PROJECT REPORT
DSC540 - ADVANCED MACHINE LEARNING

PREDICTING SONG POPULARITY USING SPOTIFY DATA

Prepared by GROUP 2

Zhenyu Zhou
Bowen Qiu
Natapong Sornprom
Tevfik Gurkan
Devindra Sawh

Submitted to

Professor Dr. Ilyas Ustun

DePaul University
Winter 2021

Abstract

Machine Learning techniques are widely used in data analysis areas for exploring hidden knowledge behind the features and predicting target labels. As a result, many researchers use Machine Learning techniques as tools to help them accelerate the progress of the research topics that they are focusing on. The problem that we focused on in our research here is to find what affects the popularity of a song. These results could be useful for music platforms and music producers when recommending songs to users or producing new songs and expecting them to be popular. To achieve this goal, we extracted data generated using Spotify, which is a popular music platform, from Kaggle, as the dataset to perform Machine Learning techniques on in order to probe into our questions. This dataset contains the information of 174389 different songs from 36195 singers. Each song has 19 features including "acousticness", "duration_ms", and "energy". We chose "popularity" as the target label in our study because the main goal is to analysis the knowledge behind the features and predict the popularity using these features. While there are various Machine Learning models, and each model has their own advantages and disadvantages, we compared and combined these results produced by different models to make the analysis more confident than using only one model. The models we used included "Decision Tree", "Random Forest", "K-nearest Neighbors", "Support Vector Machine", and "Ada Boosting". We applied the models mentioned above to the dataset one by one with adequate hyper-parameter tuning to find the best performance, then showed diagrams of the progress that how the results were changing by modifying the hyper-parameters. In this case, readers can have a better understanding of how these models work. Moreover, with the fine-tuned parameters, we compared the results all together and looked for the most significant attribute that determines whether a song is popular or not.

Introduction

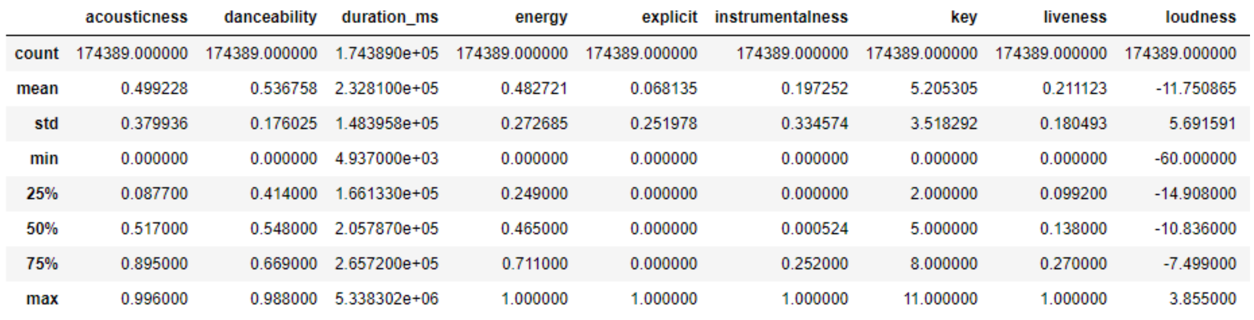
Music industry chain is a mature industry chain nowadays and music producers have to start considering how to produce popular songs to make more profit. Thus, what affects the popularity of a song becomes an open question in the music industry area. Additionally, as Machine Learning techniques grow in many areas, more and more researchers moved their eyes to this field and tried to use these techniques to solve problems in other areas. Thus, finding the attribute that can determine the popularity of a song using Machine Learning techniques becomes a hot topic.

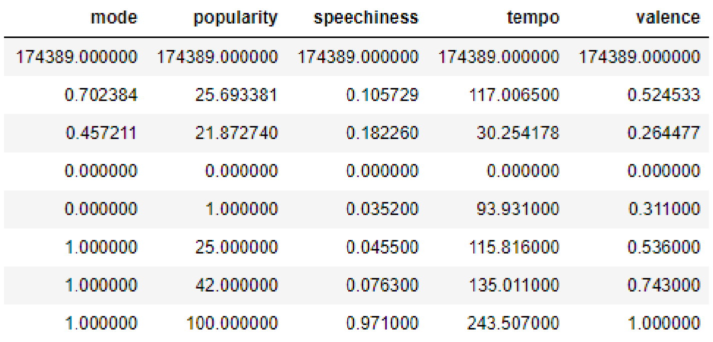
The evaluation criterion of songs seems subjective because there does not exist a common rule to judge if a song is good or bad. However, we can still measure the popularity by counting the number of listeners of a song to reflect how popular a song is. As a result, studying the relationship between popularity and other features could be a good start point to solve this music popularity problem.

In our study, we extracted data from Kaggle using Spotify, so the reliability of the dataset is credible. Since there are various Machine Learning models that exist, we chose 6 of them to do this job, they are: “Decision Tree”, “Random Forest”, “K-nearest Neighbors”, “Support Vector Machine”, “Logistic Regression” and “Ada Boosting”. For each model, we showed the progress of how we tuned the hyper-parameters to achieve the best performance. Moreover, we put all these results together for a comparison and a more detailed analysis to demonstrate which attribute affects the popularity and how this happened.

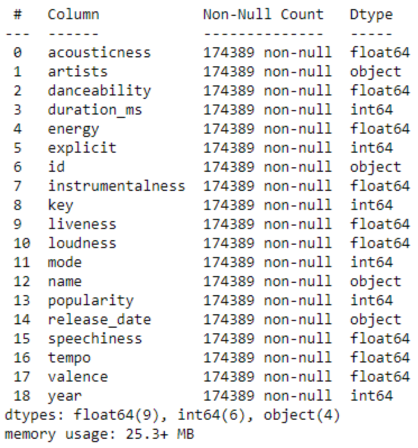
Exploratory Data Analysis (EDA)

The dataset we used in our study is generated from Kaggle using Spotify. This dataset contains 174,389 different songs from 36,195 singers. Each song has 19 features. There are no missing values being detected.



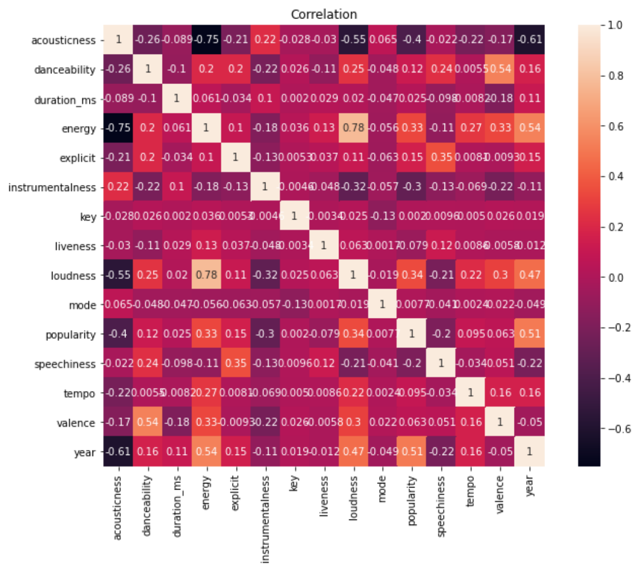


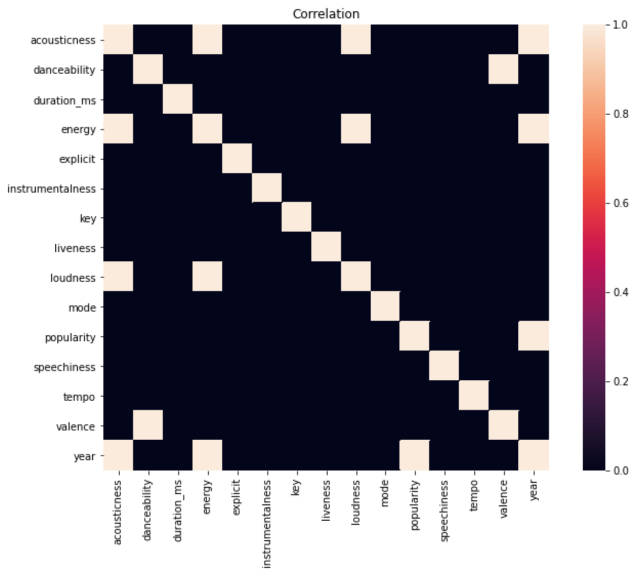
	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	popularity	speechiness	tempo	valence
count	174389.000000	174389.000000	1.743890e+05	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000	174389.000000
mean	0.499228	0.536758	2.328100e+05	0.482721	0.068135	0.197252	5.205305	0.211123	-11.750865	0.702384	25.693381	0.105729	117.006500	0.524533
std	0.379936	0.176025	1.483958e+05	0.272685	0.251978	0.334574	3.518292	0.180493	5.691591	0.457211	21.872740	0.182260	30.254178	0.264477
min	0.000000	0.000000	4.937000e+03	0.000000	0.000000	0.000000	0.000000	0.000000	-60.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.087700	0.414000	1.661330e+05	0.249000	0.000000	0.000000	2.000000	0.099200	-14.908000	0.000000	1.000000	0.035200	93.931000	0.311000
50%	0.517000	0.548000	2.057870e+05	0.465000	0.000000	0.000524	5.000000	0.138000	-10.836000	1.000000	25.000000	0.045500	115.816000	0.536000
75%	0.895000	0.669000	2.657200e+05	0.711000	0.000000	0.252000	8.000000	0.270000	-7.499000	1.000000	42.000000	0.076300	135.011000	0.743000
max	0.996000	0.988000	5.338302e+06	1.000000	1.000000	1.000000	11.000000	1.000000	3.855000	1.000000	100.000000	0.971000	243.507000	1.000000



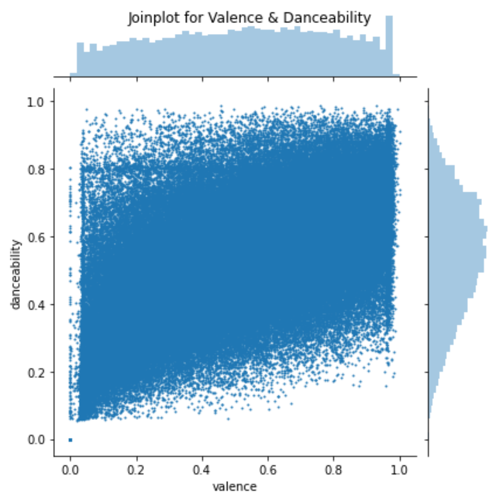
```
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   acousticness      174389 non-null  float64
 1   artists           174389 non-null  object
 2   danceability      174389 non-null  float64
 3   duration_ms       174389 non-null  int64
 4   energy            174389 non-null  float64
 5   explicit          174389 non-null  int64
 6   id                174389 non-null  object
 7   instrumentalness  174389 non-null  float64
 8   key               174389 non-null  int64
 9   liveness          174389 non-null  float64
 10  loudness          174389 non-null  float64
 11  mode              174389 non-null  int64
 12  name              174389 non-null  object
 13  popularity        174389 non-null  int64
 14  release_date      174389 non-null  object
 15  speechiness       174389 non-null  float64
 16  tempo             174389 non-null  float64
 17  valence           174389 non-null  float64
 18  year              174389 non-null  int64
dtypes: float64(9), int64(6), object(4)
memory usage: 25.3+ MB
```

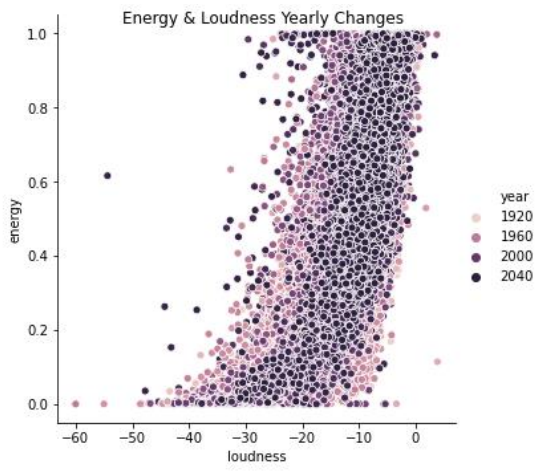
We chose “popularity” as the target label in our study, dropping variables 'id', 'release_date', 'name', 'artists' to continue our analysis. We discovered correlations among all variables except 'id', 'release_date', 'name', 'artists'. Then we added a filter on correlation plot that only shows correlation > 0.5.

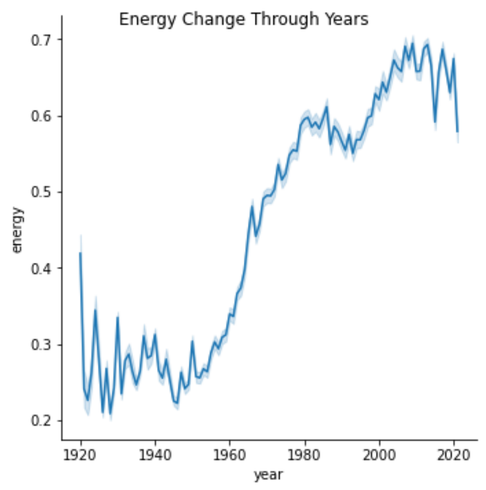


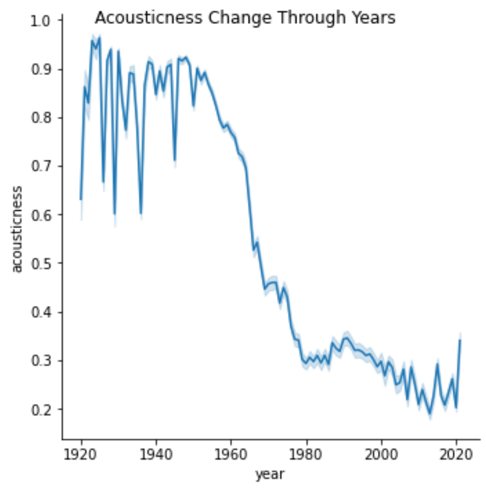


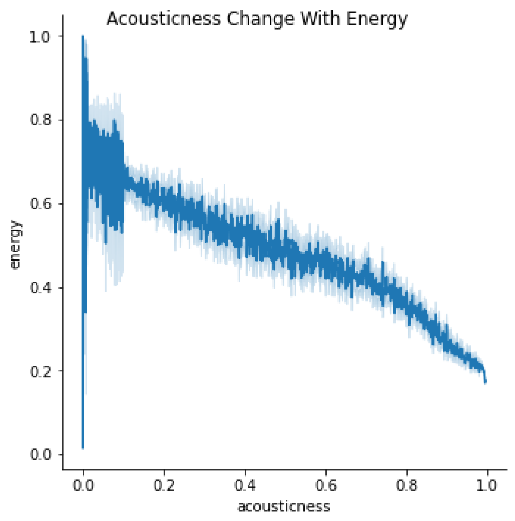
Valence & Dancibility are 54% correlated, Year & Energy are 54% correlated, Loudness & Energy are 78% correlated, Acousticness & Year are -60% correlated, Acousticness & Loudness are 55% correlated, Acoustincness & Energy are -75% correlated, Year & Popularity are 51% correlated. Then we made a few more visualizations to discover variable relationships. From the graphs we can see that popularity is 51% correlated with year. All other correlations are below 50%. Its -40% correlated with acousticness, -30% correlated with instrumentalness, 33% correlated to energy, 34% correlated with loudness, 12% correlated with dancibility, 15% correlated with explicit. All other variables are very little to no correlations. We can also see some skewness in our data, and class imbalance, such as acousticness, has more data close to 0 and 1 than other data. For further analysis, class balancing and scaling is required. We decided to use standardized scaling methods to scale our data, and apply different sampling methods to improve class balancing.

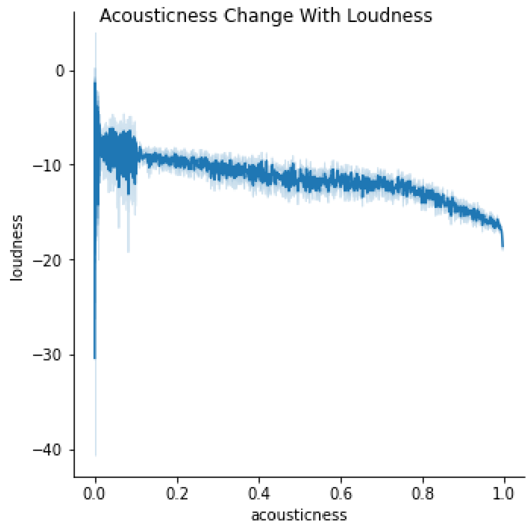

Joinplot for Valence & Danceability
danceability
valence
0.0
0.2
0.4
0.6
0.8
1.0

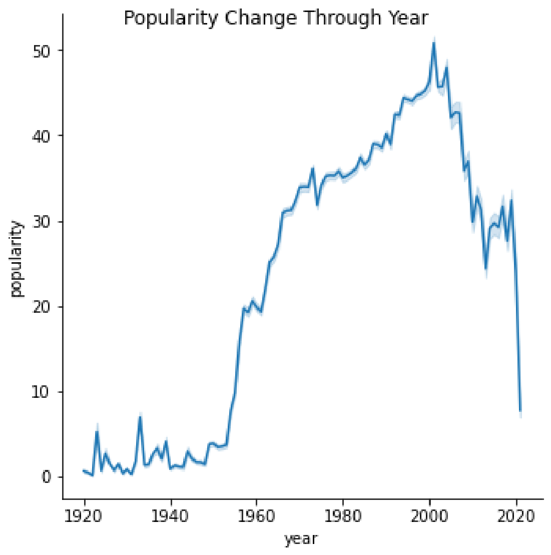

Energy & Loudness Yearly Changes
energy
loudness
year
1920
1960
2000
2040
−60
−50
−40
−30
−20
−10
0


Energy Change Through Years
energy
year
1920
1940
1960
1980
2000
2020


Acousticness Change Through Years
acousticness
year
1920
1940
1960
1980
2000
2020

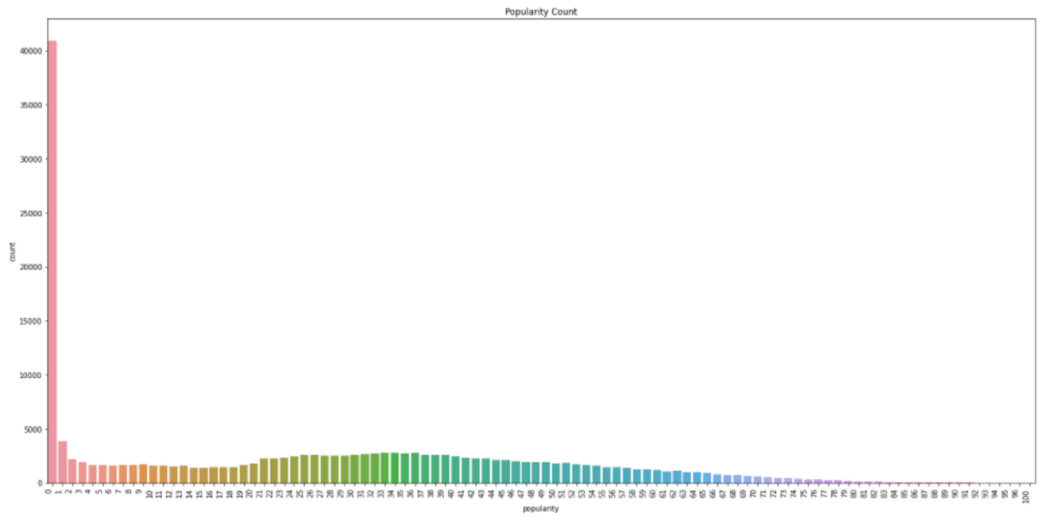

Acousticness Change With Energy
energy
acousticness

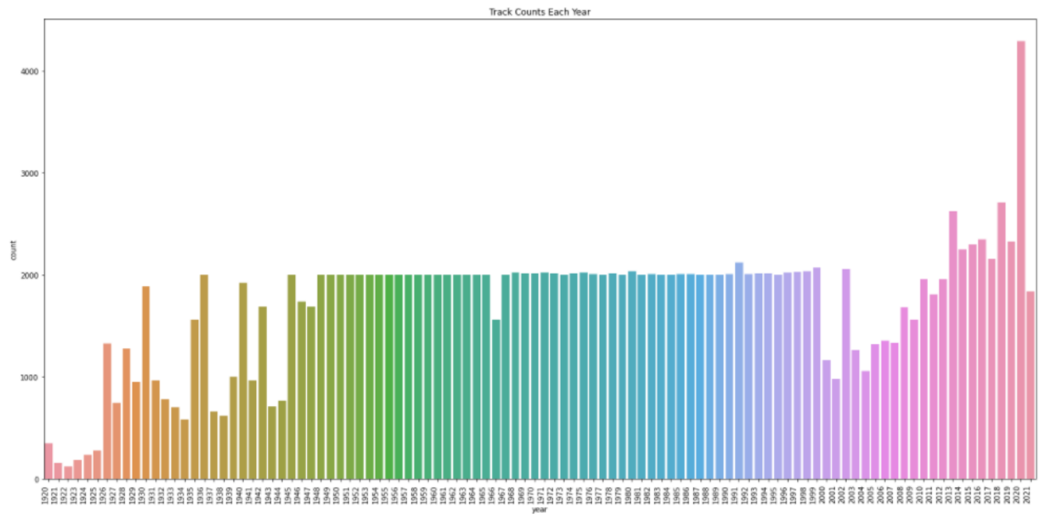

Acousticness Change With Loudness
loudness
acousticness
0
−10
−20
−30
−40



From the visualizations above we can see, the larger the valence, the more danceable the song is. High valence tracks are happy, cheerful and positive, and are more danceable. There are more positive tracks than low energy songs. And for loudness, it is mainly in a range of -10 to -40. It does match with common sense that the louder the track is, the more energy is being released. For energy, from 1920 to 1940, energy is in a range from 0.2 to 0.4, after 1940, there is a surge in energy, which means there is a big shift in music type. After matching with music history, we found out from 1920 - 1945, main street music is more jazz and blues, from 1945 - 1995, music trend shifted to a more upbeat type of music, such as pop, vocal pop, rock... After 1995, music style transited to rock, punk rock, metal, hip hop… Acousticness decreased after 1940, songs with high acousticness will consist mostly of natural acoustic sounds (think acoustic guitar, piano, orchestra, the unprocessed human voice). Songs with a low acousticness will consist of mostly electric sounds (think electric guitars, synthesizers, drum machines, auto-tuned vocals). The larger the acousticness the smaller the energy. Large acousticness means there's more guitar, human vocal, piano, which has less noise compared with electric guitar, auto tune… We can see loudness is in the range of -5 to -20, there are no significant changes in loudness, which indicates when tracks are being produced, the loudness is controlled within this range.

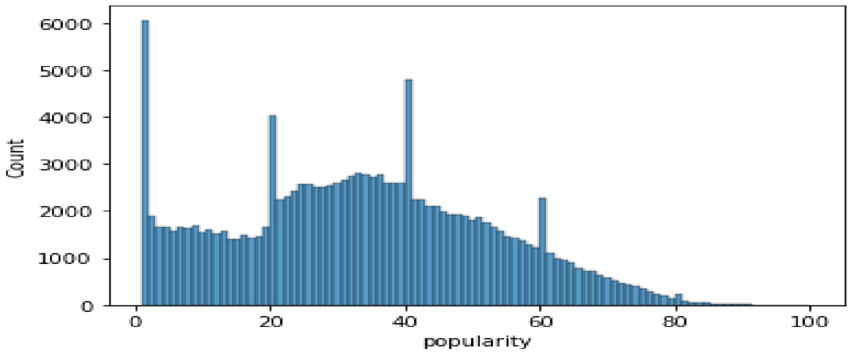
For target variable "popularity", there is more rating 0 than the rest of the data, after removing popularity 0, the most rating is popularity 1. From the previous analysis we can see that popularity started to decay after 2000. One interest finding in exploratory analysis is, there are most records being rated in 2020, since the popularity rating started to drop after 2000 and almost reached the lowest score in 2020, we can assume all the tracks being rated in 2020 have low rating scores.



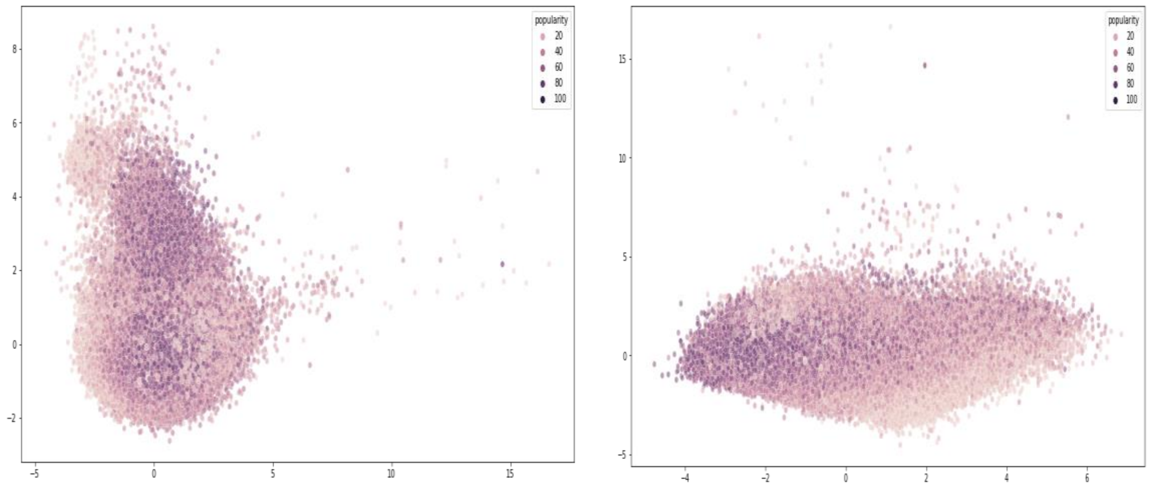


Target Variable Analysis

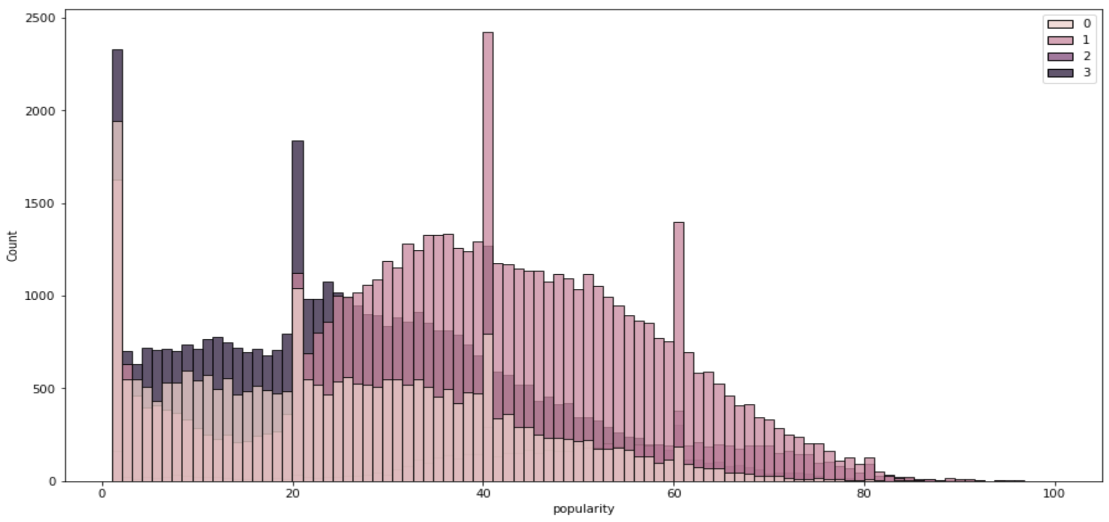
Our target variable is popularity. It is a metric that ranges between 0-100. Upon further analysis we realized that the popularity score of 0 actually represents unknowns in many cases, including new artists or new content on the app. As a group we decided to reserve the cases with the popularity score of 0 as a set aside test set. This way we can later attempt to predict their popularity with future (2021) data, the dataset covers up to dates in 2020. After this decision we got a decent distribution with a medium right tail.



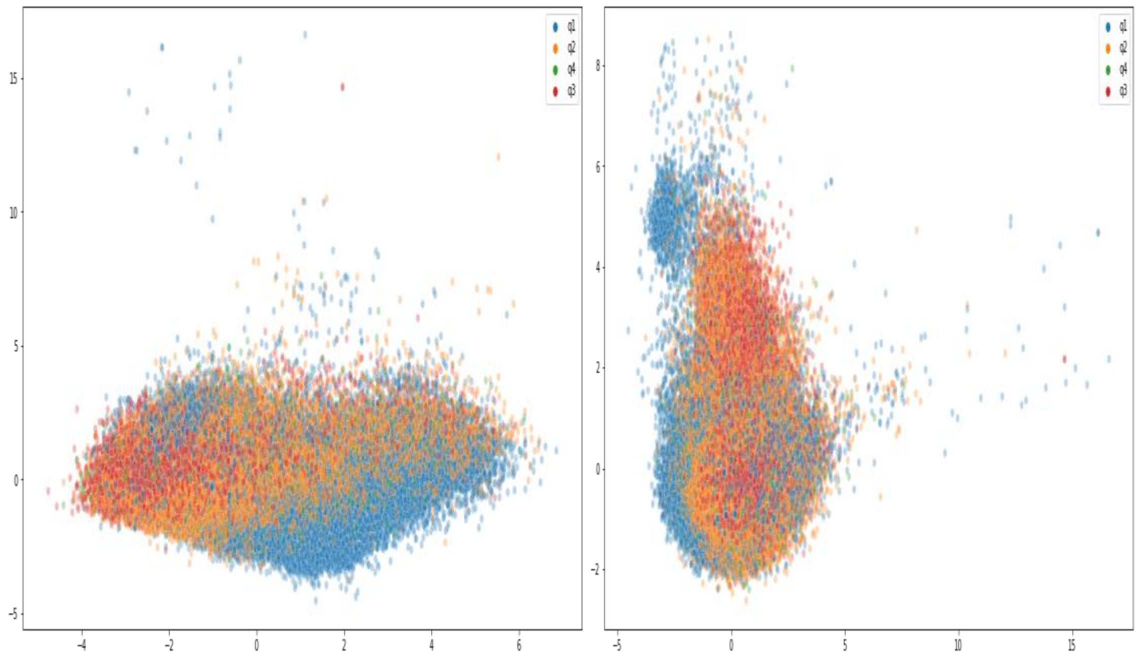
The next step was handling the target variable for classification tasks. The reason why we chose classification over regression for our dataset is because in reality popularity assessment happens between tiers and not exact numbers. When we think about popularity we don't usually think about tracks that are let's say 5 points apart, instead we say this track is very popular, or not popular at all. Starting from here we considered two different approaches, the first one was a clustering approach with PCA visualization.



The first plot covers about 35% variance and the second covers about 20% with the three PCs covering a total of 45% variance. According to the plots we wouldn't have any clear clusters capturing a certain set of popularity scores, they are mostly bundled together. The coloring is based on the popularity scores of individual points and here you can see that higher values appear with the lower ones but there is a tiered shift of colors. Meaning, you don't get from purple to light pink directly without having the middle values. Using KMeans clustering I created 4 clusters to capture a good distinction between the popularity scores. However, as you see below the clusters represented an intermingled set of popularity scores with some transition between varying popularity scores.



The second approach we discussed was bucketing based on quartiles. For this I used the PCA visualization from earlier but I colored the individual points based on their quartiles instead of using popularity ranges. In both PCA plots, we observed that the majority of transitions happened between quartiles that are away from each other by one. For instance the majority blue point (q1) clusters were followed by orange (q2) etc.



Since the overall variance differences are better captured with this representation, as a group, we decided to label popularity based on quartiles.

Research questions

Research question - 1

Does the attributes' significance change throughout the year?

EDA:

From the previous analysis we can see that popularity started to decay after 2000. One interest finding in exploratory analysis is, there are most records being rated in 2020, since the popularity rating started to drop after 2000 and almost reached the lowest score in 2020, we can assume all the tracks being rated in 2020 have low rating scores.

Research question - 2

How much unidentified variance can we get from our best model? Can we reach to some conclusions about the non-music related factors and their effects?

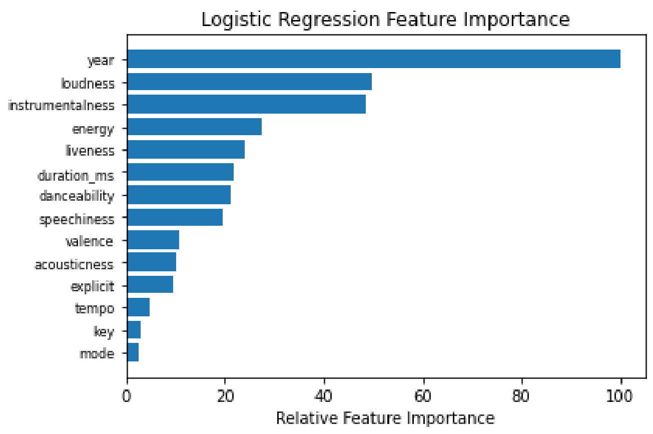
AdaBoost:

AdaBoost model optimization resulted in an overall accuracy score of 65.2%. Taking this as the best performing model we have 34.8% unidentified variance. This could be due to a lot of missing predictors such as PR and Marketing budgets. Especially at the higher popularity cases, our models struggled to properly classify with the available predictors. Knowing the music industry domain, we can make the assumption that more popular artists have access to higher marketing and social media budgets and therefore this unidentified parameter plays a bigger role in higher level popularity predictions.

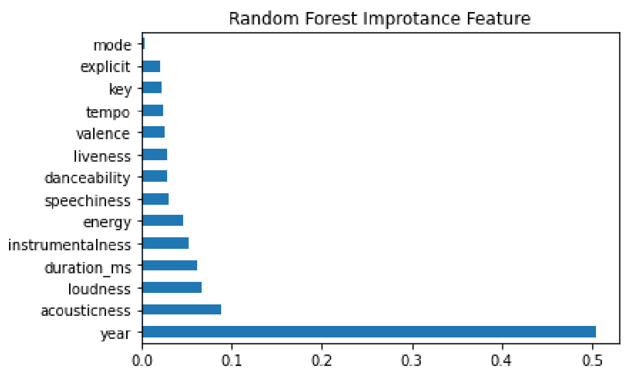
Research question - 3

Based on the result of machine learning models, what determines whether a song is popular or not?

Logistic regression:



Random Forest:



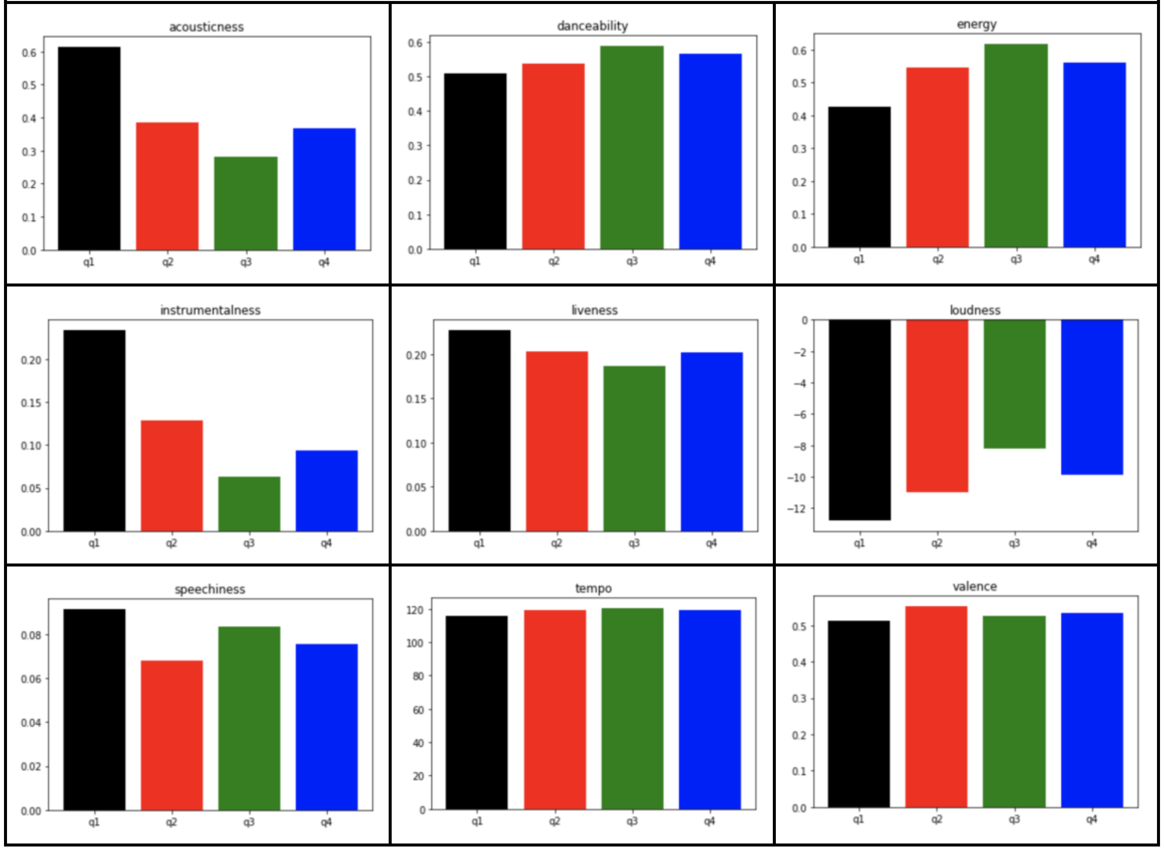
Research question - 4

How would the unpopular songs be possibly remixed based on the audio features in order to get better popular ratings?

Methodology: As the targets are divided into 4 groups ranging from low to high - q1,q2,q3,q4. This would benefit this research question in terms of pattern comparison. In order to differentiate the popular and unpopular songs, The lowest popularity of q1 is used as a benchmark to compare with q2 - q4 (more popular). The range of q2 - q4 is used as the recommended scale where the scale of remixed music should be located.

The graph below shows the mean values of each audio feature. It is observed that q1 in each feature has either higher or lower value from the rest of q2, q3 and q4 in the same feature. Only tempo, valence and danceability has q1 similar to q2,q3 and q4. Thus, they would not be considered in the summarized table.

Mean values for audio features



```
Summarized table
+----+------------------+-----------------------+-------------------------+------------------------+
|    | Variable         |   Unpopular Mean Scale |   Recommended Min Scale |   Recommended Max Scale |
|----+------------------+-----------------------+-------------------------+------------------------|
|  0 | acousticness     |              0.6148    |               0.280323  |              0.384852  |
|  1 | energy           |              0.425602  |               0.545132  |              0.618329  |
|  2 | instrumentalness |              0.233986  |               0.0632217 |              0.128539  |
|  3 | liveness         |              0.227538  |               0.186045  |              0.203293  |
|  4 | loudness         |            -12.8089    |             -11.0122    |             -8.21085   |
|  5 | speechiness      |              0.0914623 |               0.0678186 |              0.0832132 |
+----+------------------+-----------------------+-------------------------+------------------------+
```

From the summarized table above, we can see that the first column shows the value of q1, the lowest popular music. We can give suggestions to the music industry to remake this unpopular song based on our discovery of the recommended range of popular songs on 3rd and 4th columns. For example, energy could be adjusted from 0.4 to between 0.55 and 0.618.

Machine Learning

These following experiments are performed with the finalized datasets that have been modified including data scaling and variable selection. As discussed earlier we have a multi-class target variable which comprises q1, q2, q3 and q4. Therefore, we will be utilizing various algorithms such as decision tree, SVM, KNN and others in order to evaluate the performance with reference to our dataset. In addition, the scikit- learn package is used to build the machine learning object in our project.

Status	**Model**	**Experiment**
Current	Decision Tree	● Base model ● Gridsearch with base model ● Oversampled ● Gridsearch with base model
	SVM	● Default model using RBF on non-stratify dataset ● Use stratified data for training ● Hyperparameter tuning including kernel ● Analyze results for best model.
	KNN	● Tuning hyperparameter ● Standardizing the data ● Log transforming the data
	Adaboost	● Default model ● Hyperparameter tuning with grid search with balanced accuracy ● Upsampled models ● Undersampled models
	Random Forest	● Base model ● Base model sample with smote ● Base model with smote & gridsearch
	Logistic Regression	● Base model with cut off point popularity > 50 and < 50 ● Base model with cut off point popularity > 70 and < 70 ● Base model (popularity>70) with PCA ● Base model (popularity >70) with SMOTE & Gridsearch ● Smote & gridsearch model with PCA

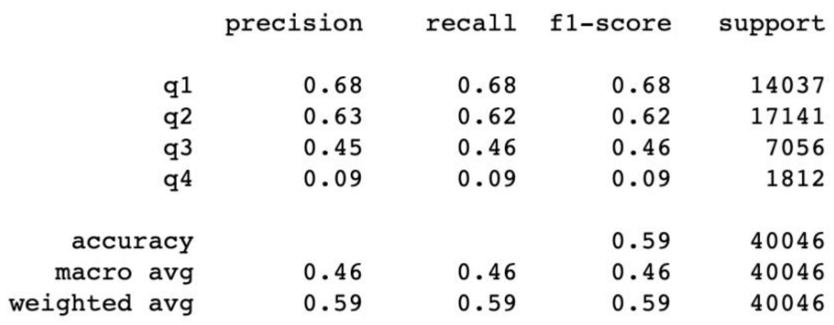
Experiment results

(1) Decision Tree Classification

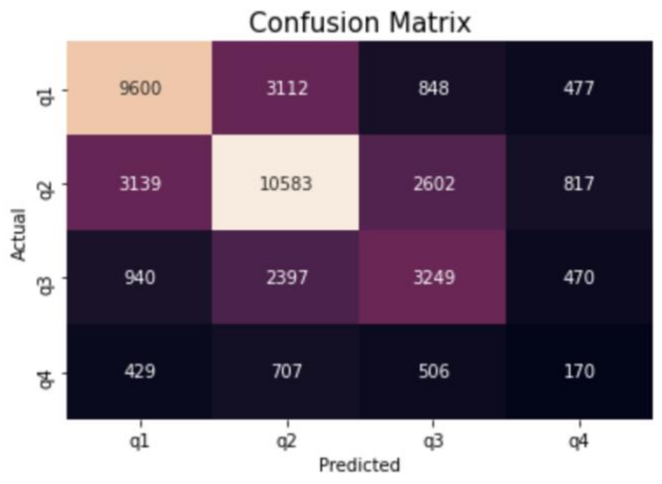
This technique is one of the most used supervised learning methods for multiclass classification. The experiments are performed with default setting, tuned model and resampling methods with both models

1.1 Decision tree experiment with default setting

Model	DecisionTreeClassifier(random_state=2020)	
	Accuracy	**Balanced Accuracy**
Train	0.9957725978723859	0.5893722219447636
Validation	0.9910643905958434	0.46389838462755434



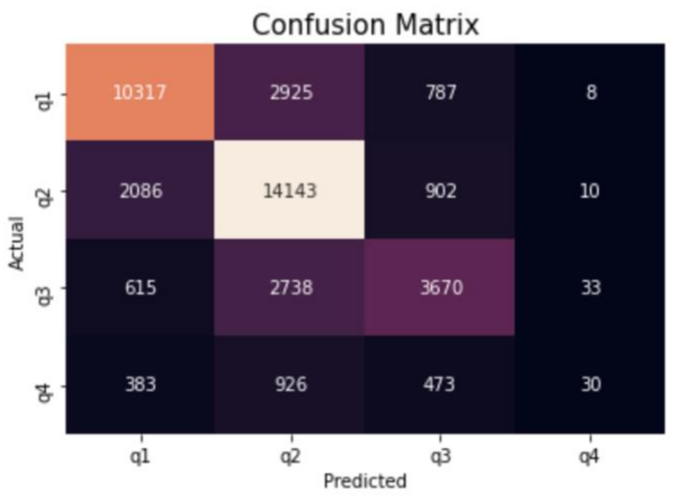
```
              precision    recall  f1-score   support

          q1       0.68      0.68      0.68     14037
          q2       0.63      0.62      0.62     17141
          q3       0.45      0.46      0.46      7056
          q4       0.09      0.09      0.09      1812

    accuracy                           0.59     40046
   macro avg       0.46      0.46      0.46     40046
weighted avg       0.59      0.59      0.59     40046
```



1.2 Decision tree experiment with the tuned hyperparameter

Model	DecisionTreeClassifier(max_depth=10, min_samples_split=20, random_state=2020)	
	Accuracy	**Balanced Accuracy**
Train	0.7238810762216658	0.5393672293365382
Validation	0.7031913299705339	0.5241912087515829

```
              precision    recall  f1-score   support

          q1       0.77      0.73      0.75     14037
          q2       0.68      0.83      0.75     17141
          q3       0.63      0.52      0.57      7056
          q4       0.37      0.02      0.03      1812

    accuracy                           0.70     40046
   macro avg       0.61      0.52      0.53     40046
weighted avg       0.69      0.70      0.69     40046
```

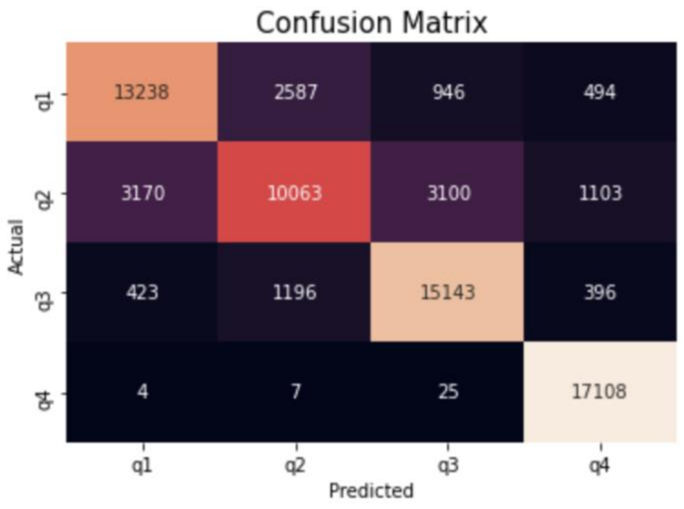


1.3 Default model with the resampling technique

Class	q1	q2	q3	q4
Original	57502	46510	23558	5914
Resample	57502	57502	57502	57502

Model	DecisionTreeClassifier(random_state=2020)	
	Accuracy	Balanced Accuracy
Train	0.9967702866370609	0.9967648230034952
Validation	0.805066446386389	0.806088752480949

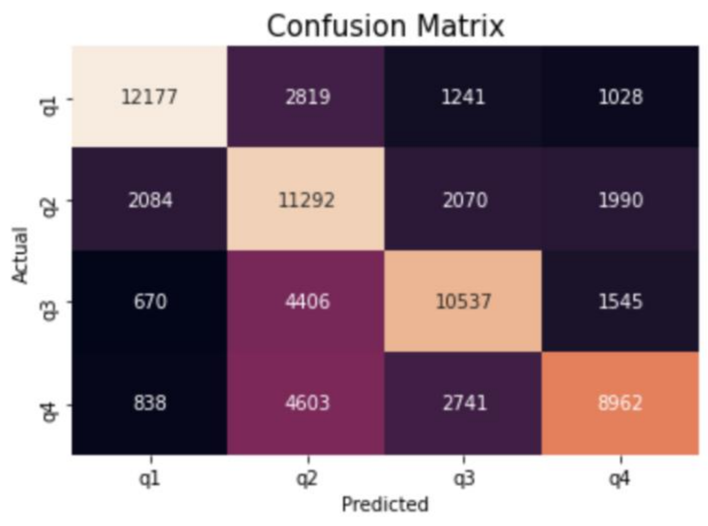
```
              precision    recall  f1-score   support

          q1       0.79      0.77      0.78     17265
          q2       0.73      0.58      0.64     17436
          q3       0.79      0.88      0.83     17158
          q4       0.90      1.00      0.94     17144

    accuracy                           0.81     69003
   macro avg       0.80      0.81      0.80     69003
weighted avg       0.80      0.81      0.80     69003
```



1.4 Tuned model with the resampling technique

Model	DecisionTreeClassifier(max_depth=14, min_samples_split=8, random_state=2020)	
	Accuracy	**Balanced Accuracy**
Train	0.6653520077016242	0.6655049439783803
Validation	0.6226975638740344	0.6224474223284273

```
              precision    recall  f1-score   support

          q1       0.77      0.71      0.74     17265
          q2       0.49      0.65      0.56     17436
          q3       0.64      0.61      0.62     17158
          q4       0.66      0.52      0.58     17144

    accuracy                           0.62     69003
   macro avg       0.64      0.62      0.63     69003
weighted avg       0.64      0.62      0.63     69003
```



Summarized table

Accuracy				
Data/ Model	**Default Model**	**Tuned Model**	**Default + Resampling**	**Tuned + Resampling**
Train	0.995773	0.723881	0.99677	0.665352
Validation	0.589372	0.703191	0.805066	0.622698

Balanced Accuracy				
Data/ Model	**Default Model**	**Tuned Model**	**Default + Resampling**	**Tuned + Resampling**
Train	0.991064	0.539367	0.996765	0.665505
Validation	0.463898	0.524191	0.806089	0.622447

Analysis

As mentioned earlier, this data set consists of multi-class targets: q1, q2, q3 and q4, each label has a different amount of data points. The minority of q4 indicates a severe imbalance class which might introduce a biased prediction for the greater major classes. For the metric measurement on this model, balanced accuracy is chosen to evaluate the model performance since it is applicable for both binary and multi-class classification. This metric put higher weight to accurate prediction of smaller classes, therefore it is reasonable to use in this imbalance class.

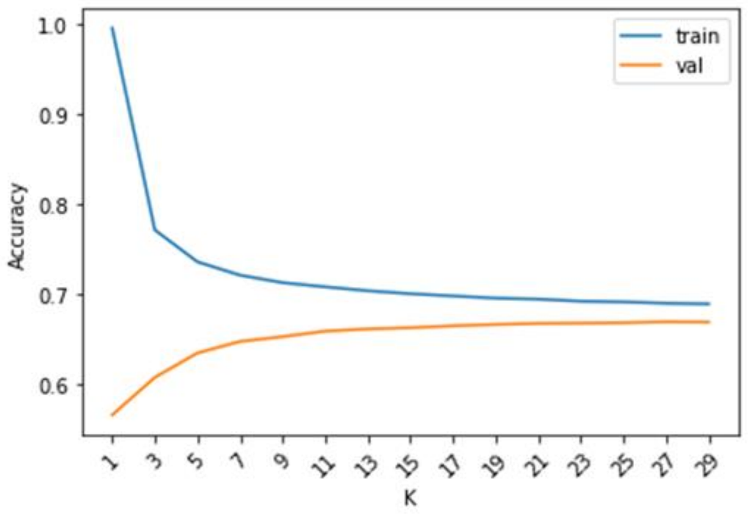
From the balanced accuracy table above, it shows that the first model with default setting has overfit the training data, the balanced accuracy on validation has around 50% lesser than on training. The overfitting gets improved when the classifier undergoes hyperparameter tuning. Even though the balanced accuracy on training is reduced but the balanced accuracy on validation is increased resulting in both values becoming closer. This implies that the model is generalized so well.

In addition, oversampling technique is used to get a better performance score. The minor classes on the training set are duplicated to get the same size as the majority. The results illustrate that the overall performance is relatively greater including overfitting, accuracy and balanced accuracy. The accuracy and balanced accuracy for the default model with resampled data get approximately 30% improvement on the validation data, from 58.9% to 80.5% accuracy and from 46.38% to 80.61% balanced accuracy. The generalization of this classifier is even better when we consider the tuned model with resampling but the smallest gap belongs to tuned model with normal data.

(2) K-nearest Neighbors

K-nearest neighbors (KNN) is a commonly used machine learning model that works based on distances among data points. The experiments include tuning hyperparameter (the value of k), performing on standardized data, and performing on log-transformed data.

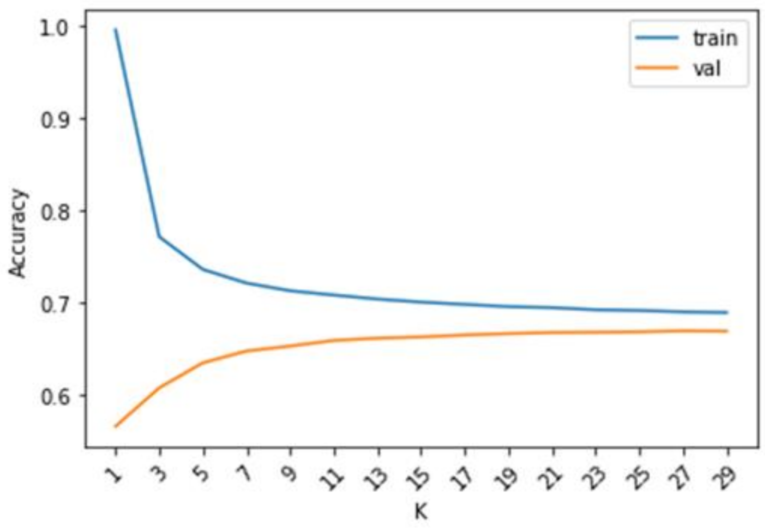
2.1 K-nearest neighbors experiment with standardized data



Summarized table

Accuracy															
K / **Data**	**1**	**3**	**5**	**7**	**9**	**11**	**13**	**15**	**17**	**19**	**21**	**23**	**25**	**27**	**29**
Training	**99.57**	**77.08**	**73.53**	**72.04**	**71.21**	**70.74**	**70.30**	**69.97**	**69.72**	**69.48**	**69.37**	**69.14**	**69.06**	**68.92**	**68.84**
Validation	**56.47**	**60.65**	**63.37**	**64.66**	**65.20**	**65.80**	**66.05**	**66.19**	**66.39**	**66.56**	**66.67**	**66.70**	**66.75**	**66.85**	**66.82**

2.2 K-nearest neighbors experiment with log-transformed data



Summarized table

Accuracy															
K Data	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29
Training	99.57	77.08	73.53	72.04	71.21	70.74	70.30	69.97	69.72	69.48	69.37	69.14	69.06	68.92	68.84
Validation	56.47	60.65	63.37	64.66	65.20	65.80	66.05	66.19	66.39	66.56	66.67	66.70	66.75	66.85	66.82

Analysis

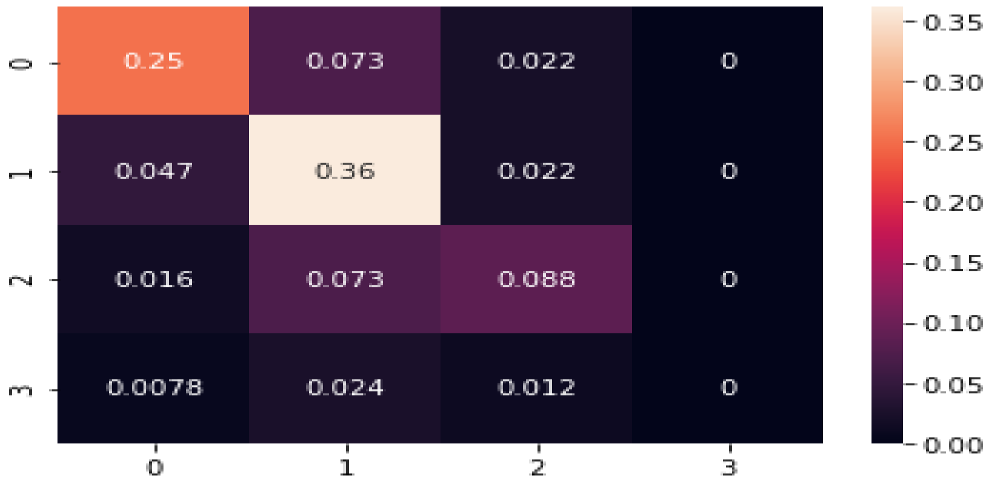
It seems that the performance using log-transformed data is exactly the same as using standardized data. This is most likely because the dataset does not contain outliers and extreme values. Compared to the results of using decision trees and SVM, the validation accuracy is lower than decision tree and close to SVM. An explanation is that KNN does not have any training process, so the only thing that affects the performance is the quality of data. A possible solution to improve the performance is to work on the data and try to improve the quality of the data.

(3) AdaBoost Algorithm

The first ensemble method we tried was AdaBoost. Due to the relatively lower accuracy scores of non-ensemble techniques this algorithm was promising due to boosting technique bias reduction. Experiments included base model testing, grid search optimization for the most promising base model, upsampling, and undersampling.

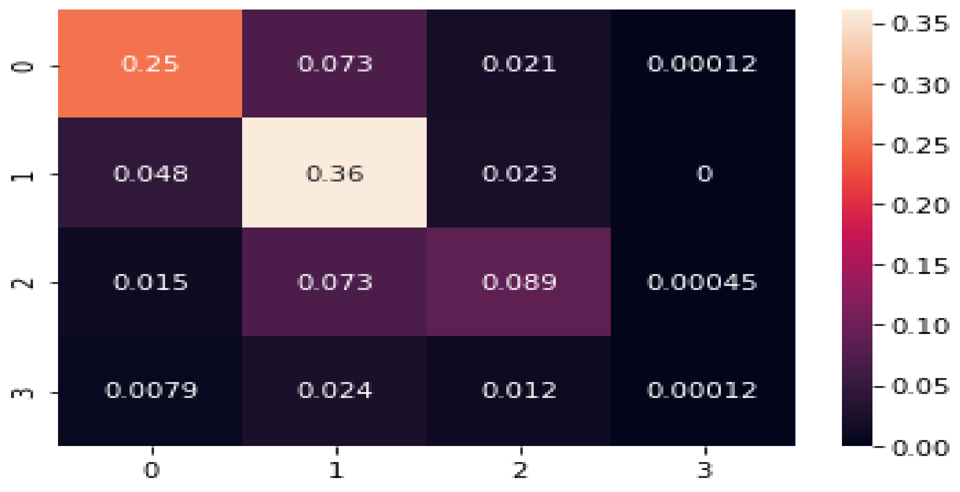
3.1 Base Model:

Model	AdaBoostClassifier()	
	Accuracy	**Balanced Accuracy**
Train	0.7015026006549798	0.5159376705599802
Validation	0.7024921340458473	0.515637639224012



3.2 Grid Search Model:

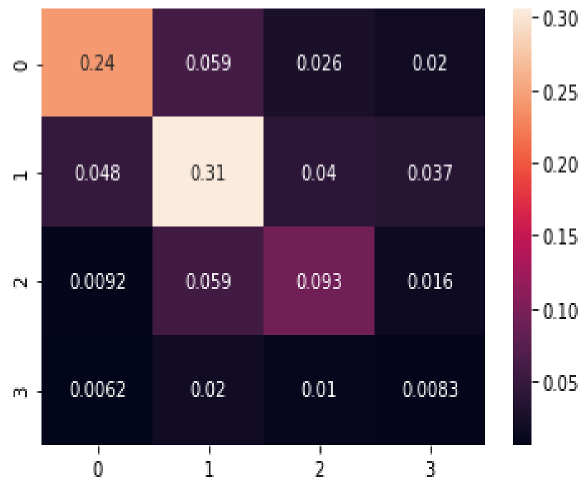
Model	AdaBoostClassifier(learning_rate=1, n_estimators=170)	
	Accuracy	**Balanced Accuracy**
Train	0.7021554399708898	0.5178667330259498
Validation	0.7036907556310243	0.5180419737447507

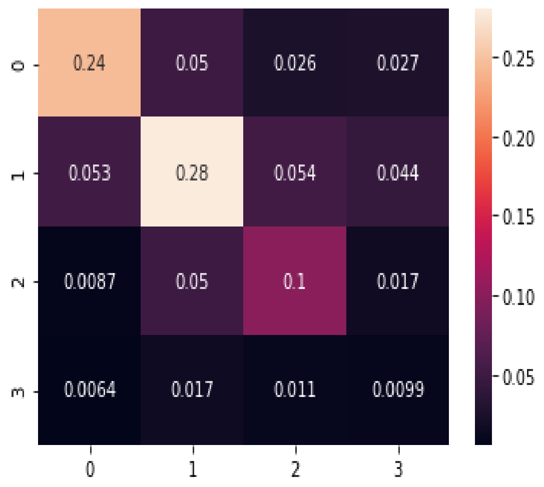


3.3 Upsampled Model

Upsampled every class by the majority class using replacement.

Model	AdaBoostClassifier()	AdaBoostClassifier(learning_rate=1, n_estimators=170)	AdaBoostClassifier(learning_rate=0.5, n_estimators=290)
	Base Accuracy	**First Grid Search Accuracy**	**Grid Search Accuracy**
Training	0.6482159292793082	0.6330721976069693	0.6486547229178706
Validation	0.6493282724866404	0.6348698996154423	0.6505268940718174

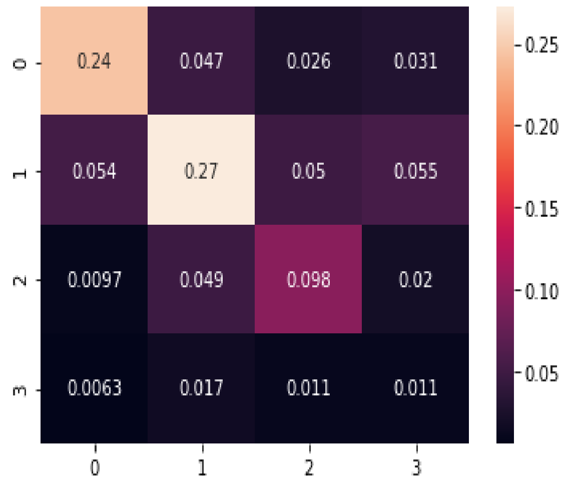


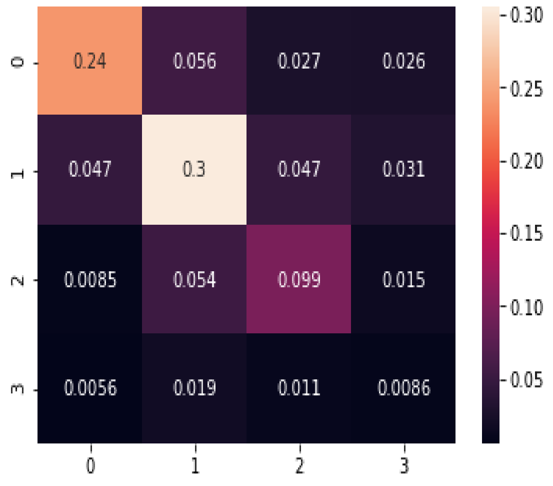


3.4 Undersampled Model

Undersampled every class by the minority class without replacement.

Model	AdaBoostClassifier()	AdaBoostClassifier(learning_rate=0.5, n_estimators=110)
	Base Accuracy	**Grid Search Accuracy**
Training	0.6224876388621332	0.6498212718594149
Validation	0.624856415122609	0.6524996254307547





AdaBoost Analysis

The non-ensemble models we experimented with had accuracy scores capped around 70%. Although for a 4-class problem this is a relatively good score, there is a good chance that there might still be room for improvement if we manage to reduce the potential bias. Boosting algorithms in general is suitable for this task and for this reason Adaboost was a good candidate.

The first experiment was with the default parameters of the AdaBoost classifier. As mentioned before, we have a heavy class imbalance issue with quartile 4 popularity scores. For this reason, experiment 1 actually failed to classify them while achieving a validation accuracy of 70.2%. Balanced accuracy was a more suitable candidate for making sure of proper classification of the minority class. Thus, for experiment 2 I ran a grid search with the balanced accuracy as the target metric. Although not perfect, the grid search version of Adaboost was able to classify some of the minority class with 70.4% accuracy with no overfitting.

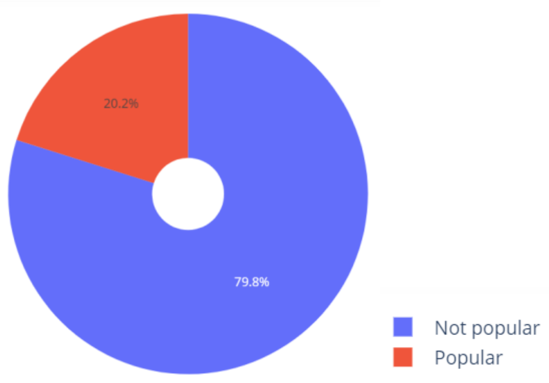
Although promising for 3 out of 4 labels, Adaboost struggled with the minority class. To address this issue I ran two more experiments using upsampling and undersampling. Upsampling was done with replacement based on the majority class q2, and the undersampling resampled the data while keeping every class the size of the minority class q4. Both of these methods presented similar results with a drop in accuracy with upsampling getting a comparably better overall accuracy of 64.9%.

Combining all the experiments, it is hard to determine a best performing model for Adaboost. If we choose to prioritize a more balanced classifier, upsampled would be ideal. However, this choice would lead to a tradeoff between quartiles 2 and 4 seen in the accuracy scores and the confusion matrices provided. The unexplained variance can also be due to the varying marketing and social media budgets which might be more significant for the higher popularity cases. In this scenario, it might be better to use the regularly sampled model and say that the model isn't really suitable for high popularity cases.

(4) Logistic regression

4.1 Base Model with cut off point popularity > 50

The first experiment we set cut off points as popularity > 50 is popular (1), less than 50 is not popular (0). One thing about the data is class imbalance, so overall we have more not popular data than popular data.

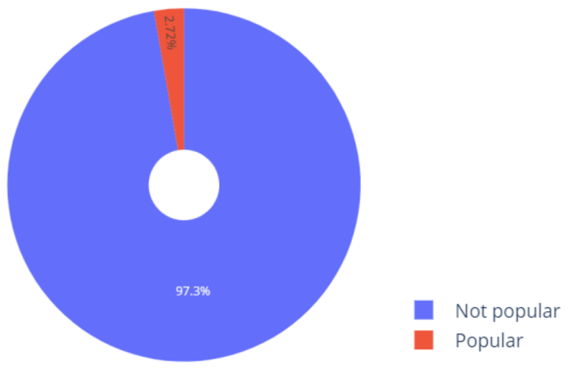


```
Training Score of Logistic Regression is: 0.8432329459106573

Validation Score of Logistic Regression is: 0.8426060030964391

R2 Score of Logistic Regression is: 0.024729890181181435

Mean Squared Error of Logistic Regression is: 0.1573939969035609

Mean Absolute Error of Logistic Regression is: 28.978649553014034
```

```
              precision    recall  f1-score   support

           0       0.87      0.94      0.91     31944
           1       0.67      0.44      0.53      8102

    accuracy                           0.84     40046
   macro avg       0.77      0.69      0.72     40046
weighted avg       0.83      0.84      0.83     40046
```

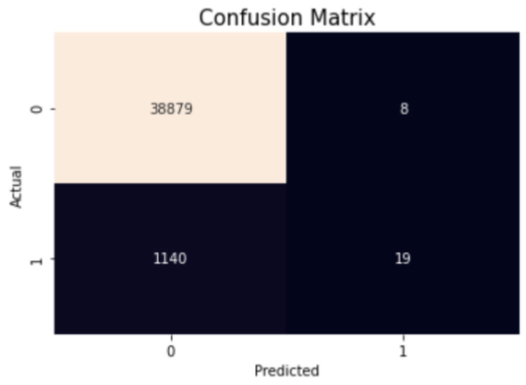
4.2 Base Model with cut off point popularity > 70

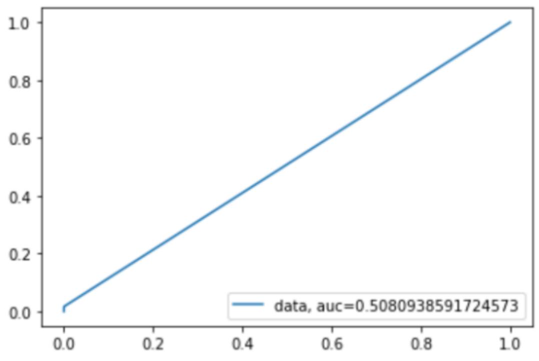
Since we have less data being rated as popular, it might improve the accuracy from narrowing to popular data. So we set the cut off point to 70. Above 70 is popular (1), less than 70 is not popular (0).



```
Training Score of Logistic Regression is: 0.9733406108863631

Training Score of Logistic Regression is: 0.971332967087849

R2 Score of Logistic Regression is: -0.020030493432299012

Mean Squared Error of Logistic Regression is: 0.028667032912151027

Mean Absolute Error of Logistic Regression is: 7.259351745492683
```

```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99     38887
           1       0.70      0.02      0.03      1159

    accuracy                           0.97     40046
   macro avg       0.84      0.51      0.51     40046
weighted avg       0.96      0.97      0.96     40046
```





4.3 Base model with Gridsearch & PCA

Apply PCA to improve accuracy and AUC. We chose 2 components.

```
Training Score of Logistic Regression is: 0.9734904428605065

R2 Score of Logistic Regression is: -0.029804304780517876

Mean Squared Error of Logistic Regression is: 0.028941717025420765

Mean Absolute Error of Logistic Regression is: 7.380137841482296
```

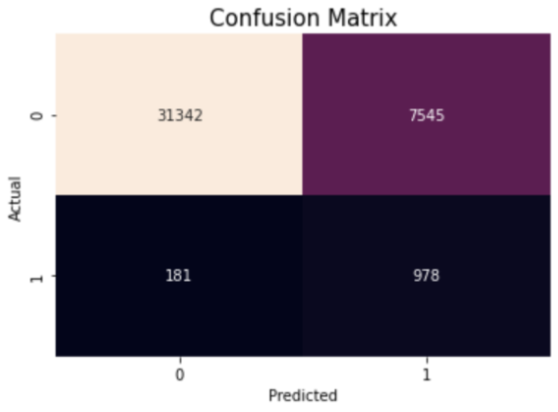
```
              precision    recall  f1-score   support

           0       0.97      1.00      0.99     38887
           1       0.00      0.00      0.00      1159

    accuracy                           0.97     40046
   macro avg       0.49      0.50      0.49     40046
weighted avg       0.94      0.97      0.96     40046
```

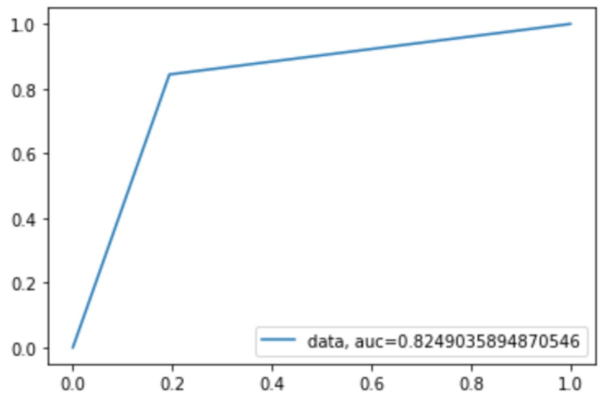
Predicted	**0**
Actual	
0	38887
1	1159

4.4 Base model with SMOTE sampling

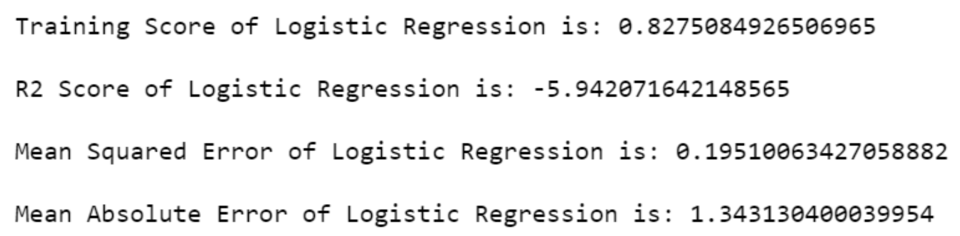
From the previous PCA experiment, we can see that there's no positive classes being predicted. So we applied SMOTE sampling to the data.

```
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     38887
           1       0.11      0.84      0.20      1159

    accuracy                           0.81     40046
   macro avg       0.55      0.82      0.55     40046
weighted avg       0.97      0.81      0.87     40046
```

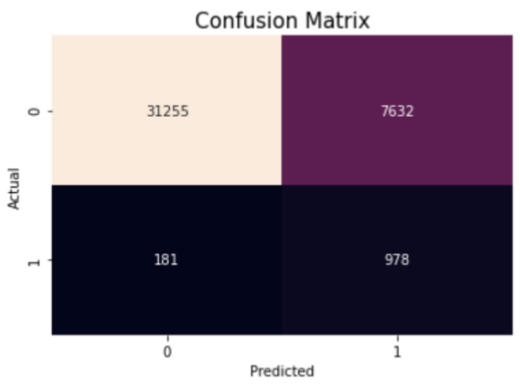


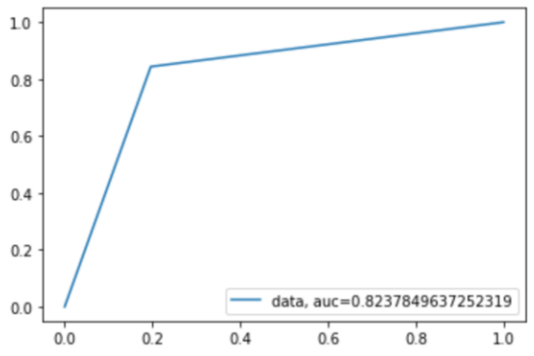


4.5 Logistic regression after SMOTE & PCA



```
Training Score of Logistic Regression is: 0.8275084926506965

R2 Score of Logistic Regression is: -5.942071642148565

Mean Squared Error of Logistic Regression is: 0.19510063427058882

Mean Absolute Error of Logistic Regression is: 1.343130400039954
```





Analysis

For logistic regression, we split predict variable popularity into 2 classes. First attempt we set the threshold to popularity larger than 50 and smaller than 50, resulting in a train score of 0.84, R2 score of 0.024, MSE of 0.16, MRSE of 29. For classification report, recall score of 0.94 for class 0 and 0.44 for class 1. We can see there is still room to improve, when the recall score for 2 classes are 0.5 away from each other, it’s possibly caused by class imbalance.

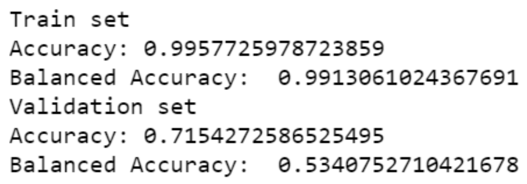
Second attempt we used a threshold larger than 70 and below 70, resulting in a train score of 0.97, R2 0, MSE 0.03, RMSE 7, which improved a lot from previous. For classification report, recall score of 1 and 0.02, which still indicates it’s not the best model even with auc of 0.5. Clearly the way we split data worsened class imbalance problems, there is only 2% tracks being rated popularity above 70, even with improved accuracy, the recall score decreased for class 1. Then we tried to use PCA, but it didn't help and made the performance worse. It seems there are no positive classes being predicted at all when we use PCA and only contain 2 components.

Since class imbalance is the biggest issue in our data, next step we tried to use smote to help class imbalance issues. Resulting Training Score of Logistic Regression is: 0.83, R2 Score of Logistic Regression is: -5.94, Mean Squared Error of Logistic Regression is: 0.19, Mean Absolute Error of Logistic Regression is: 1.34. For the classification report, recall score of 0.81 and 0.84, even with less training score, smaller recall for negative class, we improved recall for class 1 from 0.02 to 0.84, with an auc of 0.83. By sacrificing the accuracy, the model achieved a more balanced class with similar recall score.

(5) Random Forest Model

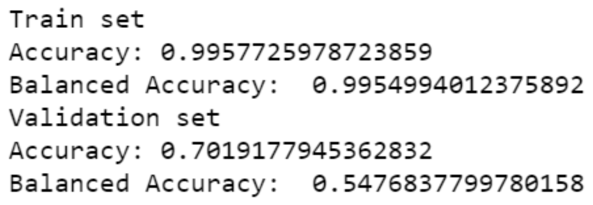
5.1 Base model

Base random forest model indicates overfitting, training accuracy is 0.99 but validation accuracy is only 0.72.

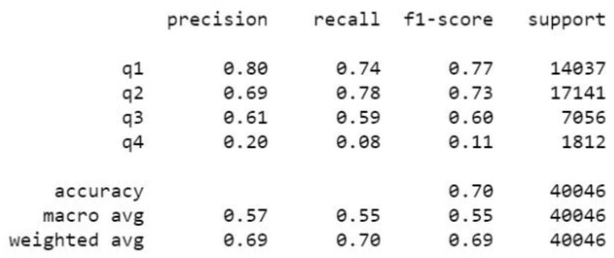


```
Train set
Accuracy: 0.9957725978723859
Balanced Accuracy:  0.9913061024367691
Validation set
Accuracy: 0.7154272586525495
Balanced Accuracy:  0.5340752710421678
```

5.2 Random Forest model with SMOTE

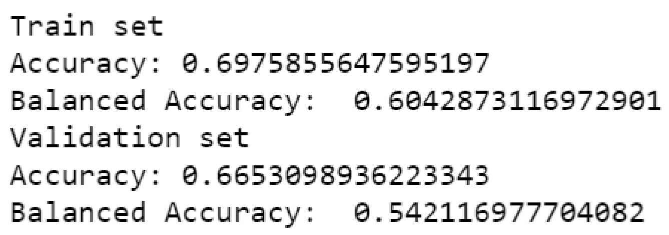


```
Train set
Accuracy: 0.9957725978723859
Balanced Accuracy:  0.9954994012375892
Validation set
Accuracy: 0.7019177945362832
Balanced Accuracy:  0.5476837799780158
```

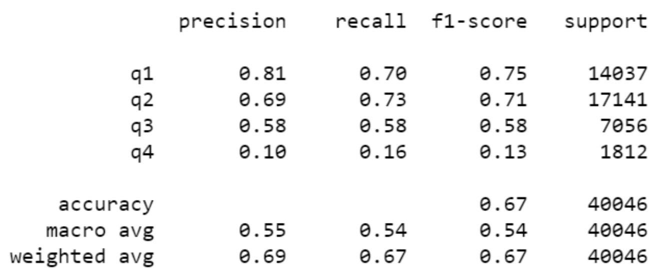


```
              precision    recall  f1-score   support

          q1       0.80      0.74      0.77     14037
          q2       0.69      0.78      0.73     17141
          q3       0.61      0.59      0.60      7056
          q4       0.20      0.08      0.11      1812

    accuracy                           0.70     40046
   macro avg       0.57      0.55      0.55     40046
weighted avg       0.69      0.70      0.69     40046
```

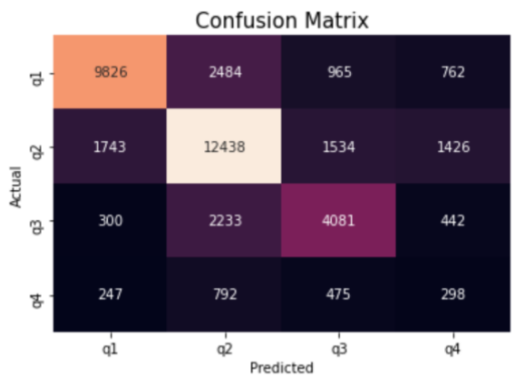
5.3 Random Forest model with SMOT & Gridsearch



```
Train set
Accuracy: 0.6975855647595197
Balanced Accuracy:  0.6042873116972901
Validation set
Accuracy: 0.6653098936223343
Balanced Accuracy:  0.542116977704082
```

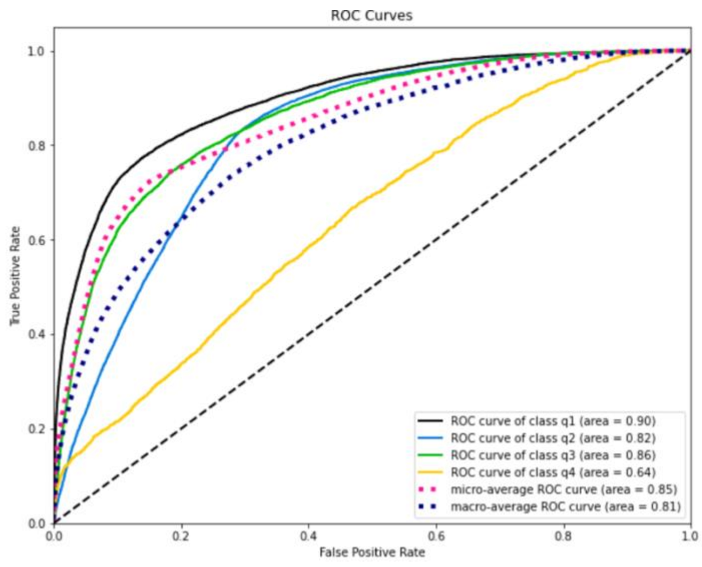


```
              precision    recall  f1-score   support

          q1       0.81      0.70      0.75     14037
          q2       0.69      0.73      0.71     17141
          q3       0.58      0.58      0.58      7056
          q4       0.10      0.16      0.13      1812

    accuracy                           0.67     40046
   macro avg       0.55      0.54      0.54     40046
weighted avg       0.69      0.67      0.67     40046
```



Confusion Matrix

Actual \ Predicted	q1	q2	q3	q4
q1	9826	2484	965	762
q2	1743	12438	1534	1426
q3	300	2233	4081	442
q4	247	792	475	298



Analysis

Random forest default model training set accuracy: 0.99, Balanced Accuracy: 0.99, Validation set, Accuracy: 0.72, Balanced Accuracy: 0.53. Model appears overfitting. Since we have class imbalance problems, we tried to use smote to balance the class. Result showing accuracy is not improved, model is still overfitted.

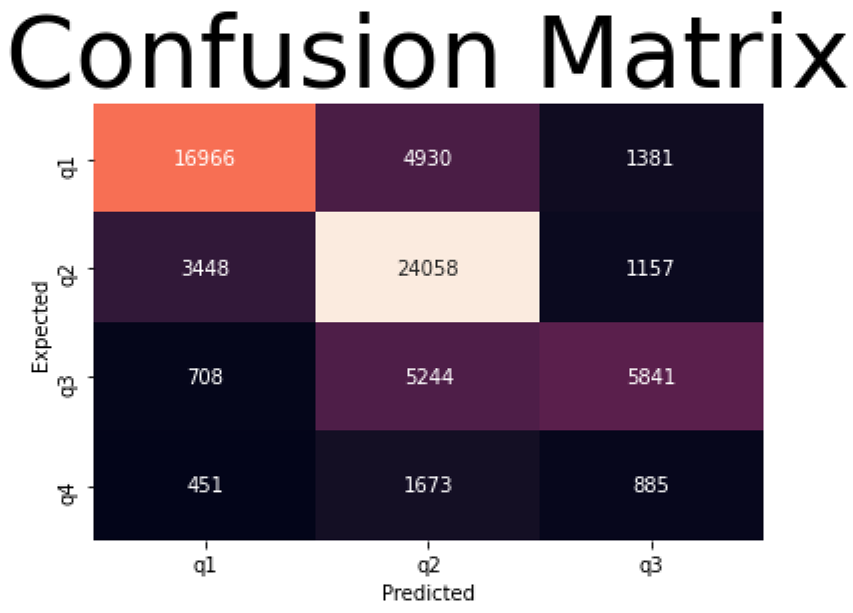
So we used grid search to do some tuning on parameters. The best parameters are {'max_depth': 12, 'min_samples_split': 12, 'n_estimators': 300}. Resulting in a Train set accuracy: 0.7, Balanced Accuracy: 0.6, Validation set accuracy: 0.67, Balanced Accuracy: 0.54, we sacrificed accuracy but the model is not overfitted. The recall score for q4 is improved from 0.08 to 0.16, since there is the least data in q4, recall score and ROC score will be less than q1, q2, q3. ROC is close to the previous model, q1 0.9, q2 0.82, q3 0.86, q4 is 0.64.

(6) Support Vector Classification

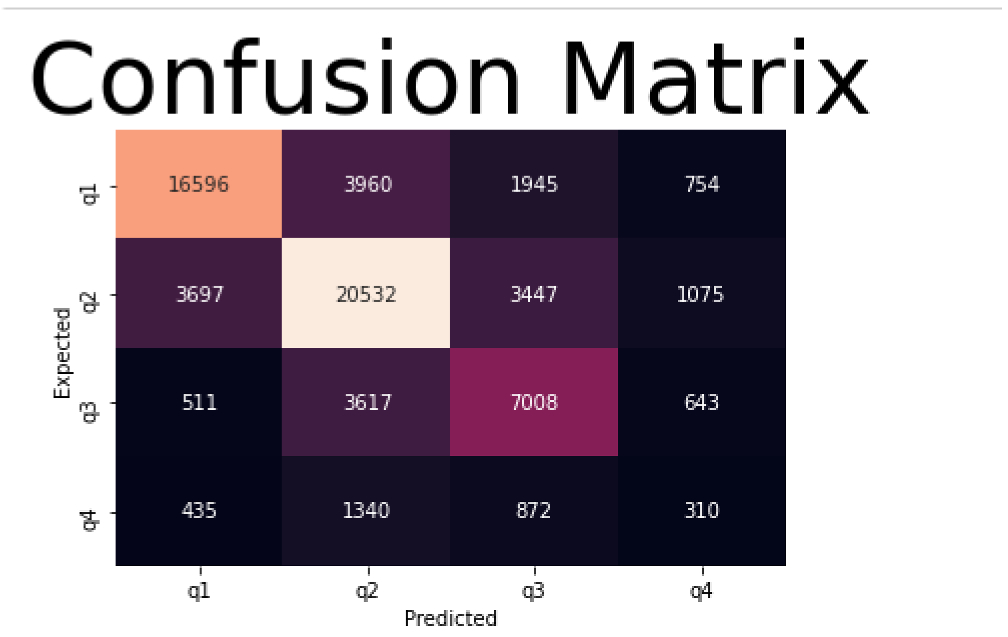
To perform SVC, the data was preprocessed to have the popularity into 4 quartiles. Then the data was standardized as SVCs modeling will perform best with normalized/standardized data.

The first model was split for training and validation using a 50% split with non-stratified data. This model has a good overall accuracy of 70.22 % but looking at the confusion matrix, one can see that 4th quartile did not have any classifications. Therefore, the next step was to test stratified data.

Confusion matrix for SVC with non-stratified data

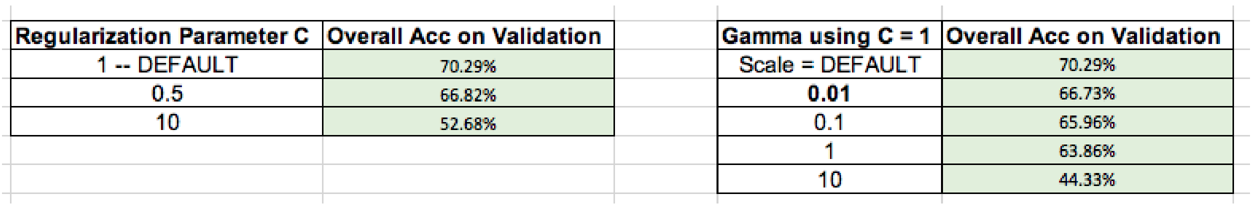


The 2nd model was done with the stratified data but included the hyperparameter of class weight being balanced. This netted out an overall accuracy of 66.6% and quartile 4 was classified.



To hypertune the model, the model was re-ran with class weight as none, the accuracy was higher at around 70% but again in the confusion matrix, the 4th quartile was not classified. Therefore class weight as balanced was kept as a hyperparameter. The next two hyperparameters to tune C and Gamma while keeping the class weight as balanced. The regularization parameter C was tuned first and then the best C was used to find the best Gamma. Using the table below, the default values for C=1 and the default gamma were the best.

Note: Gamma leverages the influence of a record, so the default is scaled which means the gamma is 0.071 based on the formula "1 / (n_features * X.var())". This makes sense as the gamma shrinks the accuracy was increasing for the model.



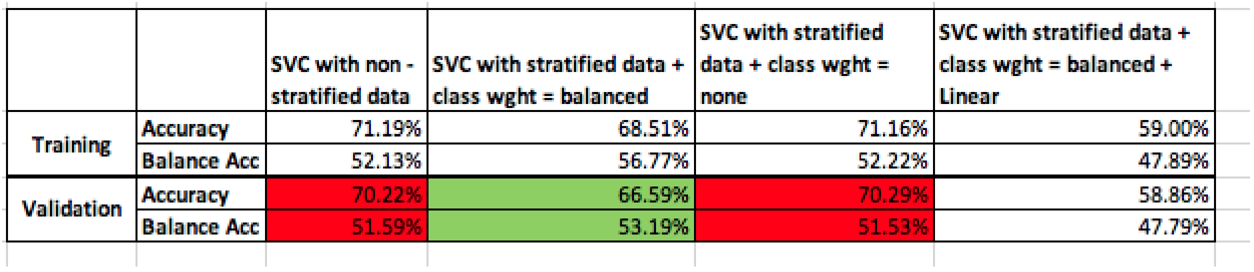
Regularization Parameter C	**Overall Acc on Validation**
1 -- DEFAULT	70.29%
0.5	66.82%
10	52.68%

Gamma using C = 1	**Overall Acc on Validation**
Scale = DEFAULT	70.29%
0.01	66.73%
0.1	65.96%
1	63.86%
10	44.33%

The table below for the comparison of the main models used for SVC shows that the ones in red had better accuracies but the confusion matrix shows no classification for the 4th quartile. Hence, the best model is the SVC using Stratified data with class weight as balanced with the other hyperparameters defaulted.

The last hypertuning was done using a 4th model to change the kernel to Linear to see if it can outperform the RBF kernel but it performed poorly so RBF was kept as hyperparameter for the SVC model.

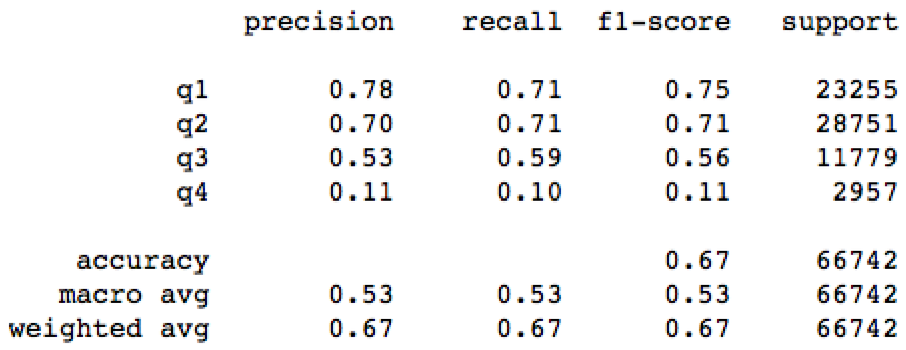
SVC Models:



		SVC with non - stratified data	SVC with stratified data + class wght = balanced	SVC with stratified data + class wght = none	SVC with stratified data + class wght = balanced + Linear
Training	Accuracy	71.19%	68.51%	71.16%	59.00%
	Balance Acc	52.13%	56.77%	52.22%	47.89%
Validation	Accuracy	70.22%	66.59%	70.29%	58.86%
	Balance Acc	51.59%	53.19%	51.53%	47.79%

One thing to note using the classification report of model 2(best SVC model), the precision, recall and f-1 scores for the 4th quartile were low. Meaning that to classify an unpopular song is easier than classifying a popular one. Therefore, SVC can be used to model to build a ML model but the insight for our research question of the best features to influence the model is not easily attained. In the future, the best model can be re-ran by taking out each feature one at a time to see which feature affects the accuracy the most to determine the most important feature.

SVC Classification Report for best model(model 2) :



```
              precision    recall  f1-score   support

          q1       0.78      0.71      0.75     23255
          q2       0.70      0.71      0.71     28751
          q3       0.53      0.59      0.56     11779
          q4       0.11      0.10      0.11      2957

    accuracy                           0.67     66742
   macro avg       0.53      0.53      0.53     66742
weighted avg       0.67      0.67      0.67     66742
```

Result and Conclusion

All research questions were able to be addressed by different methods and models. Given the pros and cons of each model, not all were used for each research question.

The below table summarizes our models used for this data set. Excluding Logistic regression and focusing on the multi-class classification models, the KNN model was the best. For model to model comparison, the overall accuracy was used to rank them, but within each model a combination of metrics were used that were primarily Overall Accuracy and Balanced Accuracy.

Model Name	**Modification**	**Overall Accuracy**
Decision Tree	Gridsearch + oversampling	62%
AdaBoost	Gridsearch + undersampling	65%
Random Forest	Gridsearch + SMOTE	67%
K-nearest Neighbors	Gridsearch + undersampling	68%
Support Vector Classification	Stratified + manual hypertuning	66.6%
Logistic regression (Binary)	Gridsearch + SMOTE	97%

For future work the following could be considered:

- Use the data set aside for popularity equal to zero as a test data set to predict an actual popularity. To verify the accuracy we would need to verify against actual values and in place of this data set from spotify, we could use other data sets such as billboards to see if the song ever placed.
- Add a feature for marketing budget to represent how much money was invested into the song such as cost for music distribution or music video cost.
- Add the artist back into the data-set and see if an artist plays a role in the popularity versus the other features.

Reference

Brownlee, J. (2020, March 18). Step-By-Step Framework for Imbalanced Classification Projects. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/framework-for-imbalanced-classification-projects/

Mithrakumar, M. (2019, November 11). How to tune a Decision Tree? Retrieved from https://towardsdatascience.com/how-to-tune-a-decision-tree-f03721801680

Mindus. (2021, February 27). Spotify descriptive and exploratory data analysis.Retrieved from https://www.kaggle.com/mindus/spotify-descriptive-and-exploratory-data-analysis

Qiao, F. (2019, January 08). Logistic regression model tuning with scikit-learn - part 1. Retrieved March 18, 2021, from https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5

Koehrsen, W. (2018, January 10). Hyperparameter tuning the random forest in Python. Retrieved March 18, 2021, from https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74

Bo Chen, Chunying Ma, Witold F. Krajewski, Pei Wang. Feipeng RenLogarithmic transformation and peak-discharge power-law analysis. Published in Hydrology Research (2020) 51 (1): 65–76. https://iwaponline.com/hr/article/51/1/65/71146/Logarithmic-transformation-and-peak-discharge

Hsu, C, Chang C, Lin C. (2010, April 15). A Practical Guide to Support Vector Classification retrieved: https://www.researchgate.net/profile/Chenghai-Yang/publication/272039161_Evaluating_unsupervised_and_supervised_image_classification_methods_for_mapping_cotton_root_rot/links/55f2c57408ae0960a3897985/Evaluating-unsupervised-and-supervised-image-classification-methods-for-mapping-cotton-root-rot.pdf