INDEX

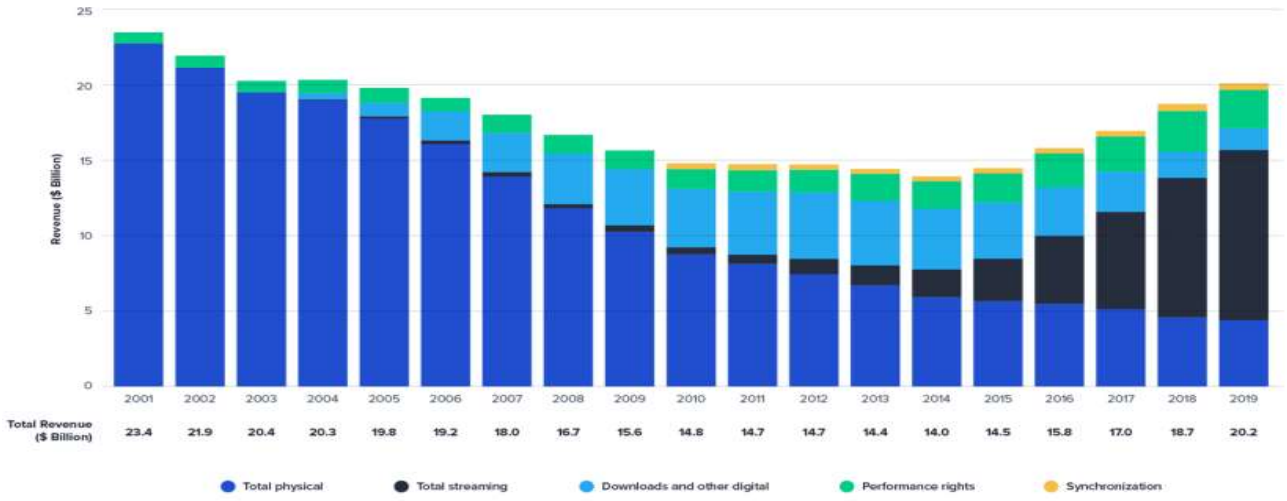
1.Introduction:

1.1 The Music Industry:

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 a 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 supervised learning algorithms over the time, the music industry has emphasized 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 its dataset for Kaggle. Over the past 3 years it has lead to understanding the importance of various features that give 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.

1.2 Music Royalties and Record Labels:

Music royalties are payments that go to recording artists, songwriters, composers, publishers, and other copyright holders for the right to use their intellectual property. Record labels market and distribute an artist's original work. They often have the master rights to a recorded song.Record labels generate income from mechanical and public performance royalties. They issue contracts that allow them to exploit the recordings in exchange for royalty payments over a set length of time. The artist then receives a flat rate or percentage of these record label royalties.

1.3 Plan of Implementation

The project can be broken down into 7 main steps which are as follows:

1. Understand the dataset.
2. Clean the data.
3. Analyze the candidate columns to be Features.
4. Process the features as required by the model/algorithm.
5. Train the model/algorithm on training data.
6. Test the model/algorithm on testing data.
7. Tune the model/algorithm for higher accuracy.

1.4 Task

Buying rights to all songs are not equally beneficial - the maximum revenue is generated from the most popular songs. Hence, it becomes important to predict beforehand the expected popularity of the song to maximize the profit. A major record label is planning to invest 10,000 (10k $ units) on 4000 songs. We are expected to help them in ensuring that they do not encounter losses with promotion and distribution of the track. Using the features given, we need to predict the songs as belonging to 5 classes of popularity categories: ‘Not popular’ , ‘average’ , ‘Most popular’ in such a way that the revenue is maximized.

2. ABSTRACT

We all know how music is addictive and how it has become a part of our busy lives. Stats say, on average, Americans spend 32.5 hours per week listening to music. That's roughly 4 hours 50 minutes a day. Everyone wants better songs/music to listen to everyday. There is a gap where sometimes people don't often get the songs they need or want. Estimating the success of a song before its release is an important music industry task. Current work uses audio descriptors to predict the success (popularity) of a song, where typical measures of success are chart measures such as peak position and streaming measures such as listener-count. Currently, a wide range of datasets is used for that purpose, but most of them are not publicly available; likewise, available datasets are restricted either in size, available features, or popularity measures. This substantially impedes the evaluation of the predictive power of a wide range of models. The dataset contains 12227 songs. To demonstrate the use of the datasets, we perform a regression and a classification (popular/unpopular) task on both datasets using a wide variety of models to predict song popularity for all target measures.

3.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.

3.1 Python Libraries

Our project somewhere will be revolving around Python and its libraries. The libraries used are described below:-

1. **NumPy:** In Python, NumPy is another library that is used for mathematical functions. The NumPy library is popular for array and matrix processing using a set of mathematical functions. This library is mostly used in machine learning computations.

   We have to import NumPy as follows:

   ```
   import numpy as np
   ```

2. **Pandas**: Pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualization, Aggregation, etc.

To import this library to our Python program, we have to write the following code before starting with any program.

```
import pandas as pd
```

From the above statement, we import the pandas library into the program, and we can access this library by referring to “pd” as an alias to the pandas whenever a property or method of pandas needs to be called in the program.

3. **Matplotlib:** Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

- Create publication quality.
- Make interactive figures that can zoom, pan, and update.
- Customize visual style and layout.
- Export to many file formats.
- Embedded in JupyterLab and Graphical User Interfaces.
- Use a rich array of third-party packages built on Matplotlib.

Once Matplotlib is installed, import it into your applications by adding the import module statement.

```
import matplotlib.pyplot as plt
```

4. Seaborn : Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data.

Seaborn offers the following functionalities:

1. Dataset oriented API to determine the relationship between variables.
2. Automatic estimation and plotting of linear regression plots.
3. It supports high-level abstractions for multi-plot grids.
4. Visualizing univariate and bivariate distribution.

To initialize the Seaborn library, the command used is:

```
import seaborn as sns
```

Using Seaborn we can plot wide varieties of plots like:

1. Distribution Plots
2. Pie Chart & Bar Chart
3. Scatter Plots
4. Pair Plots
5. Heat maps

5. Scikit-Learn: It is a Python library to work with complex data. Scikit Learn is an open-source library that supports machine learning. It supports variously supervised and unsupervised algorithms like linear regression, classification, clustering, etc. This library works in association with NumPy and SciPy.

There are a lot of changes being made in this library. One modification is the cross-validation feature, providing the ability to use more than one metric. Lots of training methods like logistics regression and nearest neighbors have received some little improvements.

Features Of Scikit-Learn

Cross-validation: There are various methods to check the accuracy of supervised models on unseen data.

Unsupervised learning algorithms: Again there is a large spread of algorithms in the offering – starting from clustering, factor analysis, principal component analysis to unsupervised neural networks.

Feature extraction: Useful for extracting features from images and text (e.g. Bag of words).

Model evaluation: Fitting a model to some data does not entail that it will predict well on unseen data. This needs to be directly evaluated. We have just seen the train_test_split helper that splits a dataset into train and test sets, but sci-kit-learn provides many other tools for model evaluation, in particular for cross-validation

Pipelines: chaining pre-processors and estimators: Transformers and estimators (predictors) can be combined together into a single unifying object: a Pipeline. The pipeline offers the same API as a regular estimator: it can be fitted and used for prediction with the fit and predict.

Related Work

The problem of predicting popularity is one that has been heavily researched. Salganik, Dodds, and Watts conducted an experimental study on popularity that focused heavily on the social influence of popularity. They found that the quality of a song only partially influences whether or not a song becomes popular, and that social influence plays an extremely large role.

Therefore, our project aims to use both acoustic features and metadata features to create a more accurate prediction model. The work by Koenignstein, Shavitt, and Zilberman, which predicts billboard success based on peer-to-peer networks, potentially captures this social influence on song popularity. This group was extremely thorough with their work and used multiple regression and classification algorithms for their predictions. Bertin-Mahieux et al. found that machine learning techniques can be used to apply labels to songs based on

acoustic features. They created a model for predicting social tags from acoustic features on a large music database by using AdaBoost and FilterBoost. While this group was extremely thorough with considering the possible models to use and sanitizing their features, using SVMs instead of AdaBoost with FilterBoost may have been a better option. However, Pachet and Roy investigated the problem of making predictions of song popularity and made the blunt claim that the popularity of a song cannot be learnt by using state-of-the-art machine learning [6]. In order to test the effectiveness of current machine learning algorithms, they test the improvement of their classification models to a generic random classifier.

Similarly to our work, Pachet and Roy consider both acoustic features and metadata; however, the study deals with an extremely large number of features (over 600) but does not mention any type of feature selection algorithm. As a result it is extremely likely that their model was subjected to overfitting. Pacet and Roy also considered features commonly used for music analysis which potentially could have affected the success of their results. However, Ni et al have responded to the above definitive claim with more optimistic results on music popularity prediction, using a Shifting Perceptron algorithm to classify the top 5 hits from the top 30-40 hits (a slightly different problem from the aforementioned study. However, this study also uses more novel audio features which is a likely factor in their improved results.

4. Exploratory Data Analysis

4.1 Correlation inspection using heatmap

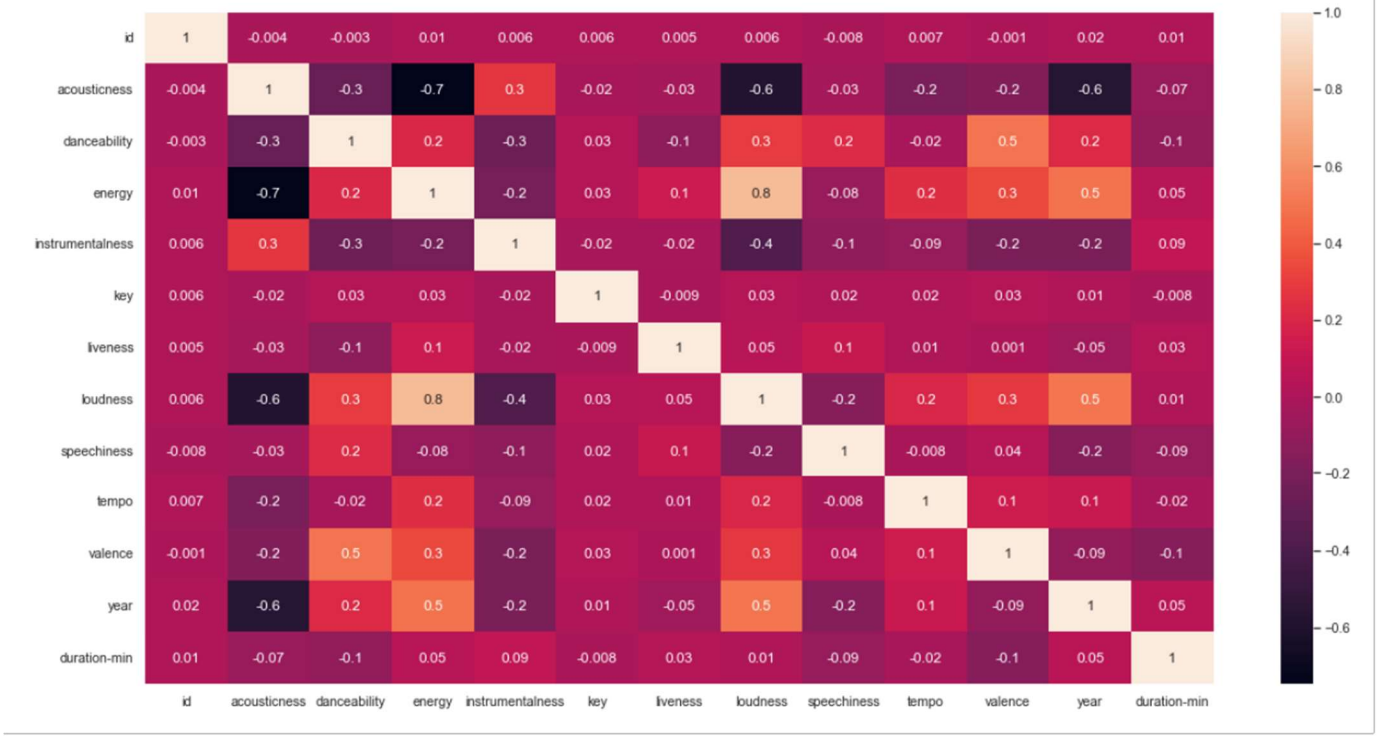
Heatmaps visualize the data in a 2-dimensional format in the form of colored maps. Heatmaps can describe the density or intensity of variables, visualize patterns, variance, and even anomalies. Heatmaps show relationships between variables. These variables are plotted on both axes. We look for patterns in the cell by noticing the color change. It only accepts numeric data and plots it on the grid, displaying different data values by varying color intensity.

How to Read a Correlation Matrix

1. -1 indicates a perfectly negative linear correlation between two variables.
2. 0 indicates no linear correlation between two variables.
3. 1 indicates a perfectly positive linear correlation between two variables.

```
plt.figure(figsize=(20,10))
sns.heatmap(df.corr(), annot = True, fmt='.1g',square=False)
```

```
<AxesSubplot:>
```



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.

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.

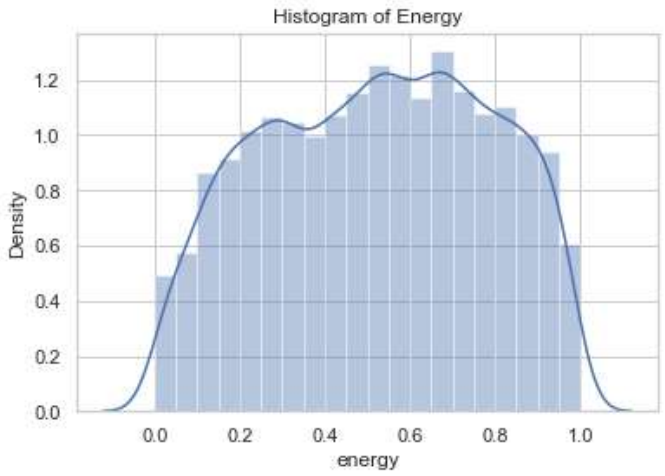
i: Histogram

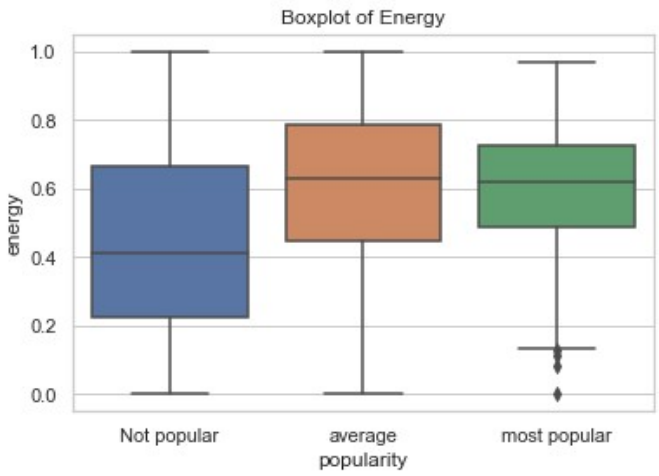
A histogram is a graphical representation that organizes a group of data points into user-specified ranges. Similar in appearance to a bar graph, the histogram condenses a data series into an easily interpreted visual by taking many data points and grouping them into logical ranges or bins.

ii: Boxplot

A box plot (or box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates comparisons between variables or across levels of a categorical variable. The box shows the quartiles of the dataset while the whiskers extend to show the rest of the distribution, except for points that are determined to be “outliers” using a method that is a function of the inter-quartile range.

1. Energy





1: Energy is a continuous variable which ranges from 0-1.

2: The distribution looks like above. Maximum number of values ranging from 0.6-0.8.

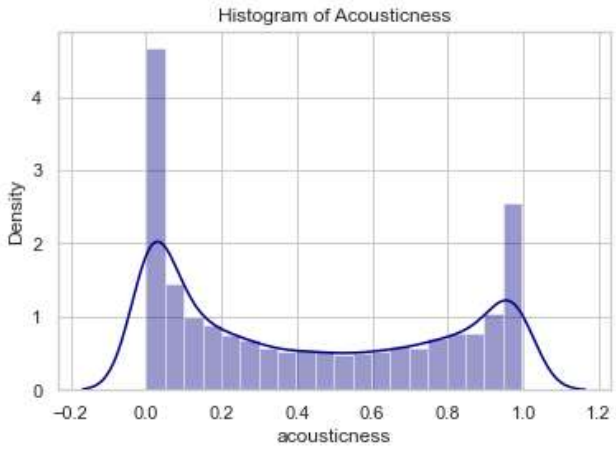
3: It is a left skewed distribution.

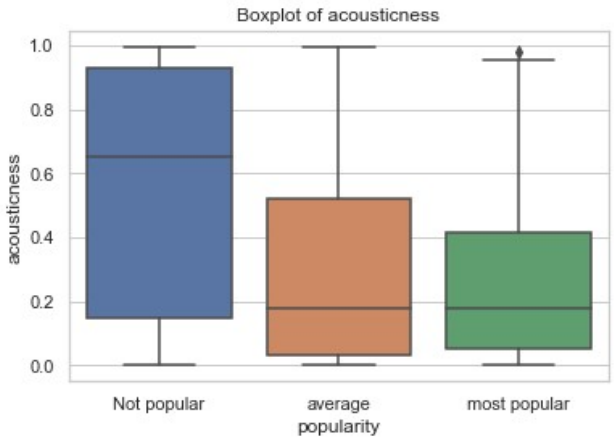
4: The mean of 'energy' is 0.6

5: The first and third quartiles of the energy data are 0.4 and 0.8 respectively.

6: By this, we can say that we have more energetic songs in the data. (As mentioned above, higher the energy value, higher the energy of the song)

2. Accoustiness



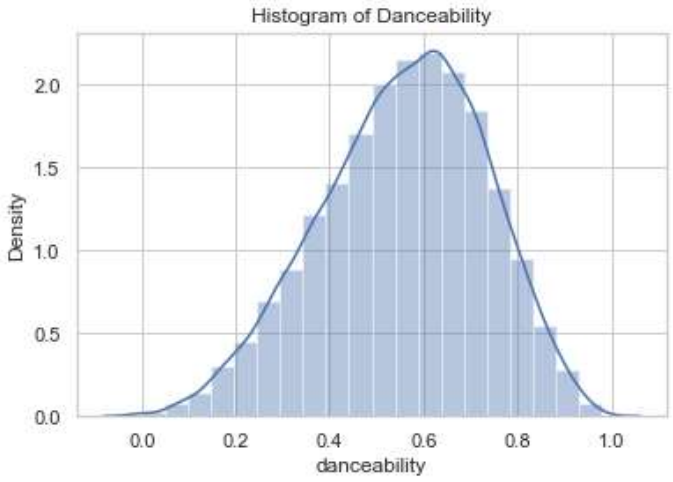


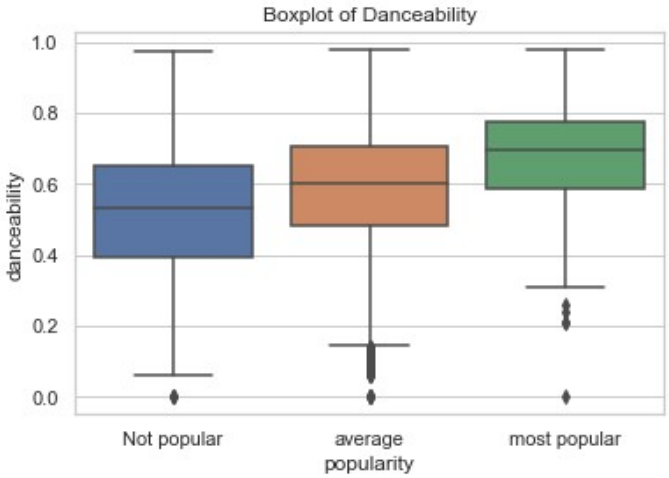
1: The mean of Acousticness lies between 0.3-0.4

2: The first and third quadrant are respectively 0.05 and 0.8

3: There are no outliers in the acousticness.

3. Danceability





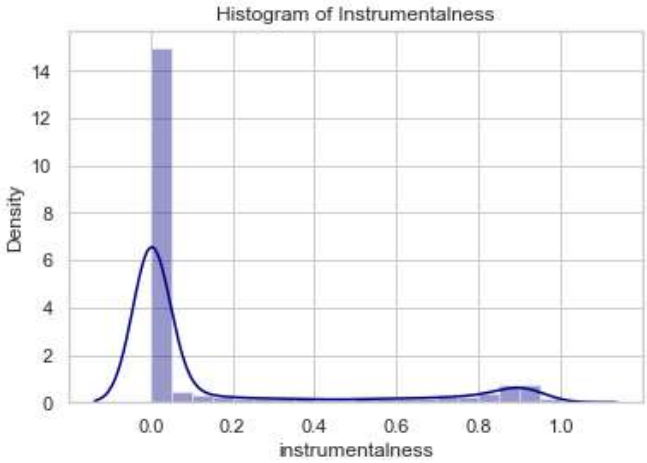
1. The distribution almost looks a normal distribution which is a good sign

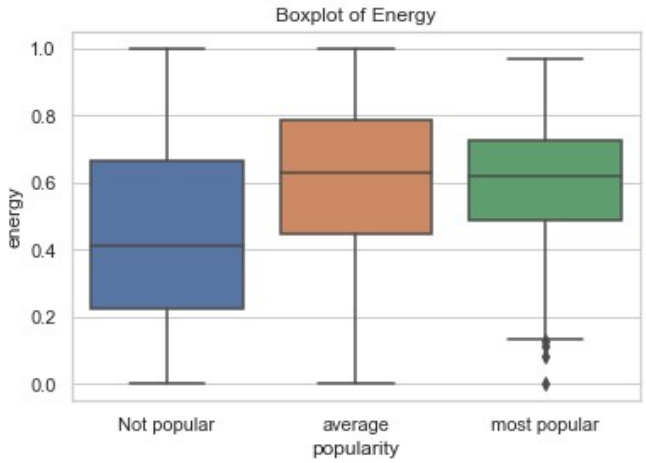
   The mean of Danceability lies between 0.55-0.6.

2. There are no outliers in danceability.

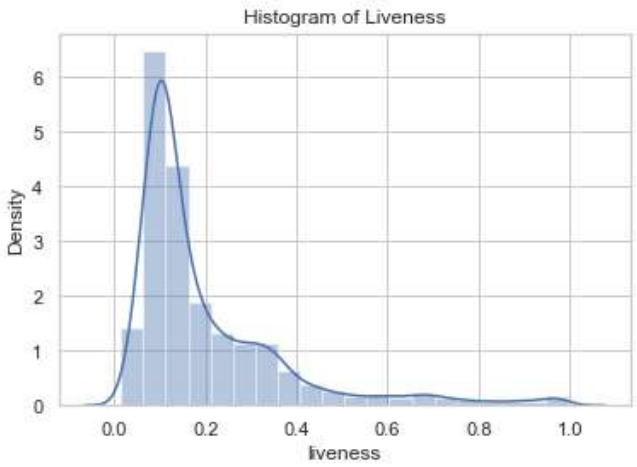
4. Instrumentalness

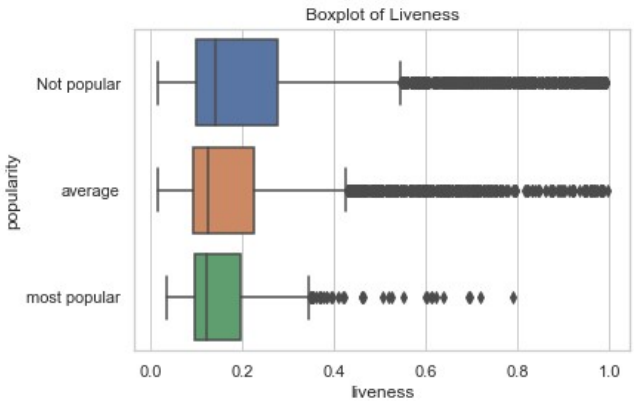
The box plot of instrumentalness also looks in a similar manner as a complete right skewed distribution.





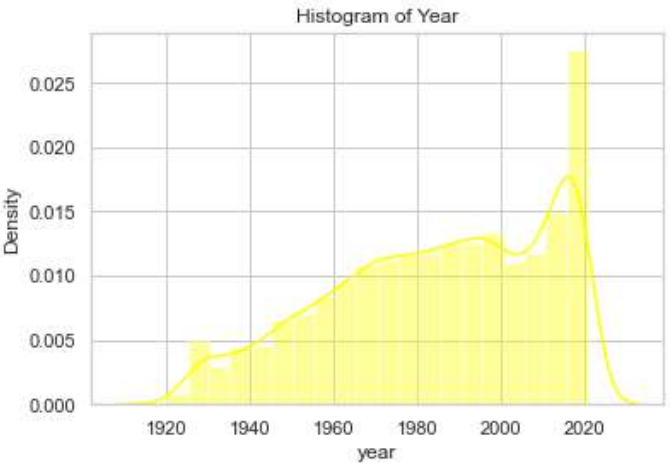
5. Liveness

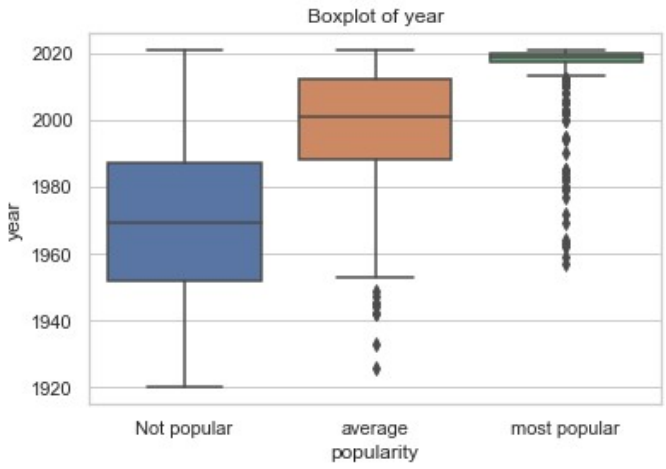




1. It has a Right skewed distribution
2. As per the definition, if the value is more than 0.8, then it was said to be recorded live.
3. Seems like there aren't much live recorded music tracks in the database
4. The boxplot shows that the mean of liveness is in between 0.1 and 0.2
5. It also says that any value greater than 0.5 is an outlier.
6. This means that there are very few data points which are recorded live and they are outliers.

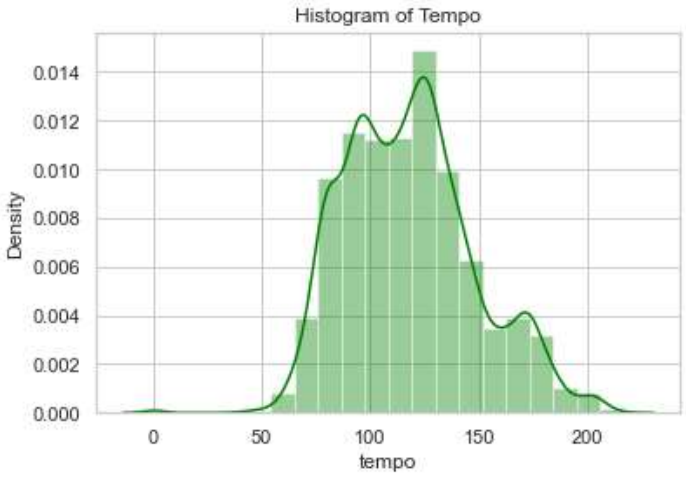
6. Year

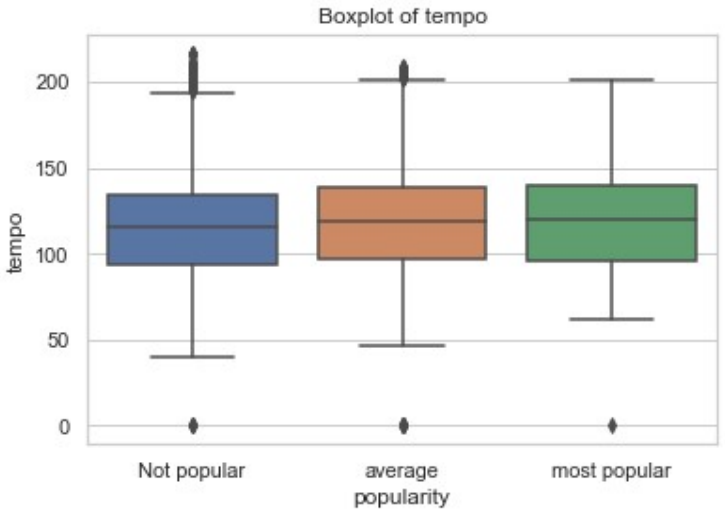




1. It has a Right skewed distribution
2. As per the definition, most of the song was popular between 2015 and 2020.

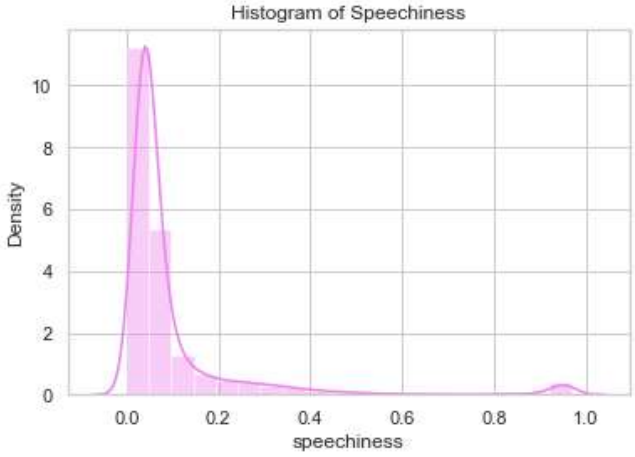
7. Tempo

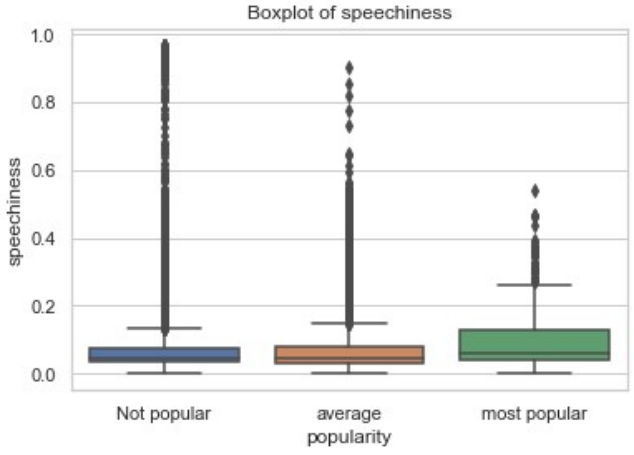




1. It is a continuous variable which ranges from 30 to 243
2. It is almost a bimodal distribution with values from 70-100 and 12-140 as modes.
3. The mean of Tempo lies between 110 - 120
4. There are outliers to the right of the distribution. All the values after 205 are outliers.

8. Speechness





1. Right skewed Distribution
2. As the definition says, if the speechiness is less than 0.33, then the track doesn't have any vocals. In the data, we have more tracks which don't have any vocals
3. We see some outliers at 0.9 speechiness which are probably rap songs

5. Data Preprocessing

Data preprocessing is a technique which is used to transform the raw data in a useful and efficient format.

5.1: Steps in Data Preprocessing:-

1. **Data cleaning**

   Data cleaning means filling missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data.

2. **Data integration** –

   It is a technique to merge data from multiple sources into a coherent data store, such as a data warehouse.

3. **Data Reduction –**

   It is a technique used to reduce the data size by aggregating, eliminating redundant features or Clustering for instance.

4. **Data Transformation –**

   It means data are transformed or consolidated into forms appropriate for ML model training, such as normalization, may be applied where data are scaled to fall within a smaller range like 0.0 to 1.0.

5. **Data Discretization –**

   It is a technique that transforms numeric data by mapping values to interval or concept labels.

5.2 Encoding of Categorical Features

Encoding is a technique of converting categorical variables into numerical values so that it could be easily fitted to a machine learning model.

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

1. popularity_map = {'Not popular':0,'average':1,'most popular':2}
2. explicit_map = {'No':0,'Yes':1}
3. mode_map = {'Minor':0,'Major':1}

5.3 One hot encoding

One hot encoding is one method of converting data to prepare it for an algorithm and get a better prediction. With one-hot, we convert each categorical value into a new categorical column and assign a binary value of 1 or 0 to those columns.

```
#Mapping the different labels to different integers
mode_map = {'Minor':0,'Major':1,}
df['mode'] = df['mode'].map(mode_map)
```

Here we have converted the labels into binary in order to deploy in model

```
#Mapping the different labels to different integers
popularity_map = {'Not popular':0,'average':1,'most popular':2}
df['popularity'] = df['popularity'].map(popularity_map)
df.head()
```

```
#Mapping the different labels to different integers
explicit_map = {'No':0,'Yes':1,}
df['explicit'] = df['explicit'].map(explicit_map)
```

6. Feature Selection

Feature Selection is the method of reducing the input variable to our model by using only relevant data and getting rid of noise in data

It is the process of automatically choosing relevant features for your machine learning model based on the type of problem you are trying to solve. We do this by including or excluding important features without changing them. It helps in cutting down the noise in our data and reducing the size of our input data.

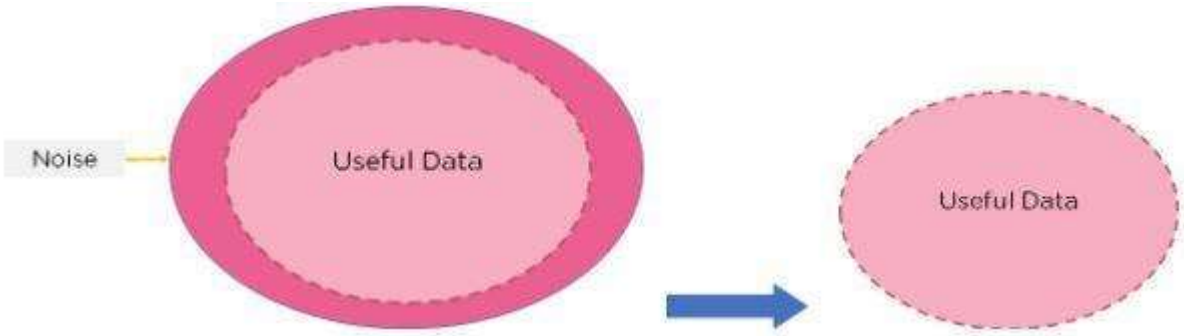


Figure 6.1: Feature Selection

6.1 What is Feature Selection?

Feature Selection is the method of reducing the input variable to your model by using only relevant data and getting rid of noise in data.

It is the process of automatically choosing relevant features for your machine learning model based on the type of problem you are trying to solve. We do this by including or excluding important features without changing them. It helps in cutting down the noise in our data and reducing the size of our input data.

6.2 Why Feature Selection?

Machine learning models follow a simple rule: whatever goes in, comes out. If we put garbage into our model, we can expect the output to be garbage too. In this case, garbage refers to noise in our data.

To train a model, we collect enormous quantities of data to help the machine learn better. Usually, a good portion of the data collected is noise, while some of the columns of our dataset might not contribute significantly to the performance of our model. Further, having a lot of data can slow down the training process and cause the model to be slower. The model may also learn from this irrelevant data and be inaccurate.

Feature selection is what separates good data scientists from the rest. Given the same model and computational facilities, why do some people win in competitions with faster and more accurate models? The answer is Feature Selection. Apart from choosing the right model for our data, we need to choose the right data to put in our model.

How to select features and what are Benefits of performing feature selection before modeling your data?

Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise.

Improves Accuracy: Less misleading data means modeling accuracy improves.

Reduces Training Time: fewer data points reduce algorithm complexity and algorithms train faster.

Feature Selection Methods:

These are the 3 Feature selection techniques that are easy to use and also gives good results.

1. Univariate Selection - statistical tests can be used to select those features that have the strongest relationship with the output variable.

The scikit-learn library provides the SelectKBest class that can be used with a suite of different statistical tests to select a specific number of features. ex = chi-squared (chi^2) statistical test .

2. Feature Importance - WE can get the feature importance of each feature of your dataset by using the feature importance property of the model.

Feature importance gives you a score for each feature of your data, the higher the score more important or relevant is the feature towards your output variable.Correlation Matrix with Heatmap - As shown above

NOTE: We had tried building some other features as well, but they did not add significant value to our model and so we have not discussed them here.

Selecting features based on correlation matrix

X=df[["acousticness","danceability","energy","instrumentalness","liveness","loudness","speechiness","tempo","valence","year","duration-min","explicit","key","mode"]]
Y = df["popularity"]

Splitting Data into Training set and Testing Set

train_test_split of Sklearn

```
]: ##Split the data to train and test
   X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

6.3 Data Preprocessing: Best practices

Here's a short recap of everything we've learnt about data preprocessing:

I. The first step in Data Preprocessing is to understand our data. Just looking at our dataset can give us an intuition of what things we need to focus on.

```
df.head()
```

	id	acousticness	danceability	energy	explicit	instrumentalness	key	liveness	loudness	mode	release_date	speechiness	tempo	valence	year	du
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	
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	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	
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	
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	

```
df.describe()
```

	id	acousticness	danceability	energy	instrumentalness	key	liveness	loudness	speechiness	tempo	
count	12227.000000	12227.000000	12227.000000	12227.000000	12227.000000	12227.000000	12227.000000	12227.000000	12227.000000	12227.000000	1222
mean	8094.034350	0.430578	0.556353	0.522129	0.149321	5.205202	0.201365	-10.668687	0.097680	118.167495	
std	4690.929822	0.366893	0.175373	0.262482	0.297954	3.526954	0.173987	5.506888	0.155895	30.200064	
min	1.000000	0.000001	0.000000	0.000020	0.000000	0.000000	0.014700	-43.738000	0.000000	0.000000	
25%	4026.000000	0.058950	0.438000	0.303000	0.000000	2.000000	0.096200	-13.656000	0.034700	95.050500	
50%	8093.000000	0.354000	0.569000	0.534000	0.000115	5.000000	0.132000	-9.584000	0.045600	116.915000	
75%	12180.000000	0.805000	0.685000	0.739000	0.055650	8.000000	0.252000	-6.571500	0.078900	136.108500	
max	16227.000000	0.996000	0.980000	1.000000	1.000000	11.000000	0.997000	1.006000	0.968000	216.843000	

Seems like there are no missing values in the dataset. Because, there no NaN values as well as all the minimum values and maximum values are according to the standards. But there are some fields that needs to be clean Let's look at the columns one by one and analyse them to see if anything needs to be imputed or cleaned

ii. Use statistical methods or pre-built libraries that help us to visualize the dataset and give a clear image of how your data looks in terms of class distribution. As already shown above.

iii. Summarize your data in terms of the number of duplicates, missing values, and outliers present in the data.

iv. Drop the fields you think have no use for the modeling or are closely related to other attributes. Dimensionality reduction is one of the very important aspects of Data Preprocessing.

v. Do some feature engineering and figure out which attributes contribute most towards model training.

Let's look at if any missing values are present

```
: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12227 entries, 0 to 12226
Data columns (total 17 columns):
 #   Column            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
dtypes: float64(10), int64(3), object(4)
memory usage: 1.6+ MB
```

We see that there are no NaN values in our dataset. There are 12227 entries for all the columns. Trying with describe() function

6.4 Dropped Features

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

```
# Dropping unecessary features

data = df.drop(labels=['id', 'release_date'], axis=1)
data.shape
```

```
(12227, 15)
```

```
#Selecting features based on correlation matrix
X = df[["acousticness","danceability","energy","instrumentalness","liveness","loudness","speechiness","tempo","valence","year","
Y = df["popularity"]
```

7. Model Architecture

At a glance, our problem looks like one pertaining to classification (since there are 3 labeled classes of popularity). But the values for popularity are ordered from high to low and hence, we tried a regression model as well. However, the performance of the regression model was pretty poor and so we decided to stick to classification models only.

Classification is a subset of supervised learning and is the process of grouping objects into a fixed number of preset classes or labels. There are a wide variety of classification models available and are used in a variety of fields such as biometric identification, document classification, text and image recognition, etc. We will be using these models to build an efficient and robust song popularity prediction model.

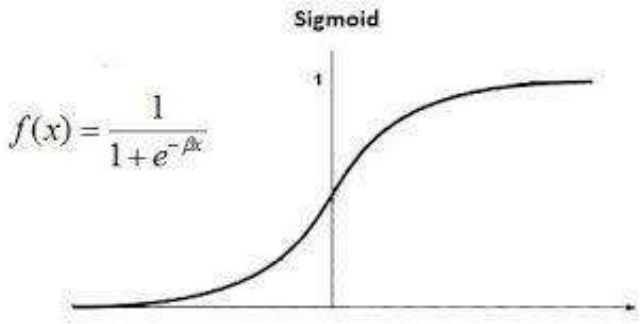
We have used some of the commonly used classification models which are reliable for giving high quality output and have their own unique mechanism.
For our required 5-class classification problem, we have used: Bagging models like Random Forest Classifier; Boosting models like XGBoost and LightGBM as well as simple classifiers like Support Vector Classifier and Decision Trees.

The dataset was severely imbalanced with songs belonging to “Very High” popularity constituting less than 4% of the entire data. We have used SMOTE to upsample the minority class and ensure a comparable proportion of all 5 classes in hope of better prediction results.

We have tried to compare the performances of each of these models on the validation set and then stacked the best performers (using Voting Classifier) to make the model robust and less prone to overfitting on unseen data.

7.1 Logistic Regression:

Logistic regression is a process of modeling the probability of a discrete outcome given an input variable. The most common logistic regression models a binary outcome; something that can take two values such as true/false, yes/no, and so on. Multinomial logistic regression can model scenarios where there are more than two possible discrete outcomes. Logistic regression is a useful analysis method for classification problems, where you are trying to determine if a new sample fits best into a category. As aspects of cyber security are classification problems, such as attack detection, logistic regression is a useful analytic technique.



Why Logistic Regression:|

logistic regression is used to estimate the relationship between a dependent variable and one or more independent variables, but it is used to make a prediction about a categorical variable versus a continuous one.

Implementation:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train,y_train)
```

```
C:\Users\prem kumar\anaconda3\envs\pytho\lib\site-packages\sklearn\linear_model\_logist
led to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
LogisticRegression()
```



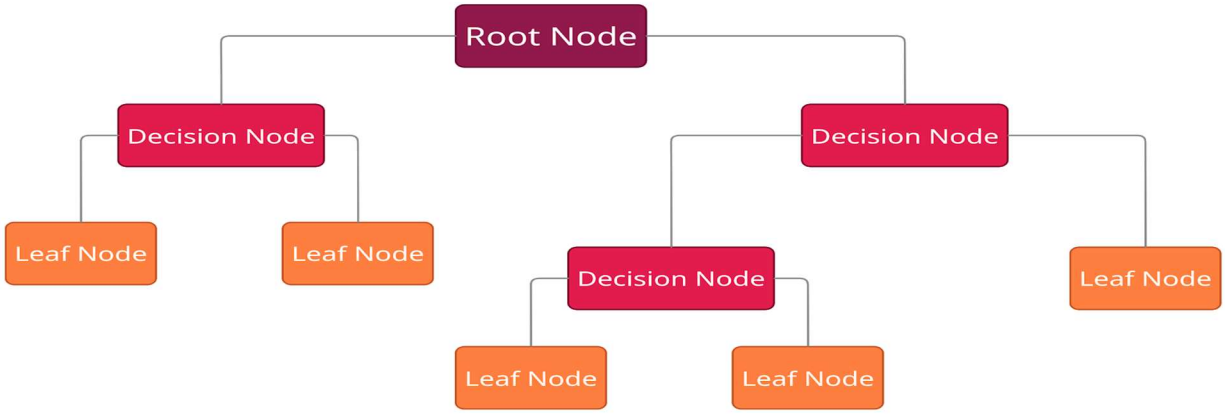
```
classifier.score(X_test,y_test)
```

```
0.6631234668847097
```

7.2 Decision Trees:

Theory:

Decision tree is a predictive model which uses a tree based algorithm..The decision trees are built by splitting the source set, constituting the root node of the tree, into subsets—which constitute the successor children. The splitting is based on a set of splitting rules following a greedy approach.This process is repeated on each derived subset in a recursive manner until the subset at a node has all the same values of the target variable,



or when splitting no longer adds value to the predictions

Figure : 7.1 Decision Tree

Why Decision Trees?

Decision trees have a powerful representation and are one of the best performers on a large, disjunctive hypothesis space. They also work well when the data is noisy, the target function is discrete-valued and the instances are describable by a fixed set of attributes and their values. These reasons make Decision Trees a sensible choice as a starting model.

How does the Decision Tree algorithm work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of the root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and moves further. It continues the process until it reaches the leaf node of the tree. The complete algorithm can be better divided into the following steps:

- **Step-1:** Begin the tree with the root node, says S, which contains the complete dataset.
- **Step-2:** Find the best attribute in the dataset using the **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contain possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step

  Continue this process until a stage is reached where you cannot further classify the nodes and call the final node as a leaf node.

Implementation:

```
[45]: from sklearn import tree
      classifier = tree.DecisionTreeClassifier()
      classifier = classifier.fit(X_train,y_train)
      print(classifier)

      DecisionTreeClassifier()

[46]: classifier.score(X_test,y_test)

t[46]: 0.6831561733442355
```

7.3 Random Forest Classifier:

Theory:

Random forest is a bagging model which consists of a large number of individual decision trees generated parallely which are the base learners and operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction.

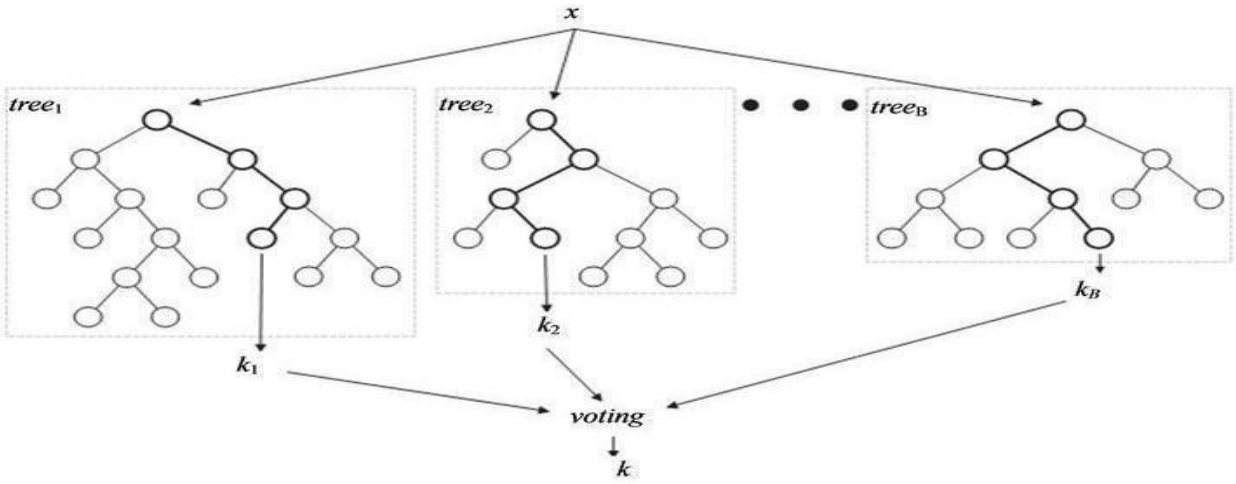


Figure : 7.2 Random forest

Why Random Forests?

Being a bagging model, random forest allows each individual tree to randomly sample from the dataset with replacement, resulting in different trees trained on different sets of data as well as different subset of features. Thus, independent trees are grown, which help to simultaneously improve accuracy and reduce variance (and thus overfitting)

How does the Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision trees, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority of votes.

Implementation:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators= 944,
 min_samples_split= 10,
 min_samples_leaf= 2,
 max_features= 'sqrt',
 max_depth= 10,
 bootstrap= True,random_state = 14)
classifier.fit(X_train,y_train)
```

```
RandomForestClassifier(max_depth=10, max_features='sqrt', min_samples_leaf=2,
                       min_samples_split=10, n_estimators=944, random_state=14)
```

```
classifier.score(X_test,y_test)
```

```
0.7616516762060507
```

7.4 Gradient Boosting Algorithm:

Gradient boosting is a widely used technique in machine learning. Applied to decision trees, it also creates ensembles. However, the core difference between the classical forests lies in the training process of gradient boosting trees.

The goal is to train multiple trees one stage at a time. In each, we construct a single tree to correct the errors of the previously fitted ones

Advantage

Gradient boosting trees can be more accurate than random forests. Because we train them to correct each other's errors, they're capable of capturing complex patterns in the data. However, if the data are noisy, the boosted trees may overfit and start modeling the noise.

why gradient boosting

Gradient boosting trees can be more accurate than random forests. Because we train them to correct each other's errors, they're capable of capturing complex patterns in the data. However, if the data are noisy, the boosted trees may overfit and start modeling the noise.

Implementation:

```
from sklearn.ensemble import GradientBoostingClassifier
classifier = GradientBoostingClassifier()
classifier.fit(X_train,y_train)
```

```
GradientBoostingClassifier()
```

```
classifier.score(X_test,y_test)
```

```
0.7567457072771873
```

7.5 The Main Differences with Random Forests

There are two main differences between the gradient boosting trees and the random forests. We train the former sequentially, one tree at a time, each to correct the errors of the previous ones. In contrast, we construct the trees in a random forest independently. Because of this, we can train a forest in parallel but not the gradient-boosting trees.

The other principal difference is in how they output decisions. Since the trees in a random forest are independent, they can determine their outputs in any order. Then, we aggregate the individual predictions into a collective one: the majority class in classification problems or the average value in regression. On the other hand, the gradient boosting trees run in a fixed order, and that sequence cannot change. For that reason, they admit only sequential evaluation.

8. ENSEMBLE METHODS:

As we have seen above, the models perform more or less similarly if we are judging based on the F1 score or accuracy but do give different values. Since we are primarily concerned with that, it makes sense to try ensemble learning techniques. Also, we chose to exclude Logistic Regression since it was performing the worst.

We have used the inbuilt Voting Classifier model, giving more weightage to Random Forest, as it was the best performing individual model. Also, we have made use of Random Forest's class weight parameter , intentionally biasing the model towards the top 2 classes because of two reasons:

1. To take care of the imbalance in the dataset.
2. Keeping in mind the fact that a higher bid can still fetch us the song (though at a higher price.

Revenue Constraint:

At this point of time, it becomes important to discuss a bit about the revenue being generated, because our main goal is maximizing revenue, and not accuracy. As is the case usually, we formed our validation set by splitting our given training set into 2 parts of 80% and 20%. For a particular validation set (random_state=123), a perfect prediction on each and every observation would require us to bid a total of $7,341 units suitably. Now, we have been asked to bid a total of $10,000 units on 4,000 songs. Taking into account this 2.5 multiplier factor and our validation set having 2446 observations, we get a threshold of $6115 (2446*2.5), which is around 83% of the total required money. Thus, inherently, we can't get a very high accuracy. Because of the same reason, it's impossible to get the max revenue with that amount which is 6115*2 = 12230 (since we get double the revenue on each song as is the bid value). So far, our revenue values have been around $8,500 units which is around 70% of the maximum amount (which, as we discussed, is impossible to achieve). Therefore, we can say that our model is actually performing better than the suggested accuracy, as far as the goal of maximizing revenue goes.

9. System Design

Design is a meaningful engineering representation of something that is to be built. It is the most crucial phase in the development of a system. Software design is a process through which the requirements are translated into a representation of software. Design is a place where design is fostered in software Engineering. Based on the user requirements and the detailed analysis of the existing system, the new system must be designed. This is the phase of system designing. Design Is the perfect way to accurately translate a customer's requirement in the finished software product. Design creates a representation or model, provides details about software data structure,architecture, interfaces and components that are necessary to implement a system. The logical system design arrived at as a result of systems analysis is converted into physical system design.

5.1 System development methodology

System development method is a process through which a product will get completed or a product gets rid from any problem. Software development process is described as a number of phases, procedures and steps that give the complete software. It follows a series of steps which is used for product progress. The development method followed in this project is the waterfall model.

5.1.1 Model phases

The waterfall model is a successive programming improvement process, in which advance is seen as streaming relentlessly downwards (like a waterfall) through the periods of Requirement start, Analysis, Design, Implementation, Testing and upkeep.

Prerequisite Analysis: This stage is worried about gathering the necessity of the framework. This procedure includes producing record and necessity surveys.

Framework Design: Keeping the prerequisites at the top of the priority list the framework details are made an interpretation of into a product representation. In this stage the fashioner emphasizes calculation, information structure, programming design and so on.

Coding: In this stage developer begins his coding with a specific end goal to give a full portrayal of At the end of the day framework particulars are just changed over into machine coherent register code.

Usage: The execution stage includes the genuine coding or programming of the product. The yield of this stage is regularly the library, executables, client manuals and extra programming documentation.

Testing: In this stage all projects (models) are coordinated and tried to guarantee that the complete framework meets the product prerequisites. The testing is worried with check and approval.

Support: The upkeep stage is the longest stage in which the product is upgraded to satisfy the changing client need, adjust to suit change in the outside environment, right mistakes and oversights beforehand undetected in the testing stage, improve the proficiency of the product.

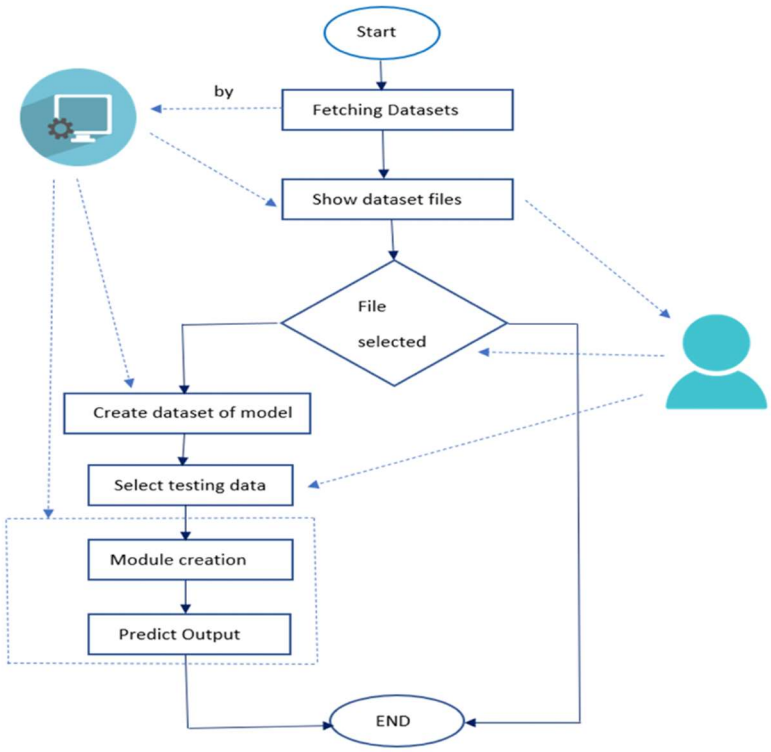


Figure -1: System architecture diagram

Hardware Requirement

We find that the following list represents the minimum requirements needed to install Python and associated applications:

- Modern Operating System:
- Windows 7 or 10
- Mac OS X 10.11 or higher, 64-bit
- Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)
- x86 64-bit CPU (Intel / AMD architecture)
- 4 GB RAM
- 5 GB free disk space

Recommended System Requirements

- Processors:Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAMIntel® Xeon® processor E5-2698 v3 at 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64 GB of DRAMIntel® Xeon Phi™ processor 7210 at 1.30 GHz (1 socket, 64 cores, 4 threads per core), 32 GB of DRAM, 16 GB of MCDRAM (flat mode enabled)
- Disk space: 2 to 3 GB
- Operating systems: Windows® 10, macOS*, and Linux*

Software Requirement

Python 3.5 or higher in Jupyter Notebook is used for Data Pre-processing, EDA, Model Training and Prediction

10. Conclusion

Above we presented the gradient-boosting trees and compared them to random forests. While both are ensemble models, they build on different ideas. We can fit and run the former ones only sequentially, whereas the trees in a random forest allow for parallel training and execution.

Going into this endeavor, we were uncertain if it is even possible to predict, better than random, if a song will be popular or not. After testing our model on new songs pulled from Spotify, we observe that it is significantly simpler to correctly predict a bad song rather than a hit. It may have been easier to predict non hit songs because our data was skewed, with only 1,200 hit songs. Moving forward, we would like to explore how additional features such as artist location or release date can influence a song's popularity. In addition, we may consider using the full dataset to see if we can improve our models. Another alternative is to use Spotify API to collect our own data.

Here are the results:

- Here we have compared all the algorithms by importing pycaret which is a library used to preprocess the data and compared the respective algorithm simultaneously on its own.
- Here we can see that the Gradient Boosting Classifier Algorithm has the highest accuracy i.e 76%.

	Model	Accuracy
gbc	Gradient Boosting Classifier	0.7644
rf	Random Forest Classifier	0.7635
lightgbm	Light Gradient Boosting Machine	0.7629
et	Extra Trees Classifier	0.7537
ada	Ada Boost Classifier	0.7369
ridge	Ridge Classifier	0.7303
lda	Linear Discriminant Analysis	0.7263
dt	Decision Tree Classifier	0.6935
lr	Logistic Regression	0.6823
nb	Naive Bayes	0.6665
knn	K Neighbors Classifier	0.6556
dummy	Dummy Classifier	0.5224
svm	SVM - Linear Kernel	0.4954
qda	Quadratic Discriminant Analysis	0.4575

11. Reference

1. https://medium.com/@albert.w.berger/what-makes-a-song-popular-in-a-certain-country-feat-p%CC%B6i%CC%B6t%CC%B6b%CC%B6u%CC%B6l%CC%B6l%CC%B6-spotify-cd705abc59

2. https://ai-ml-analytics.com/encoding/#:~:text=Encoding%20is%20a%20technique%20of,different%20types%20of%20categorical%20variables

3. https://towardsdatascience.com/seaborn-python-8563c3d0ad41

4. https://scikit-learn.org/stable/user_guide.html

5. https://www.geeksforgeeks.org/machine-learning/

6. Junghyuk Lee and Jong-Seok Lee. Predicting music popularity patterns based on musical complexity and early stage popularity. In Proceedings of the Third Edition Workshop on Speech, Language & Audio in Multimedia, SLAM '15, pages 3–6, New York, NY, USA, 2015. ACM.

7. http://cs229.stanford.edu/proj2015/140_report.pdf

8. https://www.researchgate.net/publication/341420234_Predicting_Music_Popularity_on_Streaming_Platforms