OPEN IIT DATA ANALYTICS 2021

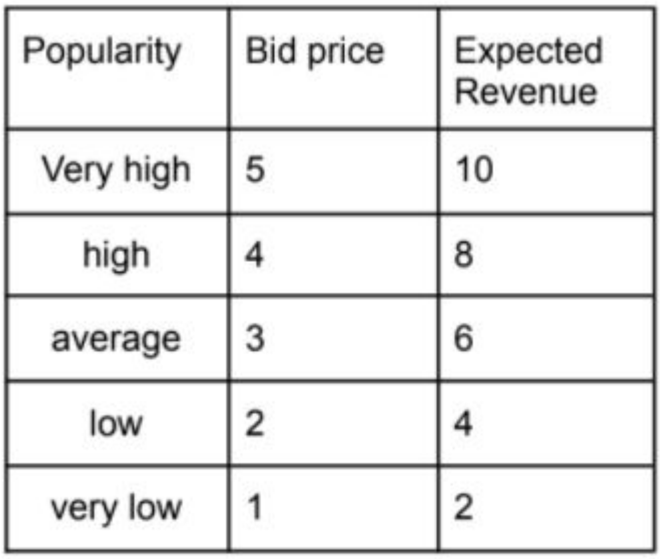
REPORT

-TEAM 36-

Index

Introduction

The report outlines the analysis done on the music popularity dataset collected in different years ranging from **1920 to 2021**. The objective of this analysis is to generate the highest possible revenue. We are given with a total amount of 10,000 and we need to generate the highest possible revenue that we can generate by bidding it with the 4000 songs.



Popularity	Bid price	Expected Revenue
Very high	5	10
high	4	8
average	3	6
low	2	4
very low	1	2

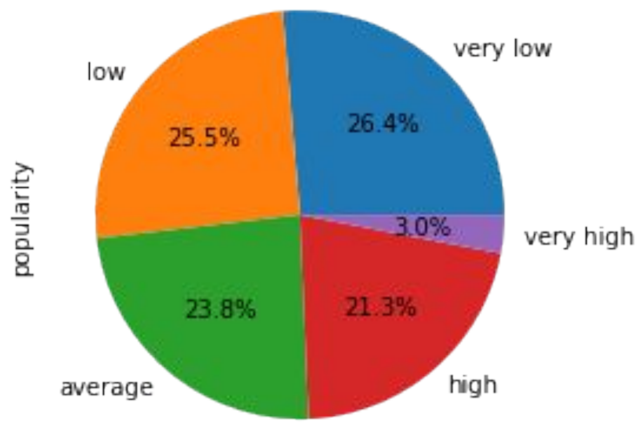
The dataset contains different independent variables like **danceability, energy, year, key, instrumentainess, liveness, loudness, mode, tempo** etc and using these independent variables the popularity (dependent variable) of the dataset has to be calculated.

The dataset has been divided into train set and test set and different models like random forest, XGBoost, Decision tree classification, Support vector classification, K-Neighbors Classification has been used for the prediction of the popularity of the song.

Analyzing Data

The train dataset provided here contains around 12,000 data points ranging from the year 1920 to 2021..

Every song in the dataset contains 19 features categorized by audio analysis like duration, mode, loudness, key, tempo; and song related features like release-date, year etc. Mostly audio features and year released are used to predict the popularity of the songs and finally to generate the highest possible revenue.

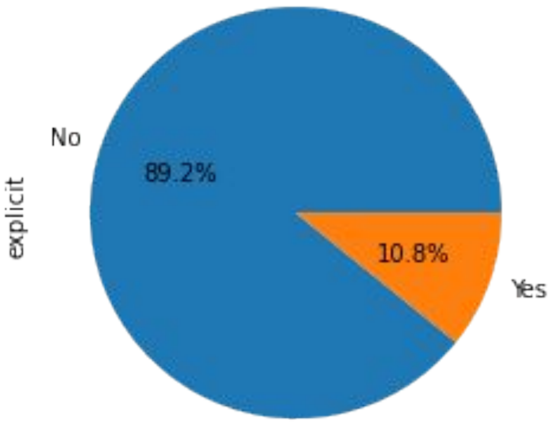


As you can see from the graph on the right, the data points of each category are mostly equally distributed except in the case of very high category

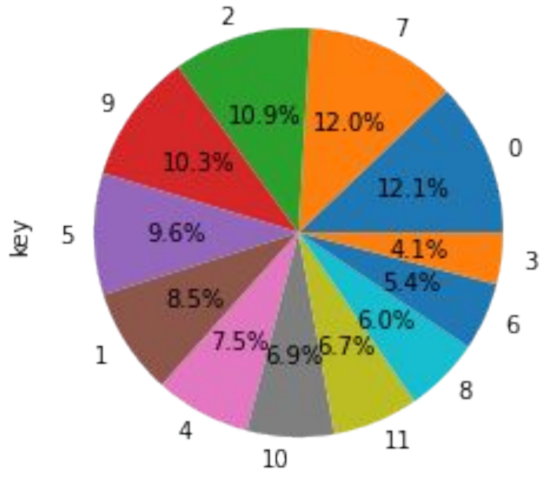
This was reflecting while building our model - we saw in the confusion matrix that category 5 has very low accuracy when compared to other categories..

So we used sampling techniques like SMOTE and Randomized Under Sampling to get a more balanced dataset.

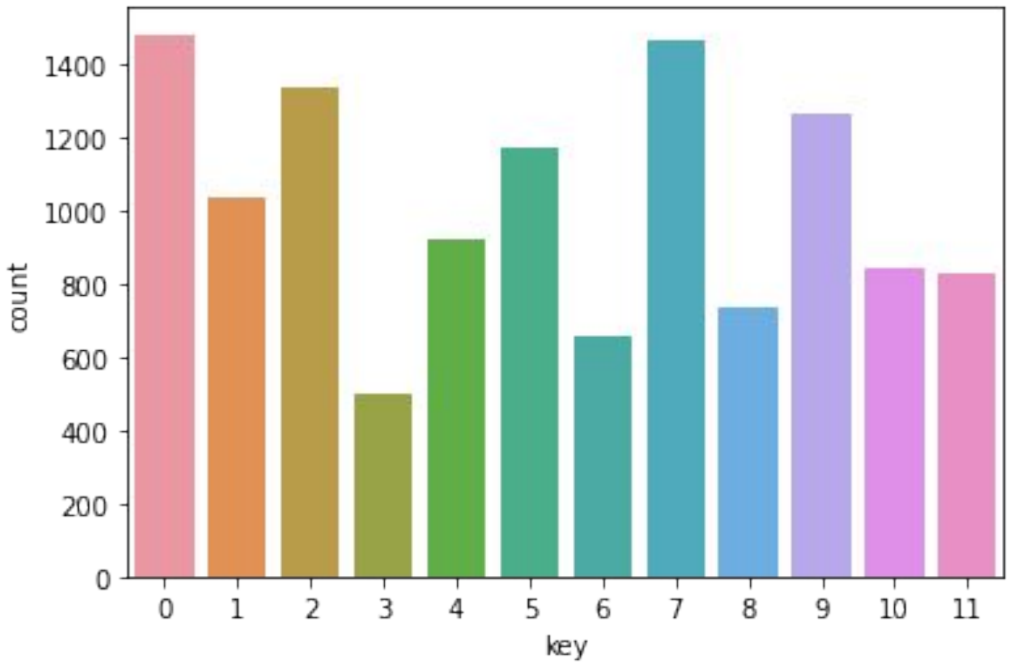
Data Distribution



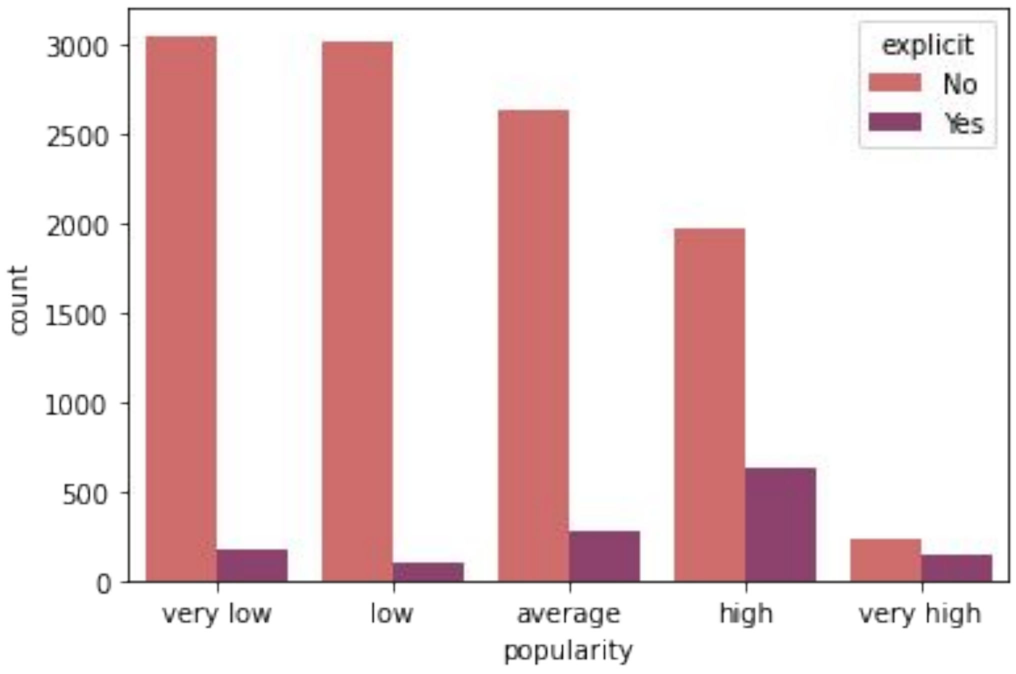
In this you can see that most of the data points are in the 'No' category and only very few are in 'Yes'



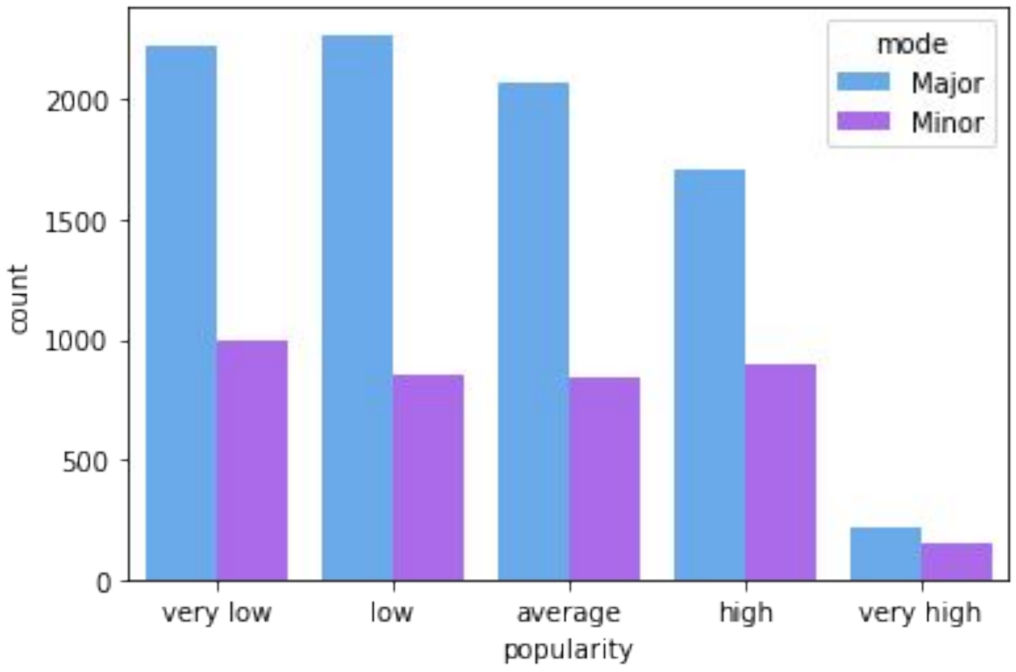
The model is pretty evenly distributed in terms of 'key' value.



Data Distribution

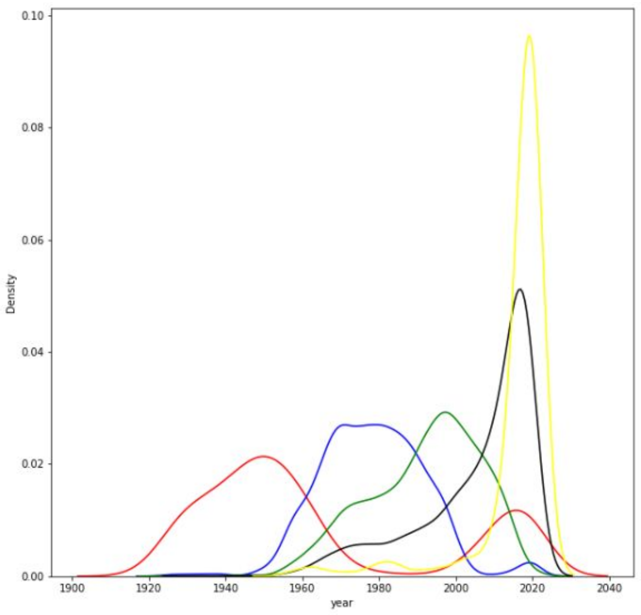


From the above graph, We can see that most of the songs which are explicit have average or high popularity.

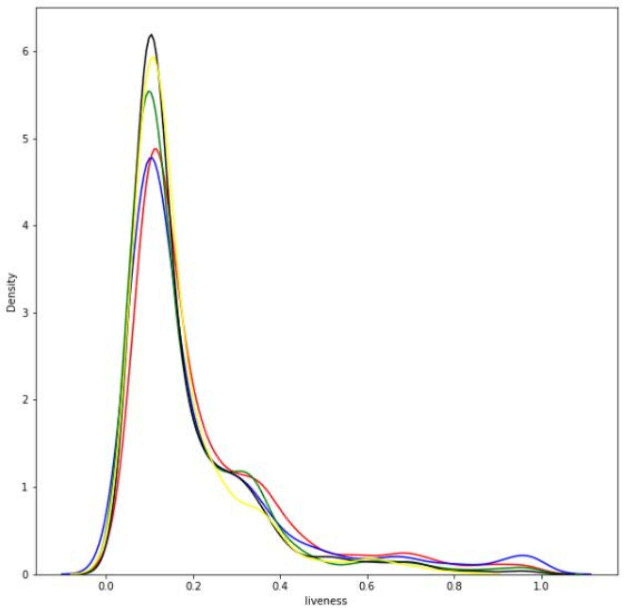


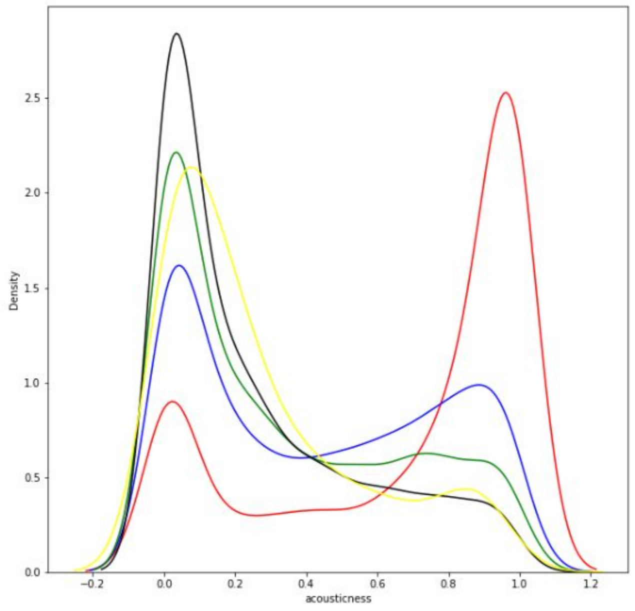
Based on the mode, the distribution seems to be very similar for both the cases.

Analyzing Popularity



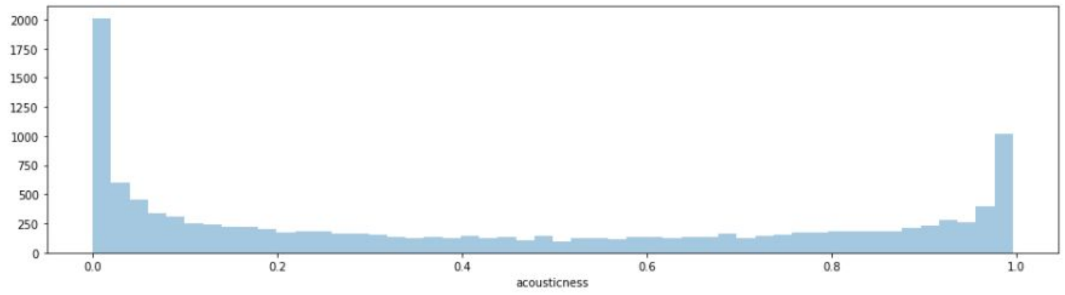
We splitted the training set into 5 different parts based on the popularity value and checked whether there are any anomalies or relationship variation with any other value. But as expected, except the year value, the distribution for all other variables came out to be nearly similar.

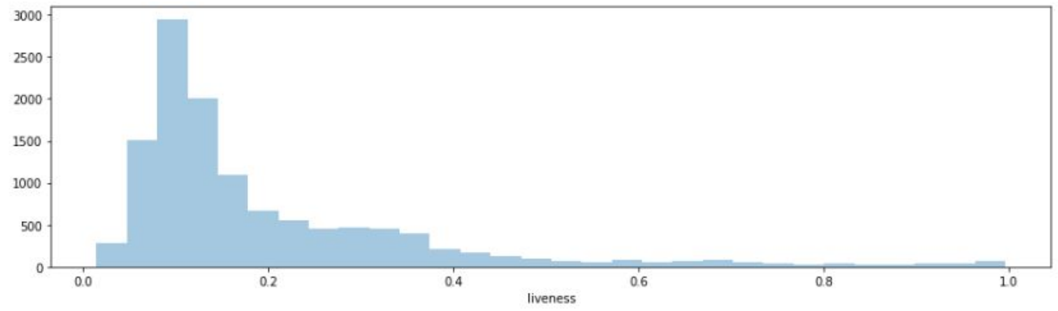




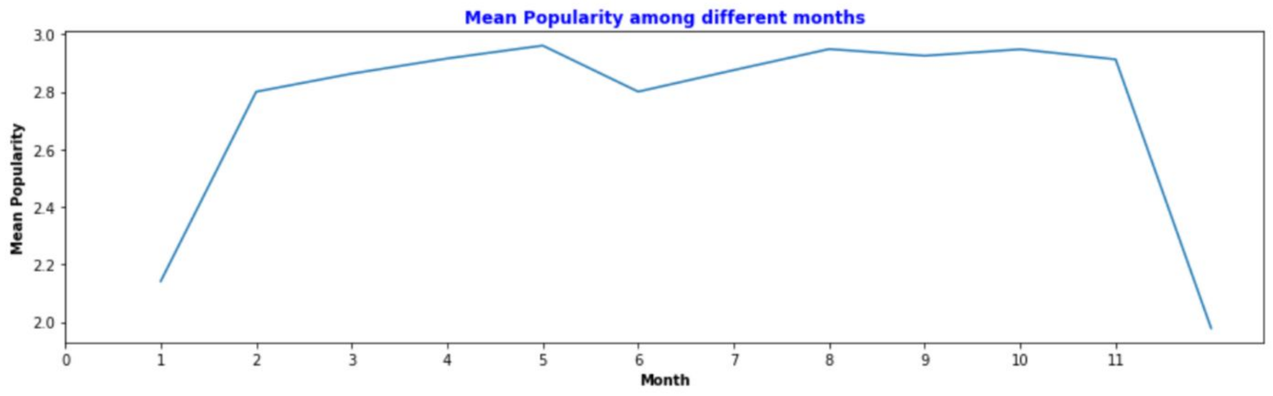
For acousticness,we saw that the values are very discrete - either very close to zero or very close to one. So we created 3 new categories for them - if the values are below 0.05, if the values lie between 0.05 and 0.95, and finally if the values are greater than 0.95.

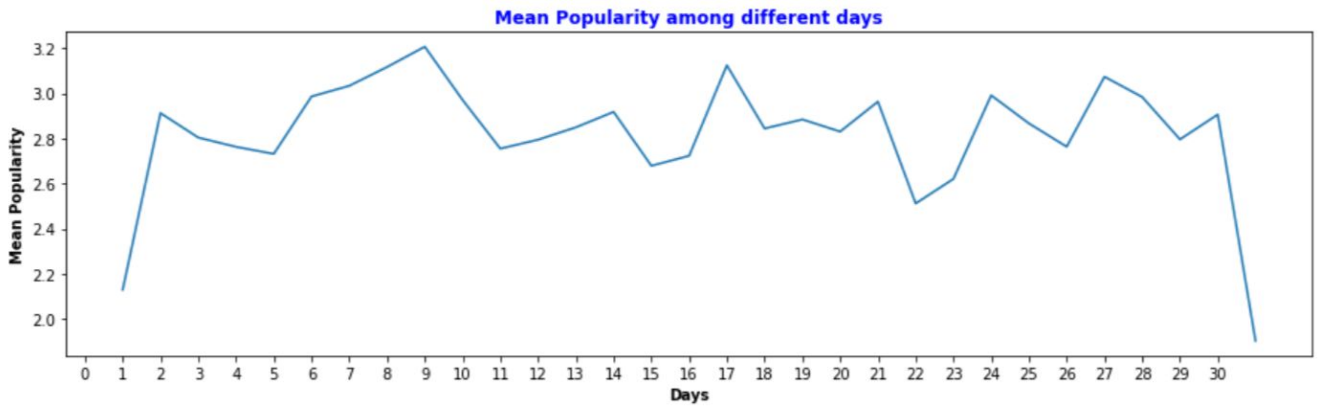
And also only the values with low acousticness seemed to be more popular in general.





Feature Engineering

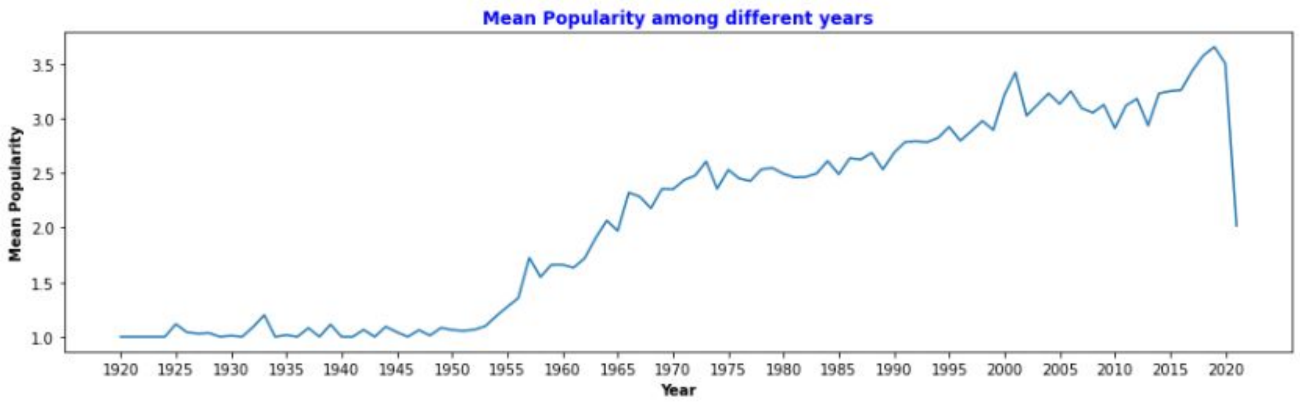




We tried plotting the mean values of popularity for different months and different days of the month and saw an odd relationship.
In the months, as you can can see from the above graph, the popularity value is coming out to be less for January and December whereas all other months remain nearly constant. And also in case of days, for the first and last day of the month, the value is coming out to be less when compared to other days.

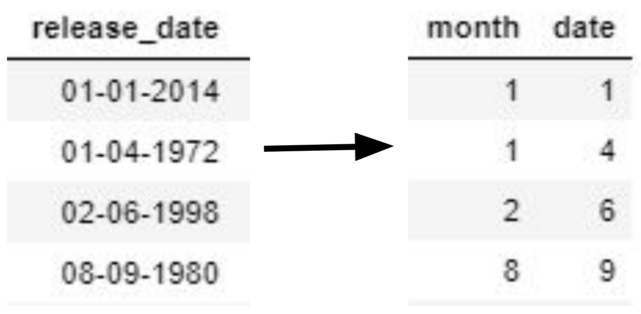
So we created 2 new features -

1. ‘Month_curse’ - which will be 0 when the month is January or December and 1 in other cases.
2. ‘Date_curse’ - which will be 0 on the first and last days of the month and 1 in other cases.



And also as you can see in general the popularity of the songs is increasing with the year. If the song is released relatively recent then it can be somewhat assumed to be more popular.

Feature Engineering - Part 2

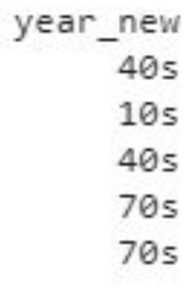


release_date
01-01-2014
01-04-1972
02-06-1998
08-09-1980

month	date
1	1
1	4
2	6
8	9

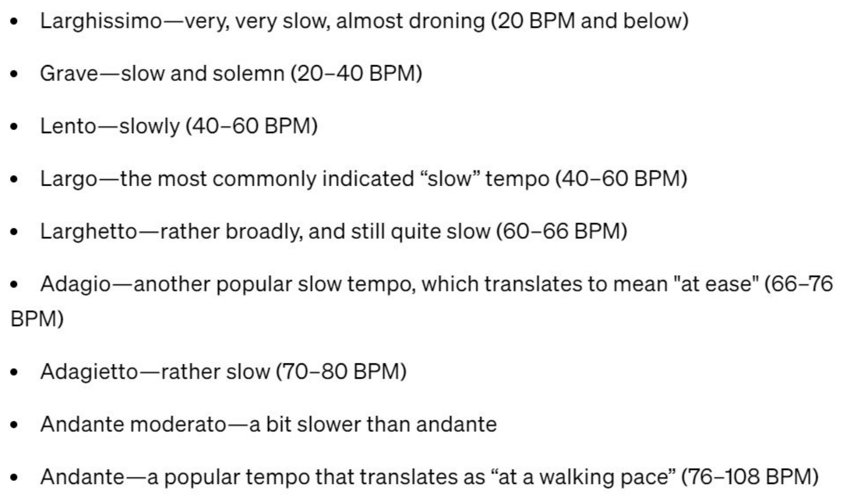
The ‘release_date’ column was converted into month and date. Year was already given in the ‘year’ column of the dataset.

From this we also created a new column ‘days’ which had the number of days passed from the starting of 2020. But latter it was found to have high correlation with the year column. So it was removed from the dataset.



year_new
40s
10s
40s
70s
70s

A new categorical column was also created to keep an eye of the decade in which the song is published. If the song is published between 1940 and 1950, then it will be categorized as 40s and so on. During the model evaluation, we found out that this was creating the same impact as the year column. So we removed the year column and used this for our evaluation.



- Larghissimo—very, very slow, almost droning (20 BPM and below)
- Grave—slow and solemn (20–40 BPM)
- Lento—slowly (40–60 BPM)
- Largo—the most commonly indicated “slow” tempo (40–60 BPM)
- Larghetto—rather broadly, and still quite slow (60–66 BPM)
- Adagio—another popular slow tempo, which translates to mean "at ease" (66–76 BPM)
- Adagietto—rather slow (70–80 BPM)
- Andante moderato—a bit slower than andante
- Andante—a popular tempo that translates as “at a walking pace” (76–108 BPM)

We read on a website that tempo can be classified into different categories based on it’s value range. So we created a new column to classify them based on their values and also one-hot encoded them to make sure there is no problem while classifying.

K-Means: Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. Using the K-means clustering and the elbow method, the optimal K value is found. Using this K value new feature is generated.

Features Used

Music Data

Categorical

- Explicit
- Mode
- Year Class ‼ NEW FEATURE ‼
- Month Curse ‼ NEW FEATURE ‼
- Date Curse ‼ NEW FEATURE ‼
- Tempo Category ‼ NEW FEATURE ‼
- Month ‼ NEW FEATURE ‼
- Date ‼ NEW FEATURE ‼
- Acoust criteria ‼ NEW FEATURE ‼
- Instru criteria ‼ NEW FEATURE ‼
- K-MEANS cluster ‼ NEW FEATURE ‼

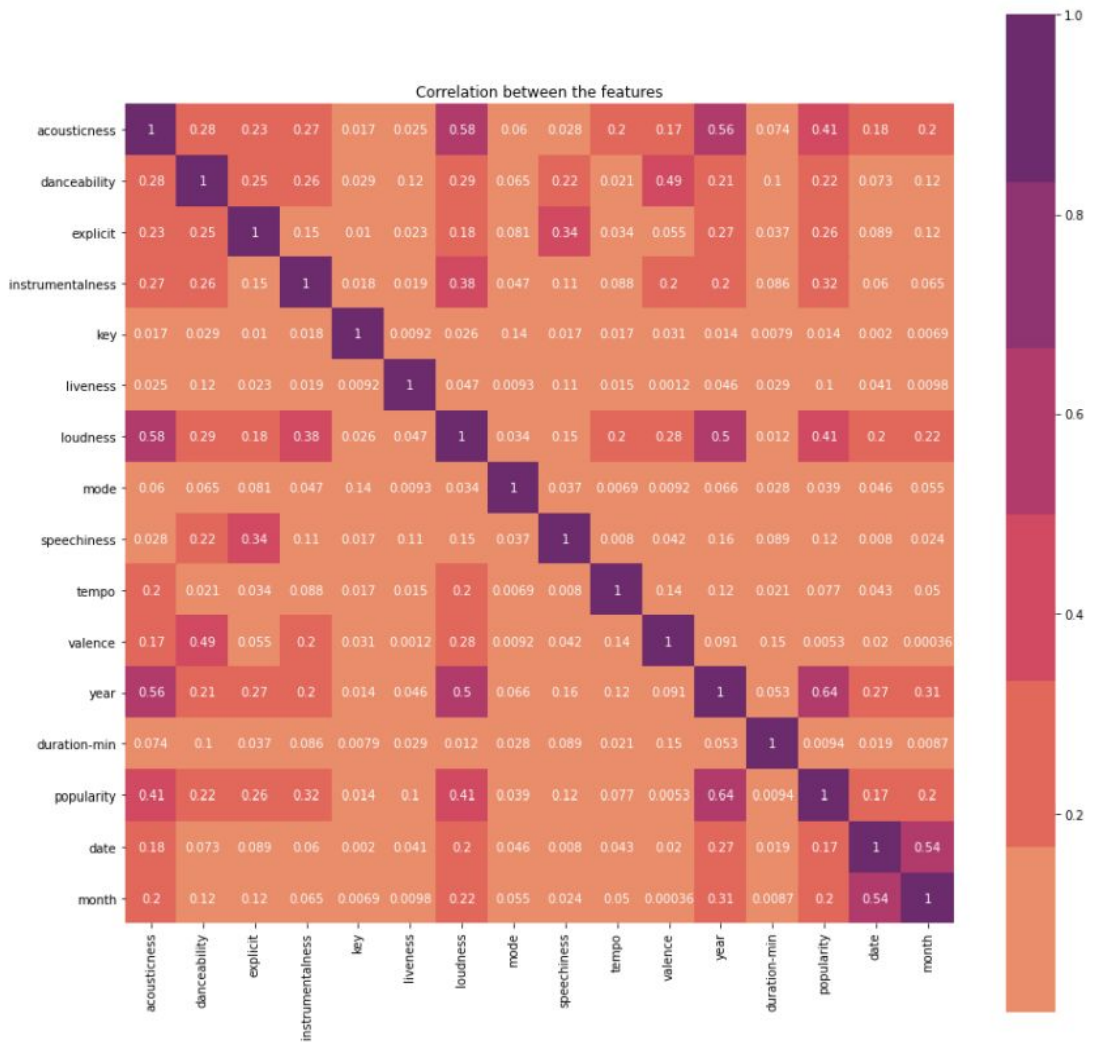
Continuous

- Acousticness
- Danceability
- Energy
- Instrumentalness
- Key
- Liveness
- Loudness
- Speechiness
- Tempo
- Valence
- Duration-min
- Year
- Release date
- ‼ NEW FEATURE ‼ Days

Feature Scaling

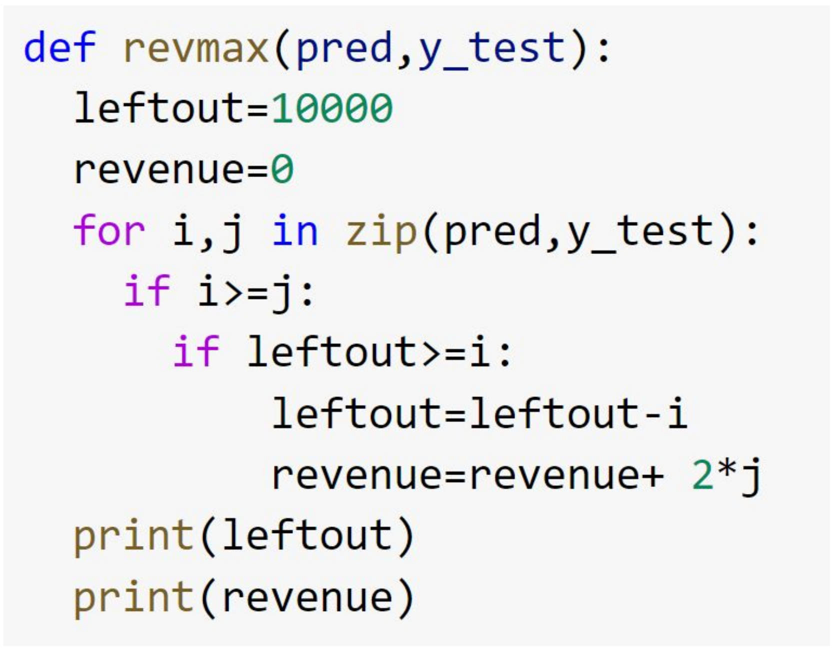
Most of the columns in the dataset were lying in the range of 0 to 1, so there was no need to scale them. But there are some columns with high values - like duration-min, tempo, loudness etc.. These columns are scaled using Min-Max Scaler to make sure the values between 0 and 1. This will not provide a boost in accuracy if we are using a tree based or ensemble based classification technique. But if we were using Linear Regression or K-Means which involves euclidean distance, then this will be helpful.

Correlation between Features



If some features have high correlation between then - like in the case of loudness and energy, one feature is removed.

Model Evaluation



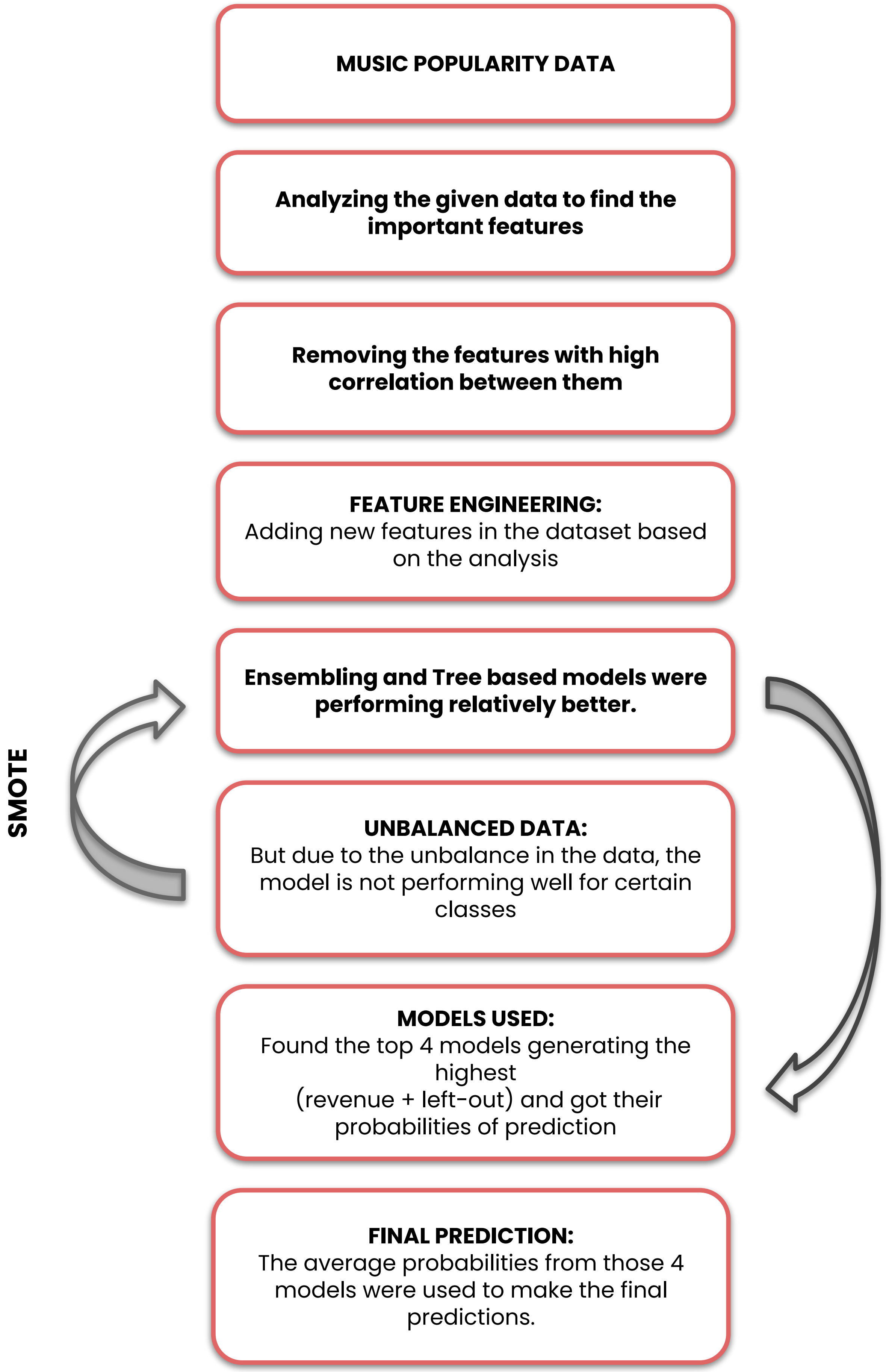
```
def revmax(pred,y_test):
  leftout=10000
  revenue=0
  for i,j in zip(pred,y_test):
    if i>=j:
      if leftout>=i:
          leftout=leftout-i
          revenue=revenue+ 2*j
  print(leftout)
  print(revenue)
```

The final predictions are going to be used for bidding. So we created a new function for the model evaluation. We have split the model into 70:30, so the test set will be having approximately 4000 data points.

Then we have given the model a amount of 10.000 and saw how much revenue is being generated in different cases and also how much money is left out from our budget after the bidding.

‘Accuracy_score’ was also kept in mind. We tried to balance both the evaluation metrics - revmax and accuracu_score. In some cases, the accuracy score was coming out to be very low, but the revenue was higher and vice versa.

Model Approach



MUSIC POPULARITY DATA

Analyzing the given data to find the important features

Removing the features with high correlation between them

FEATURE ENGINEERING:
Adding new features in the dataset based on the analysis

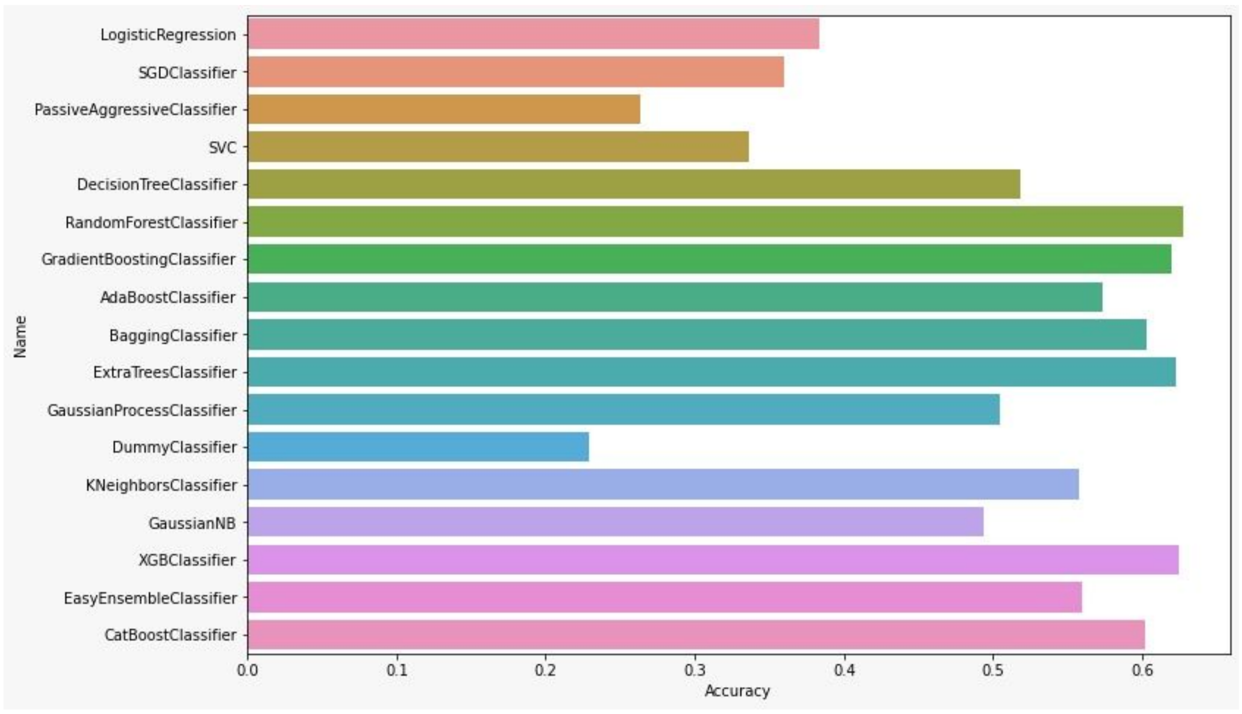
Ensembling and Tree based models were performing relatively better.

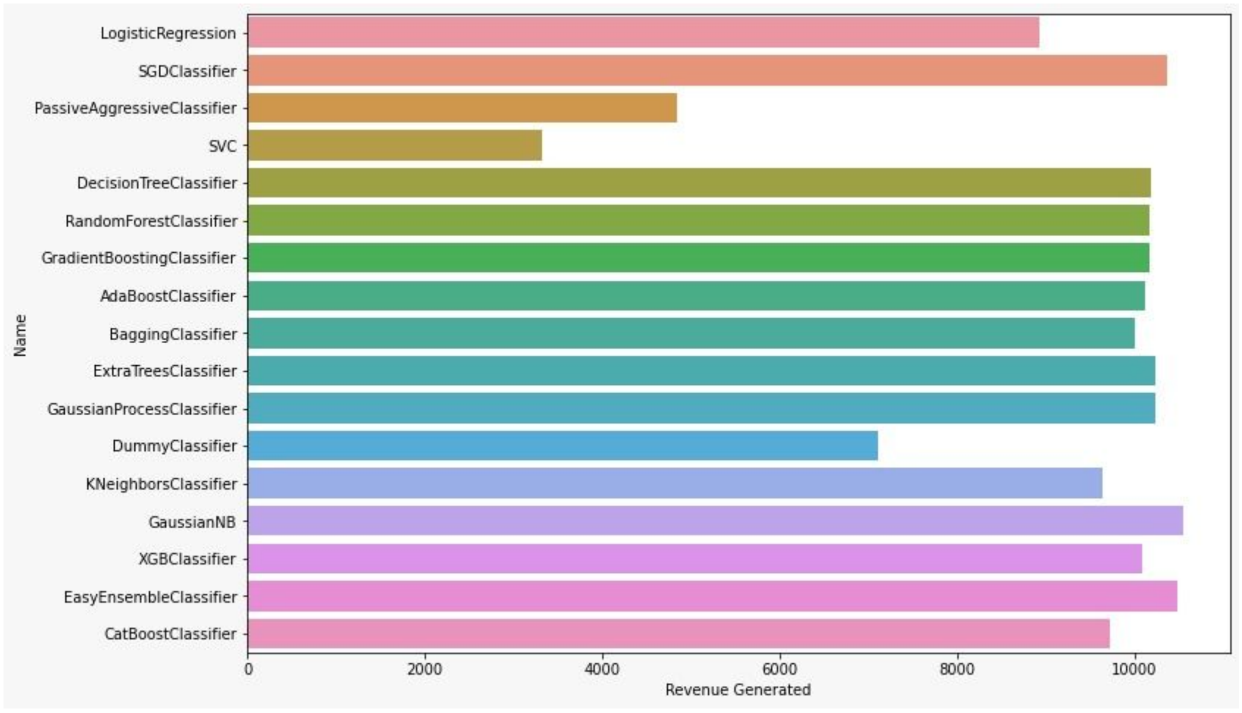
UNBALANCED DATA:
But due to the unbalance in the data, the model is not performing well for certain classes

MODELS USED:
Found the top 4 models generating the highest
(revenue + left-out) and got their probabilities of prediction

FINAL PREDICTION:
The average probabilities from those 4 models were used to make the final predictions.

Model Selection





We tried different algorithms to see which type of algorithms are performing better. All the values used are taken from cross validation to ensure there is no bias in the predictions.

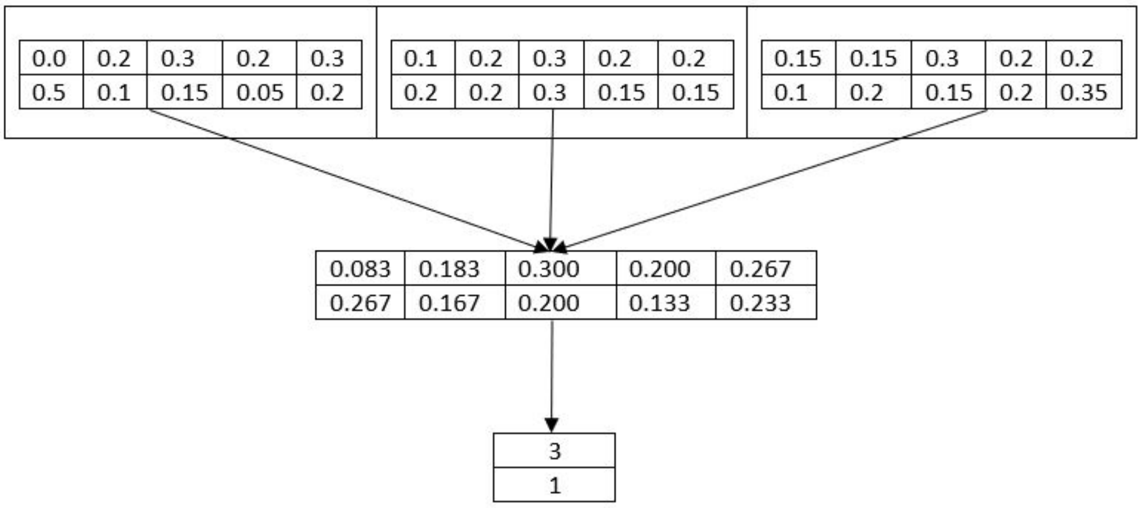
Model Design

We ran the predictions on our top performing models and calculated the probability values we are getting for each of the predictions.

Then for the final prediction, we used the average of those probability values from each of the prediction from different model.

Like each model will be giving out 5 values when you use 'predict_proba()' function and it will add up to 1. Let's say 0.1,0.2,0.3,0.15,0.25. Like this we got the probability from the top performing models and then took the average of them.

Then we predict the final value using argmax() function to see which category has the highest probability.



0.0	0.2	0.3	0.2	0.3
0.5	0.1	0.15	0.05	0.2

0.1	0.2	0.3	0.2	0.2
0.2	0.2	0.3	0.15	0.15

0.15	0.15	0.3	0.2	0.2
0.1	0.2	0.15	0.2	0.35

0.083	0.183	0.300	0.200	0.267
0.267	0.167	0.200	0.133	0.233

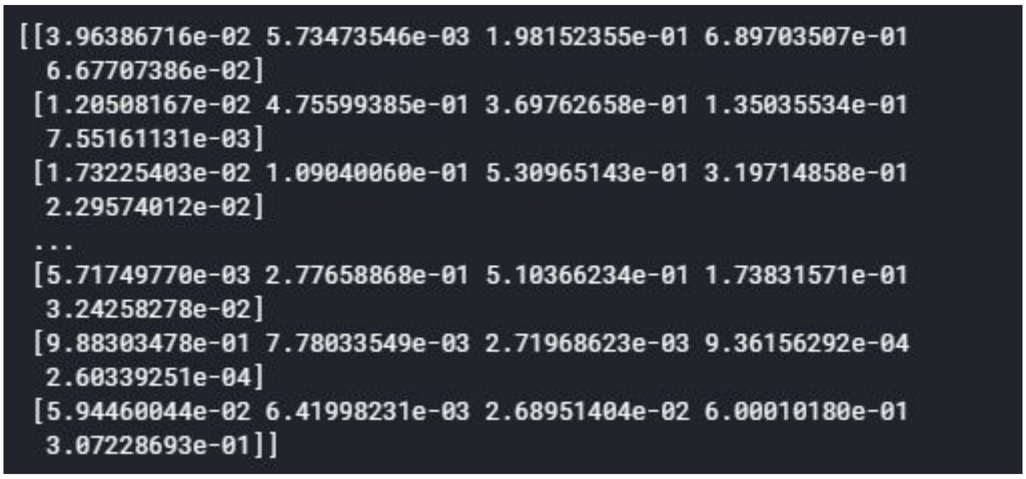
3
1

We found an imbalance in the data- the data points for category 5 i.e very high is found out to be very low. So we either need to undersample or oversample the data to get better predictions. While trying, we saw that the undersampling was performing better in terms of revenue generation. But we are afraid that many information may be lost while doing this and it may do bad in the external data. So again we used a mixture of oversampling and undersampling to make the final predictions.

Final predictions = np.argmax((prob1+prob2+prob3+prob4)/4.0))+1

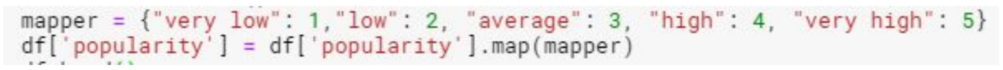
Model Flow

1. After finding out what models are performing better in which category, we tuned their hyperparameters to get the best accuracy.
2. Then we first oversampled the entire training set using SWOTE and got the probability values for all the values in the dataset using 'predict_proba()' function.
3. We followed this for 3 different types of models with different hyperparameters to make sure there is no bias in our predictions and also the predictions are robust. (As we saw the tree based models are overfitting the training set, we also included model like K Neighbors Classifier)
4. Similarly we did this after undersampling the data using RandomUnderSampler and got the 'predict_proba()' function from the top 3 performing models.
5. Then the average was taken for all those probability values and the final probabilities comes out to be like this.



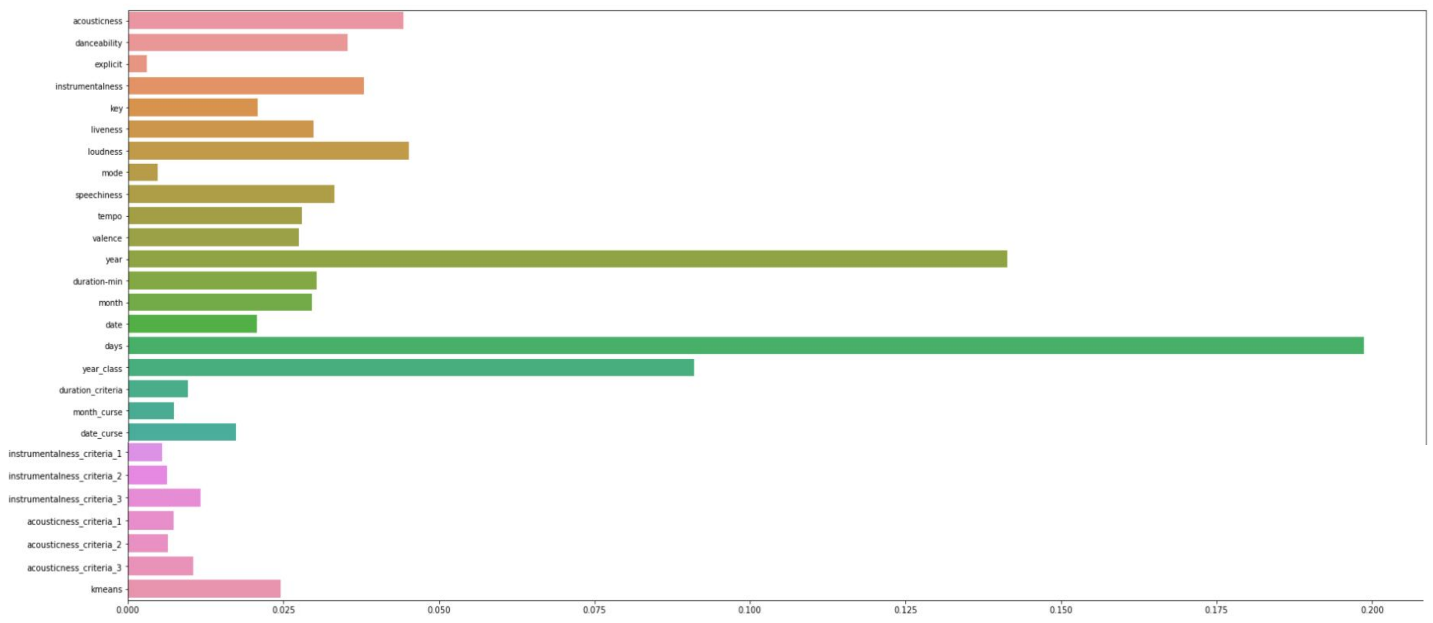
```
[[3.96386716e-02 5.73473546e-03 1.98152355e-01 6.89703507e-01
  6.67707386e-02]
 [1.20508167e-02 4.75599385e-01 3.69762658e-01 1.35035534e-01
  7.55161131e-03]
 [1.73225403e-02 1.09040060e-01 5.30965143e-01 3.19714858e-01
  2.29574012e-02]
 ...
 [5.71749770e-03 2.77658868e-01 5.10366234e-01 1.73831571e-01
  3.24258278e-02]
 [9.88303478e-01 7.78033549e-03 2.71968623e-03 9.36156292e-04
  2.60339251e-04]
 [5.94460044e-02 6.41998231e-03 2.68951404e-02 6.00010180e-01
  3.07228693e-01]]
```

6. Then we used argmax() function to get the index of the maximum probability in each row - 0,1,2,3,4. But we used 1,2,3,4,5 for the popularity values, so we added 1 to make sure it works perfectly.
7. That's it we got our final predictions and also checked with the original data popularity values. The distribution seems to be nearly similar.

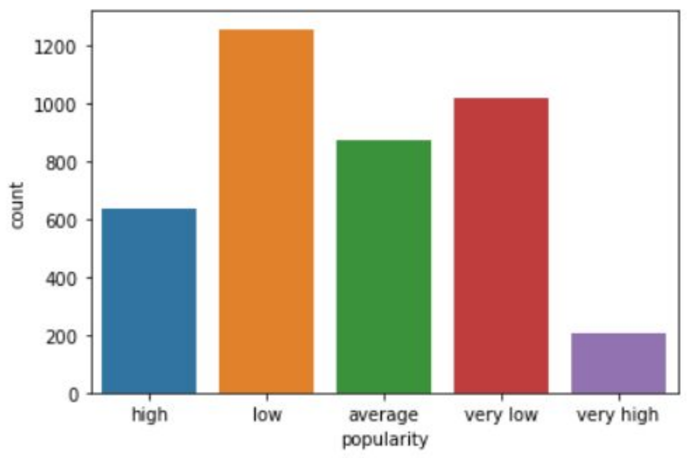


```
mapper = {"very low": 1,"low": 2, "average": 3, "high": 4, "very high": 5}
df['popularity'] = df['popularity'].map(mapper)
```

Feature Importance



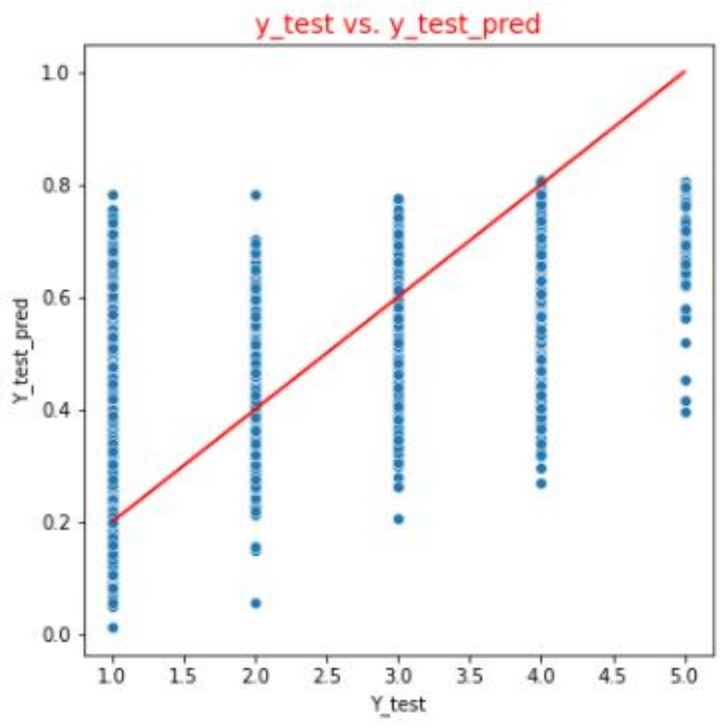
Final Predictions



The final predictions were seen to be following the same distribution as the original training set. So that way we can cross verify that out model is performing relatively better. And also as we have used different models for the prediction and used the average probability from them, our model should be able to predict the values in the external dataset correctly.

ANNEXURE

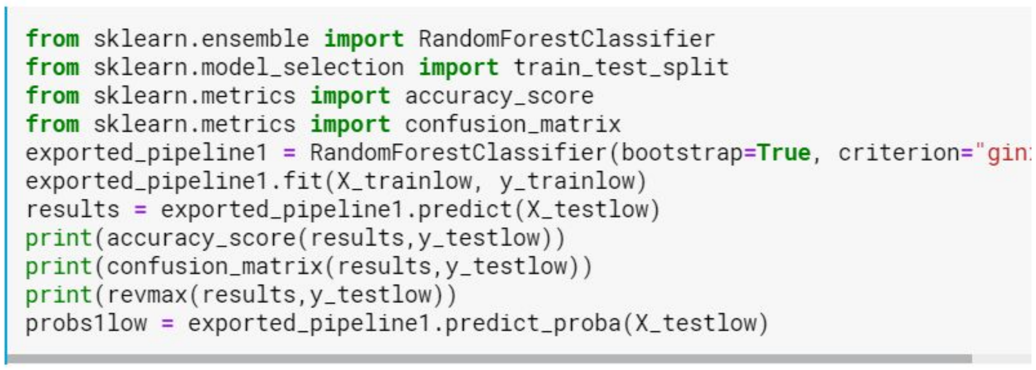
ANNEXURE-1



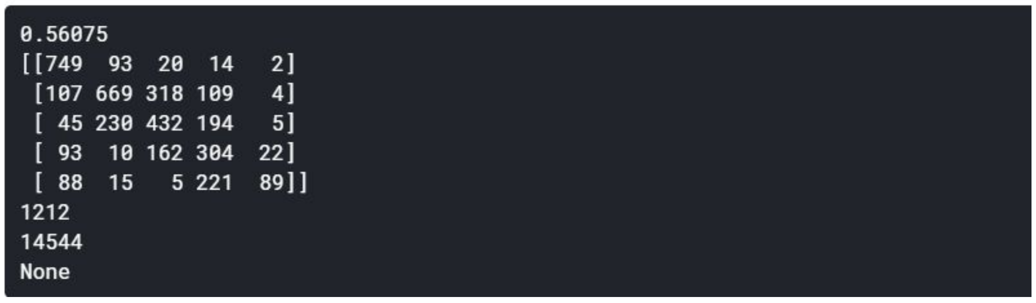
We also tried a way in which this can be approached as a regression problem and the values between (0-0.2) will be predicted as 1 and values of (0.2-0.4) will be predicted as 2 and so on. And we tried fitting a linear regression model. Was able to achieve around 50% accuracy with the test set after 70:30 split.

ANNEXURE-2

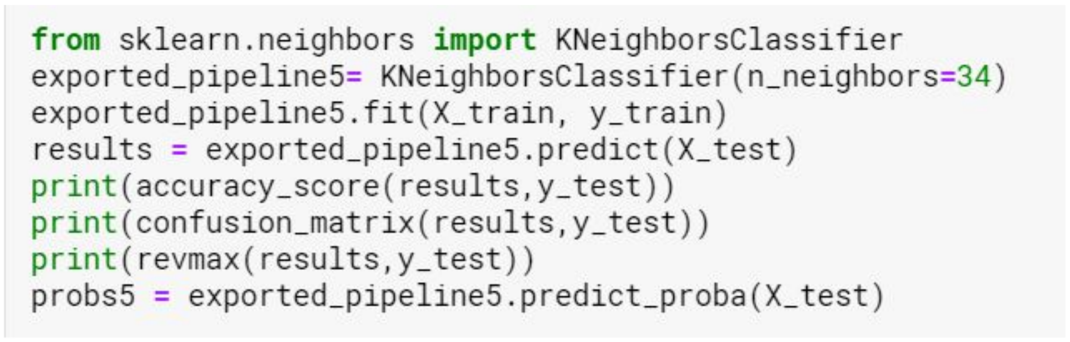
DIFFERENT MODELS USED FOR THE AVERAGE



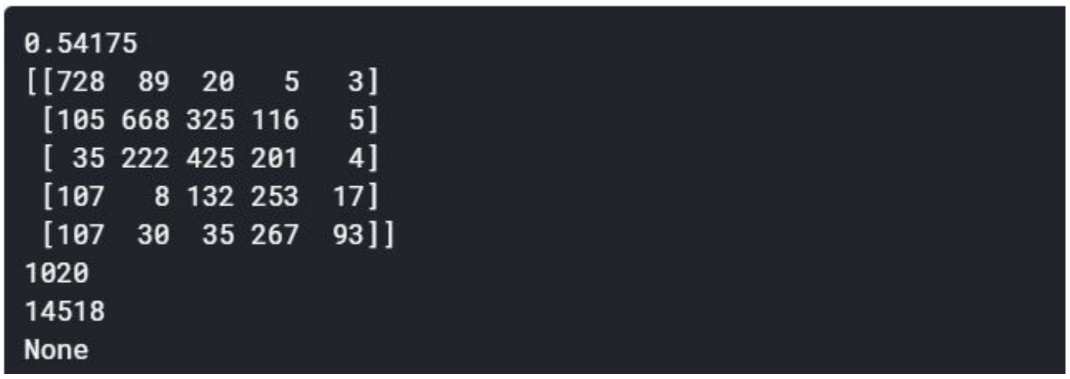
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
exported_pipeline1 = RandomForestClassifier(bootstrap=True, criterion="gin
exported_pipeline1.fit(X_trainlow, y_trainlow)
results = exported_pipeline1.predict(X_testlow)
print(accuracy_score(results,y_testlow))
print(confusion_matrix(results,y_testlow))
print(revmax(results,y_testlow))
probs1low = exported_pipeline1.predict_proba(X_testlow)
```



```
0.56075
[[749  93  20  14   2]
 [107 669 318 109   4]
 [ 45 230 432 194   5]
 [ 93  10 162 304  22]
 [ 88  15   5 221  89]]
1212
14544
None
```



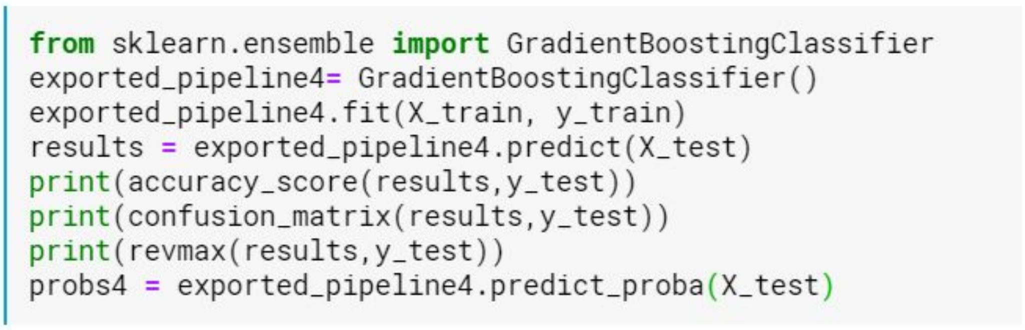
```python
from sklearn.neighbors import KNeighborsClassifier
exported_pipeline5= KNeighborsClassifier(n_neighbors=34)
exported_pipeline5.fit(X_train, y_train)
results = exported_pipeline5.predict(X_test)
print(accuracy_score(results,y_test))
print(confusion_matrix(results,y_test))
print(revmax(results,y_test))
probs5 = exported_pipeline5.predict_proba(X_test)
```



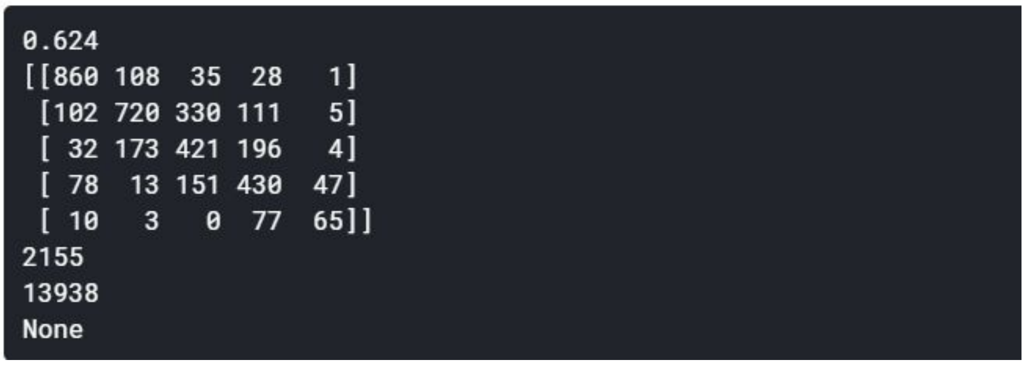
```
0.54175
[[728  89  20   5   3]
 [105 668 325 116   5]
 [ 35 222 425 201   4]
 [107   8 132 253  17]
 [107  30  35 267  93]]
1020
14518
None
```

ANNEXURE-3

DIFFERENT MODELS USED FOR THE AVERAGE

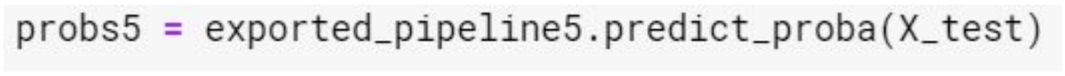


```
from sklearn.ensemble import GradientBoostingClassifier
exported_pipeline4= GradientBoostingClassifier()
exported_pipeline4.fit(X_train, y_train)
results = exported_pipeline4.predict(X_test)
print(accuracy_score(results,y_test))
print(confusion_matrix(results,y_test))
print(revmax(results,y_test))
probs4 = exported_pipeline4.predict_proba(X_test)
```

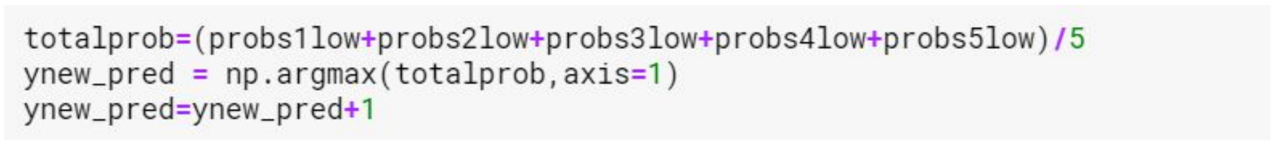


```
0.624
[[860 108  35  28   1]
 [102 720 330 111   5]
 [ 32 173 421 196   4]
 [ 78  13 151 430  47]
 [ 10   3   0  77  65]]
2155
13938
None
```

THE WAY WE CALCULATED THE FINAL PREDICTION

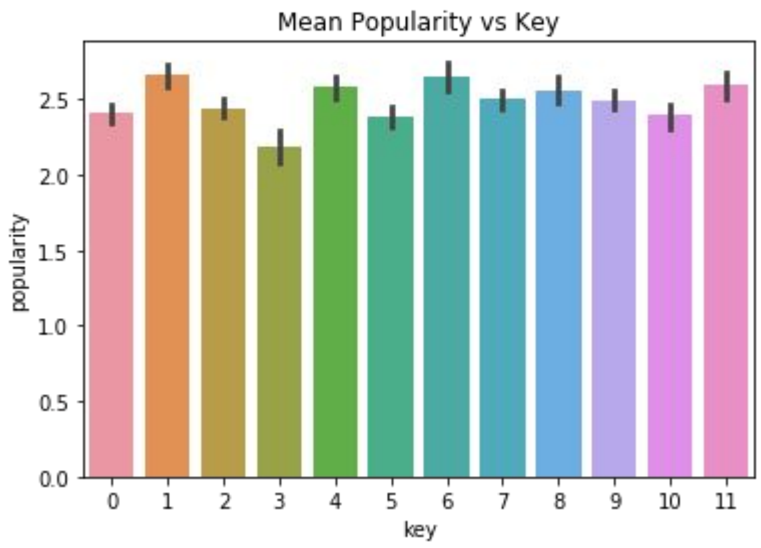


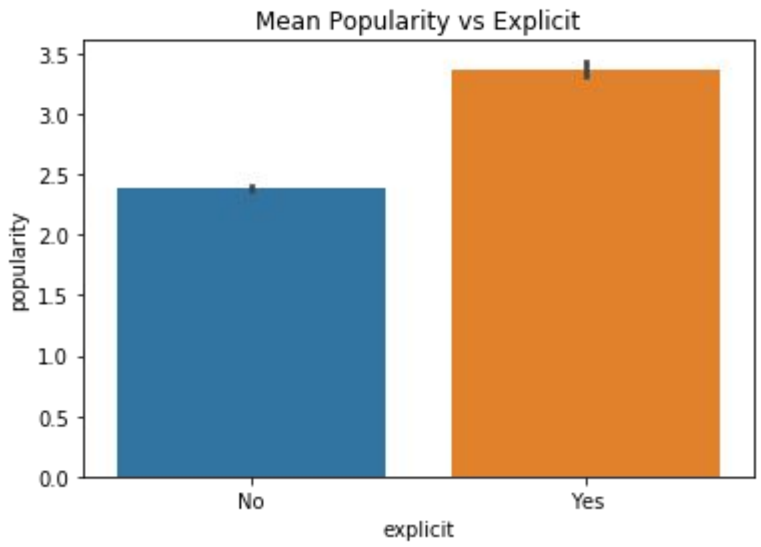
```
probs5 = exported_pipeline5.predict_proba(X_test)
```



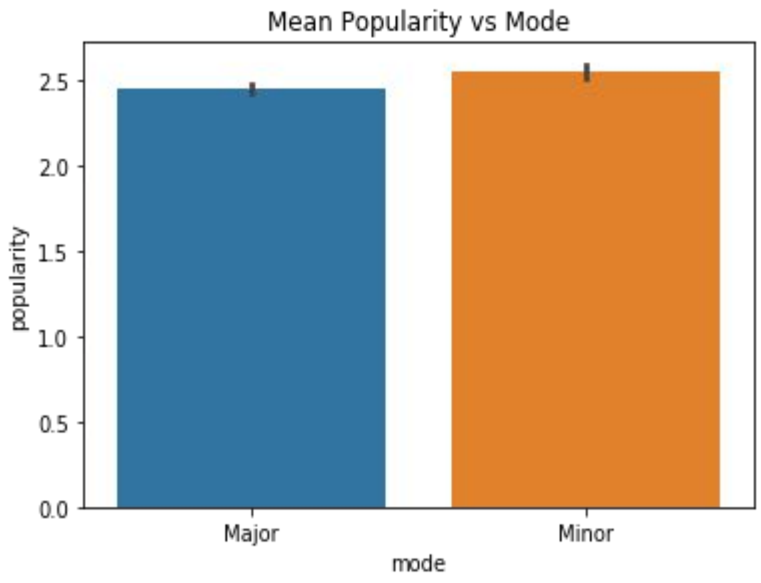
```
totalprob=(probs1low+probs2low+probs3low+probs4low+probs5low)/5
ynew_pred = np.argmax(totalprob,axis=1)
ynew_pred=ynew_pred+1
```

ANNEXURE-4





From this graph we can say that the Mean popularity is high when the Explicit value is YES.



We can say that the probability of a song being more popular is higher when the mode is Minor.