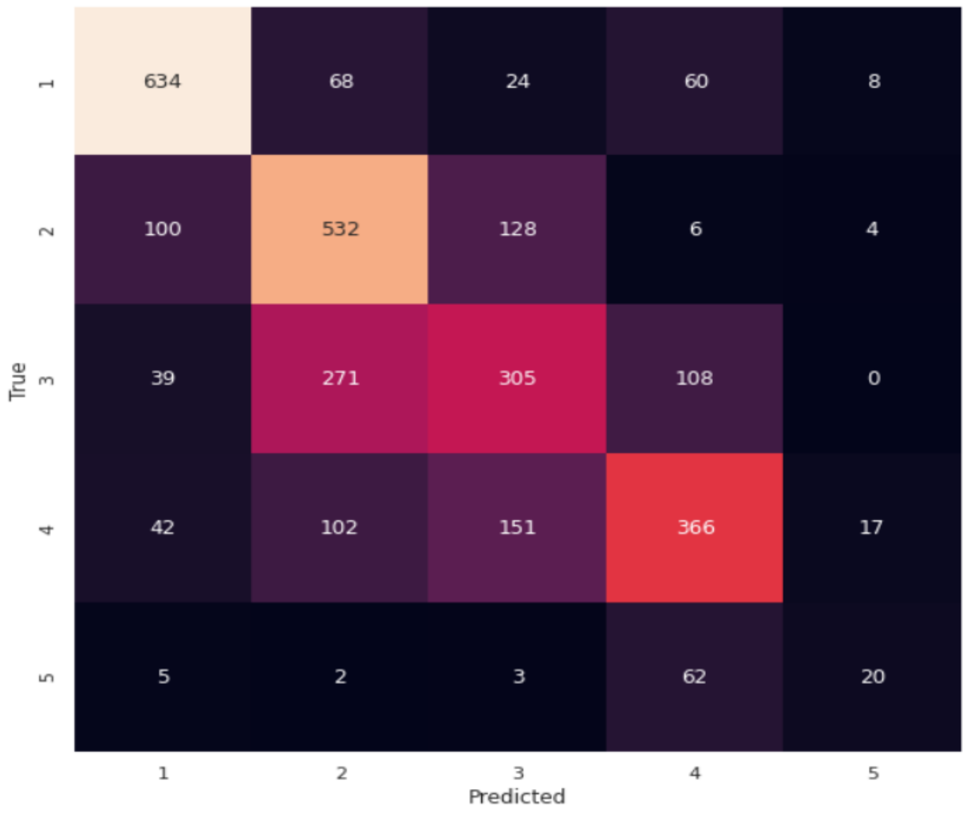
	Model Name	Train Accuracy	Validation Accuracy	Accuracy Difference
0	Random Forest	0.724156	0.607458	0.116698

```
The best hyperparameters are:  {'max_depth': 17, 'min_samples_leaf': 7, 'n_estimators': 50}
```

	0	1	2	3	4
precision	0.773171	0.545641	0.499182	0.607973	0.408163
recall	0.798489	0.690909	0.421853	0.539823	0.217391
f1-score	0.785626	0.609742	0.457271	0.571875	0.283688

Accuracy is the fraction of predictions our model got right.

Confusion Matrix

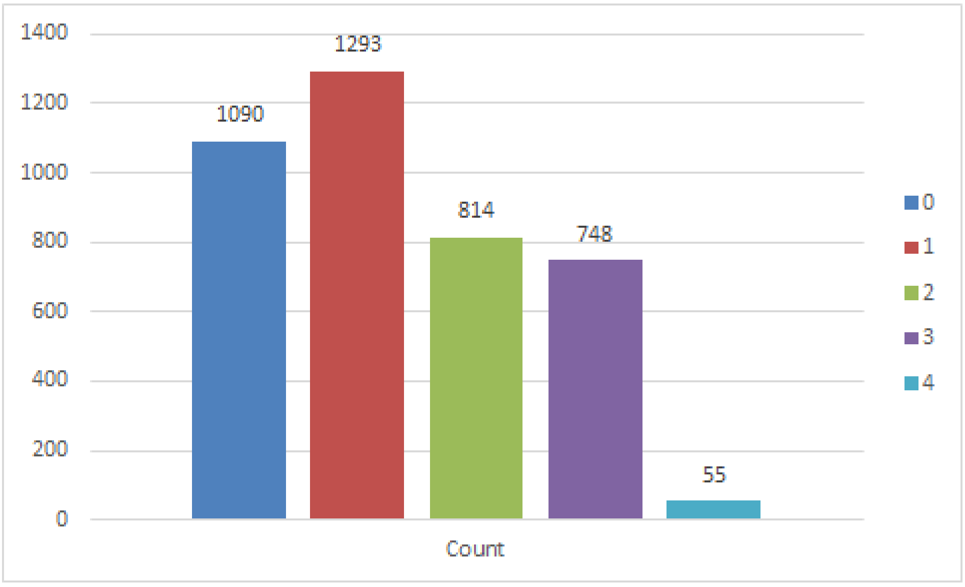


Confusion matrices are used to visualize important predictive analytics like recall, specificity, accuracy, and precision. **Confusion matrices** are useful because they give direct comparisons of values like True Positives, False Positives, True Negatives and False Negatives.

Here we can observe that most of the **right predictions are concentrated near the diagonal**, giving due confidence in the model. Moreover, concentration of wrongly predicted true values on the left hand/ lower-side of the matrix will save us from having loss on biding, thus maximising Revenue. This was also a reason to select this model.

Conclusion

We use our model to predict the popularity for 4000 test data song tracks and we obtain the following distribution of popularity.



Random Forest Model gave us the best result i.e. **72% accuracy** on Test-Set (after over-sampling.

Model can be further updated with Neural Network after increasing getting more records in train set and more songs with “high popularity” in the training set can reduce bias.

THANK YOU