Andrew Birnbaum

Song Popularity Prediction Model

Humans have greatly associated themselves with songs and music which can improve mood, decrease pain and anxiety, and facilitate opportunities for emotional expression. Research suggests that music can benefit our physical and mental health in numerous ways. I personally have a deep connection with music. Music is an emotional escape from reality which is important for me especially because reality has been quite trying over the past few years. Living through a pandemic is reason enough for anyone to need an escape. It's incredible how the mind changes when it's locked up for 2 years. I literally listen to music for about 5 hours a day…sometimes even more. Listening to good music can elevate your mood so easily, conversely, bad music can put you in a funk. The ability to make better music would be beneficial for everyone and predicting song popularity may allow us to achieve this goal as popularity dictates enjoyability en masse. Studies have been carried out to understand songs and its popularity based on specific features. Over 14,000 songs in this data set had their parameters broken down to create these features. Predicting song popularity is the project goal.

The song data set was found on Kaggle[1] and the 13 features are:

- Song Duration
- Acousticness
- Danceability
- Energy
- Instrumentalness
- Key
- Liveness
- Time Signature
- Audio Valence
- Loudness
- Audio Mode
- Speechiness
- Tempo

1. https://www.kaggle.com/datasets/yasserh/song-popularity-dataset

Andrew Birnbaum

The data set must first be cleaned to be able to use different machine learning algorithms. First the data set must be visualized. A countplot of the popularity is shown. (Figure 1) The scale is from 0-100. Most of the values fall between 40-60 which makes sense as most songs are average. There are only a few very popular songs, 80-100. There is also a big jump of points at zero, this is due to many songs being very unpopular. Some are due to no-name artists, and some are due to just really bad sounding songs. The next step in cleaning the data was getting rid of NA values and duplicate songs (1st of each was kept).

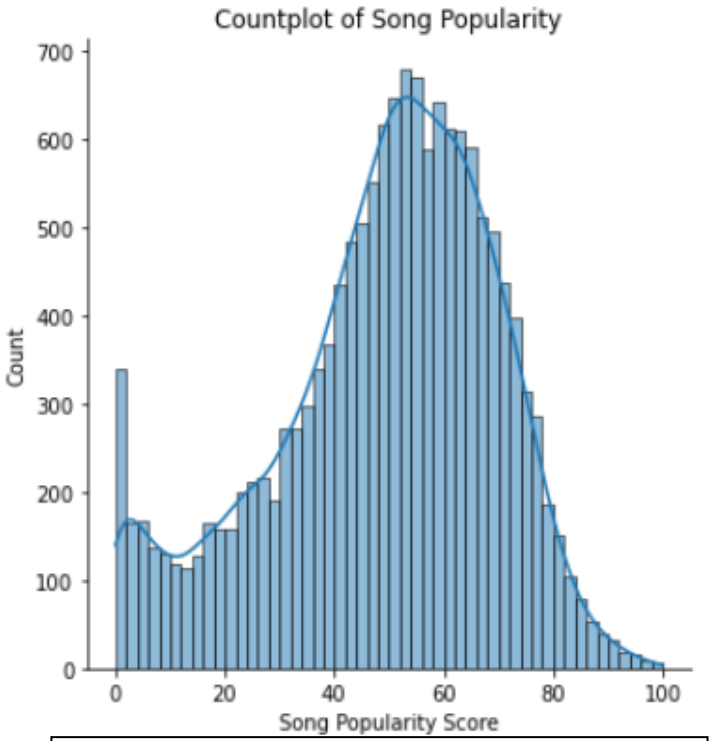

Figure 1: Countplot of Song Popularity (0-100)

These are the 10 most popular songs with Spotify streams from the data set. (Figure 2) Before removing the duplicate songs, “Happier” was all of the top songs. All of these songs have a ton of streams, almost a billion each. The data set was not from Spotify as the streaming values are not in order.

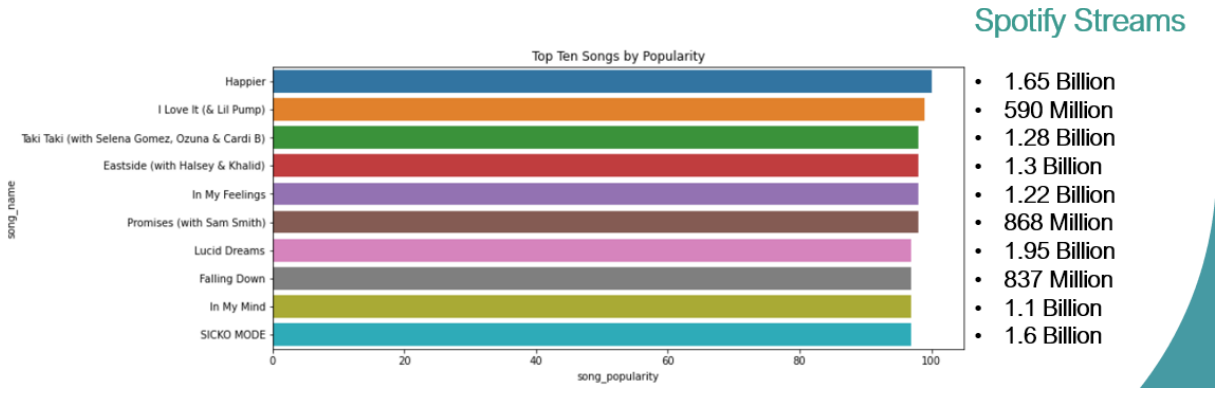

Figure 2: Top 10 Songs from Data Set

These are the 10 least popular songs with Spotify streams from the data set. (Figure 3) All of these songs have a very few streams, all in the low millions or less. Some of these songs are from really famous artists like T.I. and Meek Mill. This shows that the unpopular songs aren't necessarily unpopular because the artist isn't well known.

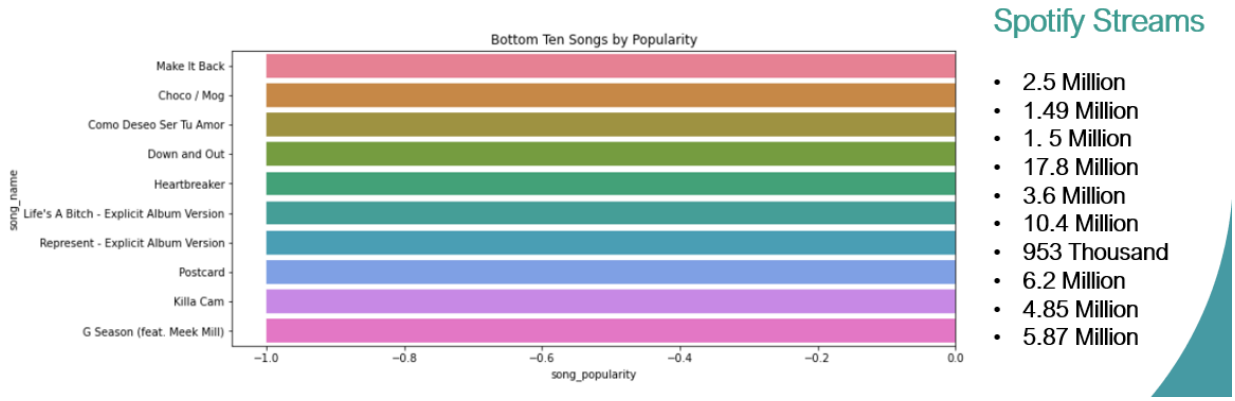

Figure 3: Bottom 10 Songs from Data Set

The next step of cleaning the data was choosing the useful features. The correlation between features was visualized using a heat map. (Figure 4) A problem that arises is multicollinearity which happens when one predictor variable in a model can be linearly predicted from the others with a high degree of accuracy. This can lead to skewed or misleading results. If categories are highly correlated, then one must be removed to get rid of the overlap in the data. Most of the features have little or no correlation. The only features with correlation are Energy, Loudness, and Acousticness. Energy & Loudness have a 0.77 correlation. Energy & Acousticness have a -0.68 correlation. Loudness & Acousticness have a -0.57 correlation. Energy is highly correlated with Loudness and more highly negatively correlated with Acousticness, and therefore, it was removed as it had the largest multicollinearity. (Figure 5)

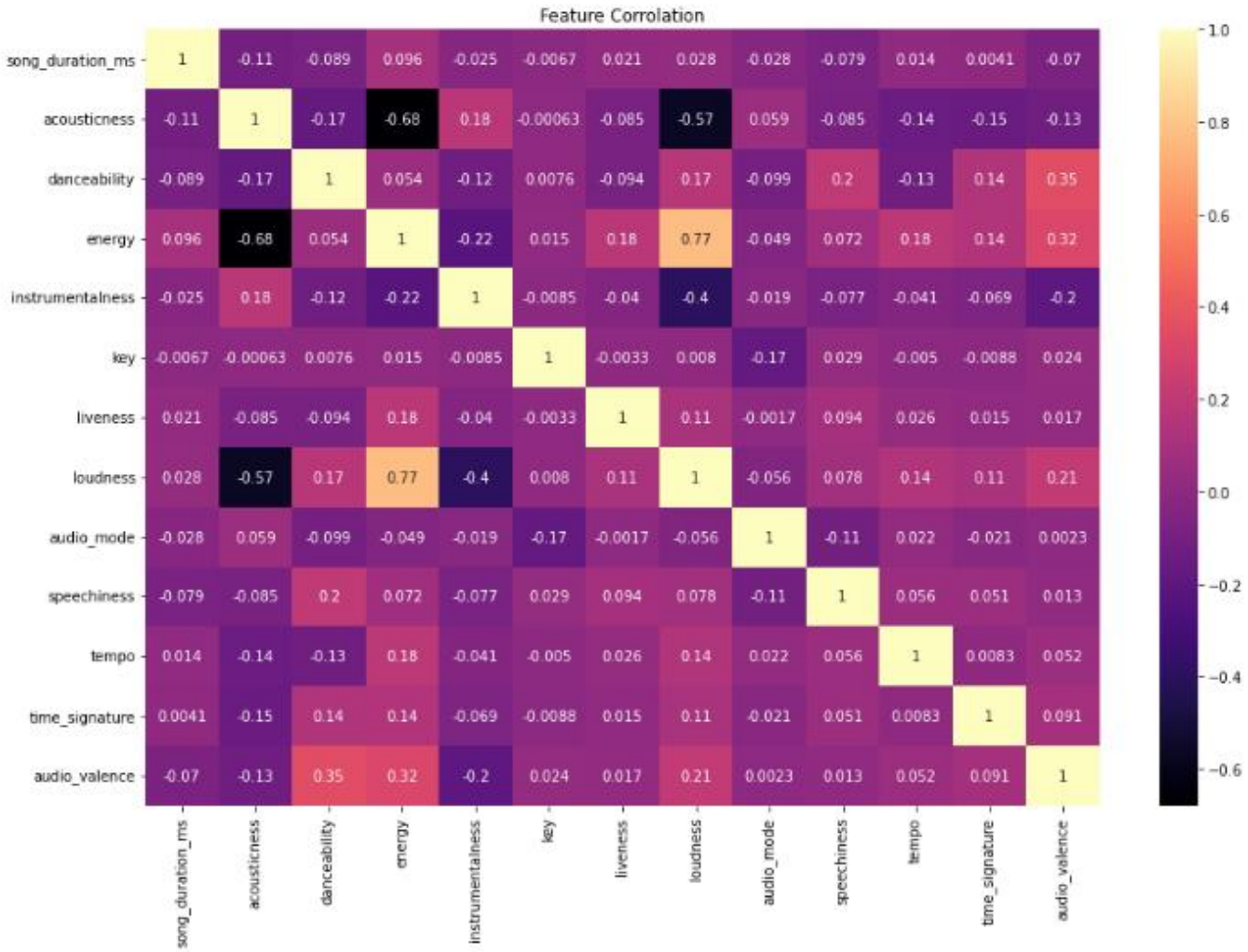

Figure 4: Heatmap to Show Correlation Between Features

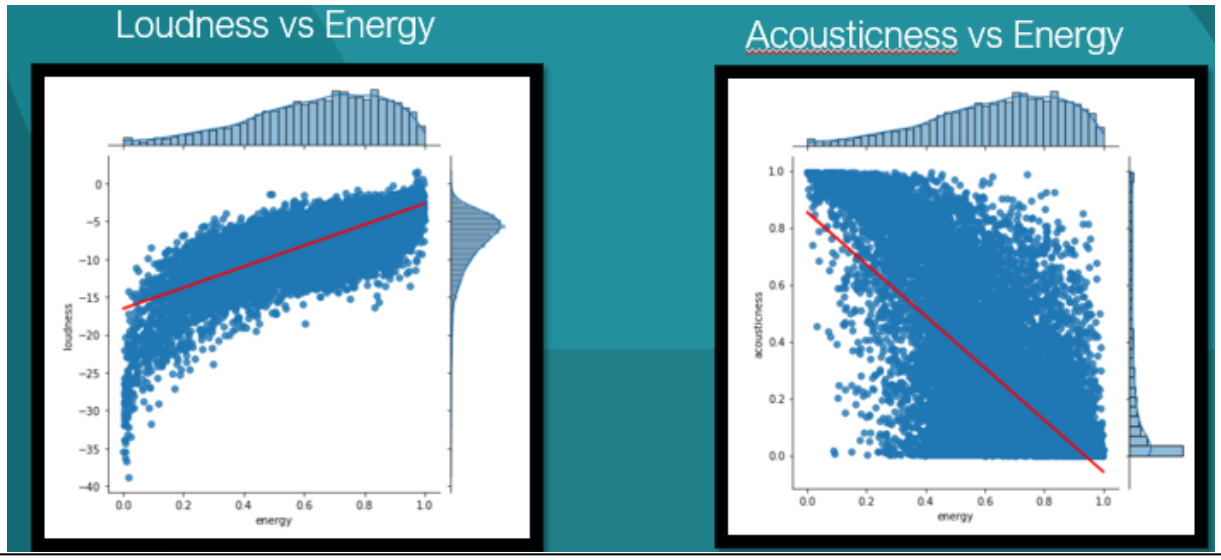

Figure 5: Plots of Energy vs Loudness and Energy vs Acousticness that show heavy correlation

Andrew Birnbaum

Countplots of each of the features were made to visualize all of the features to see if anything stands out as abnormal. (Figure 6) Instrumentalness seems to be incomplete, and investigation must be done to see if the feature is important or should be removed.

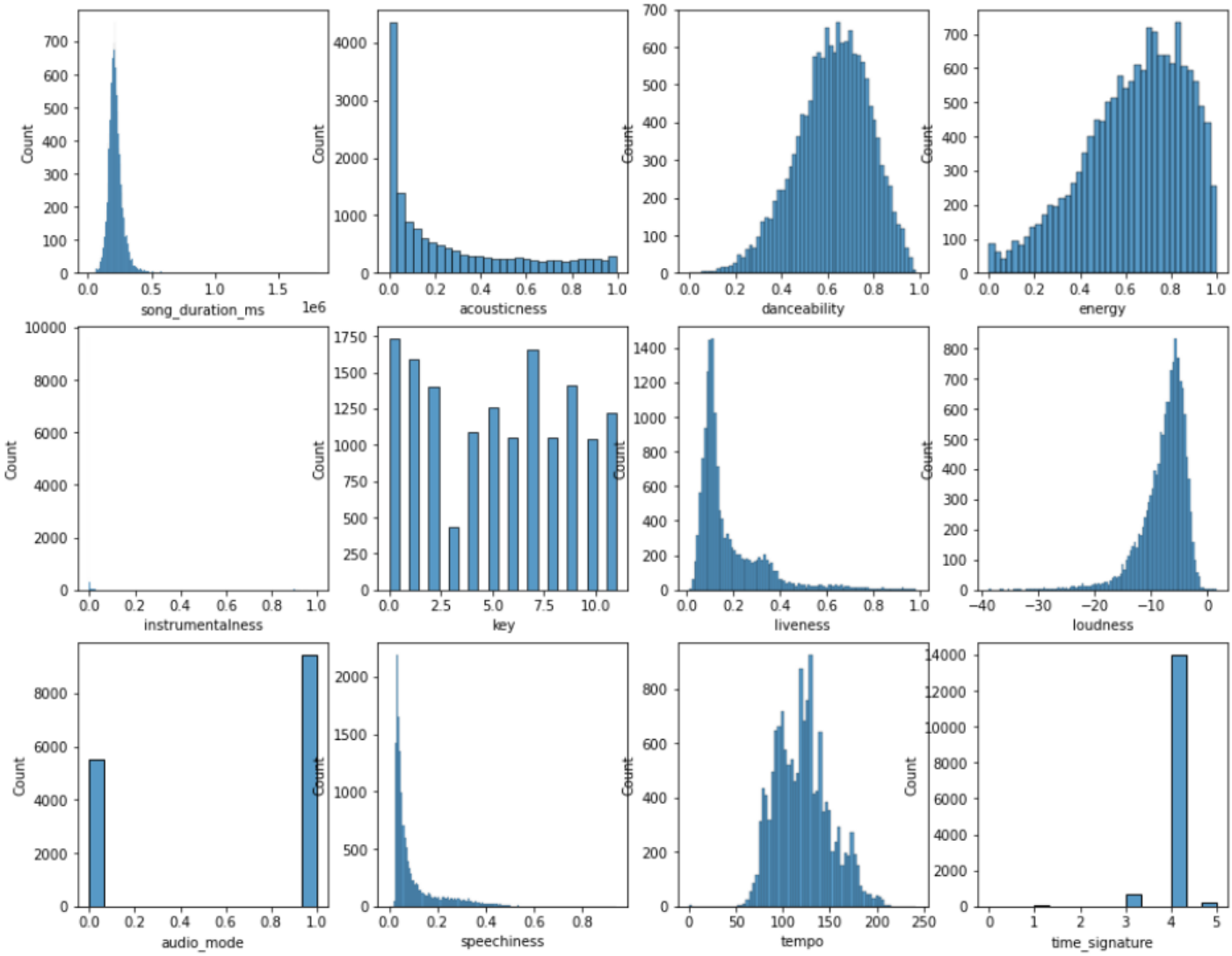

Figure 6: Countplots of all of the Features

First a correlation plot with the remaining features and Song Popularity was made. (Figure 7) Instrumentalness is very negatively correlated with Song Popularity. Instrumentalness vs Song Popularity is seen in the scatterplot/jointplot. (Figure 8) The popular songs all have very low Instrumentalness, this is a very good indicator of popularity, and the feature was kept.

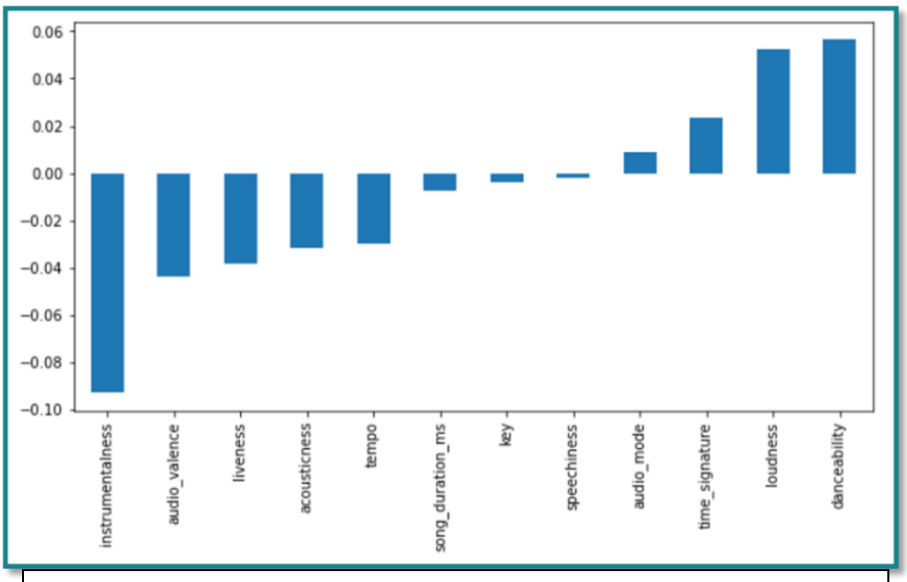

Figure 7: Barplot Showing Correlation with Song Popularity

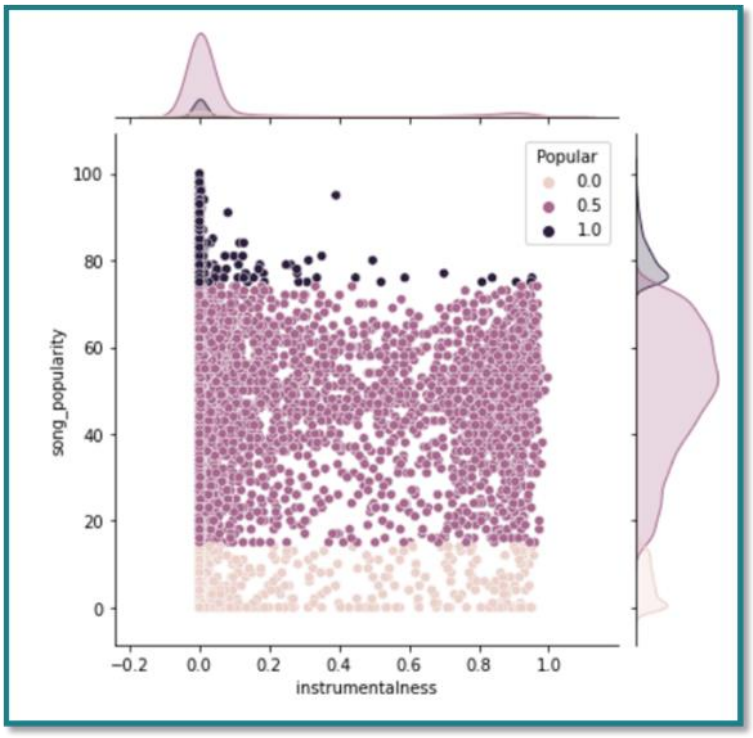

Figure 8: Jointplot Showing Correlation of Instrumentalness with Song Popularity

The data set is now properly cleaned, and the important features were saved to be used in the test. The data set was then split using sklearn with a 70/30 train test split. (Figure 9) The X is the 12 remaining features, and the y is the song popularity from 0-100 making this a multi-class classification problem. The split data was then scaled using MinMAxScaler to make sure that the features aren't unevenly weighted.

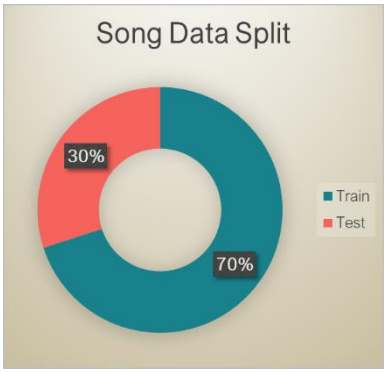

Figure 9: Pi-Chart Showing Train/Test Split

7 tests were done using 5 shallow learning algorithms (2 KNN's) and 1 deep learning model. (Figure 10)

- Test 1: Logistic Classifier
- Test 2: Decision Tree Classifier
- Test 3: Random Forest Classifier
- Test 4: K Nearest Neighbor (K=5) Classifier
- Test 5: K Nearest Neighbor (K=32) Classifier
- Test 6: Support Vector Machine Classifier
- Test 7: Keras Classifier (Deep Learning)

```
logmodel = LogisticRegression(class_weight='balanced')
dtree = DecisionTreeClassifier(class_weight='balanced',min_samples_leaf=15)
rfc = RandomForestClassifier(n_estimators=600,class_weight='balanced')
knn = KNeighborsClassifier(n_neighbors=5,weights='uniform')
knn = KNeighborsClassifier(n_neighbors=32,weights='uniform')
svc_model = SVC(class_weight='balanced',probability=True)
```

Figure 10: Python Code of Shallow Learning Models

An error rate plot was made to pick the nearest neighbor with the lowest error, this was 32 as seen below. (Figure 11) The default (5) and the 32 models were both used for comparison.

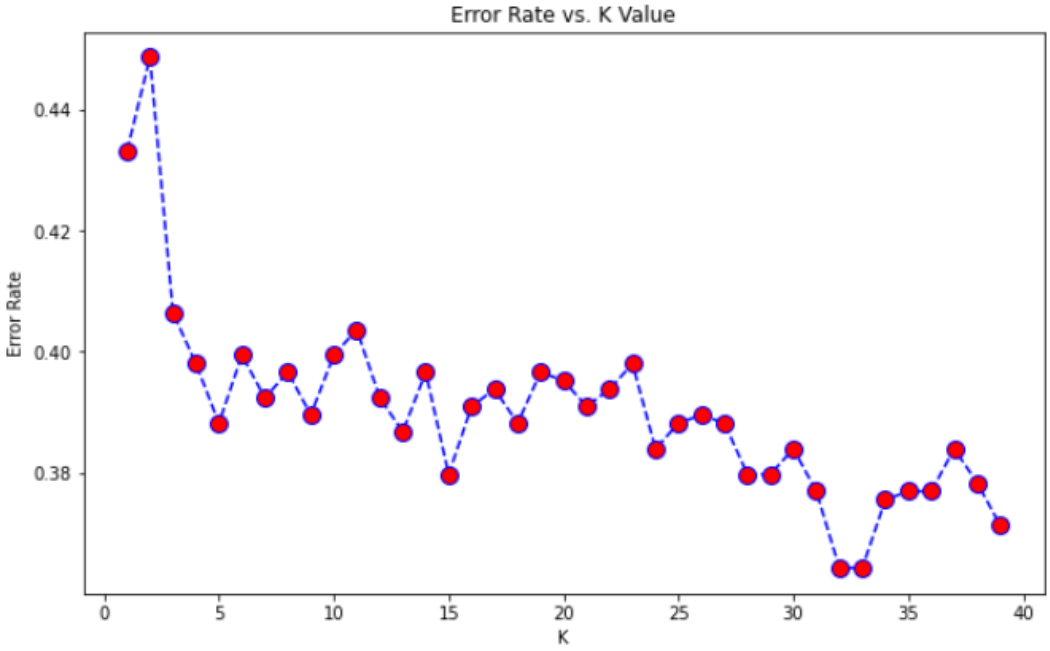

Figure 11: Error Plot of KNN=1:40 to Decide on Best Model

The Keras model had layer of 12 for the number of features tested and then a drop off of half to be 6 and finally a 1 for the sigmoid (binary) level (these layers performed the best). (Figure 12)

```
model = Sequential()

model.add(Dense(12,activation='relu'))

model.add(Dense(12,activation='relu'))

model.add(Dense(6,activation='relu'))

model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam',loss='binary_crossentropy', metrics=["accuracy"])
```

Figure 12: Python Code of Deep Learning Model

These tests performed very poorly as shown by the mean error of each of the classifiers which was about 50. (Figure 13) If a prediction has an error of 50 out of 100 total values, it is basically just a guess. The scatterplot also shows the predicted values vs. real values. (Figure 14) This also shows that most of the predictions are around 50. There are a few reasons for this low performance. Firstly, most of the data was "Average" which is in the 40-60 range. Since the data wasn't evenly distributed, the tests failed. Another problem is that most of the classifiers are better at recognizing binary classifications not multi-class. The models that can solve multi-class need one hot encoding to properly test the data.

```
Test: Logistic Classifier

Mean Error =  52.73638289858402

Test: Decision Tree Classifier

Mean Error =  52.73700436916154

Test: Random Forest Classifier

Mean Error =  52.73925890429801

Test: K Nearest Neighbor (K=5) Classifier

Mean Error =  52.73965509267507

Test: K Nearest Neighbor (K=32) Classifier

Mean Error =  52.74013065231383

Test: Support Vector Machine Classifier

Mean Error =  52.73989173608027

Test: Keras Classifier

Mean Error =  51.82424906763959
```

Figure 13: Mean Squared Error for All Models

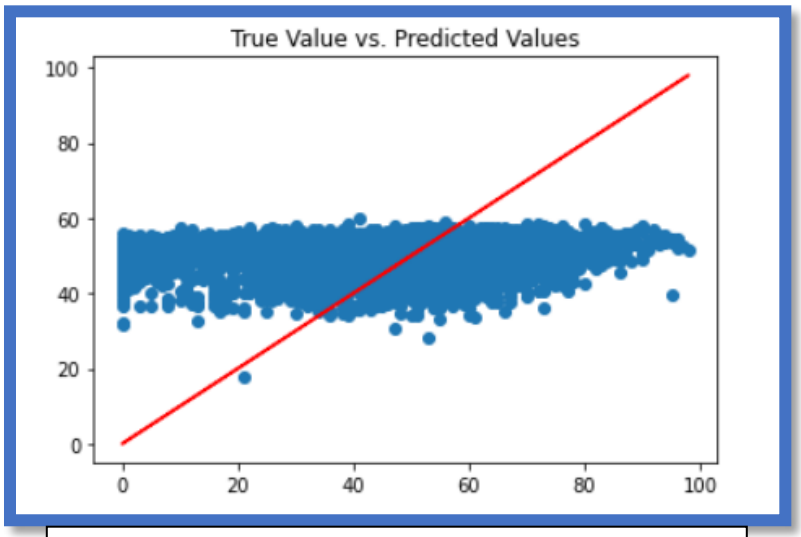

Figure 14: Scatterplot of True vs Predicted Values

The data was split into two categories to solve the problems with multi-class and turn it into a binary classification. This gave some good-looking results if it was based on accuracy and area under the ROC curve. (Figure 15) The Logistic Regression and Keras models both gave a .805 area under the curve, the SVM gave a .8 value. These scores are unfortunately not correct as they are heavily biased. This is due to the data split. Anything over 85% was “Popular” and anything below 85% was “Not Popular. (Figure 16) Most of the points were “Not Popular”-14,724 and few were “Popular”-202. This skewed data was only predicting “Not Popular” with no consequence as there were so many more values. The accuracy was exceedingly high despite getting all of the “Popular” samples wrong. This is shown in the classification reports/confusion matrices. (Figure 17)

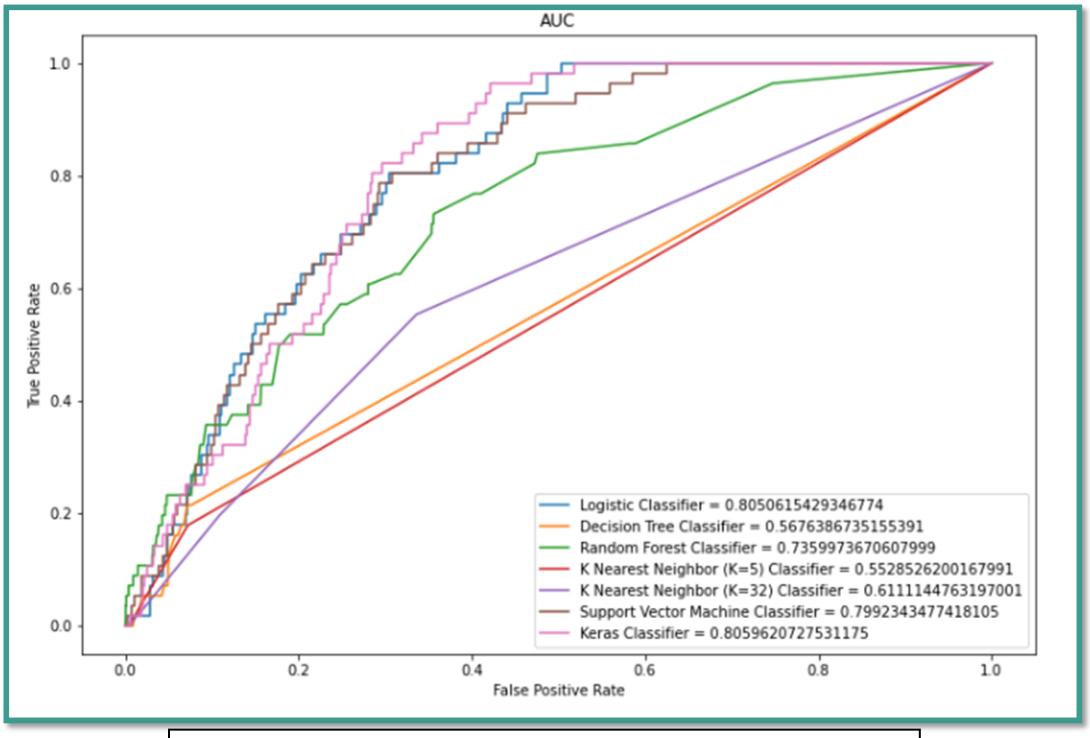

Figure 15: Area Under the ROC Curve for 7 Models

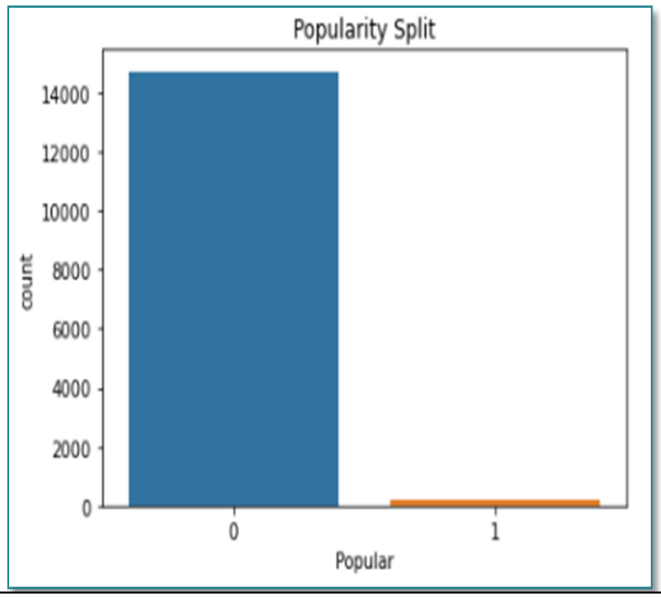

Figure 16: Data Split of Popular (1) & Not Popular (0)

Test: K Nearest Neighbor (K=5) Classifier

	precision	recall	f1-score	support
0	0.99	1.00	0.99	4422
1	0.00	0.00	0.00	56
accuracy			0.99	4478
macro avg	0.49	0.50	0.50	4478
weighted avg	0.98	0.99	0.98	4478

```
[[4422    0]
 [  56    0]]
```

Test: K Nearest Neighbor (K=32) Classifier

	precision	recall	f1-score	support
0	0.99	1.00	0.99	4422
1	0.00	0.00	0.00	56
accuracy			0.99	4478
macro avg	0.49	0.50	0.50	4478
weighted avg	0.98	0.99	0.98	4478

```
[[4422    0]
 [  56    0]]
```

Test: Support Vector Machine Classifier

	precision	recall	f1-score	support
0	1.00	0.69	0.81	4422
1	0.03	0.80	0.06	56
accuracy			0.69	4478
macro avg	0.51	0.75	0.44	4478
weighted avg	0.98	0.69	0.80	4478

```
[[3044 1378]
 [  11   45]]
```

Test: Logistic Classifier

	precision	recall	f1-score	support
0	1.00	0.67	0.80	4422
1	0.03	0.80	0.06	56
accuracy			0.67	4478
macro avg	0.51	0.74	0.43	4478
weighted avg	0.98	0.67	0.79	4478

```
[[2967 1455]
 [  11   45]]
```

Test: Decision Tree Classifier

	precision	recall	f1-score	support
0	0.99	0.92	0.96	4422
1	0.03	0.21	0.06	56
accuracy			0.92	4478
macro avg	0.51	0.57	0.51	4478
weighted avg	0.98	0.92	0.94	4478

```
[[4087  335]
 [  44   12]]
```

Test: Random Forest Classifier

	precision	recall	f1-score	support
0	0.99	1.00	0.99	4422
1	0.50	0.04	0.07	56
accuracy			0.99	4478
macro avg	0.74	0.52	0.53	4478
weighted avg	0.98	0.99	0.98	4478

```
[[4420    2]
 [  54    2]]
```

Figure 17 : Classification Reports & Confusion Matrices for Shallow Models

To fix this issue, the data set must be properly split. To do this, the data was split into 3 columns, "Popular"-1,141 anything above 75% popularity, "Not Popular"-1,226 anything below 15% popularity, and "Average"-12,559 anything in between (which will be evaluated later). (Figure 18) The classes are now much more evenly split, and the tests can be done properly while avoiding bias. The area under the ROC curve gave very promising results, .785 for the Random Forest Classifier, .743 for SVM and .724 for Keras. (Figure 19) The values used for the curve were the probability values using PredictProba command instead of Predict, 1s and 0s. The classification reports/confusion matrices showed promising results as values were in all 4 boxes, meaning the models were actually predicting values. (Figure 20)

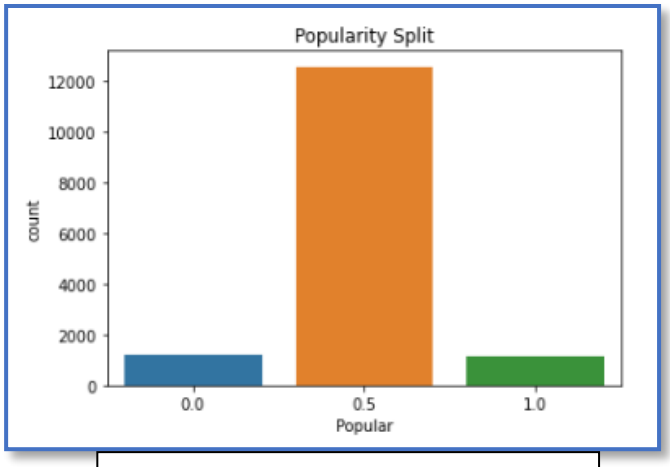

Figure 18: Data Split of Popular (1), Average (.5) & Not Popular (0)

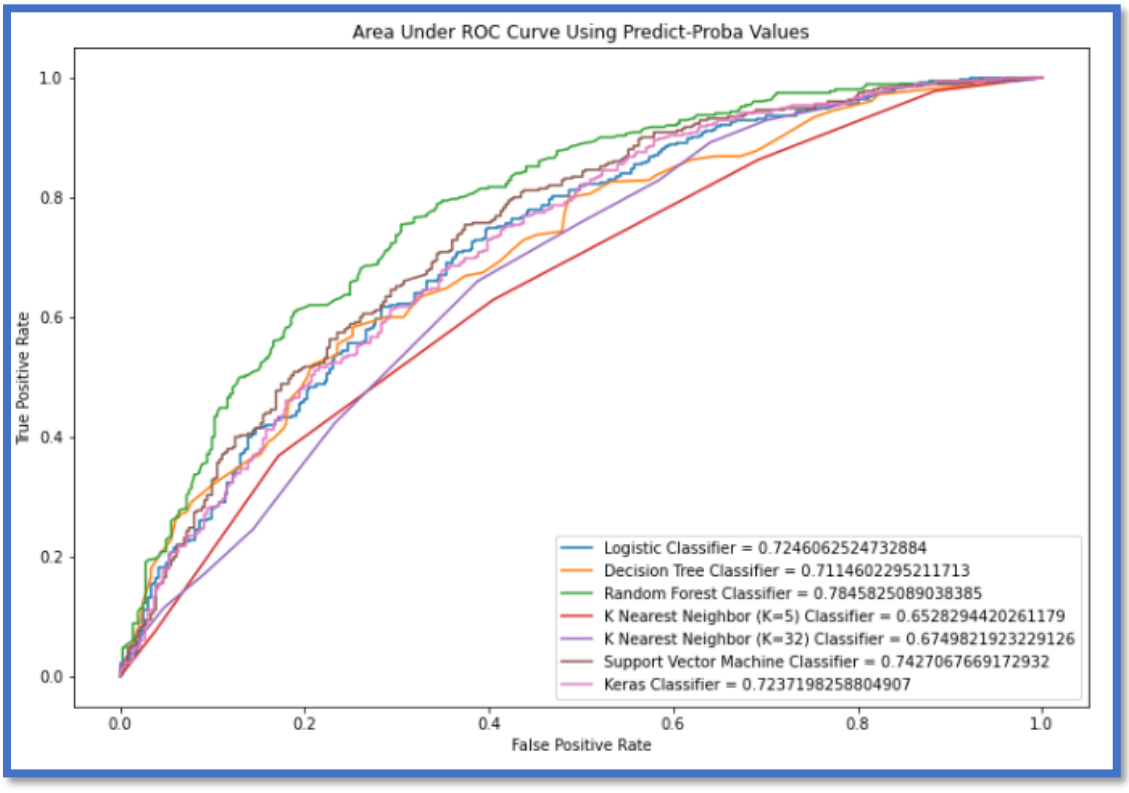

Figure 19: Area Under the ROC Curve for 7 Models

```
Test: Logistic Classifier

              precision    recall  f1-score   support

         0.0       0.71      0.59      0.64       361
         1.0       0.64      0.75      0.69       350

    accuracy                           0.67       711
   macro avg       0.67      0.67      0.67       711
weighted avg       0.67      0.67      0.67       711

[[212 149]
 [ 87 263]]

Test: Decision Tree Classifier

              precision    recall  f1-score   support

         0.0       0.65      0.67      0.66       361
         1.0       0.65      0.63      0.64       350

    accuracy                           0.65       711
   macro avg       0.65      0.65      0.65       711
weighted avg       0.65      0.65      0.65       711

[[243 118]
 [128 222]]

Test: Random Forest Classifier

              precision    recall  f1-score   support

         0.0       0.69      0.75      0.72       361
         1.0       0.72      0.66      0.69       350

    accuracy                           0.70       711
   macro avg       0.71      0.70      0.70       711
weighted avg       0.71      0.70      0.70       711

[[270  91]
 [119 231]]
```

```
Test: K Nearest Neighbor (K=5) Classifier

              precision    recall  f1-score   support

         0.0       0.62      0.60      0.61       361
         1.0       0.60      0.63      0.61       350

    accuracy                           0.61       711
   macro avg       0.61      0.61      0.61       711
weighted avg       0.61      0.61      0.61       711

[[215 146]
 [130 220]]

Test: K Nearest Neighbor (K=32) Classifier

              precision    recall  f1-score   support

         0.0       0.65      0.61      0.63       361
         1.0       0.62      0.66      0.64       350

    accuracy                           0.64       711
   macro avg       0.64      0.64      0.64       711
weighted avg       0.64      0.64      0.64       711

[[221 140]
 [119 231]]

Test: Support Vector Machine Classifier

              precision    recall  f1-score   support

         0.0       0.73      0.59      0.65       361
         1.0       0.65      0.78      0.71       350

    accuracy                           0.68       711
   macro avg       0.69      0.68      0.68       711
weighted avg       0.69      0.68      0.68       711

[[212 149]
 [ 78 272]]
```

Figure 20 : Classification Reports & Confusion Matrices for Shallow Models

The next test was to see if the best model, RFC, was able to predict the “Average” Scores and detect differences between upper and lower averages. The data was prepared using the same scaling method. The results were unclear, the slope was 10.49 (correlation) but with so many values (12,559) was this significant? (Figure 21) An ANOVA was done to test for significance and the P-Value was 4.7x10^-44 which shows that the data is indeed significant and Average scores were individual (not all 50).

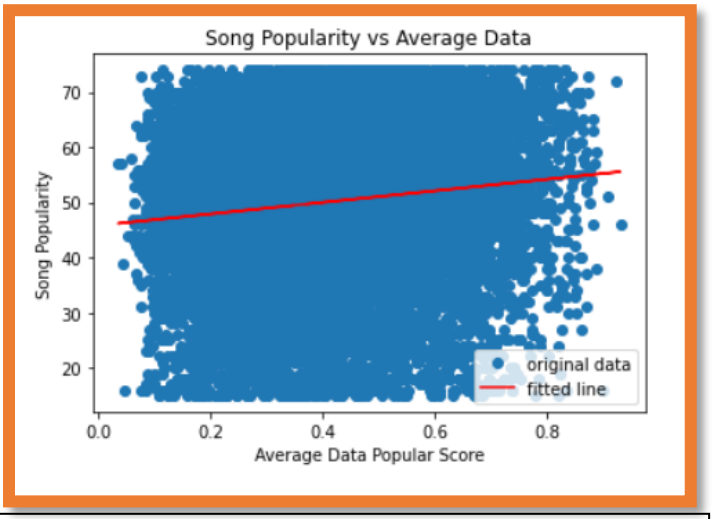

Figure 21: Scatterplot of Song Popularity vs Average

Andrew Birnbaum

In conclusion, this model (RFC) gave an impressive 78.5% accuracy, this is more than 28.5% higher than the guessing value. It would be 50% if the data was evenly split, it is a little uneven. With this model, songs can be predicted if they are going to be a hit (extremely popular) or flop (very unpopular) without even listening to it. This model can also predict with a fine accuracy where in the average range a song will fall (upper or lower average).

There are many applications for this model such as creating albums or music. When making albums, there is always 2 very popular BANGERZ! and 8 mediocre songs. This model can help advise producers on which songs to put on the album. When making music, artists can use "Popular" features to fine tune their songs. They can add parts or take them out depending on what the model tells them.

There are some problems associated with this model:

- The model has very good accuracy, but unfortunately it is still wrong 21.5% of the time. When it is incorrect, the model will tell you that a flop song will be extremely popular. This is an extreme error!
- Another problem is the actual question being tested. Is this song very popular or very unpopular? Most people could answer this question by listening to the song for a few seconds. It is easy to distinguish between extremes.
- This model also can't predict popularity on a range like the songs from the data set. Having a scale for 0-100 would be a better model.

Some possible solutions:

- Firstly, this model is not supposed to be the sole decision maker in choosing songs. You should definitely listen to the song and make proper judgement before producing it.

Andrew Birnbaum

- A harder question would be to try to distinguish between Average songs and Popular songs, if this can be done, the model will be much better.
- The model would have to be a multi-class classification using one-hot encoding to predict popularity values. The model may be more valuable if it was continuous values from 0-100 rather than 100 separate ranks.