# Spotify Top Spots: Is There Rhyme or Reason?

**S2077483** **S2595518** **S1977113**

## Abstract

This paper seeks to develop a reliable method for predicting if a song will reach the top 10 in the Spotify charts using the song's characteristics and introducing several historical popularity metrics. We found the historical popularity metrics to be key features in improving model accuracy. The best-performing models are concluded to be Random Forest classifier (75.8% accuracy, 74.2% F1-score, and 84.3% ROC AUC) and Extreme Gradient Boosting (78.6% accuracy, 77.8% F1-score, and 87.6% ROC AUC). We then use the Extreme Gradient Boosting model on newly released songs to obtain two potential global top 10 hits.

## 1 Introduction

Spotify is a widely recognised music streaming platform that offers detailed information on song characteristics such as danceability, valence, and loudness. Leveraging these features alongside metrics reflecting the artist's historical prominence, such as the number of songs previously in top charts and the average rank of those songs, may play a crucial role in predicting a song's popularity. These in conjunction can serve as an indicator of the utmost global commercial success.

Historical popularity has been explored in the past, with research that utilises low and high-level audio features and the date of the song's release with deep neural networks [7]. Another study has combined intrinsic song characteristics with social media statistics, selecting a random forest as the best-performing classifier [12]. The consideration of prior popularity was also a focal point in a separate research paper, aiming to identify the 20 most significant variables for predicting a song's popularity. This study revealed that previous popularity significantly influenced the prediction outcome [11].

This paper seeks to include several historical popularity metrics to improve machine-learning attempts at predicting if a song would reach a top 10 position on the global Spotify charts.

## 2 Data Preprocessing

The Spotify data used for this paper contains information about a song's rank on a given day during the span of January 1, 2017, to May 28, 2023. The columns in the dataset that we will focus on include *Rank*, *Title*, and *Date*. There also were seven song characteristics, including *Danceability*, *Energy*, *Loudness*, *Speechiness*, *Accousticness*, *Instrumentalness*, and *Valence*. To prepare this data for data exploration, we begin by dropping the duplicate songs that appear on the same day when multiple artists are in a given song.

To investigate our interest in the effect of an artist's previous popularity on future popularity, we create four new columns that probe into this idea. *Previous Popularity* refers to whether an artist has ever had a song reach the top ten in the Spotify rankings. *Number of Songs Previous* is the number of songs that an artist has ever shown up in the Spotify top 200 regardless of rank. *Previous Top Rank Avg* is the average top rank an artist's previous songs. Lastly, the *Number of Artists* is simply the number of artists that appear in the song. When creating these columns, it was critical to be mindful of the date so an artist's previous popularity and number of songs relate to the date the song first appeared in the charts. This means an artist's popularity later will not influence their popularity earlier. To accomplish this, we ordered the dataset by date and iterate through each row, keeping track of these variables for each artist to ensure we remained true to what was known at the time. For the case of an artist not having a value for *Previous Top Rank Avg*, we simply encoded them to 201, a value outside of the top 200. Appendix A shows the process of coding these new columns in python jupyter notebook.

The last set steps included encoding whether a song ever reached the *Top 10* spot, encoded to true or false. We also utilise the standard scaler from the scikit-learn python library to scale the song characteristics. After this, we decided to remove the earliest month in the dataset, as those songs include ones that are at the end of their popularity and would be falsely classified as scoring low on the charts. This is due to insufficient data for these songs before February 1, 2017, causing a song or artist to appear worse than reality, such as if a song did reach the top ten or top one spot before February 1, 2017, but had fallen off below top ten when our dataset begins. It is important to note that we still use the earliest month to create *Previous Popularity*, *Number of Songs Previous*, and *Previous Top Rank Avg*. We only remove the month after. Lastly, we remove all duplicates, keeping only the first occurrence so we only contain one observation for a song created by an artist(s). It is with this dataset we begin our data exploration.

## 3 EDA

There are 693 songs in our records that have reached *Top 10* within the charts and 6808 that have not. With our analysis we aim to uncover the factors that might contribute to a song's success, explaining why some songs make it to the *Top 10* while others do not.

We started by exploring the distribution of values for key song characteristics, namely *Danceability*, *Energy*, *Loudness*, *Speechiness*, *Acousticness*, *Instrumentalness*, *Valence*, (as seen in Fig. 7). Surprisingly, the visual examination of the figures did not reveal significant differences between the two groups. The disparities in means were minimal, with the highest observed in *Energy* (0.136) and the lowest in *Acousticness* (0.040). These marginal variations suggest that the intrinsic features of songs do not exhibit substantial differences between those that reach *Top 10* status and those that do not.

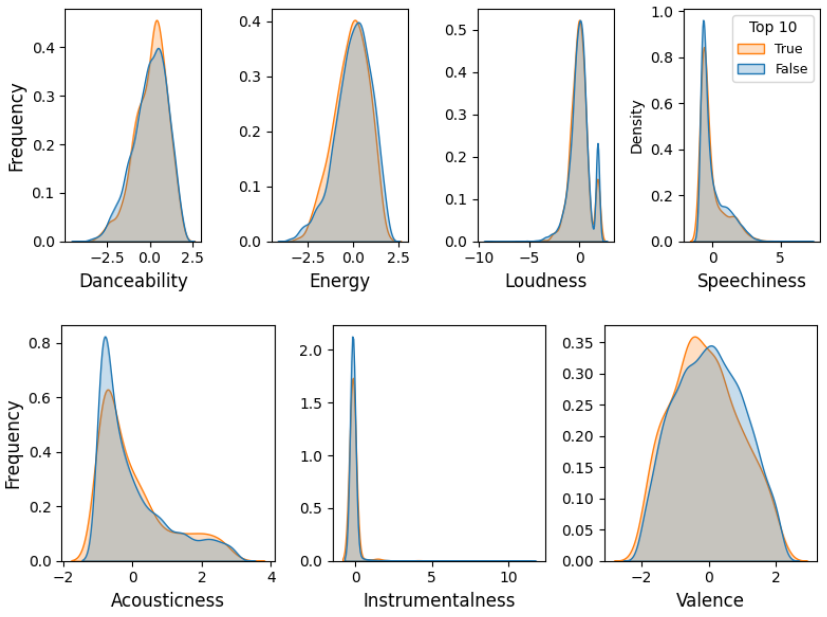


Figure 1: Distribution of song characteristics for songs in and outside the 'Top 10'.

To see if an artist's recorded past success contributes to a song's performance in the charts, we explore the historic popularity metrics, specifically *Previous Popularity*, *Number of Songs Previous*, and *Previous Top Rank Avg*. Overall, 26.0% of the songs in our dataset have a documented previous popularity, a figure that significantly rises to 49.9% for songs that have reached the top 10. This suggests a notable relationship between an artist's prior success and their likelihood of producing a chart-topping song.

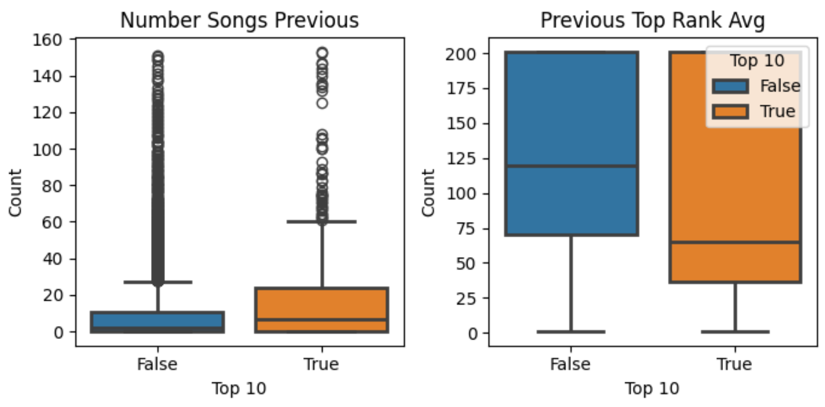


Figure 2: Distribution of popularity metrics within and out of the 'Top 10'.

For the latter two metrics, we plotted the distribution for visual inspection. Most artists do not have a high volume of songs that reach the top 200, as indicated by the distribution of counts between the two groups (as depicted in Fig. 2 on the left). This points us in the direction that there are different measures of popularity. The average of the rank of previous songs provides insight into the mean popularity of songs per artist. For songs reaching the top 10, the average position in the chart is notably higher, with a median of 63 versus 119. This implies that songs achieving top positions typically come from artists who have had songs that were consistently high in the top charts.

## 4 Learning Methods

We followed a systematic methodology to develop a predictive model for determining whether a song will attain a position in the *Top 10* of the global charts. With the cleaned dataset, we defined the features as song characteristics, the number of artists and the historical popularity metrics. The class to be predicted is the *Top 10* column. We applied an 80/20 train-test split with a stratified distribution based on the class label for final model training and evaluation, and a stratified 80/20 train-validation split on the training set for hyperparameter tuning. To address the class imbalance of the *Top 10*, we applied the Synthetic Minority Oversampling Technique (SMOTE) from the imbalanced-learn python library on each one of the sets independently.

Accuracy is a key metric we used for evaluating the models' performance. As we have balanced out the distribution of the *Top 10* class, it should give us an overview of a model's performance. Additionally, we have incorporated the F1 score, which combines precision and recall to ensure that both the false positives and false negatives are considered in the evaluation. As some of the models obtain the class prediction from probabilities, getting the Area Under the Curve of the Receiver Operating Characteristic (ROC AUC) gives us a measure of the model's ability to discriminate between classes across various classification thresholds.

### 4.1 Models

We employed **logistic regression** as a simple base classifier, giving us an initial performance to compare against the more complex models.

The **K-Nearest Neighbours** (KNN) supervised learning algorithm provides a distance based classification based on the majority class of the k-nearest points in the feature space, in our case for the 5 nearest points obtained by the Euclidean distance [8].

**Support Vector Machine Classifier** (SVM/SVC) splits the data into 2 classes in a high-dimensional feature space with a hyperplane that maximises the distance between the decision boundary and the nearest data point (support vectors) from both of the classes [5]. This margin maximisation contributes to better generalisation on classifying unseen examples [10].

The **Random Forest Classifier** is an ensemble of individual decision trees, recursively created with a random subset of data and features [6]. The variability in the creation of the model increases its generalisation capabilities. To ensure reproducibility, we set a consistent random state. The final prediction is the average of the decision tree voting. The following parameters for tree regularisation where considered in the hyperparameter tuning process:

- `n_estimators`: monitoring the number of trees to be constructed
- `max_depth`: the maximum number of levels per decision tree
- `min_samples_split`: the minimum number of samples in a node to perform a split
- `min_sample_leaf`: the minimum number of samples required to be in a childless node [1].

A different decision tree ensemble classifier, **Extreme Gradient Boosting Classifier** (XGBoost), builds decision trees sequentially, with the aim of iterative correction of errors of the previous classifier. The final prediction is formed by combining the predictions made by each tree, each of which is trained on a subset of the data. One distinction between extreme gradient-boosting and Random Forest is that extreme gradient boosting employs a more regularised model, which aids in mitigating the risk of overfitting [9]. The parameters considered and later tuned for the setup of the algorihtm are:

- `n_estimators`: the number of trees
- `max_depth`: the maximum height of the trees
- `learning_rate`: step size for adjusting the weights in the boosting process
- `subsample`: the ratio of training instance to be resampled before setting up the trees
- `colsample_bytree`: the subsample ratio of columns for each tree [4], [3].

## 5 Model Selection

We first created a base logistic regression model with a maximum number of iterations of 1000 based solely on the numerical song characteristics and excluding the historic popularity metrics. This model produced an accuracy score of 55.2%, an F1-score of 55.9%, and an ROC AUC of 59.1% on the validation set. This base model will be improved upon using the historic popularity metrics we extracted and with the models discussed in the learning methods section.

Table 1: Model performance with no parameter adjustment on the validation set

| Model | Accuracy | F1-Score | ROC AUC |
|---|---|---|---|
| Logistic Regression | 66.0% | 63.2% | 69.7% |
| KNN | 62.3% | 58.9% | 65.9% |
| SVM | 66.4% | 62.6% | 71.0% |
| Extreme Gradient Boosting | 83.3% | 81.6% | 83.3% |
| Random Forest | 77.4% | 73.0% | 86.3% |

Table 1 displays the different models with the default parameters given by the scikit-learn library on the validation set. These models now include the historic popularity metrics as features along with the song characteristics. The positive impact of the historic popularity metrics is obvious, as the **Logistic Regression** with them improves its accuracy and F1-score to 66% and 63.2%, respectively. It also greatly improved upon the base ROC AUC with a value of 69.7%. The **KNN** model performed worse than the logistic regression model but was still better than our base model. It also produced the lowest value for the ROC AUC with 65.9%, compared to the other models. **SVM** performs similarly to the logistic regression model but achieves slightly higher accuracy albeit a slightly lower F1-score. The next two models performed the best on the validation set without parameter adjustment. **Random Forest** achieves a 74.4% accuracy, a 73% F1-score, and an ROC AUC value of 86.3%. In comparison, **Extreme Gradient Boosting** boasts better accuracy and F1-score of 83.3% and 81.6%, respectively, but a lower value for the ROC AUC at 83.3%. Appendix B details the ROC AUC graph of all the models compared to the base logistic regression model. Having found the two best models, the next step is to tune the parameters in an attempt to improve the models.

## 6 Results

As the random forest and extreme gradient boosting model perform similarly, we hyperparameter tune the models to see if this would reveal any discrepancies in the two models' performance. The full details of the parameter tuning are shown in Appendix C. For the extreme gradient boosting model, we tuned the model on the validation set, comparing it against its F1-scores and found the best parameterisation of 200 estimators, a max depth of 3, a learning rate of 0.1, a colsample of 0.9, and a subsample of 1. The random forest was similarly tuned on the validation set and compared against its F1-scores. We found this model's best parameterisation to be 200 estimators, a max depth of 10, a minimum sample split of 2, and a minimum sample leaf of 3. Table 2 shows the performance metrics of the tuned models. While both models perform well, the extreme gradient boosting model outperforms the random forest model in all metrics. Notice that the extreme gradient boosting model with parameter tuning receives a worse accuracy and F1-score than its default parameter counterpart. Since the default parameters do not set a tree depth, this can lead to an overfitted model. Thus, while our tuned model appears worse on validation, it is more generalisable to unseen data. With the best parameters found, we recreate the models on the entire training set and test against the unseen test set. The performance of the models on the test set is shown in Table 3. The extreme gradient boosting model on the test set again performs the best compared to the random forest model. Thus, our best model achieves an accuracy of 78.6%, an F1-score of 77.8%, and an area under the curve of 87.6%.

Table 2: The tuned models performance on the validation set

| Tuned Model | Accuracy | F1-Score | ROC AUC |
|---|---|---|---|
| Extreme Gradient Boosting | 81.9% | 81.2% | 90.6% |
| Random Forest | 80.3% | 79.3% | 88.8% |

Table 3: The tuned models performance on the unseen test set

| Tuned Model | Accuracy | F1-Score | ROC AUC |
|---|---|---|---|
| Extreme Gradient Boosting | 78.6% | 77.8% | 87.6% |
| Random Forest | 75.8% | 74.2% | 84.3% |

Both the random forest and extreme gradient boosting models greatly utilise the historical popularity metric features, albeit in slightly different ways. The historical popularity metrics account for 50.9% of the total feature importances. *Previous Popularity* and *Previous Top Rank Avg* are the majority of this, with 21.0% and 19.4% respectively. The extreme gradient boosting model places even greater importance on the historical popularity metrics with them accounting for 60.1% of the total feature importances. However, this model primarily relies on *Previous Popularity*, with a feature importance of 44.7%. The full list of feature importances for both models are given in Appendix D.

## 7 Future Predictions

The Extreme Gradient Boosting model achieved the best of all three performance metrics of the multitude of models we implemented. With this model in hand, we scoured the web for newly released songs that we thought might reach the top 10 on Spotify. We used songs from popular artists based on the importance of the historical popularity metrics for our models. Importantly, we ensured that the song did not reach inside of the top 20 on the Spotify charts [2].

Since our data extent is limited to May 28, 2023, to pull in data for new songs, we incorporated the use of the Spotify web API and spotipy python library. This allowed us to enter in track IDs to pull in the song characteristics of new songs. We also use the same standard scaler, saved as a pickle, to ensure the song characteristics are scaled consistently. However, for the historical popularity metrics, we are restricted to what our cleaned data contained. With this, we built the features for new songs as of late November 2023. The fully constructed data for two songs, Houdini by Dua Lipa and Real Love by Martin Garrix and Lloyiso, are shown in Appendix E.

We predict that Houdini, a song released on November 10, 2023, will reach the top 10 in the Spotify charts. Dua Lipa is a very popular artist, with *Previous Popularity* set to true, 21 *Number Songs Previous*, and a *Previous Song Top Rank Avg* of 41.4. Given her impressive resume, our model has a 76.6% likelihood that the song, Houdini, would enter the top 10 rankings.

We also predict that Real Love, on September 22, 2023, will also reach the top 10 in the Spotify charts. In contrast to Dua Lipa, these artists are not popular. Based on the data up to May 28, 2023, they have a *Previous Popularity* set to false, 0 *Number Songs Previous*, and a *Previous Song Top Rank Avg* of 201. This shows that the song characteristics still help determine if a song reaches the top 10 in the charts. In this case, the model had a 54.2% likelihood that the song would enter the top 10 rankings, showing that the historic popularity metrics make a good song more likely to enter the top 10, but it is not an all-encompassing feature.

## 8 Conclusion

In conclusion, the previous popularity metrics proved to be an important part of the prediction of the song reaching Top 10 in the charts, improving our model's classification abilities. Our best-performing model is Extreme Gradient Boosting with the tuned parameters, achieving 87.6% ROC AUC.

The model can also identify songs that can reach the top 10 in the Spotify rankings without requiring "good" historical popularity scores. This is incredibly important as an ineffective model would consider all songs from a popular artist to be in the top 10 rankings or all songs from a nonpopular artist to never reach the top 10, which we know does not reflect reality. Thus, our model functions well in its intent, separating good songs capable of top-10 status from those that are not. The historical popularity metrics are some of the most important features in the random forest and extreme gradient boosting models, greatly increasing the performance and confidence in our model's decisions.

One limitation to our approach for checking artist's popularity is that the songs with multiple artists it was obtained for the collaboration. A potential way to improve this would be to establish the individual popularities and devise methods to consolidate them into one.

For the artists with no previous record in the charts, we assigned their existing rank average to be 201. However, it's worth noting that this approach may be somewhat misleading, as it implies a position just below the top 200 charts, which may not accurately reflect the actual scenario.

## References

[1] Random forest classifier documentation. `https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html` [Accessed: 15/11/23].

[2] Spotify top songs. `https://kworb.net/spotify/artist/1vyhD5VmyZ7KMfW5gqLgo5_songs.html` [Accessed: 18/11/23].

[3] Xgboost documentation. `https://xgboost.readthedocs.io/en/stable/` [Accessed: 15/11/23].

[4] Candice Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz. A comparative analysis of xgboost, 11 2019.

[5] Nello Cristianini and John Shawe-Taylor. *Support Vector Machines*, page 93–124. Cambridge University Press, 2000.

[6] Adele Cutler, David Cutler, and John Stevens. *Random Forests*, volume 45, pages 157–176. 01 2011.

[7] Michael Vötter Eva Zangerle, Ramona Huber. Hit song prediction: Leveraging low and high level audio features. 2019.

[8] Joos Korstanje. *The kNN Model*, pages 169–177. Apress, Berkeley, CA, 2021.

[9] Shubham Malik, Rohan Harode, and Akash Singh. Xgboost: A deep dive into boosting ( introduction documentation ), 02 2020.

[10] Christopher D Manning, Prabhakar Raghavan, Hinrich Schütze, and University Of Cambridge. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, 2009.

[11] Matteo Matera. *The Music Industry in the Streaming Age: Predicting the Success of a Song on Spotify*. Universidade NOVA de Lisboa (Portugal) ProQuest Dissertations Publishing, 2021.

[12] Mafas Raheem Yap Kah Yee. *Predicting Music Popularity Using Spotify and YouTube Features*. INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY, 2022.

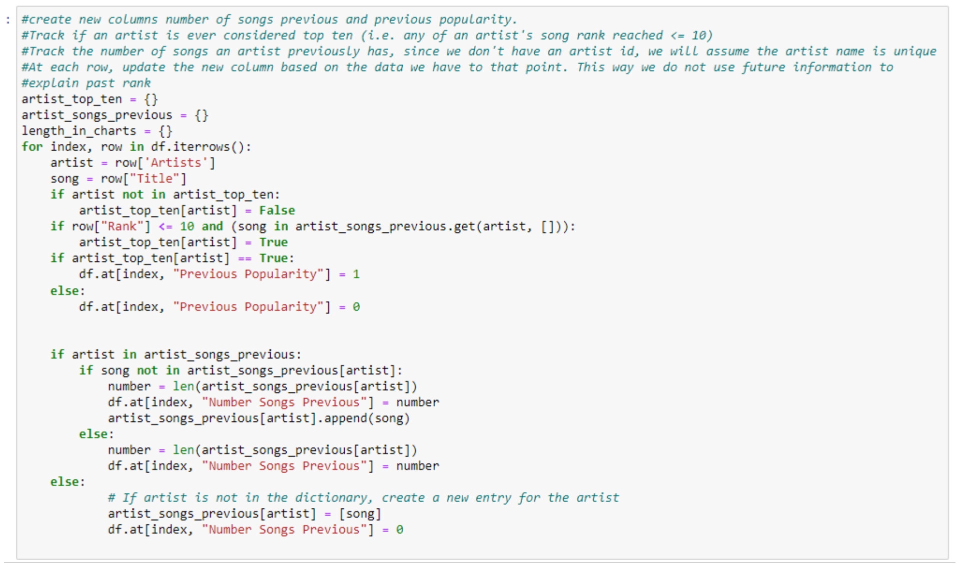
## Appendices

## A Data Processing Code

The creation of the historical popularity metrics required diligent and careful manipulation of the data in order to be in respect to time. The two figures below detail our process to create these new columns. Assume that the dataframe *df* has already dropped the duplicate rows, sorted *Date* in nondescending order, and created *Highest Rank* by taking the minimum of *Rank* as grouped by *Atists* and *Title*.

To create *Previous Popularity* and *Number Songs Previous* we iterate through the dataframe row by row, beginning at the earliest date in the dataset, January 1, 2017. Two dictionaries help keep track of these features. `artist_top_ten` holds the artist as a key, and a value of either True or False depending on if the artist has a song that has previously reached the top 10 in the charts. It also verifies that if setting the `artist_top_ten` to true, the song must already have been in the dataset. This helps fix the edge case where a song enters the top 10 on the first day of its arrival in the charts. This way, we keep it limited to *previous* popularity.

The second dictionary `artist_songs_previous` holds the artist as the key, and a growing list of songs as the value. Here, we set *Number Songs Previous* to the length of the list before adding in the current row.



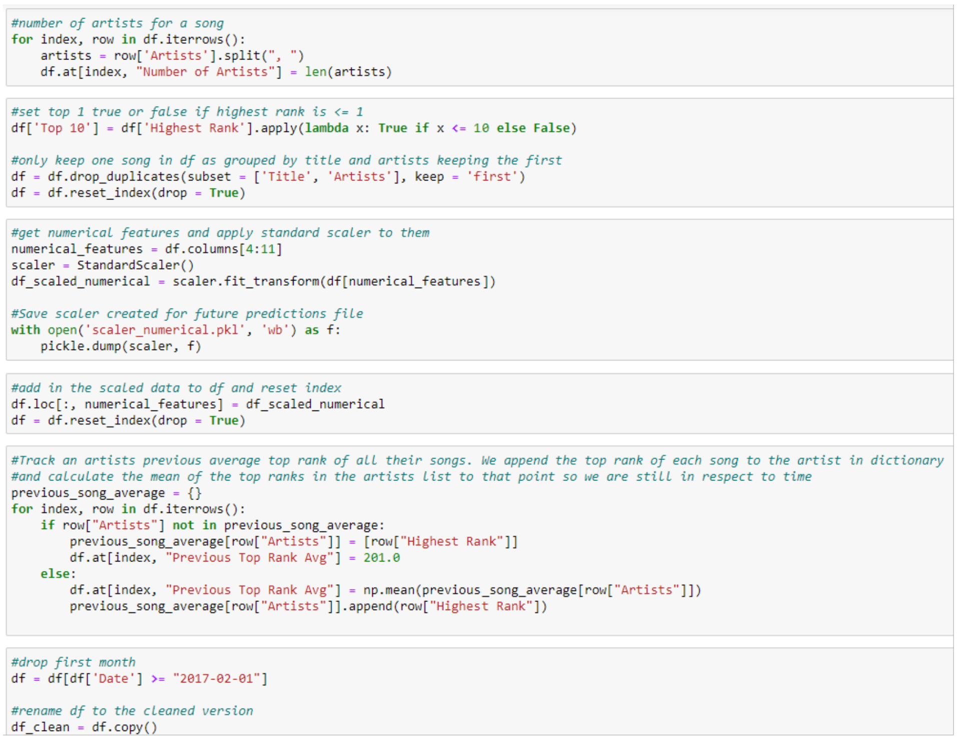
```
#create new columns number of songs previous and previous popularity.
#Track if an artist is ever considered top ten (i.e. any of an artist's song rank reached <= 10)
#Track the number of songs an artist previously has, since we don't have an artist id, we will assume the artist name is unique
#At each row, update the new column based on the data we have to that point. This way we do not use future information to
#explain past rank
artist_top_ten = {}
artist_songs_previous = {}
length_in_charts = {}
for index, row in df.iterrows():
    artist = row['Artists']
    song = row["Title"]
    if artist not in artist_top_ten:
        artist_top_ten[artist] = False
    if row["Rank"] <= 10 and (song in artist_songs_previous.get(artist, [])):
        artist_top_ten[artist] = True
    if artist_top_ten[artist] == True:
        df.at[index, "Previous Popularity"] = 1
    else:
        df.at[index, "Previous Popularity"] = 0


    if artist in artist_songs_previous:
        if song not in artist_songs_previous[artist]:
            number = len(artist_songs_previous[artist])
            df.at[index, "Number Songs Previous"] = number
            artist_songs_previous[artist].append(song)
        else:
            number = len(artist_songs_previous[artist])
            df.at[index, "Number Songs Previous"] = number
    else:
            # If artist is not in the dictionary, create a new entry for the artist
            artist_songs_previous[artist] = [song]
            df.at[index, "Number Songs Previous"] = 0
```

Figure 3: Creation of *Previous Popularity* and *Number Songs Previous*.

This last block of code drops the duplicate songs, keeping only the first. We create the standardised song characteristics using standard scaler and save it as a pickle. From there, we calculate *Previous Top Rank Avg* using a dictionary. The artist is the key with the list of each song's highest rank as the value. We initalise the mean to 201, and then calulate the mean of the list before appending in the current song's highest rank.

Lastly, we dropped the first month in the dataset, giving us our cleaned dataset.



```
#number of artists for a song
for index, row in df.iterrows():
    artists = row['Artists'].split(", ")
    df.at[index, "Number of Artists"] = len(artists)
```

```
#set top 1 true or false if highest rank is <= 1
df['Top 10'] = df['Highest Rank'].apply(lambda x: True if x <= 10 else False)

#only keep one song in df as grouped by title and artists keeping the first
df = df.drop_duplicates(subset = ['Title', 'Artists'], keep = 'first')
df = df.reset_index(drop = True)
```

```
#get numerical features and apply standard scaler to them
numerical_features = df.columns[4:11]
scaler = StandardScaler()
df_scaled_numerical = scaler.fit_transform(df[numerical_features])

#Save scaler created for future predictions file
with open('scaler_numerical.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

```
#add in the scaled data to df and reset index
df.loc[:, numerical_features] = df_scaled_numerical
df = df.reset_index(drop = True)
```

```
#Track an artists previous average top rank of all their songs. We append the top rank of each song to the artist in dictionary
#and calculate the mean of the top ranks in the artists list to that point so we are still in respect to time
previous_song_average = {}
for index, row in df.iterrows():
    if row["Artists"] not in previous_song_average:
        previous_song_average[row["Artists"]] = [row["Highest Rank"]]
        df.at[index, "Previous Top Rank Avg"] = 201.0
    else:
        df.at[index, "Previous Top Rank Avg"] = np.mean(previous_song_average[row["Artists"]])
        previous_song_average[row["Artists"]].append(row["Highest Rank"])
```

```
#drop first month
df = df[df['Date'] >= "2017-02-01"]

#rename df to the cleaned version
df_clean = df.copy()
```

Figure 4: Creation of *Previous Top Rank Avg* and standardising of the song characteristics.

## B Models ROC AUC

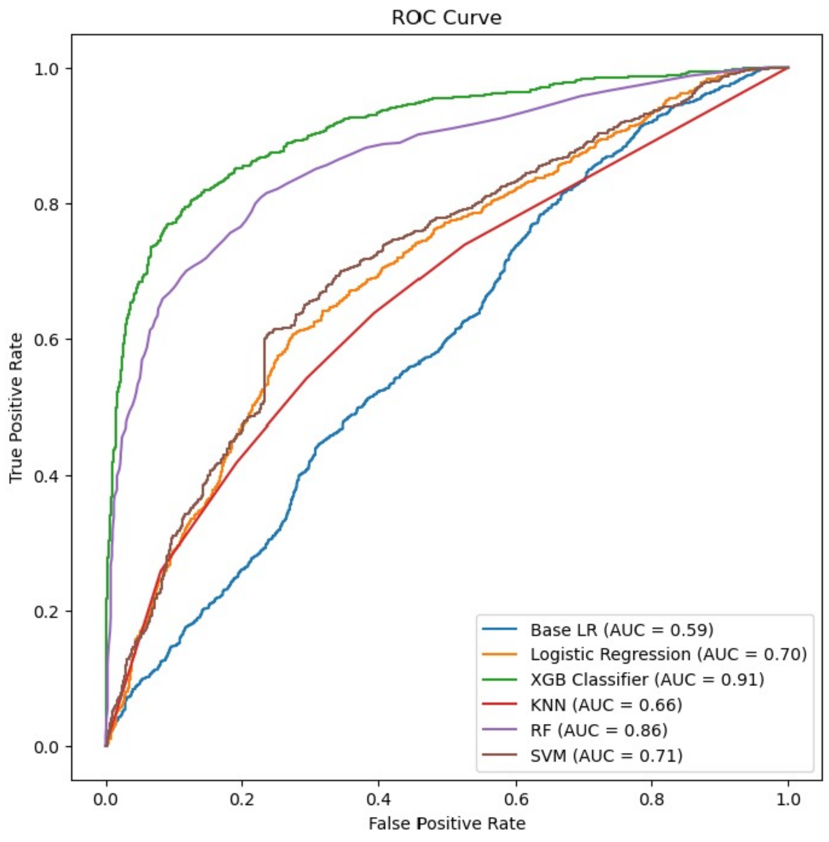


Figure 5: ROC AUC Graph for Model Selection in comparison to the Base Logistic Regression model. All models were tested on the validation set.

## C Hyperparameter tuning details

This details the parameters tested while hyperparameter tuning the random forest and extreme gradient-boosting model. This was performed on the validation set and saving the parameters with the best F1-score.

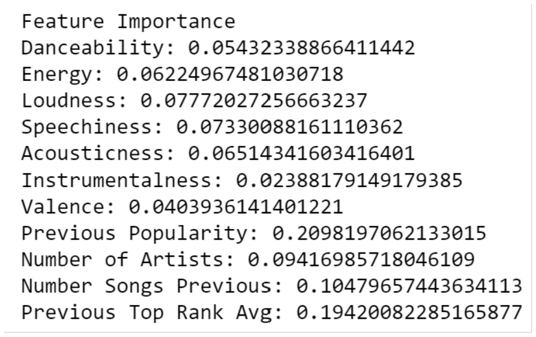
### C.1 Random Forest Hyperparameter Tuning

- Number of estimators: [100, 200, 300]
- Maximum Depth: [1, 2, 5, 10]
- Minimum Samples Split Values: [2, 4, 6]
- Minimum Samples Leaf Values: [1, 2, 3]

### C.2 Extreme Gradient Boosting Hyperparameter Tuning

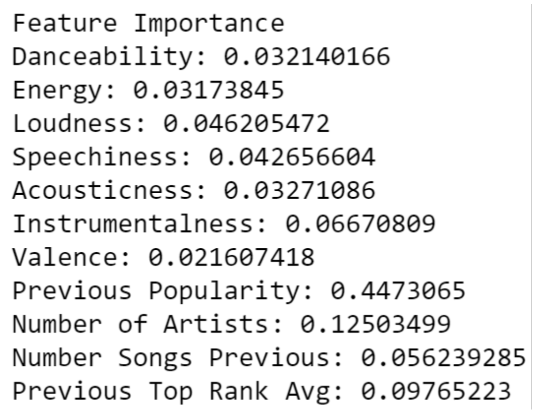
- Number of estimators: [100, 200, 300]
- Learning Rate: [0.01, 0.1, 0.2]
- Maximum Depth: [3, 5, 10]
- Subsample: [0.8, 0.9, 1.0]
- Colsample by Tree: [0.8, 0.9, 1.0]

## D Models feature importance



```
Feature Importance
Danceability: 0.05432338866411442
Energy: 0.06224967481030718
Loudness: 0.07772027256663237
Speechiness: 0.07330088161110362
Acousticness: 0.06514341603416401
Instrumentalness: 0.02388179149179385
Valence: 0.0403936141401221
Previous Popularity: 0.2098197062133015
Number of Artists: 0.09416985718046109
Number Songs Previous: 0.10479657443634113
Previous Top Rank Avg: 0.19420082285165877
```

Figure 6: Feature Importances for the hyperparameter tuned Random Forest model on the test set.



```
Feature Importance
Danceability: 0.032140166
Energy: 0.03173845
Loudness: 0.046205472
Speechiness: 0.042656604
Acousticness: 0.03271086
Instrumentalness: 0.06670809
Valence: 0.021607418
Previous Popularity: 0.4473065
Number of Artists: 0.12503499
Number Songs Previous: 0.056239285
Previous Top Rank Avg: 0.09765223
```

Figure 7: Feature Importances for the hyperparameter tuned Extreme Gradient Boosting model on the test set.

## E Future Prediction Song Features

Table 4: Features of the newly released songs for future predictions using the Spotify API for the song characteristics.

| Song Name | Artist | Danceability | Energy | Loudness | Speechiness |
|---|---|---|---|---|---|
| Real Love | Martin Garrix, Lloysio | -0.311 | 0.702 | 0.416 | -0.342 |
| Houdini | Dua Lipa | 0.595 | 1.021 | 0.780 | -0.141 |

Table 5: Features of the newly released songs for future predictions using the Spotify API for the song characteristics.

| Song Name | Artist | Acousticness | Instrumentalness | Valence |
|---|---|---|---|---|
| Real Love | Martin Garrix, Lloysio | -0.758 | -0.375 | -1.042 |
| Houdini | Dua Lipa | -0.845 | 0.021 | 1.876 |

Table 6: Features of the newly released songs for future predictions using our cleaned dataset for *Previous Popularity* and *Number Songs Previous*.

| Song Name | Artist | Previous Popularity | Number Songs Previous |
|---|---|---|---|
| Real Love | Martin Garrix, Lloysio | 0.0 | 0.0 |
| Houdini | Dua Lipa | 1.0 | 21.0 |

Table 7: Features of the newly released songs for future predictions using our cleaned dataset for *Previous Top Rank Avg* and *Number of Artists*.

| Song Name | Artist | Previous Top Rank Avg | Number of Artists |
|---|---|---|---|
| Real Love | Martin Garrix, Lloysio | 201.0 | 2 |
| Houdini | Dua Lipa | 41.4 | 1 |

## 9 Individual Contributions

**S2077483:**

**Data exploration**:

- performed EDA on the dataset, raw and cleaned
- applied the cleaning process
- data preprocessing for model predictions, synthetic sampling to balance out the class distribution (SMOTE), split up for the training/val/test sets

**Model development**: Worked on developing the final models with a focus on the non-tree dependent classifiers: Logistic Regression, SVM, KNN. Explored the potential of applying neural networks for this problem, obtaining preliminary results for a Multilayer Perceptron model. However, we have decided against including this in the report. Developed the workflow for the best model selection, with the utilisation of a baseline model.

- set up for the models
- parameter tuning
- obtaining the models performance for the
- visualisation of model's performance, ROC AUC curve

**Report**: Wrote sections for the Abstract, Introduction, EDA, Learning Methods and Conclusions.

**S2595518:**

- I researched our topic and helped to write the reflective abstract and introduction parts. Moreover, I did research about past written papers and observations and summarized them in the introduction section.
- In addition to this, I have explained some of the models and parameters we used during the project in the Learning Methods section.
- Lastly, I have helped with the future predictions part by finding famous artists' songs that were released after May 28, 2023, by using Spotify history charts. With the help of our model, we made predictions about whether the songs would be top 10 in the charts.

**S1977113:**

**EDA/Data Preprocessing/Future Predictions**:

- EDA on raw dataset
- created columns for previous popularity, number of songs previous, previous top rank avg, number of artists, and top 10.
- scaled song characteristics
- data cleaning
- set up Spotify API, pulled song characteristics of new songs.
- created script to generate the dataset for new songs to feed into the model for future predictions based on the song characteristics given by Spotify API and historical popularity metrics from the cleaned dataset.

**Model Development**: worked on the development of random forest, XGBoost, logistic regression, and Base logistic regression models. Hyperparameter tuned the xgboost model. graphed developed models on roc auc curve

**Report**: wrote sections for Data Processing, Model Selection, Results, Future Predictions and the respective appendicies. Contributed elements of the conclusion.

### 9.1 AI use declaration

We have used both Github Copilot and ChatGPT for the production of parts of the code, with the usage clearly stated in the code. used for: annotating count plots in the EDA (not used included in the report), quick generation of song characteristics from Spotify API to be joined with the cleaned dataset.