MUSIC INDUSTRY ANALYSIS

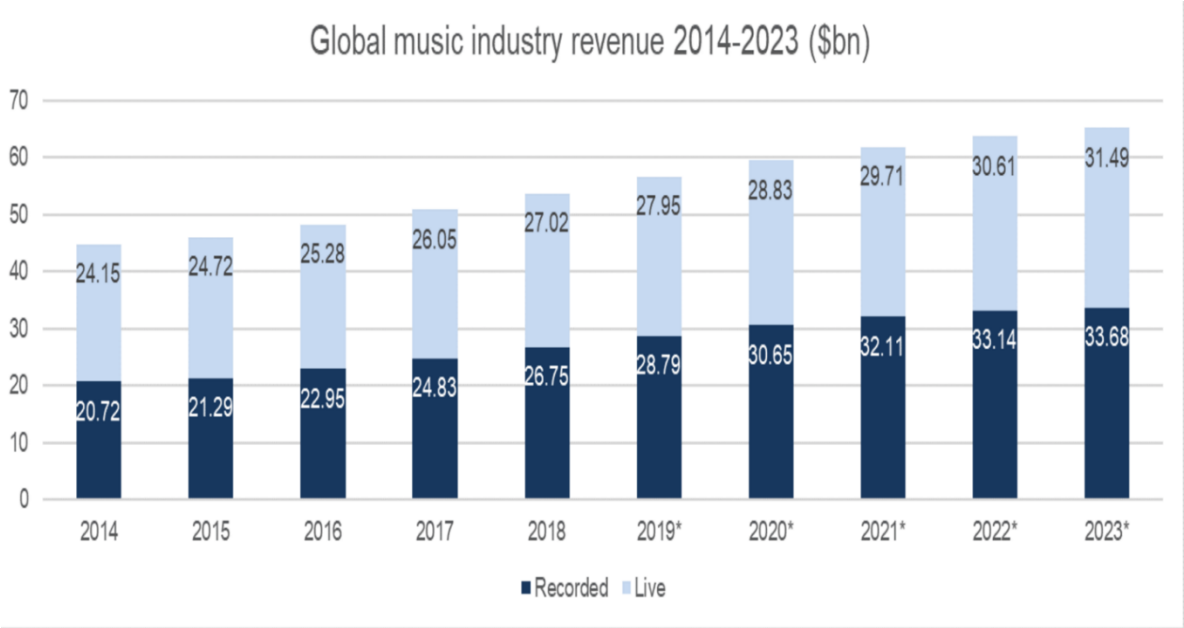
TEAM 9

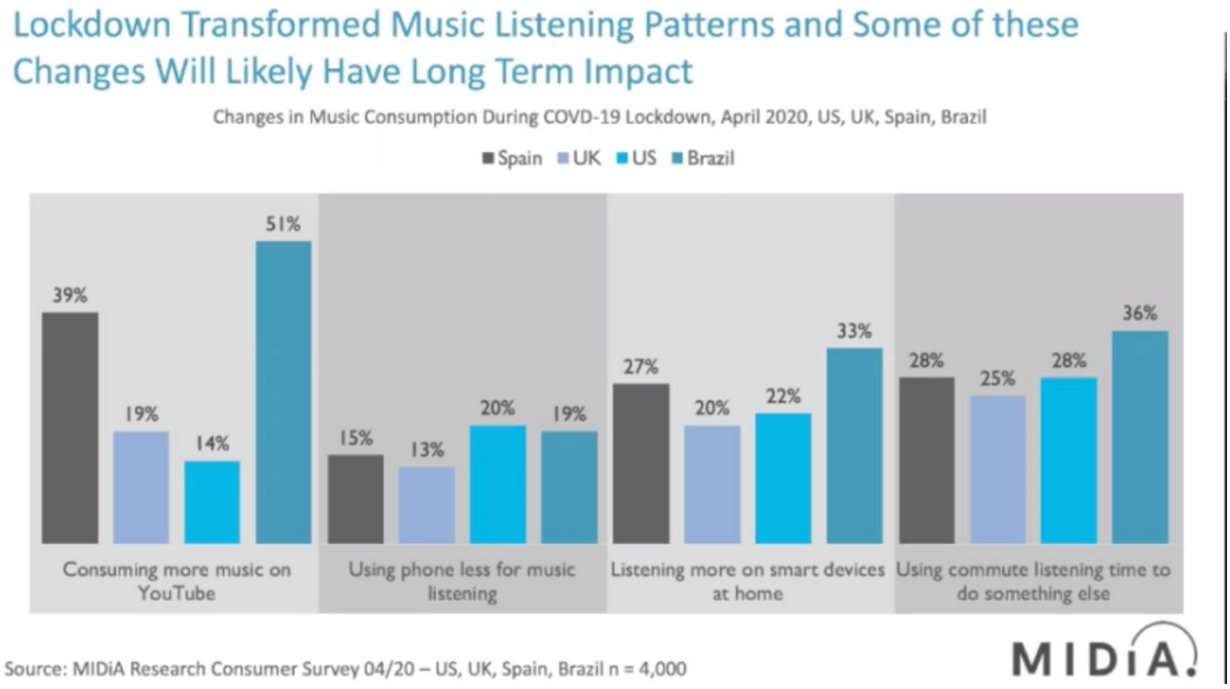
Lichang Tan, Debolina Sasmal, Jack Qu, Haripriya Vemulapati

Table of Contents

INTRODUCTION

The music industry has undergone several changes in the past few decades due to digitization of music and evolution of peer-to-peer sharing. While the effects of digitization of the profitability of the music and purchase intention of customers had been ambiguous for the longest time, there has been a positive shift with streaming platforms driving majority revenue in the current market. The issues that the industry faced due to piracy have been eliminated to a large extent by increasing audience engagement and accessibility to music. The role of data in the music industry has seen significant innovations as well. The shift in trend has enabled music producers and distributors with the ability to make more data-driven decisions to align the content closely to the kind of music that is appealing to their target market. While a number of different dimensions and metrics are available, the current project aims to summarize the current music industry by understanding the probability of a song winning critical acclaim, popularity or higher visibility. The Covid-19 pandemic has changed the way musicians interact with their audience, due to tours coming to a complete halt and digital platforms being the most widespread channel. This had created a need to fill the strategic gap between content produced and user demand as well as listening behavior.





Additionally, the paper tries to answer the following questions to understand managerial implications:

1. What attributes of a song contribute to the likelihood of it winning a Grammy?
2. What attributes of a song contribute to it being popular?
3. How to increase the outreach of the music by using the recommendation system?

With the help of the results and findings, the aim of the project is to understand the key determinants of a song's success and utilize them to make songs with a higher inclusion of these features and thus gain competitive advantage in the growing industry landscape.

DATA WRANGLING

For the purpose of this project, we have used the following datasets from Kaggle:

1. spotifyWeeklyTop200Streams.csv
2. songAttributes_1999-2019.csv
3. grammyAlbums_199-2019.csv
4. Spotify Dataset 1922-2021, ~600k Tracks

Datasets **songAttributes_1999-2019.csv** and **grammyAlbums_199-2019.csv** have been combined together using the columns "Album" and "Artist", since the Grammy dataset was missing the audio characteristics data. This merged dataset helped us to analyze Grammy winning songs have which audio attributes.

```
grammy=pd.read_csv("grammyAlbums_199-2019.csv")
attribute=pd.read_csv("songAttributes_1999-2019.csv")
attribute['won grammy'] = attribute['Album' and 'Artist'].isin(grammy['Album' and 'Artist']).astype(object)

attribute.to_csv(r'C:\Users\Debolina\Desktop\python mism 6212\updated_attributes_csv.csv', index = False)
```

spotifyWeeklyTop200Streams.csv dataset was used to compare the pre-covid and during-covid analysis of songs. We didn't perform any changes to the dataset as it was pretty clean and usable, but we had to collect the data for 2020 from the Spotify Web API.

Spotify Dataset 1922-2021, ~600k Tracks dataset was used because it contained a very important column – "year", which was very helpful in performing the trends over time analysis. This dataset didn't require any cleaning either.

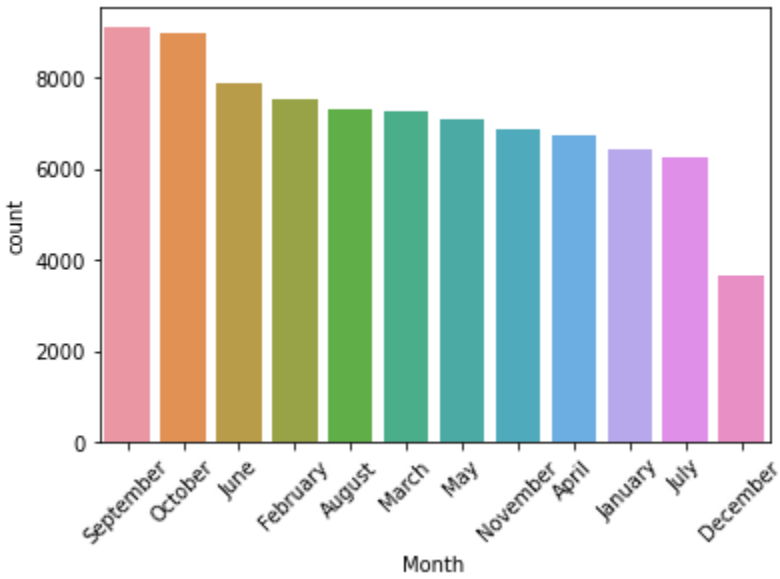
The song attributes can be explained as follows:

Attributes	Explanation
popularity	popularity of track
duration_ms	time duration of track in ms
explicit	whether it contains explicit (1) content or not (0)
danceability	how suitable the track is for dancing, 0(not danceable)->1(very danceable)
energy	how energetic the track is, 0(less energetic)->1(very energetic)
loudness	how loud the song is in dB -60(very quiet)->0(very loud)
speechiness	the ratio of spoken words to the overall, 0(instrumental)->1(talk show)
acousticness	whether the song is acoustic or not, 0(not acoustic)->1(very acoustic)
instrumentalness	the ratio of instrumental sounds overall, 0(lot of vocal sounds)->1(instrument sounds)
liveness	presence of audience, 0(studio record)->1(concert)
valence	how positive the music is, 0(sad)->1(cheerful)
tempo	tempo of track in BPM

ANALYSIS

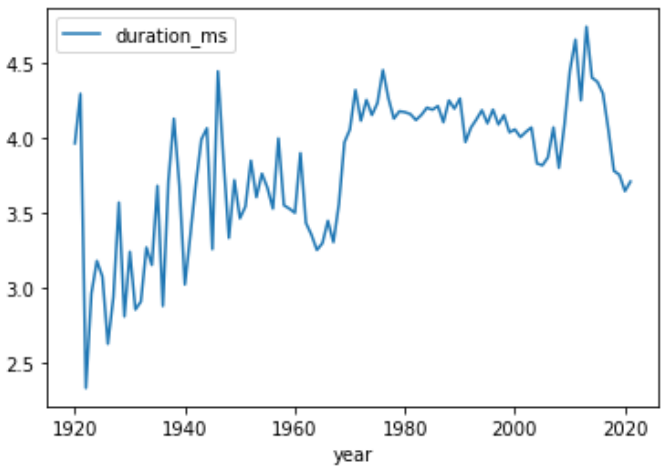
Overall Exploratory Data Analysis (EDA)

Count of songs released per month:



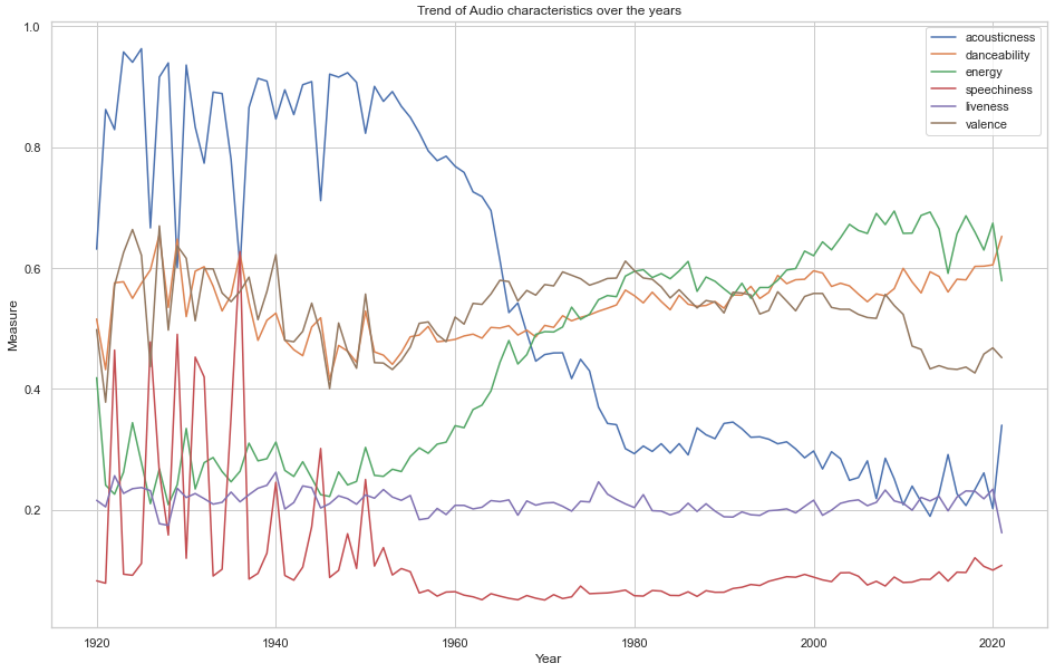
This visualization shows the count of songs released per month. It seems that most of the songs are released in the months of September and October, probably to gear up for the Holiday season. So, if a not-so-well-known artist or music agency wants to release a new album/song, it should be in the months of Dec, Jan and July when the competition is less.

Duration of song over years:

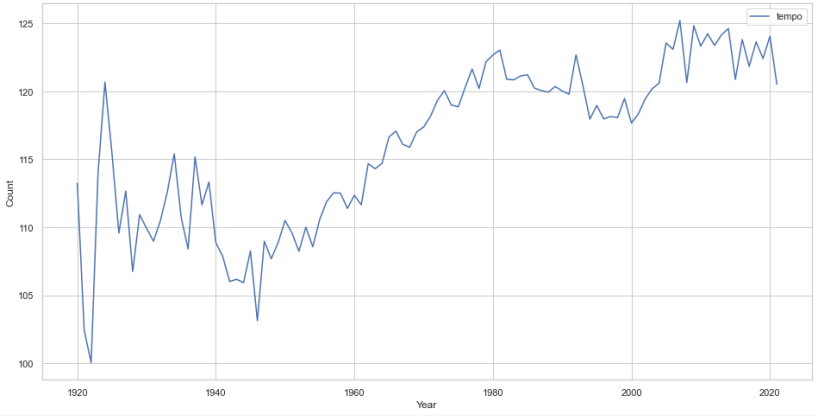


This visualization shows the average duration of songs as per the current trends. Songs are usually 3.5-4mins nowadays, but it is interesting to see that it had dropped below 2.5mins back in the 1920s. As per research, 3-4 mins is the average attention span of a listener, wherein one doesn't feel too bored or feel like the song wasn't enough (Forbes.com).

Audio characteristics over the years:



This visualization shows how the audio characteristics have changed over the years. The acousticness used to be so high, that is the use of classical instruments over the electronic ones such as electronic guitar or synthesizer, but now it has gone low. As expected, energy and danceability occupy the highest spots. Music is supposed to make us feel energetic after all. Also, the speechiness (detects the presence of spoken words in a track) did go high between 1920-1950, but it's pretty low now, and that makes sense because of the songs comprise of just music nowadays.



We have plotted the trend of tempo (In musical terminology, tempo is the speed or pace of a given piece. In classical music) over the years separately because its value is in hundreds unlike the other audio characteristics that are in decimals. Since the songs are more energetic now and with the onset of Rap and Trap music, tempo being higher is an expected result.

PART I: UNSUPERVISED MACHINE LEARNING

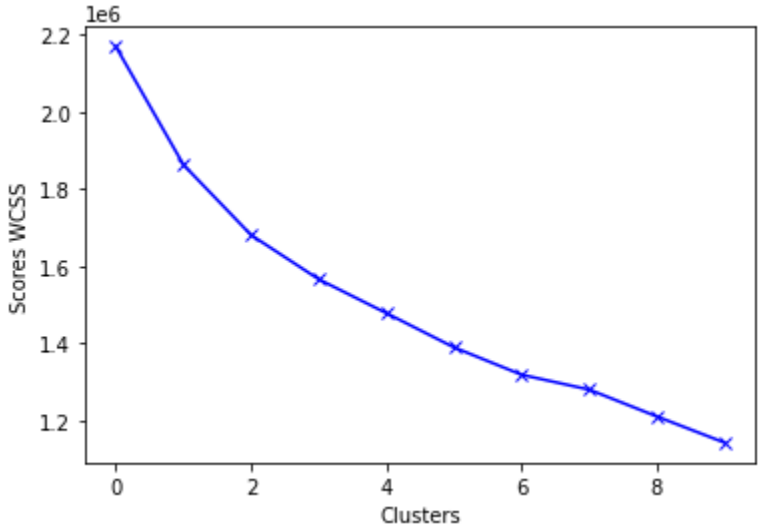
K-Means Clustering

K-means clustering is a type of unsupervised learning, which is used to find groups in the data, with the number of groups represented by the variable *K*. The algorithm works iteratively to assign each data point to one of *K* groups based on the features that are provided. Data points are clustered based on feature similarity. For this project, we have tried to check how the variables are grouped together and what features each group has.

Standard scaling of data

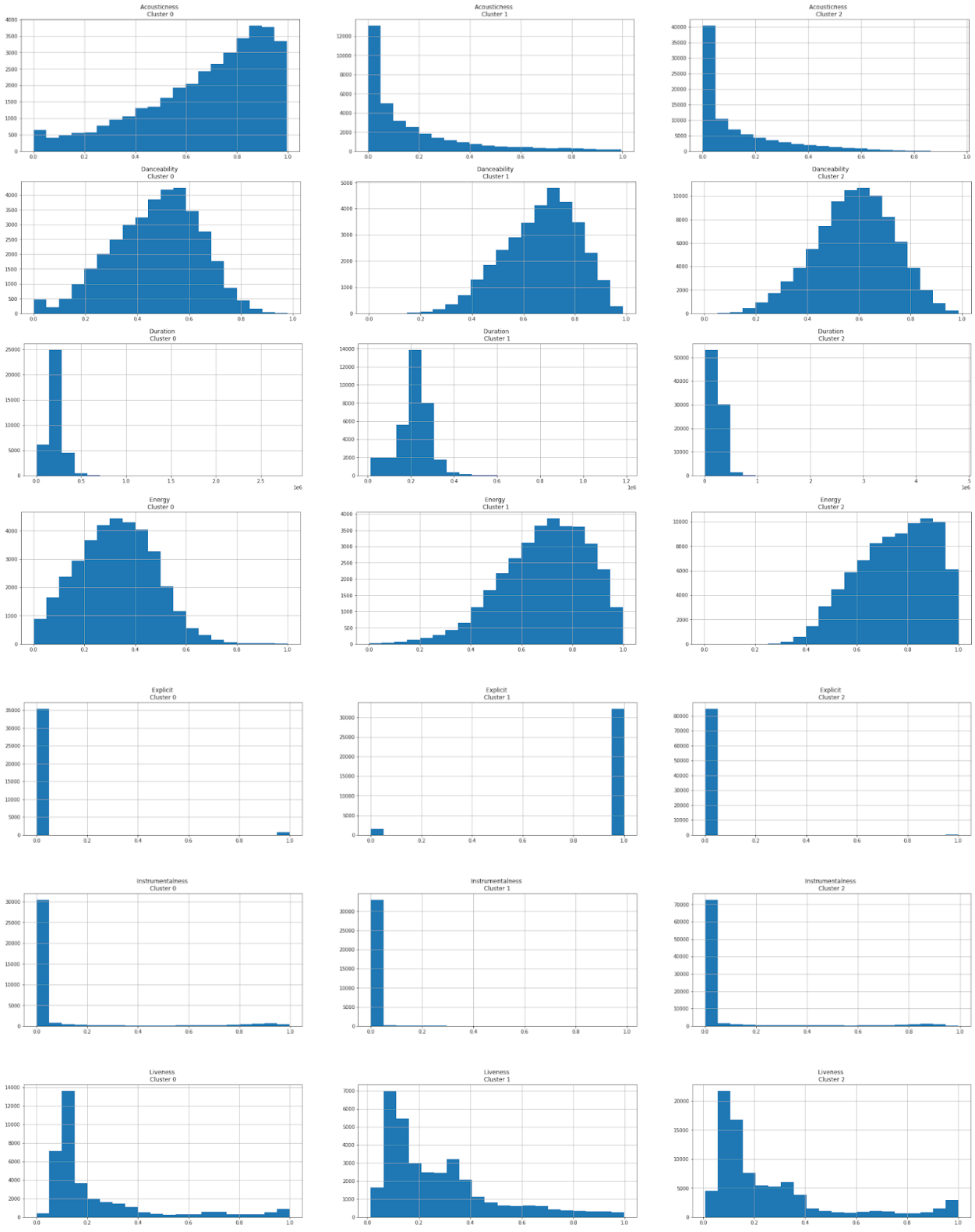
Standardization is a scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation. The distance between the data points is important, and hence we have scaled our data for clustering.

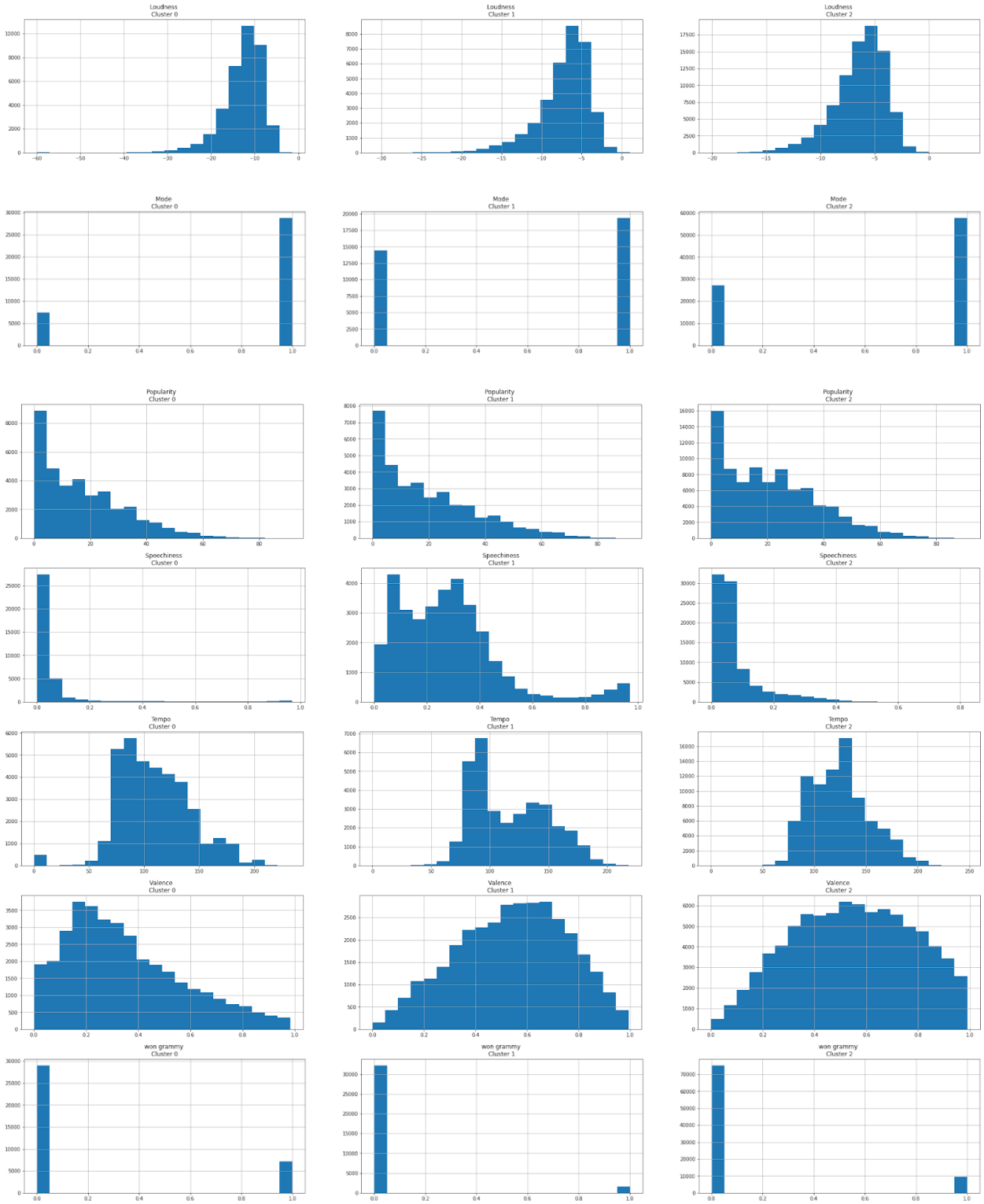
Elbow to find optimal no. of clusters for K-Means



As per the output, we decided on forming three clusters for optimal results.

Plotting histogram of each column for each cluster for better visual understanding


Acousticness
Cluster 0
Acousticness
Cluster 1
Acousticness
Cluster 2
Danceability
Cluster 0
Danceability
Cluster 1
Danceability
Cluster 2
Duration
Cluster 0
Duration
Cluster 1
Duration
Cluster 2
Energy
Cluster 0
Energy
Cluster 1
Energy
Cluster 2
Explicit
Cluster 0
Explicit
Cluster 1
Explicit
Cluster 2
Instrumentalness
Cluster 0
Instrumentalness
Cluster 1
Instrumentalness
Cluster 2
Liveness
Cluster 0
Liveness
Cluster 1
Liveness
Cluster 2



Interpretation

As per the above visualizations,

Cluster 0:

- Highest number of Grammy winning songs.
- Less explicit content.
- Least number of popular songs.
- Highest Acousticness.
- Low-medium Energy.

Cluster 1:

- Least number of Grammy winning songs.
- Most explicit content.
- Medium popularity- more than cluster 0
- Low Acousticness.
- High Energy.
- Highest Speechiness**.**

Cluster 2:

- Less Grammy winning songs.
- Least explicit content.
- Most popular.
- Low Acousticness.
- High Energy.

We can see the key differences among the three clusters. Cluster 0 and Cluster 1 are the most interesting ones if we want to analyze the likelihood of a song winning Grammy. Grammy winning songs have least explicit content (sexual, violent, or offensive in nature) and high acousticness (use of classical instruments over the electronic ones such as electronic guitar or synthesizer). These songs are more likely to have a classy and sophisticated vibe.

Interestingly, Cluster 1, which consists of the least no. of Grammy winning songs is the complete opposite of Cluster 0. Also, we can see that a song's popularity doesn't guarantee a Grammy win.

PART II: SUPERVISED MACHINE LEARNING

Classification models

Goal: Our goal is to predict which songs are going to win Grammy Awards based on the Liveness, Loudness, Mode, Popularity, Peachiness, Tempo, Timesignature, Valence. This is a binary classification problem. The label is 1 for True class and 0 for False class, respectively. The best model should be the one that gives us the highest f1 score and not overfitting.

Exploratory Data Analysis (EDA)

The most important part of the EDA we found for modelling is that this dataset has moderately imbalance classes issue, which is common in the machine learning world. The distribution of classes in this dataset is 87% of False class, 13% of True class, indicating 87% (136464) songs did not win Grammy, 13% (18467) songs won Grammy. The False class is larger than the True class by roughly 7 times.

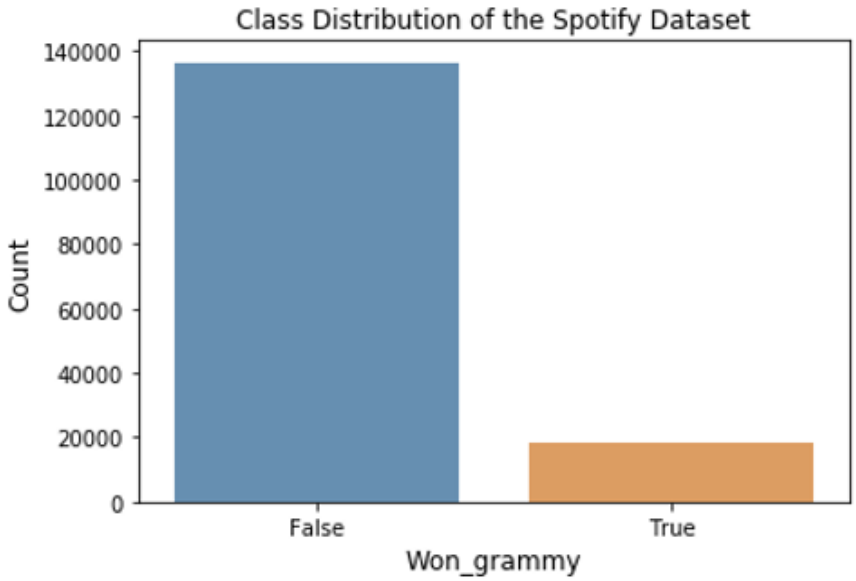


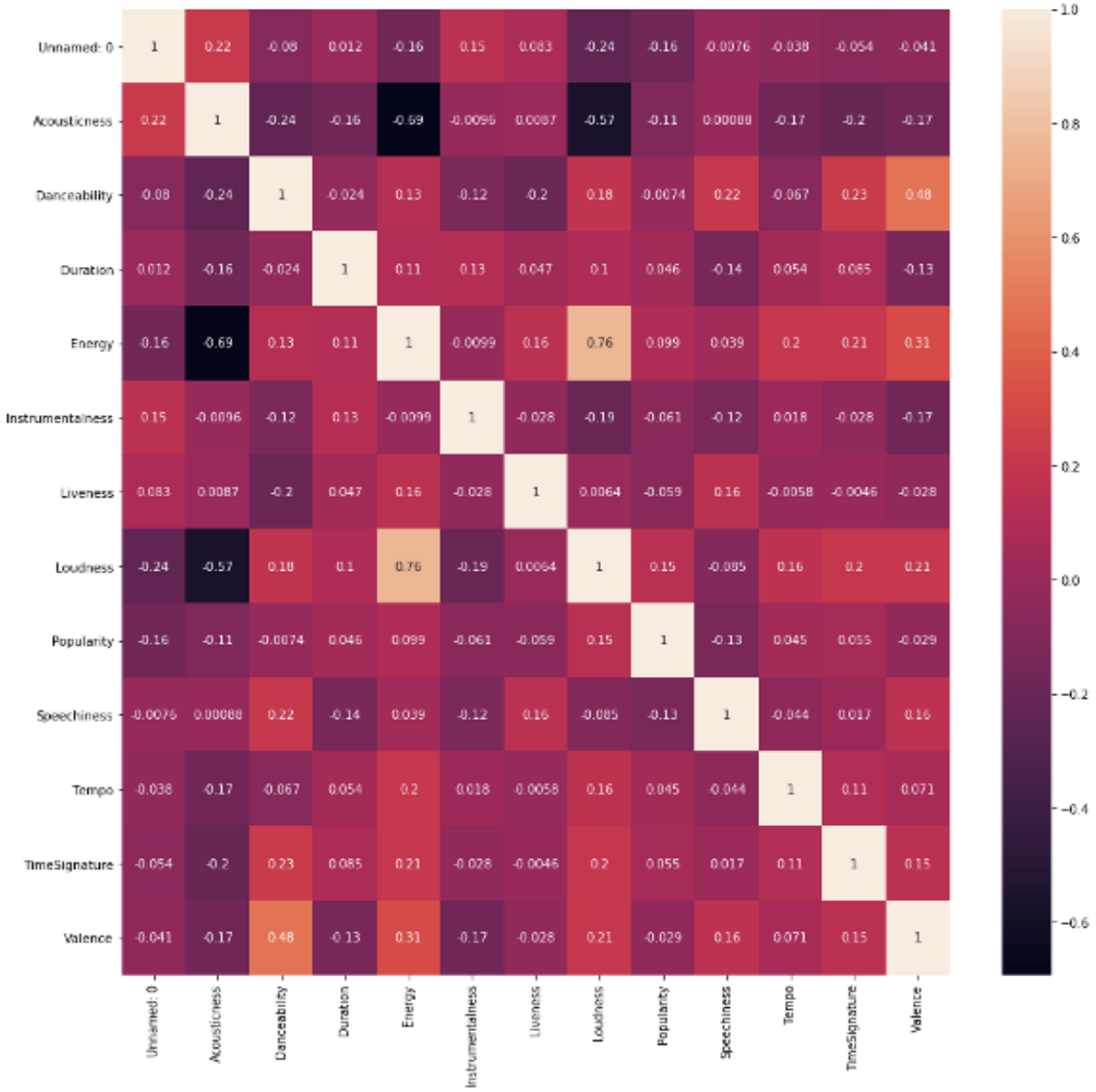
Figure 3

Data Preprocessing

This dataset has 18 variables including categorical and numerical. There are no data problems such as missing values. For the purpose of this problem, we need to drop the unnecessary variables('Album','Artist','Name'), remaining 14 variables listed under Goal for modeling. Next, we need to dummize the dataset because there are two categorical variables in the modeling dataset, then we split the dataset into predictors(x) and predicted variable(y), and then split the x,y into training set(75%) and testing set(25%).

Feature Selection

We use a heatmap to discover the correlation between variables. From the heatmap, we can see that only two variables (‘Loudness’ and ‘Energy’) are highly correlated to each other, the correlation rate is 0.78. but let's keep them for prediction, because sometimes, removing one of the correlated variables will not impact the performance. Therefore, we use all the 14 variables to train the model.



Methodologies

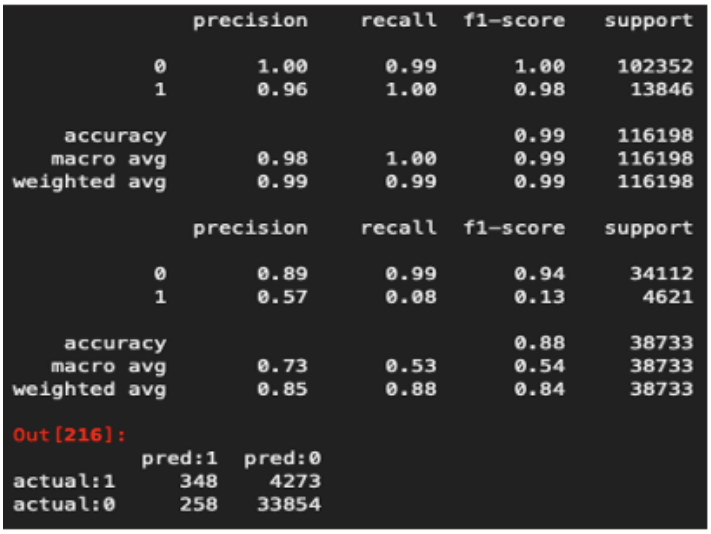
The methods we use for modeling are Cross Validation, Grid Search hyperparameters, Random Searching hyperparameters, Under sampling, Oversampling, Feature Selection by using models, Feature Importance evaluation.

Modeling

We use Random Forest, Balanced Random Forest, Random Forest + Undersampling+Grid Search hyperparameters, Random Forest + Oversampling, Extreme Gradient Boosting (XGBoost) + Undersampling, XGBoost+ Random Searching hyperparameters as our models. But we chose XGBoost+ Random Searching hyperparameters as the final model because this model gave us the highest f1 score for the minority class.

1.Random Forest

The first model we tried is Random Forest with class_weight={0:1,1:7} without tuning hyperparameters. The overall accuracy rate of the testing is good, but the recall rate is extremely low, making the f1 score of the True class to be low as well. In addition, this model is overfitting because the training result of the f1-score of the True class is extremely well (98%), but the testing result is bad (13%). Since we intend to have more values on the True class, we don't consider this model.



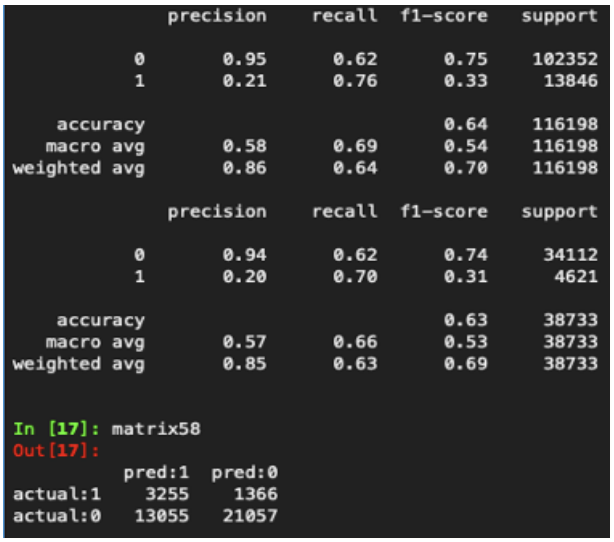
	precision	recall	f1-score	support
0	1.00	0.99	1.00	102352
1	0.96	1.00	0.98	13846
accuracy			0.99	116198
macro avg	0.98	1.00	0.99	116198
weighted avg	0.99	0.99	0.99	116198

	precision	recall	f1-score	support
0	0.89	0.99	0.94	34112
1	0.57	0.08	0.13	4621
accuracy			0.88	38733
macro avg	0.73	0.53	0.54	38733
weighted avg	0.85	0.88	0.84	38733

Out[216]:

	pred:1	pred:0
actual:1	348	4273
actual:0	258	33854

The Balanced Random Forest model with sampling_strategy='auto', performs better than the previous random forest model on the True class. The f1 score for the True class gets improved by 18%. And this model is not overfitting by checking the f1 scores of the training result and testing result. But the f1 score for the False class gets decreased by 20%. Although this model gives us higher predictions for which songs won Grammy Awards, we still want to improve the precision rate of the True class because there are many false negative misclassifications of this model.



	precision	recall	f1-score	support
0	0.95	0.62	0.75	102352
1	0.21	0.76	0.33	13846
accuracy			0.64	116198
macro avg	0.58	0.69	0.54	116198
weighted avg	0.86	0.64	0.70	116198

	precision	recall	f1-score	support
0	0.94	0.62	0.74	34112
1	0.20	0.70	0.31	4621
accuracy			0.63	38733
macro avg	0.57	0.66	0.53	38733
weighted avg	0.85	0.63	0.69	38733

In [17]: matrix58

Out[17]:

	pred:1	pred:0
actual:1	3255	1366
actual:0	13055	21057

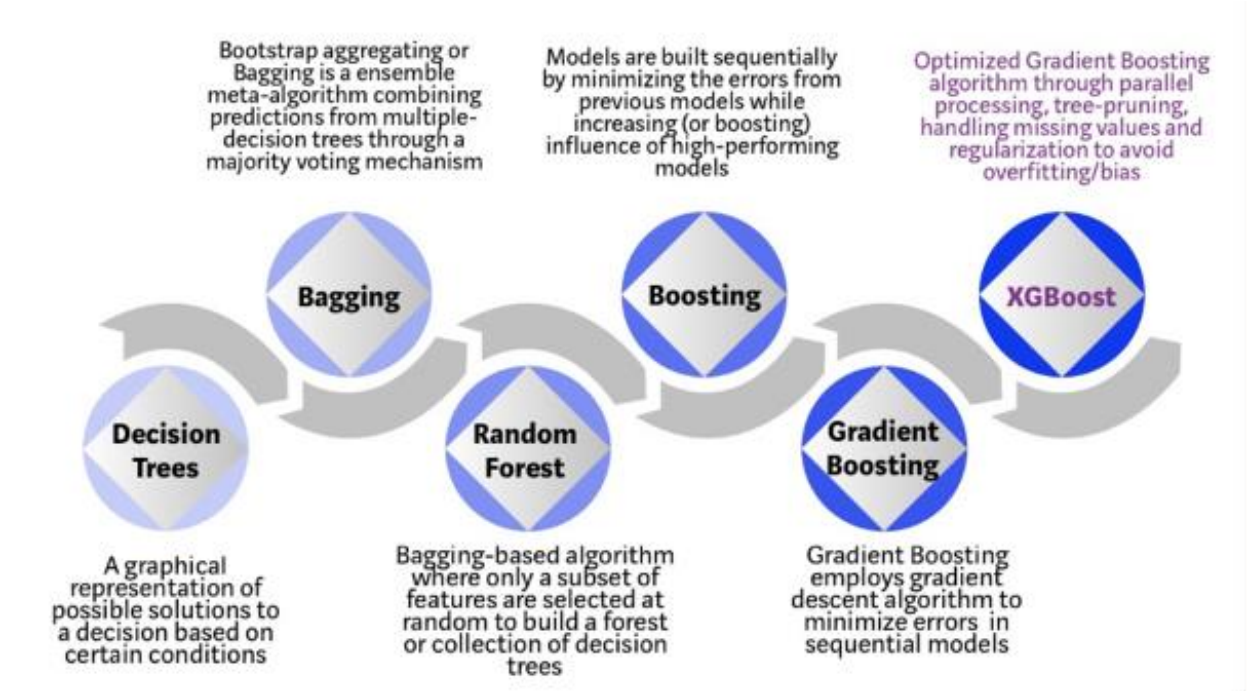
2. Random Forest & Undersampling

For the purpose of improving the precision rate of the True class, we tune the hyperparameters of the random forest model with the Undersampling method to see if the precision rate is going to be improved. We first use the RandomUnderSampler function to balance the classes from the distribution of 70/30 to 50/50. Then use GridSearchCV function with five folds cross validation to tune the hyperparameters of max_depth, min_samples_split, n_estimators, bootstrap. This model fits five folds for each of 56 candidates, totaling 280 fits. This model performs better than the Balanced Random Forest model. the overall f1 score of both True class and False class gets improved by 3% and 2 %, respectively. But this model is still overfitting on the True class.

```
In [230]: print(metrics.classification_report(y_train,y_pred_train))
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    102352
           1       0.96      1.00      0.98     13846

    accuracy                           0.99    116198
   macro avg       0.98      1.00      0.99    116198
weighted avg       0.99      0.99      0.99    116198


In [231]: print(metrics.classification_report(y_test,y_pred_test23))
              precision    recall  f1-score   support

           0       0.95      0.64      0.76     34112
           1       0.22      0.75      0.34      4621

    accuracy                           0.65     38733
   macro avg       0.59      0.70      0.55     38733
weighted avg       0.86      0.65      0.71     38733


In [232]: matrix89
Out[232]:
          pred:1  pred:0
actual:1    3481    1140
actual:0   12294   21818
```

3.Extreme Gradient Boosting (XGBoost)

With the consideration of getting better performance of the f1 score, the concern of overfitting, we use XGBoost algorithm to boost the True class and constrain the model to go overfitting. This algorithm uses advanced regularization (L1 & L2), which improves model generalization capabilities and avoids overfitting. XGBoost is basically developed by incorporating Bagging, Random Forest, Boosting, Gradient Boosting. One algorithm performs all the tasks. The chart below explains each tree-based algorithm and how XGBoost is the appropriate one for us to solve our concerns.

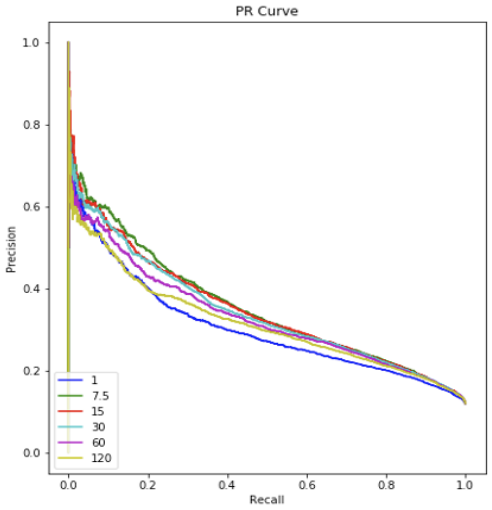


Source is from Prashant Banerjee, Data Scientist, Kaggle.

The XGBoost of the default logistic regression model gives us 88% of the f1 score of the False class and 39% of the f1 score of the True class. These results are the best of all the models, though the predicted accuracy rate of the True class is not the highest. This model gives us the most balance between precision and recall. Although the previous models received higher recall rate, meaning the predicted accuracy rate of the True class is higher, those models also gave us lower precision rate, meaning more false negative misclassifications.

We use the default tree booster to this model. We then tune 9 hyperparameters, 'max_depth', 'learning_rate', 'n_estimators', 'min_child_weight', 'subsample', 'colsample_bytree', 'colsample_bylevel', 'reg_alpha', 'reg_lambda'. These hyperparameters are the most common parameters to be tuned. The tuning range of each parameter is based on the recommendation of Amazon Developer guide. The tuning of these parameters enables us to avoid overfitting and enhance performance. Parameters such as max_depth, min_child_weight, regularization parameters (lambda, alpha) are used to control model complexity. While subsample , colsample_bytree, the reduced-size of learning rate are used to add randomness to make training robust to noise. We used RandomizedSearchCV instead of GridSearchCV due to the large data points. This model fit 5 folds for each of 10 candidates, totaling 50 fits. After we receive the best parameters, since this dataset has imbalance classes issue, we ran a for loop of selecting different weights to compare the f1 score and plot the Precision/Recall (PC) curve to compare the results visually. From this, setting up the weight to 7.5 is the optimal value that gives us the highest f1

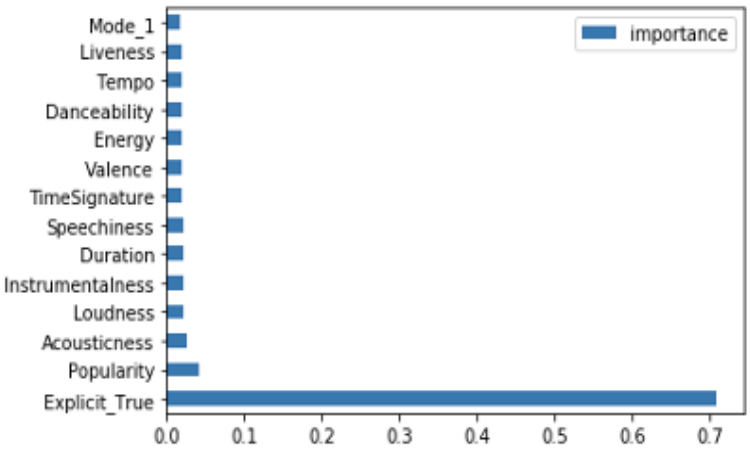
score.




```
In [318]: print(metrics.classification_report(y_train,y_pred_train49))
              precision    recall  f1-score   support

           0       0.99      0.89      0.94    102352
           1       0.54      0.95      0.69     13846

    accuracy                           0.90    116198
   macro avg       0.77      0.92      0.81    116198
weighted avg       0.94      0.90      0.91    116198


In [319]: print(metrics.classification_report(y_test,y_pred_test13))
              precision    recall  f1-score   support

           0       0.93      0.84      0.88     34112
           1       0.31      0.53      0.39      4621

    accuracy                           0.80     38733
   macro avg       0.62      0.69      0.64     38733
weighted avg       0.86      0.80      0.82     38733


In [320]: matrix52
Out[320]:
          pred:1  pred:0
actual:1    2467    2154
actual:0    5500   28612
```

Feature Importance

We use SelectFromModel function to calculate the feature importance of each input variable, essentially allowing us to test each subset of features by importance, starting with all features and ending with a subset with the most important feature. The below chart tells us that the most important feature with 71% is Explicit_True when we use all 14 variables to do the modeling. Besides, this model only uses Explicit_True for the prediction. In order to see if the model is going to get improved, we run another model that only keeps the top 7 variables. We did Random Search for the new hyperparameters. However, the f1 score of the True class is 32%, which gets decreased by 7% when we use all variables. Explicit_True is still the most important feature.



Other Thoughts:

1. Oversampling vs Undersampling

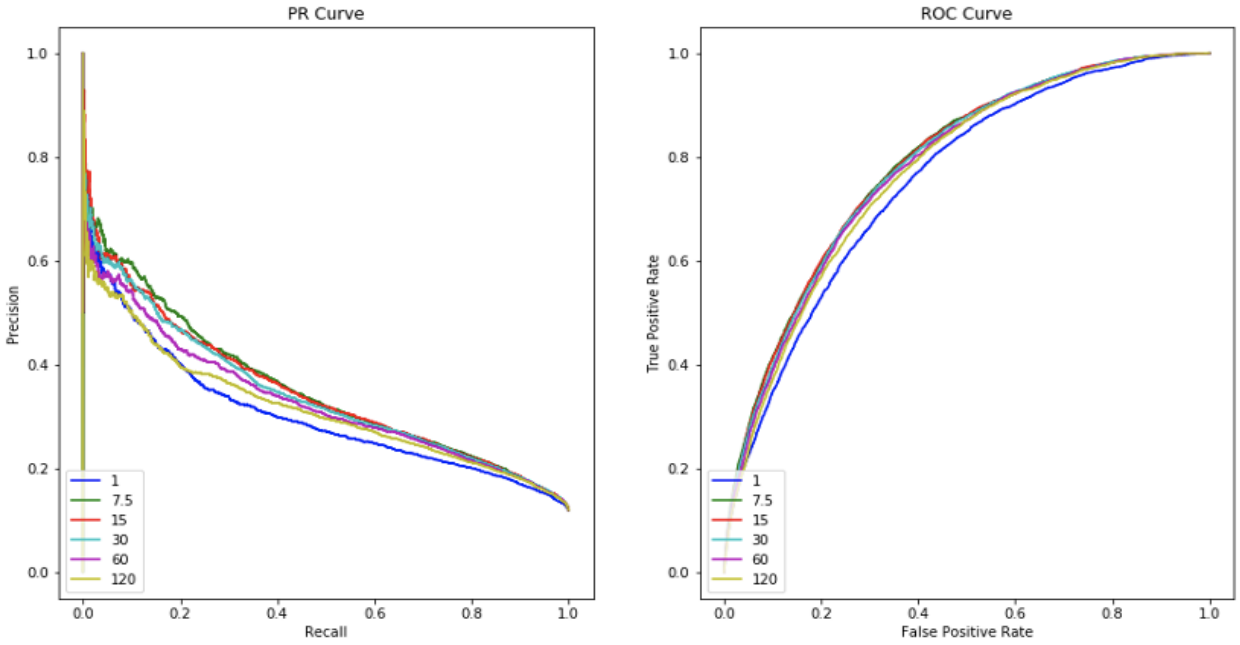
We also tried using the Oversampling method by using the SMOTE function to balance our classes before doing modeling. However, the result is less ideal than using Random Undersampling, and it took more time to implement the method. We also tried using the combination of Oversampling and Undersampling by using Pipeline to receive the best oversampling and undersampling rate. However, the model performance is not good either. Many researchers recommend using the Undersampling method if you don't have sufficient time, and most of the time, this method returns better results than the Oversampling method. But there is no guarantee for which method always outperforms the other for imbalanced data. If you have sufficient time, you should try many methods such as Oversampling, Undersampling and the combination of Oversampling and Undersampling.

2.Handle Imbalanced Data

According to the XGBoost documentation guide, we tried using parameters of both scale_pos_weight and max_delta_step to balance the classes. However, the performance of using max_delta_step is much poorer than using scale_pos_weight.

3.PC Curve vs ROC Curve

AUC score and ROC curve are two of the common methods to evaluate classification performance in the Machine Learning world. However, this is not the case when you have highly imbalanced classes. Our dataset is not highly imbalanced, just moderate, but we still don't see the ROC curve is the right fit, but the PC curve. The charts below can confirm our assumption.



For a PR curve, a good classifier aims for the upper right corner of the chart but upper left for the ROC curve.

While PR and ROC curves use the same data, you can see that the two charts tell different stories, with some weights seem to perform better in ROC than in the PR curve. For example, when weight is set to a bigger value, the PR curve tends to get worse, except the weight is equal to 1. However, it is not clear to identify which ROC curve is the best when the weight is set to 7.5,15 and 30, respectively. ROC curve is not a good visual illustration for moderately and highly imbalanced data, because the False Positive Rate (False Positives / Total Real Negatives) does not drop drastically when the Total Real Negatives is huge. Whereas Precision (True Positives / (True Positives + False Positives)) is highly sensitive to False Positives and is not impacted by a large total real negative denominator.

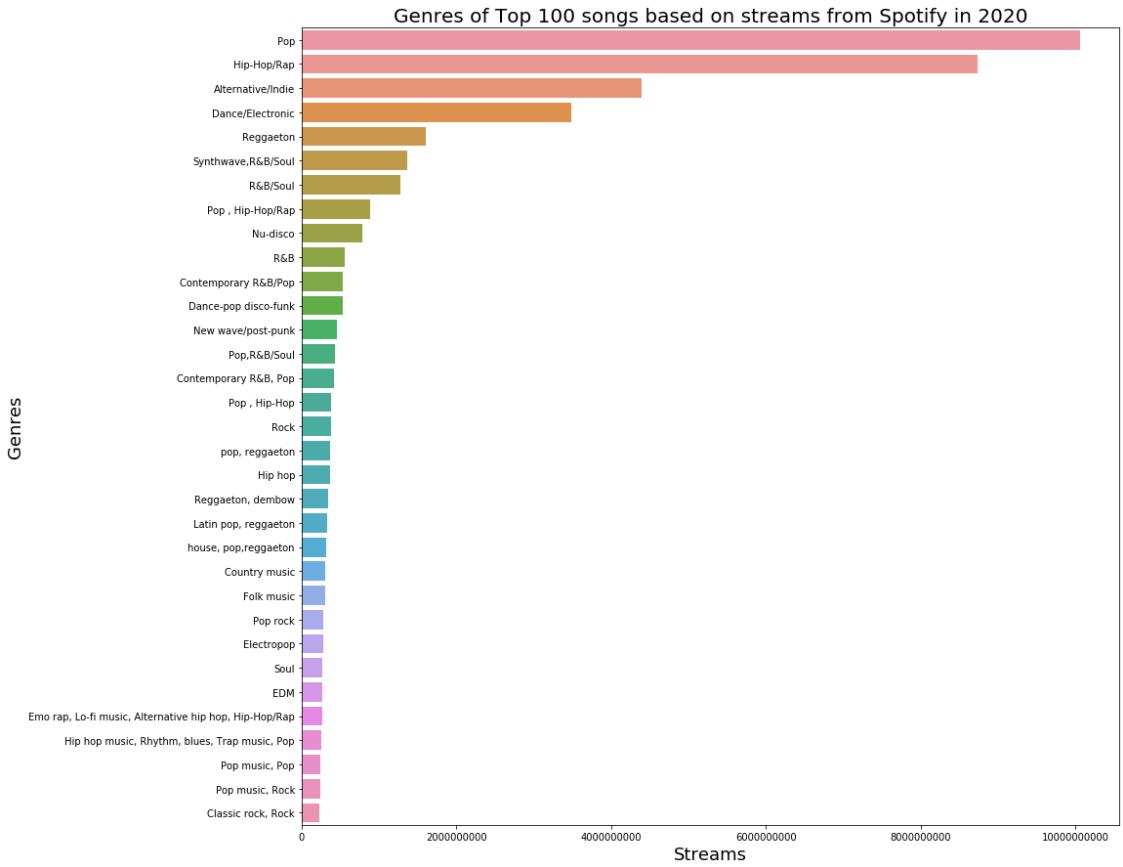
III. COMPARISON – Pre-covid vs. During covid

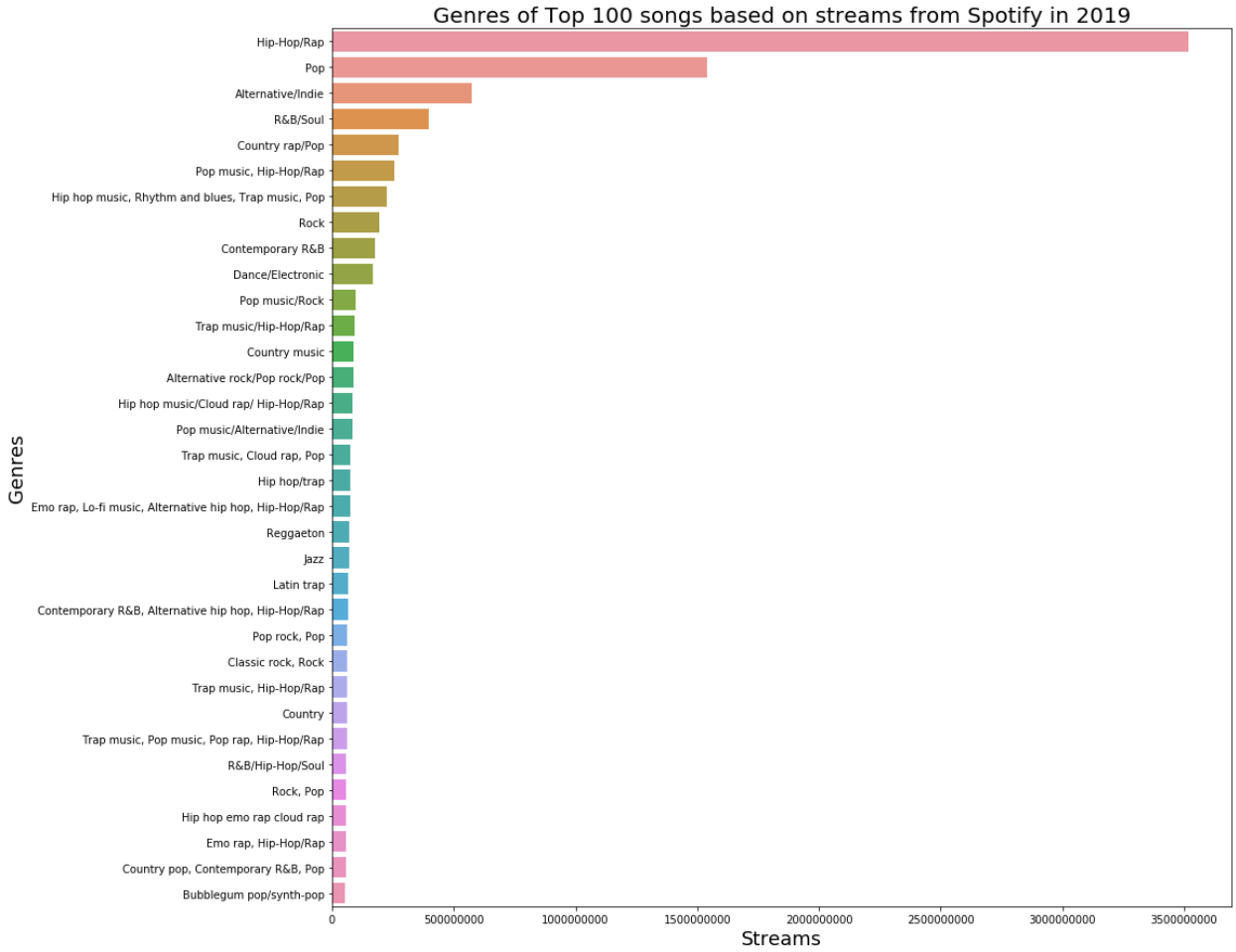
The Change of Streaming Habit

We are interested in how pandemic changes people's listening habits. Therefore, we scraped the streaming record of 2019 and 2020 from Spotify Web API, then used them to compare the differences of listening habits such as genres between before Covid-19 (2019) and Covid-19 period (2020). Understanding the trend will also help song producers, writers etc. to have a direction to produce songs that meet people's tastes. Accordingly, they could make money from this Covid dark time, or even post-Covid because things won't go back to normal at least within a few years.

Data Preprocessing:
The two datasets from the Web API are clean, well-formatted. Only four missing values from one dataset, we just removed these missing values. We then did EDA for each dataset.

COVID-19 has changed our behaviors of all kinds since March 2020. This includes people's streaming habits and music tastes. The first change was the stream frequency. The stream frequency in 2020 was nearly three times more than the stream frequency in 2019. In other words, people listened to music more. The second change was the music genres. Overall, Pop,Hip-Hop,R&B/Soul,Altertative/Indie were the most popular streamed genres both in 2019 and 2020. However, the genre of Dance/Electronic jumped to no.4 from no.10 in 2020. In addition, the genre of Reggaeton also jumped to no.5 from no.22 in 2020. The genres of Dance-pop disco-funk, New wave/post-punk and NU-disco were new to show up to the most popular list and Jazz music was gone from the list in 2020. It is not hard to understand why these dance type genres were popular during social distancing. The electronic dance type music is frequently upbeat, lyricless nature, thriving club scenes. Public events and clubs were closed, and this relaxational & experimental music helped people to release themselves in the dark times, at least, at home. However, this doesn't show any conclusive COVID-19-related behavior. Correlation but not causation. But we still can recommend that music producers produce more relaxational & experimental related music, at least within 2021 since pandemic is still ongoing.
The comparisons are given below:





PART IV. RECOMMENDATION ENGINE

Content-Based Recommendation Engine

YouTube, Netflix, and other various other streaming platforms have been using recommendation engines to drastically increase the time people spend on using their services. Amazon and other e-commerce sites use recommendations to improve sales of products that would otherwise go unnoticed by the consumers.

Here, I have developed a content-based recommendation engine that recommends similar songs based on the artists and genres. This would improve sales by making the customers more aware of other viable songs, which would likely lead to the purchase of such songs.

I am using a small dataset due to computational limitations. The features that the engine uses to make recommendations could be easily scaled up to include even more features given enough data.

Feature selection and Data Processing

We used the "grammyAlbums_1999-2019.csv" due to its small size. The content-based recommendation engine will be using "Artist" and "Genre" to make recommendations. As I stated from above, this can easily scale up to include more features.

In order to process the text information, I needed to combine the texts under "Artist" and "Genre" into one string that will then be fed into the vectorizer for processing. To avoid misrecognition of the text data, I had to clean out any irrelevant spaces, marks, symbols, and punctuations included in the data. For example, [Lady Gaga & Bradley Cooper] will be recognized as 5 separate words but we do not want that. Thus, by cleaning out the spaces and symbols we get [ladygaga bradleycooper], which will be recognized as two words.

Each data column must be carefully analyzed to make sure the resulting text information will be in the appropriate format. The two features I used required different processing procedures to ensure the accuracy of the resulting string.

Methodology

Term Frequency–Inverse Document Frequency (TF-IDF)

The TF-IDF is a method that will place weights on words depending on the frequency it appears. This will assign a numeric value to the metadata string that we created above, which will then be evaluated by the cosine-similarity method. I have also experimented with the count-vectorizer method, but its results were inferior compared to the TF-IDF method.

After performing the above procedures, the resulting string will be something like this [ladygaga bradleycooper pop]. This contains all the metadata information about the pop song "Shallow" with singers Lady Gaga and Bradley Cooper. Each song in the data file will have a string like this and all of these strings will be fed into the TfidfVectorizer to produce a TF-IDF matrix.

Cosine Similarity

There are a few similarity measures that I have tried (Euclidean, Hyperbolic Tangent Kernel) and the cosine similarity performed the best. The cosine similarity measures the angle between two vectors and uses this to determine how similar the two vectors are. With these measures we can then rank them given the input and compute the recommendations.

Recommendation Function

The input will be a song title and the function will find a list of cosine similarity scores for all the other songs in the data file. This list will then be sorted based on the highest similarity and find the top 10 songs associated with the top 10 scores.

Results

Overall, the output recommendations were mostly good with some random suggestions. The list of recommendations will produce songs made by the same artist and are in the same genres. This works for most titles but for certain titles the results will be very random. This is expected due to the nature of recommendation engines and natural language processing. However, randomness is good in the business perspective because user data from online streaming platforms often suggest that users will have random preferences that do not stick to a certain ruleset.

A quick example, given the song “this is America” by Childish Gambino (genre is general), the top 10 recommendations are:

32	Redbone	R&B	Childish Gambino
189	Single Ladies (Put a Ring on it)	General	
366	Walk on	General	
387	Beautiful Day	General	
46	Hello	General	
149	Rolling in the Deep	General	
367	Fallin'	General	
210	Please Read the Letter	General	
297	Daughters	General	
23	24k Magic	General	

As you can see, Redbone is by Childish Gambino and it got picked up by the recommendation engine. There is only one song by Childish Gambino because he only has two songs in the dataset (“This is American” and “Redbone”). The other songs were selected based on the genre of the input song which is in General.

RECOMMENDATIONS & FUTURE SCOPE

Classification modeling:

We can try Grid Searching for the hyperparameters with a wider range of each parameter for the XGBoost model. We did not use this method this time because we have a large dataset, it would take at least 20 hours to finish the tuning. We actually tried to tune nine hyperparameters by using this method initially, but we waited for 11 hours, and it hadn't finished up tuning. In order to save time, we used Random Searching instead, which took 2 hours to train.
Second, we consider if other features such as genre of the songs have been added to this dataset, the performance might have improved further. In the future, we can try to add more features to the model.
Third, we can also try to tune hyperparameters of the Balanced Random Forest model. We did not do it this time. First because it will also take a long time. Second it is because many researchers commented that even with the tuning method, the problem of low precision rate of the minority class might still exist.

Recommendations-

From our classification model, whether the song is explicit or not impacts the most for winning a Grammy Award. This result is actually matched with the clustering result from my teammate, Debolina's part. An explicit track is one that has curse words or language or art that is sexual, violent, or offensive in nature. Therefore, we can recommend song producers avoid making songs explicit. Winning a Grammy means an accomplishment. Songwriters, producers, and other behind-the-scenes roles often enjoy the best benefits from Grammy wins. Those awards are more of a big deal for them. And it doesn't matter what they won for – winning a Grammy opens all kinds of doors in just about any industry. Grammy winners don't take cash back home from the moment they receive the Awards. But money will come later. Before winning a Grammy, producers on average charge $30,000 to $50,000 per track, after winning a Grammy, that would be $75,000 per track(Marketplace,2021). Any performer or producer who wins on the music industry's biggest night can expect to see at least 55% more in concert ticket sales as compared to before they won(Harding,2021). Apart from the concerts, more businesses will invite these winners for commercial events. During social distancing, the music industry is struggling to survive. Our classification model can help music producers to understand the chances based on some song attrictures to win a Grammy and what impacts the most to win a Grammy. This opens a direction for music producers to make songs in the future.

Recommendation System:

Improvement to the recommendation engine could be made by also incorporating collaborative filtering of user data. This will provide a good starting point for the content-based engine. Another layer of recommendation algorithm could be added based on the numerical features in the dataset. The company could also use past user data to develop pre-filtering layers. For example, if a user finishes listening to a pop song and then immediately searches for classical music, then the algorithm could add in a few classical music recommendations to users that listen

to pop songs. This randomness simply cannot be done just by the content-recommendation engine and would require past user data. This way we can manually create the desired "randomness" that would fit the business purpose.

Reference

Heidenreich, H. (2018, August 24). Natural Language Processing: Count Vectorization with scikit-learn. Medium. https://towardsdatascience.com/natural-language-processing-count-vectorization-with-scikit-learn-e7804269bb5e

Document Clustering with Python. (n.d.). Brandonrose. Retrieved April 16, 2021, from http://brandonrose.org/clustering

sklearn.feature_extraction.text.CountVectorizer — scikit-learn 0.24.1 documentation. (n.d.). Scikit-Learn. Retrieved April 16, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html

6.2. Feature extraction — scikit-learn 0.24.1 documentation. (n.d.). Sickit-Learn. Retrieved April 16, 2021, from https://scikit-learn.org/stable/modules/feature_extraction.html

sklearn.metrics.pairwise.cosine_similarity — scikit-learn 0.24.1 documentation. (n.d.). Sickit-Learn. Retrieved April 16, 2021, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html

GeeksforGeeks. (2019, October 3). Implement sigmoid function using Numpy. https://www.geeksforgeeks.org/implement-sigmoid-function-using-numpy/

Wasi, A. W. (2017, February 2). Creating a TF-IDF Matrix Python 3.6. Stack Overflow. https://stackoverflow.com/questions/42002859/creating-a-tf-idf-matrix-python-3-6

Lavin, M. J. (2019, May 13). Analyzing Documents with TF-IDF. Programming Historian. https://programminghistorian.org/en/lessons/analyzing-documents-with-tfidf

Scott, W. (2019, May 21). TF-IDF from scratch in python on real world dataset. Medium. https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

Chen, S. (2021, January 13). What Is a Recommendation Engine and How Does It Work? Appier. https://www.appier.com/blog/what-is-a-recommendation-engine-and-how-does-it-work/

Sharma, P. (2020, December 23). Comprehensive Guide to build a Recommendation Engine from scratch (in Python). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/

Tan,J. (2020,November 11). How to deal with imbalanced data in Python. Towards Data Science. Retrieved from https://towardsdatascience.com/how-to-deal-with-imbalanced-data-in-python-f9b71aba53eb

Brownlee,J.(2016,August 27).Feature Importance and Feature Selection With XGBoost in Python. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/

Amazon Developer Guide. Tune an XGBoost Model. Retrieved from https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-tuning.html

XGBoost Documentation. Notes on Parameter Tuning. Retrieved from https://xgboost.readthedocs.io/en/latest/tutorials/param_tuning.html

XGBoost Documentation. XGBoost Parameters. Retrieved from http://devdoc.net/bigdata/xgboost-doc-0.81/parameter.html

RandomUnderSampler. Retrieved from https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html

BalancedRandomForestClassifier. Retrieved from https://imbalanced-learn.org/stable/references/generated/imblearn.ensemble.BalancedRandomForestClassifier.html

Kaggle. (2017).Imbalanced data & why you should NOT use ROC curve. Retrieved from https://www.kaggle.com/lct14558/imbalanced-data-why-you-should-not-use-roc-curve

Harding,A.(2021,March 13). Do Artists Get Money For Winning a Grammy Award? This Is How Much Grammys Are Worth. Showbiz. Retrieved from https://www.cheatsheet.com/entertainment/do-artists-get-money-winning-grammy-award-how-much-grammys-worth.html/

INDIVIDUAL CONTRIBUTIONS:

Lichang Tan – Supervised Learning - Classification modeling, comparison, and recommendation from classification modeling, the web scraping from Spotify API for the comparison dataset.

Jack Qu – Recommendation Engine

Haripriya Vemulapati – Business Problem, Recommendations and Future scope

Debolina Sasmal - Dataset wrangling, EDA, Unsupervised Learning – K-Means Clustering