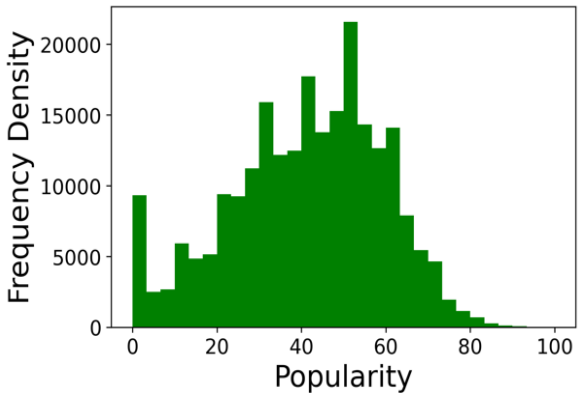
Can a song's popularity on Spotify be predicted?

1. Introduction

1.1 Context and Problem

Being known to the general public is the key to success in the music industry. Nowadays, an artist is considered popular if they manage to break into the top charts of Spotify, a digital music, podcast, and video streaming platform. Being able to predict the popularity of a song can help an artist to improve their track and adapt it to trends before its release. For a label, it can guide their selection of which artists to sign or which track on an album to promote as a single. Popularity of songs will therefore be predicted based on statistics about the song's audio (tempo, key, danceability) and data like artist name and track genre.

1.2 Introduction to the dataset



The dataset used for this analysis is generated from the Spotify API (1) and spans 176,774 songs. It contains 18 features with information about the song, two of them are non-predictive and one of them, popularity, is this report's goal field. Popularity is a value between 0 and 100 generated by Spotify, its computation is based on the total number of plays compared to other tracks as well as how recent those plays are.

Figure 1.1: Frequency of Popularity values 0-100

(Histograms for frequency density of other attributes in the dataset can be found in Appendix A)

1.3 Data preparation

Firstly, duplicate songs had to be dropped. Some tracks with the same individual track ID were listed multiple times with different genre values. By removing these, the dataset was reduced from 232,725 songs to 176,774.

Secondly, the non-predictive attributes were removed. These included the track ID, which is just a tag attached to the song so it can be identified and the track name, which, because of its high cardinality, made pre-processing very complex. Extracting key words from the track name was considered as they could have high predictive values, but this was not carried out. The data then only contained predictive values and the goal field which were separated in x and y values, with y being the 'popularity' and x being the other features.

The data was split, 80% for training and 20% for testing and validation. Repeated k-fold cross validation was used so the testing and validation did not need to be allocated separately. The attributes were sorted into numerical and categorical groups to be processed differently. The categorical group was then split again to separate *artist_name*. Through using a pipeline and a column transformer the pre-processing steps could be chained and the groups of attributes could be treated differently:

- The numerical data was cleaned and then scaled so the attributes were inter-comparable. This was done by using *SimpleImputer* and the *StandardScaler.*
- The categorical data, except *artist_name*, was encoded so each value was represented by an integer. *OneHotEncoder* allows these variables to be converted into a binary form.
- The *artist_name* was encoded differently because of its high cardinality. By using target encoder, the dimensions of the dataset were not altered.

Through using a column transformer, these steps in the pipeline could be chained and the relevant transformation applied to each subset of the attributes.

2. Classification Prediction

2.2 Threshold selection

In order to do classification predictions, the dataset was split between a popularity of ≥ 70 and < 70 and therefore converted to binary values: 1 being popular and 0 being unpopular. The reason for this was that it was observed that a popularity value ≥ 70 roughly aligned with what could be considered 'viral' songs. This would mean the predictions output by these models would be more optimal for tasks such as finding the right single to promote, as the cost of failure can be high due to associated marketing costs.

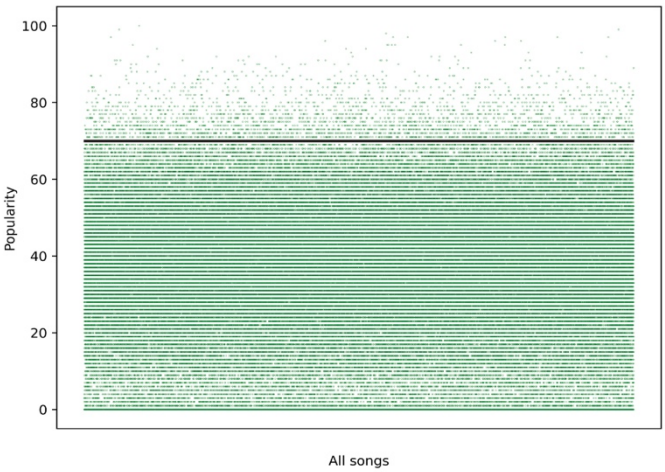


Figure 2.1: Scatter plot of popularity values with threshold in black

2.3 Balancing the dataset

Following this threshold selection, a new dataset was created to train our algorithm on balanced data with the same number of popular songs and unpopular songs. This under-sampling isn't likely to result in a loss of important information as the balanced dataset still has a total of 7'670 rows.

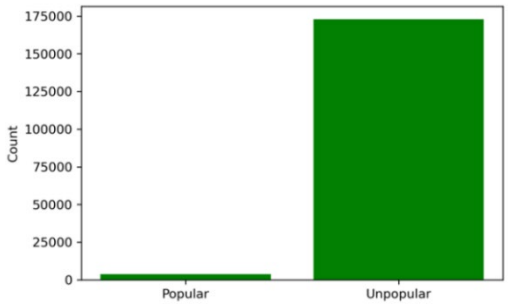


Figure 2.2: Unbalanced dataset

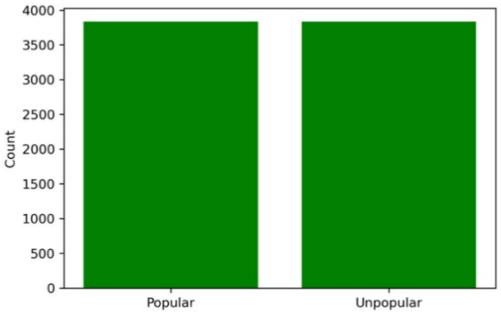


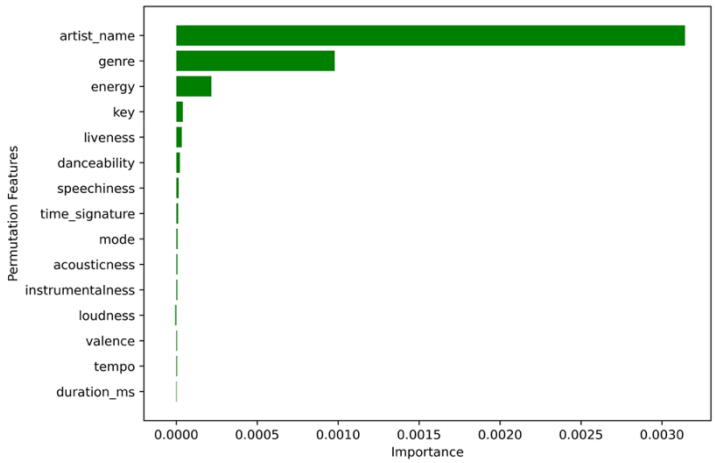
Figure 2.3: Balanced dataset

2.4 Classification prediction with artist name

A logistic regression model was used to predict popularity with all the features of the binary unbalanced dataset in order to evaluate the importance of the features across the whole dataset.

Table 2.1: Logistic Regression Results with *artist_name*

		In Sample	Out of Sample
Logistic Regression	Accuracy	98.02%	97.80%
	Precision	100%	100%
	Recall	98%	98%



With all the values, it was found that the *artist_name* variable is more than three times more important than any other attribute. However, as we are trying to predict the popularity of a song according to its audio features to help artists and labels succeed in the music industry, the *artist_name* variable isn't really relevant. For example, Drake would not need this algorithm to assume his next song is going to be popular. Therefore, following this model prediction, *artist_name* was dropped from the x features.

Figure 2.4: Feature Importance of Logistic Regression Model with *artist_name*

2.5 Classification prediction without artist name

2.5.1 Logistic Regression

Forward Selection

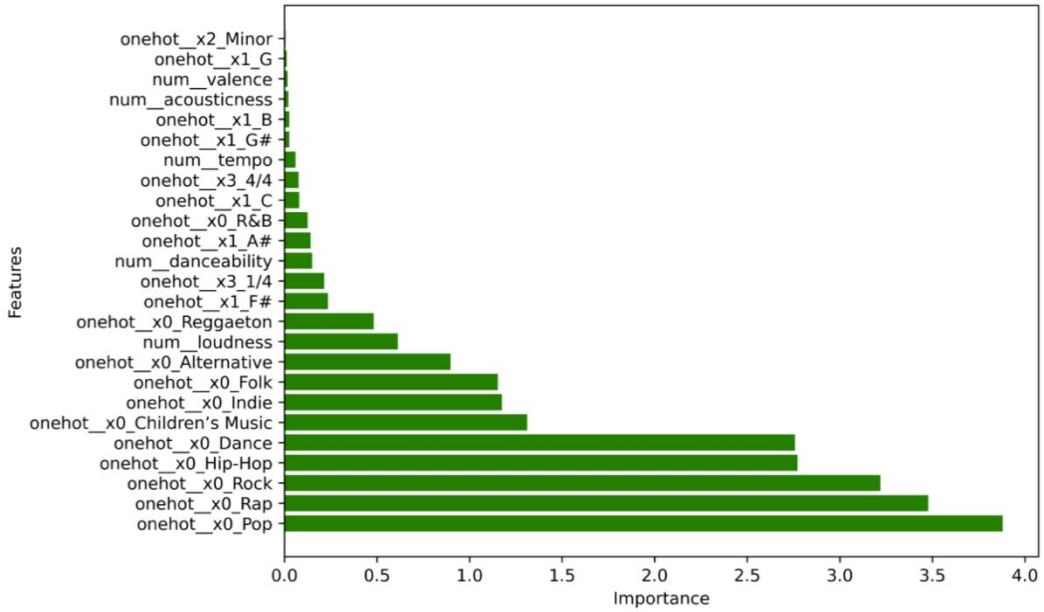
Forward and backward selections were tried out manually but, in this case, there is no need to feature select because there are so many rows and not many columns. There is low risk for overfitting, as indicated by our best models so there is no real need to feature select.

Results

Table 2.2: Logistic Regression Results without *artist_name*

		In Sample	Out of Sample
Logistic Regression	Accuracy	85.79%	87.09%
	Precision	87%	86%
	Recall	86%	88%

Feature Importance (using the encoded feature names using the model coefficient)



The one-hot encoded features give deeper insight into what the model values when predicting popularity. The most popular genres such as Pop and Rap show the highest importance, however other factors such as loudness take priority over more niche genres.

Figure 2.5: Feature Importance of Logistic Regression Model without *artist_name*

2.5.2 Decision Tree

Results

Table 2.3: Decision Tree Results

		In Sample	Out of Sample
Decision Tree	Accuracy	88.45%	83.96%
	Precision	88%	88%
	Recall	89%	81%

The decision tree is also very accurate both in and out of sample. Once again, the decision tree's features' importance shows that genre is the main feature to predict if a song is going to be viral. Loudness is once again an important feature. Encoded feature names and coefficients could not be used in this case.

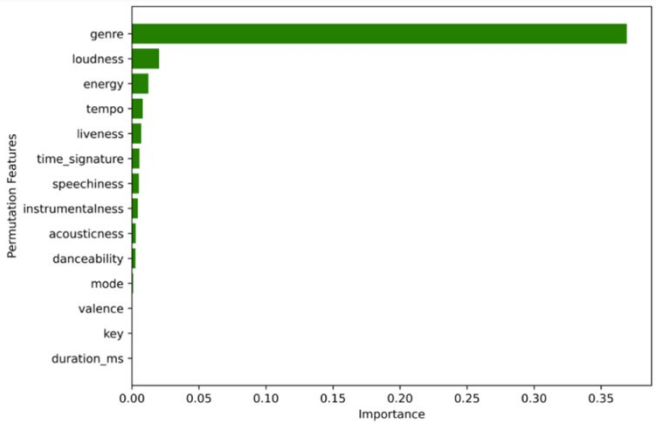
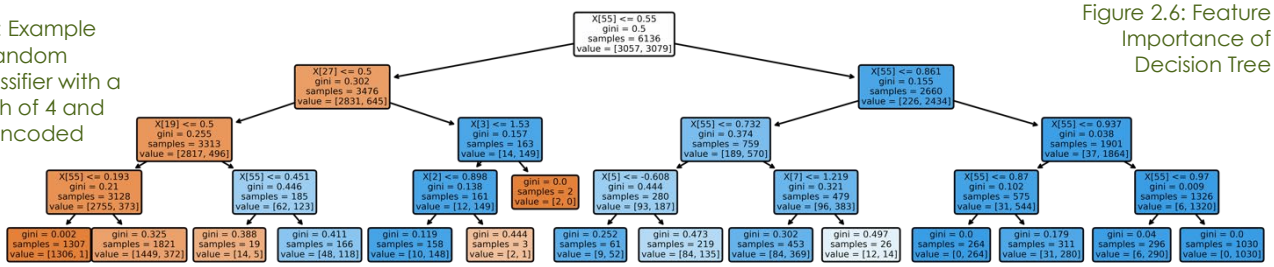


Figure 2.6: Feature Importance of Decision Tree

Figure 2.7: Example tree for Random Forest Classifier with a max depth of 4 and one-hot encoded features



2.5.3 Random Forest Classifier

Table 2.4: Results of Random Forest Classifier

		In Sample	Out of Sample
Random Forest Classifier	Accuracy	100%	85.85%
	Precision	100%	88%
	Recall	100%	84%

The random forest classifier was surprisingly performant in sample. It was tested multiple times and always came up to a perfect value.

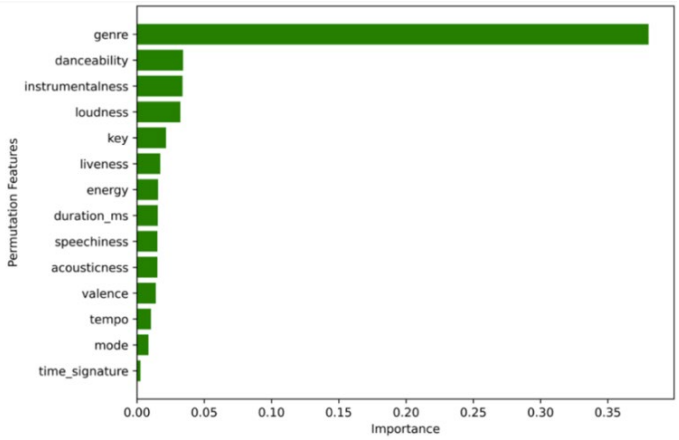


Figure 2.8: Feature Importance of Random Forest Classifier

Feature importance in this case shows a better repartition across all attributes even though genre remains the main most important feature. Encoded feature names and coefficients could not be used.

2.6 Models comparison

Table 2.5: Comparison of performance metrics on test set for models without *artist_name*

	Accuracy	Precision	Recall
Logistic Regression	87.09%	86%	88%
Decision Tree	83.96%	88%	81%
Random Forest Classifier	85.85%	88%	84%

Logistic Regression was found out to be the best model out of the three with the highest accuracy value of 87.09%. The two other models however also had very high accuracies. Precision and recall values are both very close for the three models. LogReg has the highest recall value of 88% which is great as it means the algorithm has a low false negative rate and will therefore not prevent an artist from entering the music market by falsely predicting his/her song as unpopular.

Feature Importance

Table 2.6: Most important features for models without *artist_name*

Most important features	First	Second	Third
Logistic Regression (encoded)	Genre	Instrumentalness	Loudness
Decision Tree	Genre	Loudness	Energy
Random Forest Classifier	Genre	Danceability	Instrumentalness

This comparison is reassuring as for all models the same features have the biggest importance and impact on the prediction.

2.7 Models restriction

This classification prediction method however lacks an output of a specific popularity. Record labels investing money into a track or artist would benefit from a specific popularity output rather than a binary value. 2 songs could both be classified below 70 but clearly a popularity of 69 would deserve more investment than a song of popularity 0. Therefore, numerical prediction was conducted with two different models: Linear Regression and Random Forest Regressor.

3. Numerical Prediction

3.1 Popularity Values

In this section, popularity values between 0 and 100 were predicted by two different models. The previous threshold and binary values were removed to have a more complex prediction. Accuracy will be the only performance metric as the data is now numerical and that simple confusion matrices are not applicable anymore to compute the precision and recall values.

3.2 Linear Regression

Firstly, a model predicting the popularity number without *artist_name* including all the features was made. Then, the *artist_name* was reintroduced, improving the prediction as shown in Figures 3.1 and 3.2.

Table 3.1: Results of Linear Regression without *artist_name*

		In Sample	Out of Sample
Linear Regression	Accuracy	69.06%	69.02%

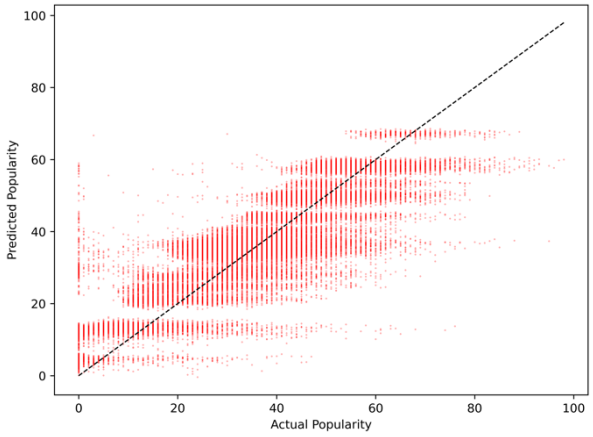


Figure 3.1 Predicted vs Actual Popularity without *artist_name*

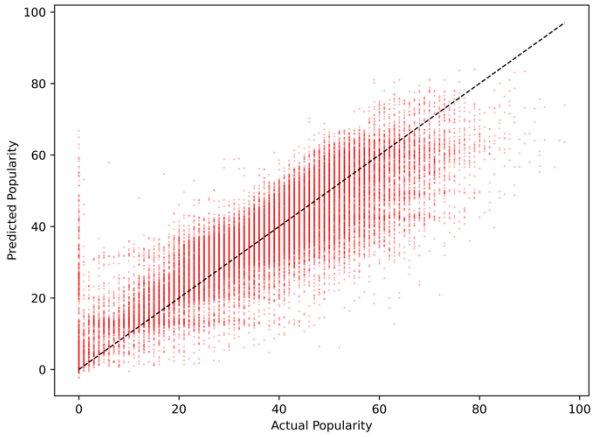


Figure 3.2 Predicted vs Actual Popularity with *artist_name*

3.3 Random Forest Regressor with Artist Name

Finally, a random forest regressor with the *artist_name* variable and therefore including all the features was used. Its accuracy out of sample came up to 75%, which is satisfying knowing that it predicts an integer between 0 and 100 and therefore has 101 possible popularity outputs for every single song.

Table 3.2: Results of Random Forest Regressor with *artist_name*

		In Sample	Out of Sample
Random Forest Regressor	Accuracy	97.09%	74.94%

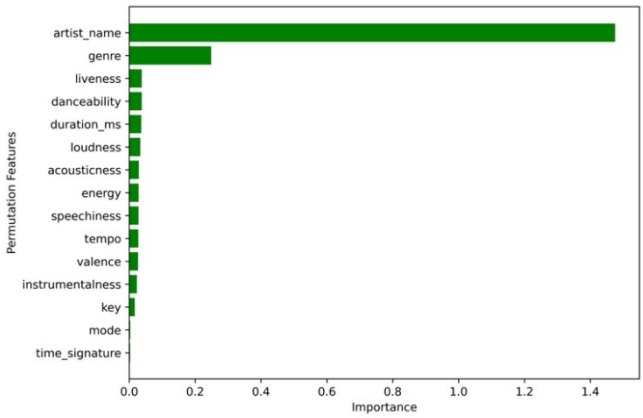


Figure 3.4: Permutation Features of Random Forest Regression with *artist_name*

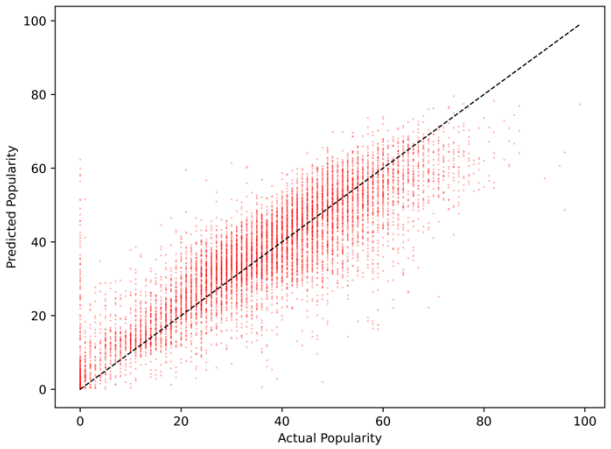


Figure 3.3: Predicted vs Actual Popularity with *artist_name*

It can once again be seen that *artist_name* accounts for almost 5 times the importance of the second most important feature.

4. Conclusion

The popularity of a song is time dependent and can be seen as very subjective. It was however shown that it can be predicted depending on audio features with an emphasis on the genre of the song, its loudness, instrumentalness and danceability. The Logistic Regression was chosen as the best model to predict if the song of an unknown artist is going to be viral and reach a popularity over 70 with an accuracy of 87.09%, a precision of 86% and a recall of 88%. Regarding the numerical prediction taking into account the artist's name, the Random Forest Regressor performed the best at predicting the popularity value between 0 and 100 with an accuracy of 74.94%. These models however rely on the data reported by Spotify's API and are therefore limited as they fully depend on the algorithm calculating the popularity value.

5. References

(1) SPOTIFY, *Get Audio Features for a Track* [online], Web API, Docs, [Accessed at:] https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/

Dataset obtained via:
(2) Hamidani Zaheen, *Spotify Tracks DB* [online], Music Database, Kaggle, [Accessed at:] https://www.kaggle.com/zaheenhamidani/ultimate-spotify-tracks-db

(3) BROWNLEE Jason, *Difference Between Classification and Regression in Machine Learning* [online], December 11, 2017, last updated May 2019, Machine Learning Mastery, [Accessed at:] https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/

6. Appendices

Appendix A – Graphs

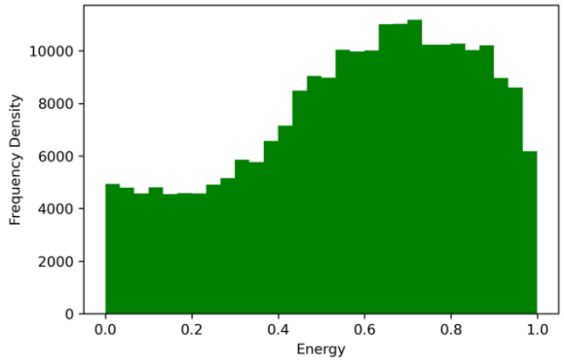
Audio Features Description from Spotify's API (1)

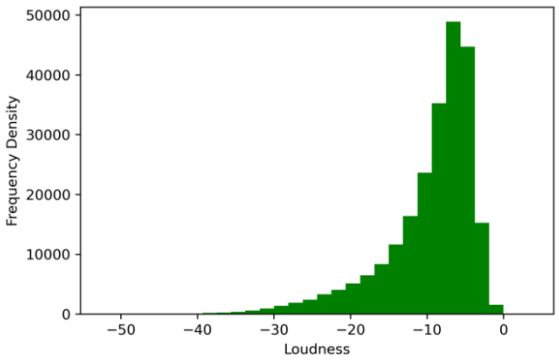
Variable	Value Type	Value Description
artist_name	string	The artist's name.
track_id	string	The Spotify ID for the track.
track_name	string	The name of the song.
popularity	int	Value between 0-100. Popularity is track-based and a measure of how many plays a track received and how recent those plays are. Then artist popularity is derived from that.
genre	string	The genre of the song.
duration_ms	int	The duration of the track in milliseconds.
key	int	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on. If no key was detected, the value is -1.
mode	int	Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
time_signature	int	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).
acousticness	float	A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic.
danceability	float	Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.
energy	float	Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy.

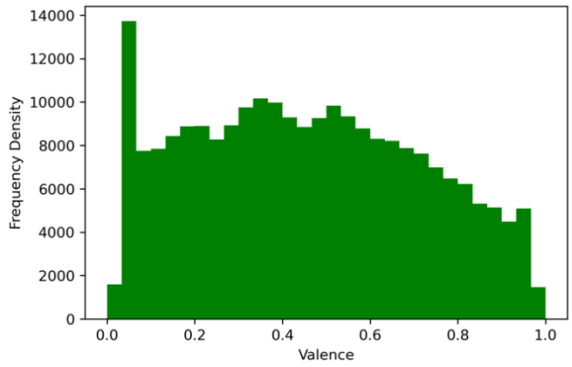
instrumentalness	float	Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.
liveness	float	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live.
loudness	float	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.
speechiness	float	Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.
valence	float	A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
tempo	float	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration.

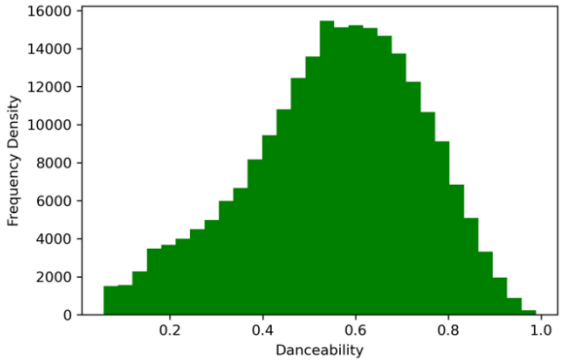
Appendix B – Graphs

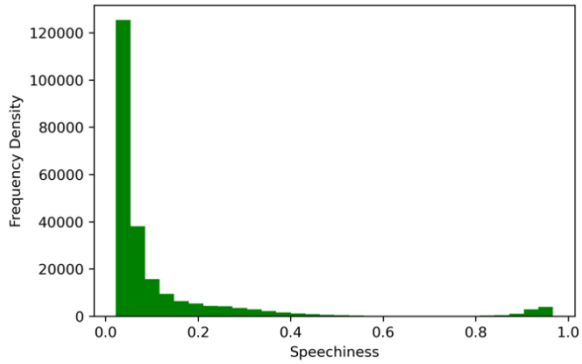
Frequency Density of Features

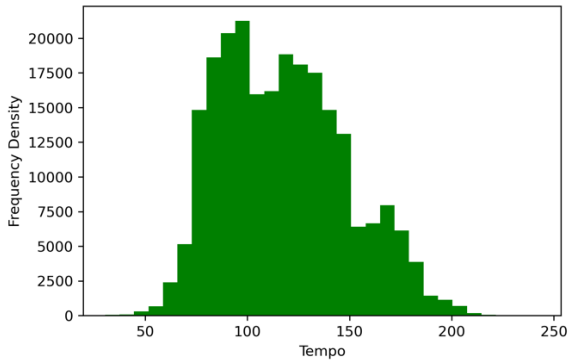


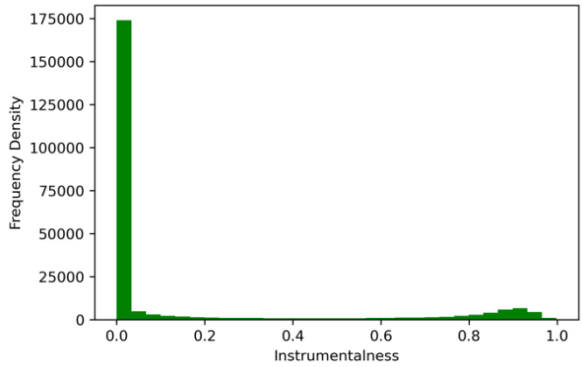


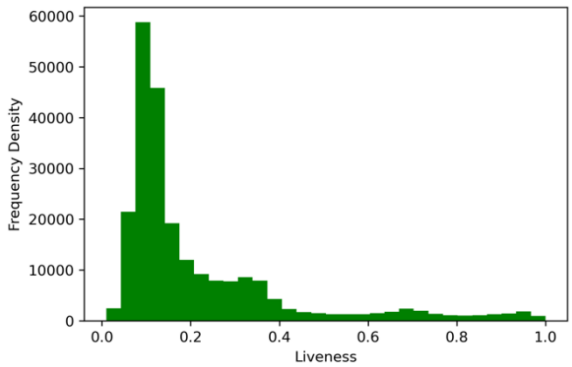




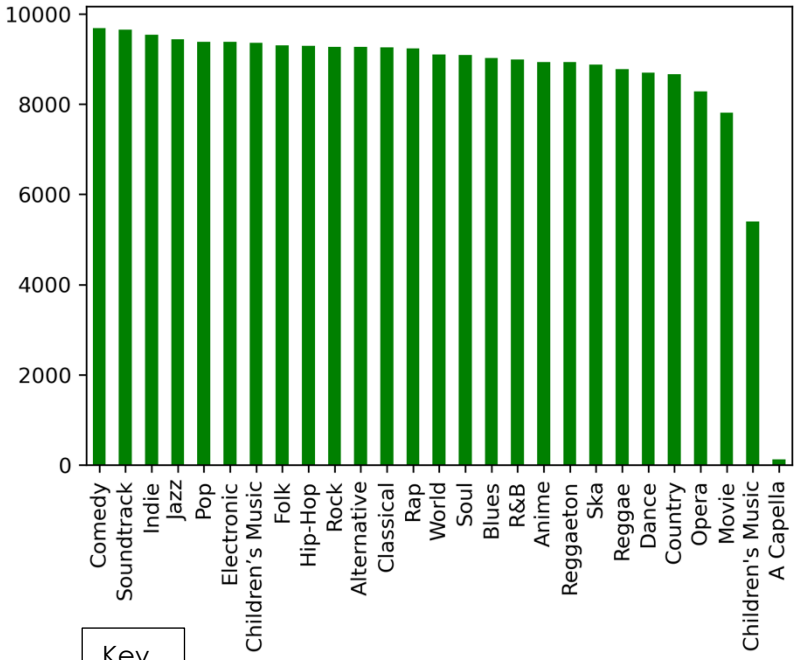




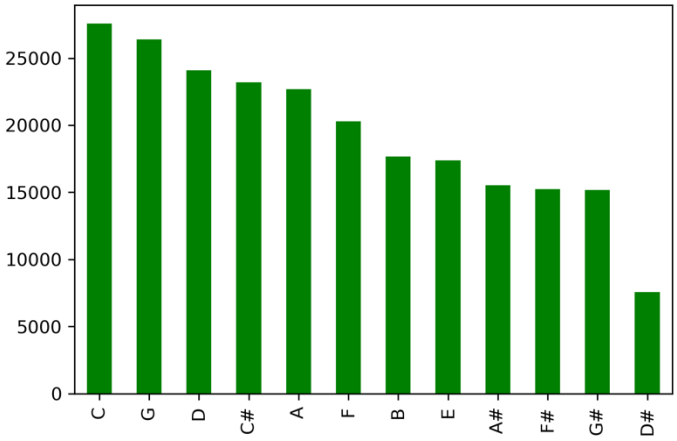




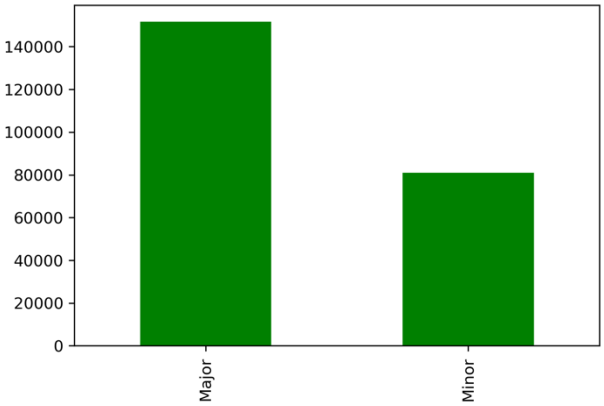
Genre



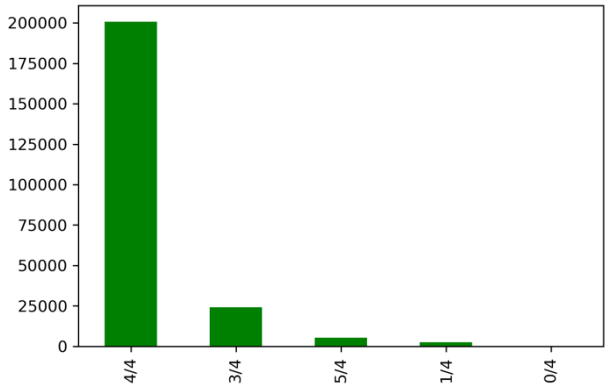
Key



Mode



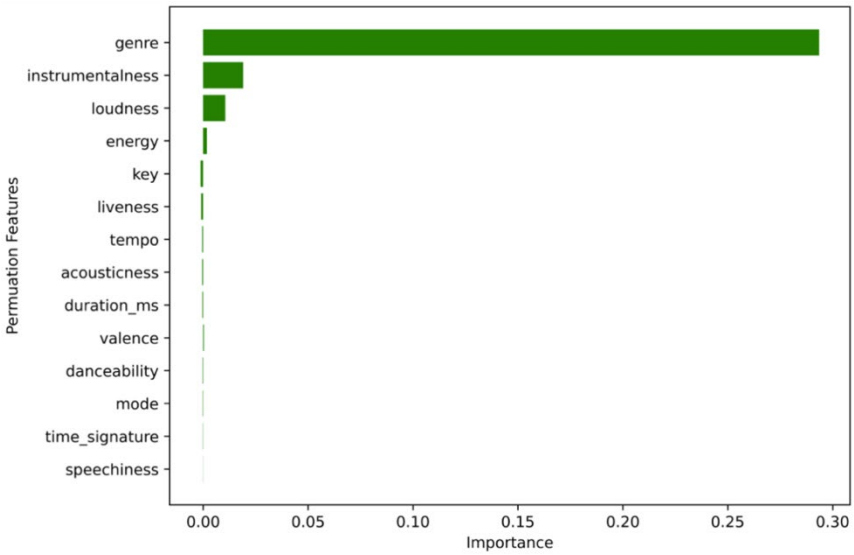
Time Signature



Classification prediction without artist name

Logistic Regression

Permutation Feature importance (having de-encoded the feature names)



Appendix C – Confusion Matrices

Classification Prediction

Logistic Regression with artist_name

```
Test Values:

TP: 34461   FP: 119
FN: 658     TN: 117
Recall: 0.98
Precision: 1.00
```

```
Training values:

TP: 137969   FP: 390
FN: 2412     TN: 648
Recall: 0.98
Precision: 1.00
```

Logistic Regression without artist_name

```
TP: 673   FP: 106
FN: 92    TN: 663
Recall: 0.88
Precision: 0.86
```

```
Training values:

TP: 2677   FP: 386
FN: 429    TN: 2644
Recall: 0.86
Precision: 0.87
```

Decision Tree without artist_name

```
TP: 647   FP: 156
FN: 90    TN: 641
Recall: 0.88
Precision: 0.81
```

```
Training values:

TP: 2691   FP: 372
FN: 321    TN: 2752
Recall: 0.89
Precision: 0.88
```

Random Forest Classifier

```
TP: 681   FP: 91
FN: 126    TN: 636
Recall: 0.84
Precision: 0.88
```

```
Training values:

TP: 3063   FP: 0
FN: 0    TN: 3073
Recall: 1.00
Precision: 1.00
```

Appendix D – Code

```
!pip install category_encoders

import pandas as pd
import numpy as np

# preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

# transformers
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from category_encoders import TargetEncoder

# model Scoring & Validation
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RepeatedKFold

# models to try
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
```

```
# Confusion Matrix
from sklearn.metrics import confusion_matrix

# graphing stuff
import matplotlib.pyplot as plt
```

```
# 1. INITIAL SETUP
data = pd.read_csv('/content/SpotifyFeatures.csv')
data = data.drop_duplicates(subset='track_id')
data = data.drop(columns=['track_id','track_name'])
data = data.dropna()

# If classification
data['popularity'].values[data['popularity'].values < 70] = 0
data['popularity'].values[data['popularity'].values >= 70] = 1

total = len(data)
nb_popular = data['popularity'].sum()
popular1 = (data.popularity == 1).sum()
nb_unpopular = total - nb_popular
unpopular = data.loc[data['popularity'] == 0].sample(nb_popular)
popular = data.loc[data['popularity'] == 1]
resampled_data = pd.concat((unpopular, popular))
resampled_data.head()

x = resampled_data.drop(columns=['popularity'])
y = resampled_data['popularity']

# If Regression
# x = data.drop(columns=['popularity'])
# y = data['popularity']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)
# 2. PREPROCESSING
# Split into Numerical and Categorical
num_f = ['acousticness', 'danceability', 'duration_ms', 'energy',
         'instrumentalness', 'liveness', 'loudness', 'speechiness',
         'tempo', 'valence']
cat_f = ['genre', 'key', 'mode', 'time_signature']
cat2_f = ['artist_name']


num_trans = Pipeline([('nimputer', SimpleImputer()),
                      ('scaler', StandardScaler())])

cat_trans = Pipeline([
                     ('onehot', OneHotEncoder(handle_unknown='ignore'))])

cat2_trans = Pipeline([
                     ('target', TargetEncoder())])

preprocesser = ColumnTransformer([
                                  ('num', num_trans, num_f),
                                  ('cat', cat_trans, cat_f),
                                  ('cat2', cat2_trans, cat2_f)
])
```

```
reg = Pipeline([
                ('preprocessor', preprocesser),

# classification models
                # ('logistic', LogisticRegression())
                # ridiculously good if cateogorising popularity
                # ('tree', tree.DecisionTreeClassifier(max_depth=4, max_leaf_nodes=50))
                # ('classifier', RandomForestClassifier())

# regression models
                # ('regressor', LinearRegression())
                # ('regressor', RandomForestRegressor())

])

param_grid = {}

kfold = RepeatedKFold(n_splits=4, n_repeats=3, random_state=125)

grid = GridSearchCV(reg, param_grid=param_grid, cv = kfold,
                    return_train_score=True, verbose=10)

grid.fit(x_train, y_train)

in_sample = grid.score(x_train, y_train)
out_sample = grid.score(x_test, y_test)
print(f'In Sample Accurcy: {in_sample*100:.2f}%')
print(f'Out of Sample Accurcy: {out_sample*100:.2f}%')
```

```
# Generating Report
y_true = y_test
y_pred = grid.predict(x_test)
print(classification_report(y_true, y_pred))
```

```
from matplotlib.pyplot import figure
figure(num=None, figsize=(8, 6), dpi=800, facecolor='w', edgecolor='k')
y_pred = grid.predict(x_test) # getting the predicted values on test set
plt.scatter(y_test, y_pred, s=0.02, c='r', marker='o')
plt.xlabel('Actual Popularity')
plt.ylabel('Predicted Popularity')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',
         lw=1)
plt.show()
```