Machine Learning

Project Green Jade

ML Model to Predict Song Preferences

Introduction

After using Spotify for about an year, I wanted to know what my favourite songs had in contrast to the ones I disliked. I wondered what my favourite songs would reveal about me and if there was any hidden pattern in my preferences.

Spotipy Is a python library which allows one to access all the playlists, and the best part is you can get the features of all audio tracks. Features like Energy, Danceability, Valence etc can be used as attributes which can be used to understand your listening preferences.

Goal:

- Find discernible differences between my liked and disliked songs. Visualize it
- Build a classification model to predict which songs I would potentially like, given a playlist.

Understand Audio Features:

Speechiness: This indicates how wordy a given track is. A value higher than 0.7 could indicate a podcast.

Valence: This indicates the mood of a song. A higher value indicates happy, perky, cheerful tracks, low values indicate sad, depressed, angry feelings.

Liveness: This detects the presence of a live audience in a track. A high number could mean a live performance.

Instrumentalness: Indicates how vocal a particular song is, long “aah” and “oooh” are instrumental.

Energy: High energy songs sound fast, loud and noisy.

Danceability: This is a combination of tempo, beats, rhythm etc. A higher value here indicates an upbeat song, and you could dance to it. (if you know to dance)

Acousticness: A higher acoustic score could mean that the track is mostly just music like, study tracks.

Project Approach

1. Create playlists of songs in Spotify, you Like and Don’t like (I just named mine as *GOOD* and *BAD*).
2. Sign up for the Spotify developer dashboard to get credentials which allow you to access songs.
3. Use *Spotipy* and fetch Your Liked and Disliked songs.

4. Explore the dataset and see what the differences are between your liked and disliked songs.
5. Build models using appropriate techniques.
6. Test Model on unseen data.
7. See how the model is performing on new songs from the *Daily-mix* playlists and *Discover* playlist.

Surprisingly, the First step is the most cumbersome, because I had to manually like the songs and segregate them into *GOOD* and *BAD.* I was able to get up to around 200 songs in each playlist.

Using Spotipy get the two playlists and collate them into one dataframe and shuffle the rows before writing it as a csv.

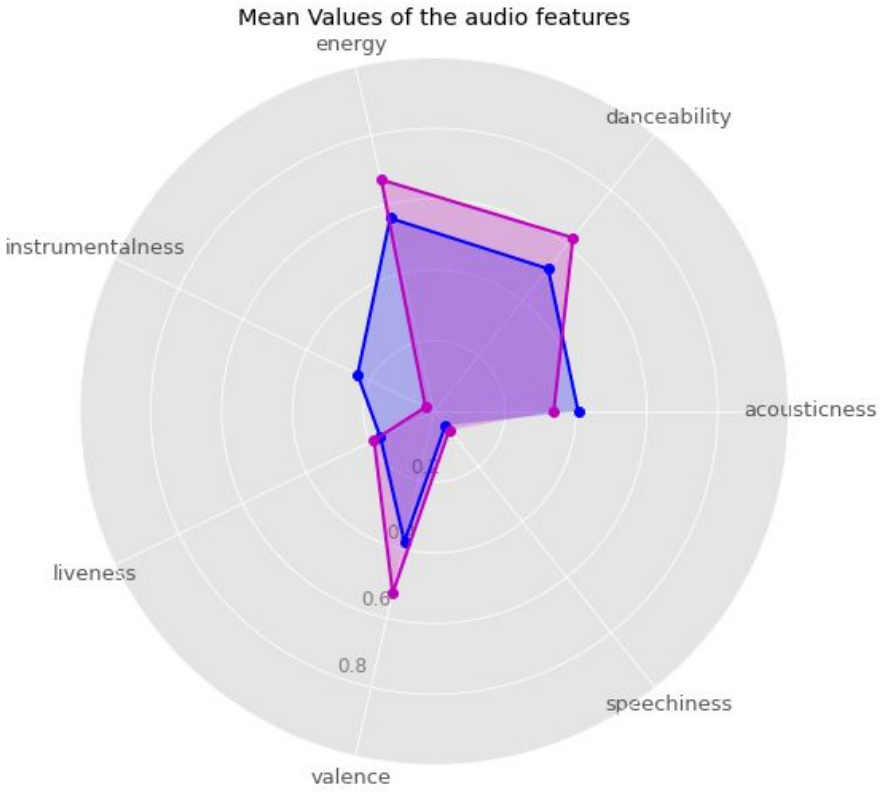
EDA

Let's Visualize the differences between liked and disliked songs.

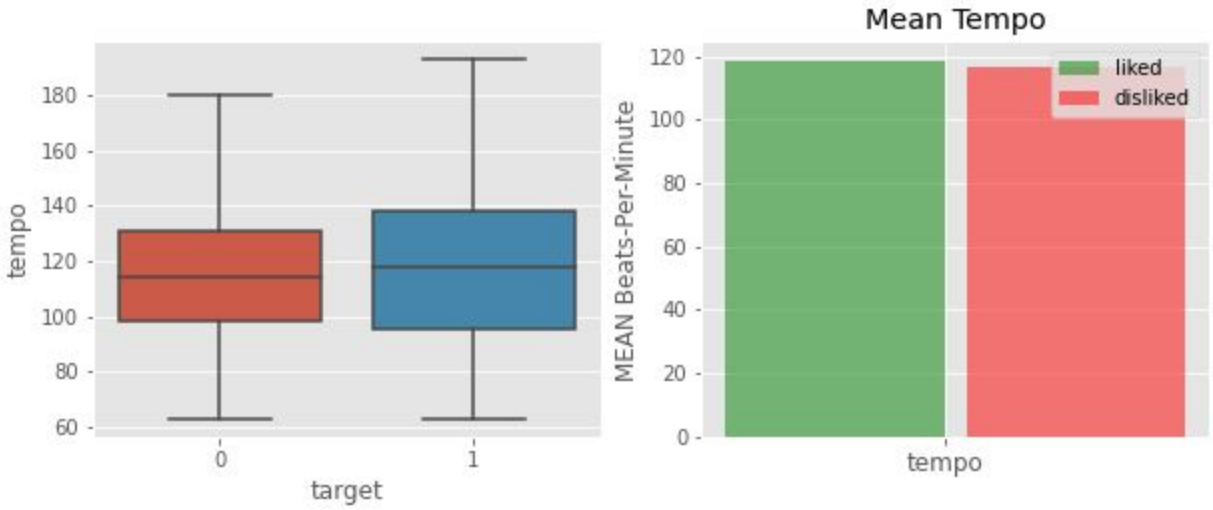
The graph below shows the *mean* of the audio features of liked and disliked tracks.

Looking at it, one can infer that:

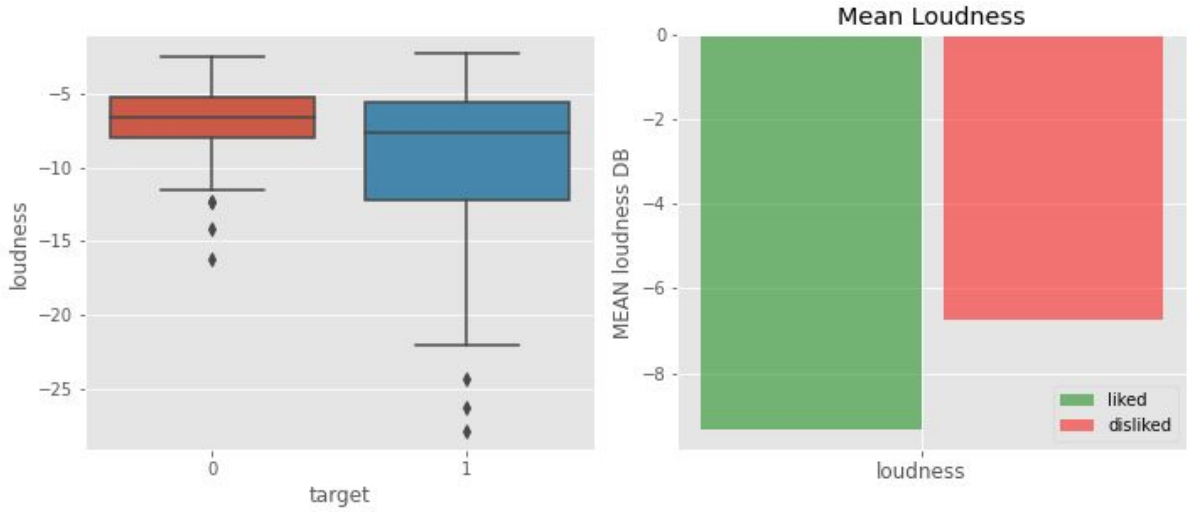
- The songs I dislike tend to be slightly more speechy and are significantly more cheery and happy (valence).
- I tend to like *instrumental* songs, songs which probably contain a lot of elongated "aah" and "ooh" etc. This is a distinct characteristic of Bollywood songs.
- On average I seem to dislike very high *energy* songs but I prefer it in the sweet spot.
- *Danceability* is a mixture of various characteristics such as tempo, beat strength, etc and I seem to dislike highly danceable tracks. This is primarily because I listen to songs mostly when I am working and danceable songs are very distracting.


Mean Values of the audio features
energy
danceability
instrumentalness
acousticness
liveness
speechiness
valence
0.6
0.8

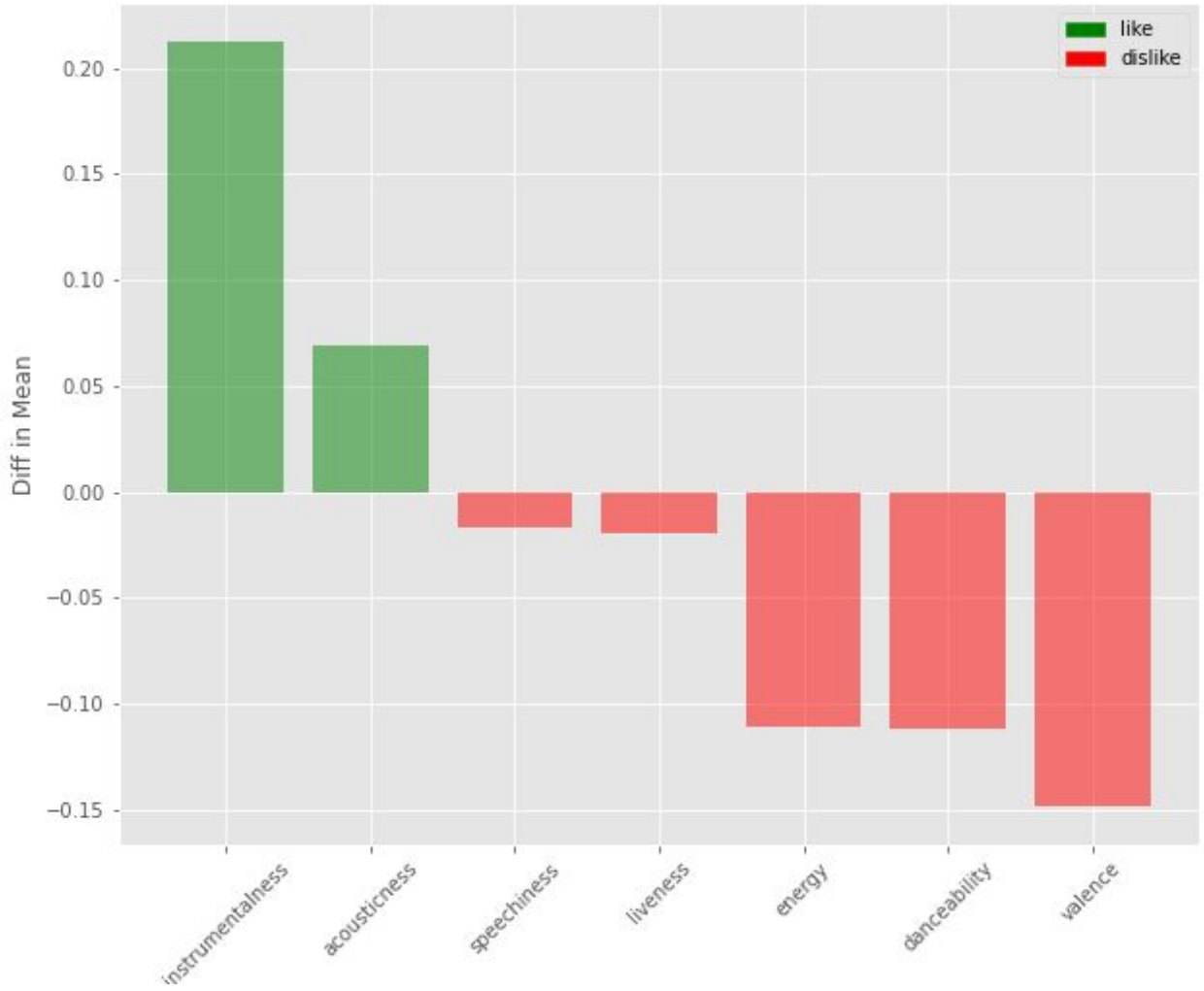
Tempo is also an important factor to consider. A slow tempo might be a melodious, relaxing song (if valence is high) and depressing(if valence is low). The graph below shows that there is almost no difference in the tempo of liked and disliked songs.



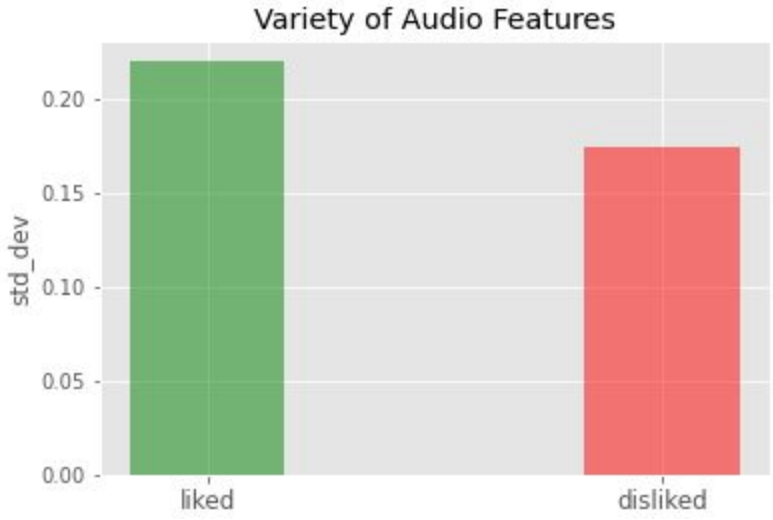
This graph below showing *Loudness,* shows that the majority of songs I like are much less loud compared to the disliked songs. These songs could have a lot of Bollywood romantic songs, which tend to be calmer.

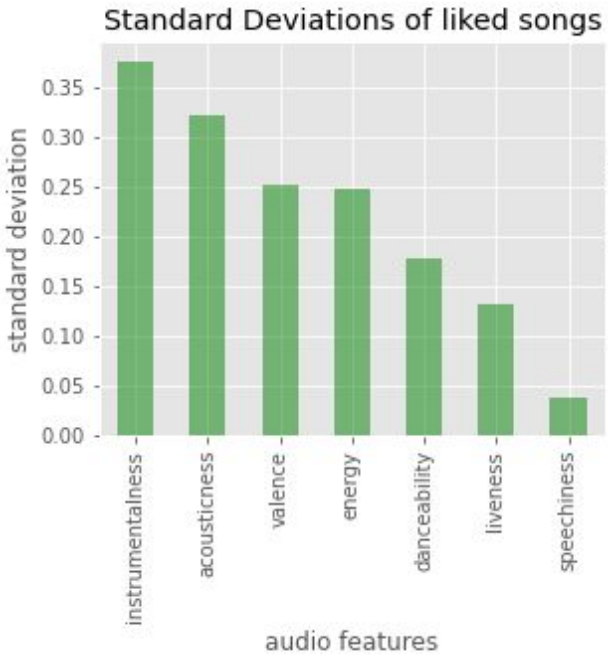


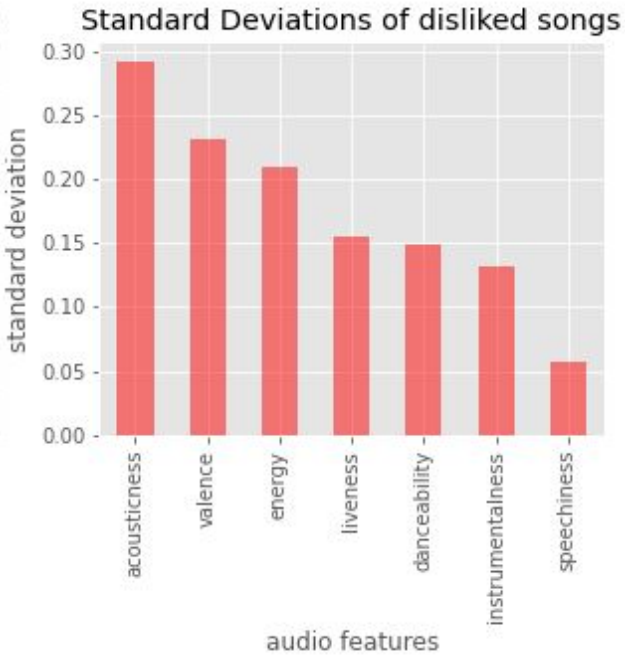
The below graph shows the difference between the mean of audio features. High valence, energy, danceability are what probably cause them to be disliked. Whereas high Instrumental and acoustic songs are liked.



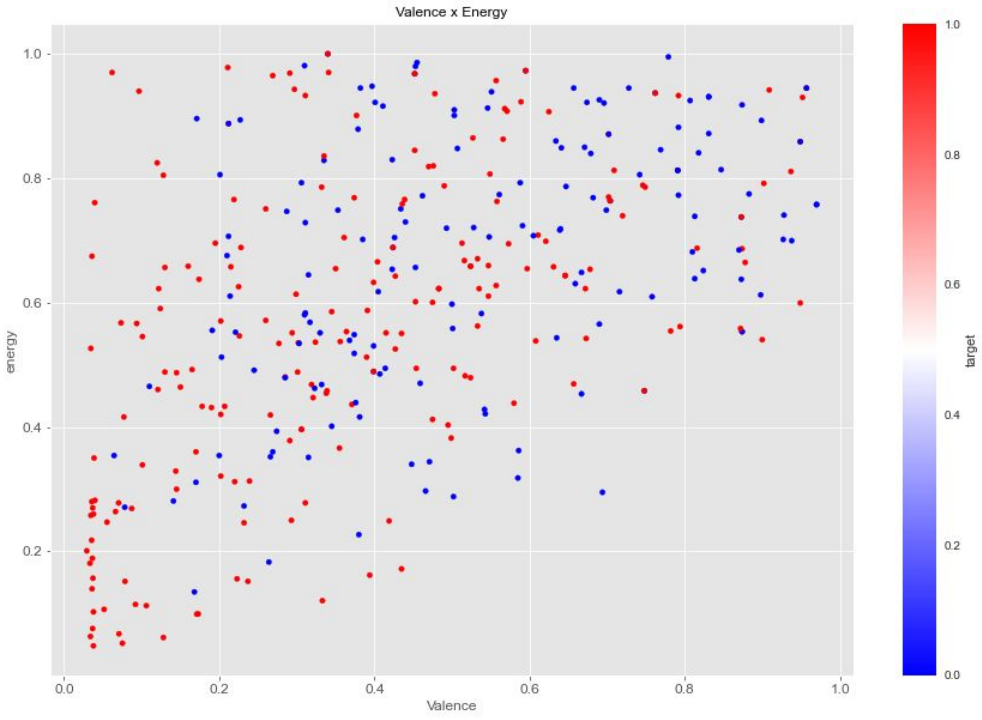
Exploring the overall variety in songs, we can use standard deviation to determine that. The below graph shows that liked songs are more varied in nature.



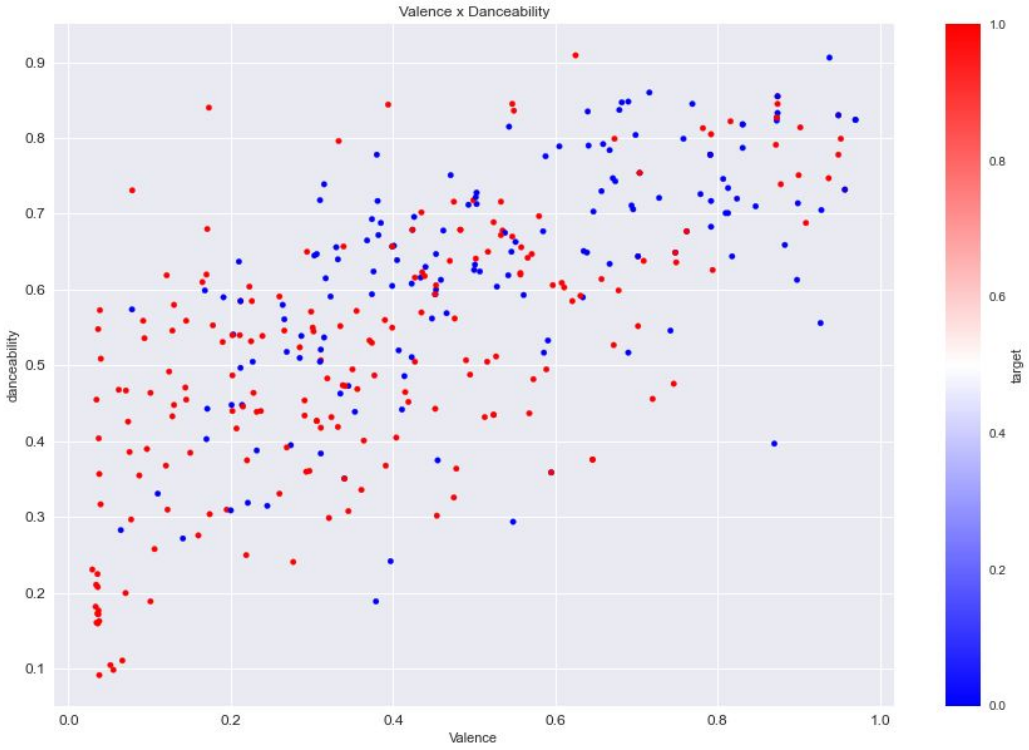




Analysing the scatter plot of Valence and Energy, there is a small cluster of songs which are low energy and low valence (sad, depressing songs) and low valence High Energy (screaming, angry songs). What's also interesting is there are no songs which are high valence, low energy (Over the top happy bubbly songs)



The scatterplot below between Valence and danceability, there is again a cluster of songs at low valence around low to medium danceability.



Model Building

I wanted to build a model starting incrementally from basic algorithms such as logistic regression and working towards more sophisticated algos such as Random Forest Classifier to even more advanced boosting algorithms like lightgbm, xgboost or catboost.

In the process of finding the best model, I wanted to visualize things such as learning curve, time to fit and model score.

I will try not to include too many code snippets.

I have seen many projects where train and test dataset are combined together before preprocessing such as Categorical encoding, Feature scaling are done, I think that although doing it that way is OK, it is not ideal because not every categorical encoder which was fit on the train data is able to handle unseen classes in the test set. So I had to carefully consider the kind of encoder I used. Then I found CatBoostEncoder as part of the Categorical_encoder package.

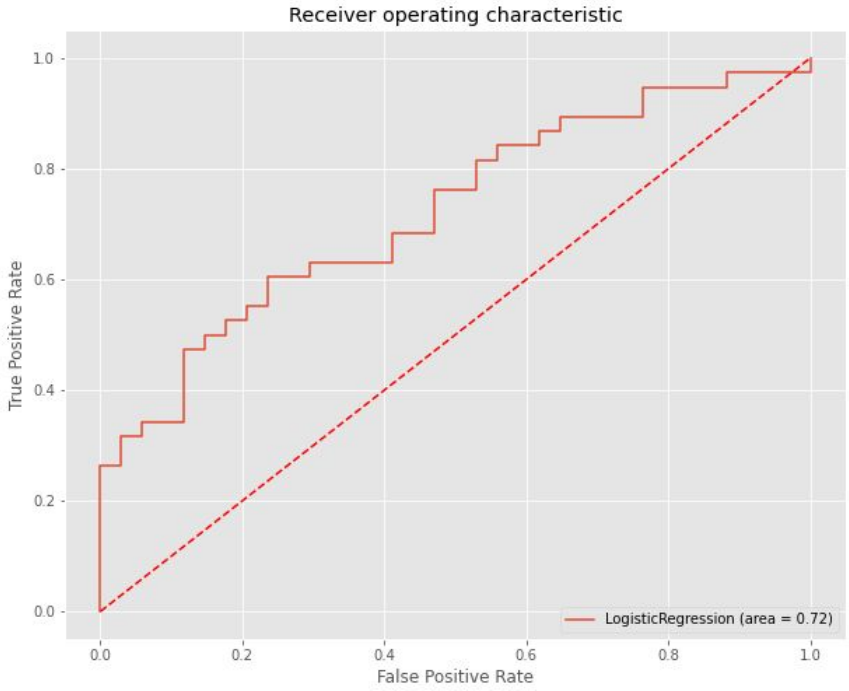
Logistic Regression:

The first LogisticRegression model without any parameter tuning, here are the performance numbers.

With no hyperparameter tuning the recall and precision are close together and AUROC is also ok. We can probably do better.

```
AUCROC Train: 0.74
Recall: 0.84
Precision: 0.77
```

```
AUCROC Test: 0.63
Recall: 0.87
Precision: 0.61
```



Logistic Regression Hyperparameter Tuned:

Looking at the documentation, here is the param grid that seems optimum.

```
param_grid = {
'classifier': [LogisticRegression()],
'classifier__penalty' : ['l1', 'l2'],
'classifier__C': np.logspace(-4, 4, 20),
'classifier__solver': ['liblinear']
}
```

The best parameters after GridSearchCV:

```
{'classifier': LogisticRegression(C=78.47599703514607,
                   penalty='l1', solver='liblinear'),
 'classifier__C': 78.47599703514607,
 'classifier__penalty': 'l1',
 'classifier__solver': 'liblinear'}
```

Here are the perf numbers on Train and Test dataset:

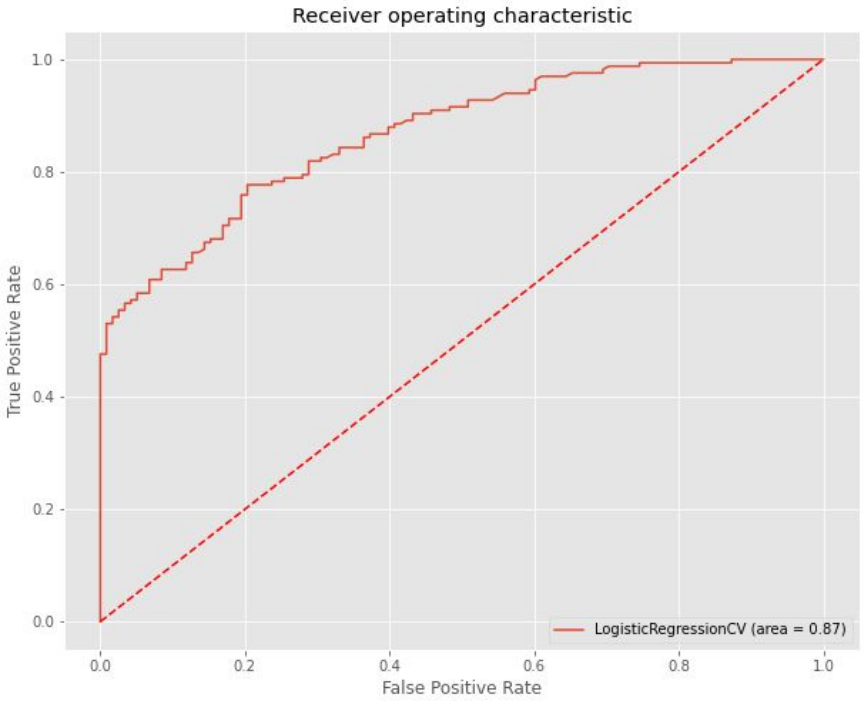
AUCROC Train: 0.76 **AUCROC Test: 0.601**

Recall Train: 0.82 **Recall Test: 0.789**

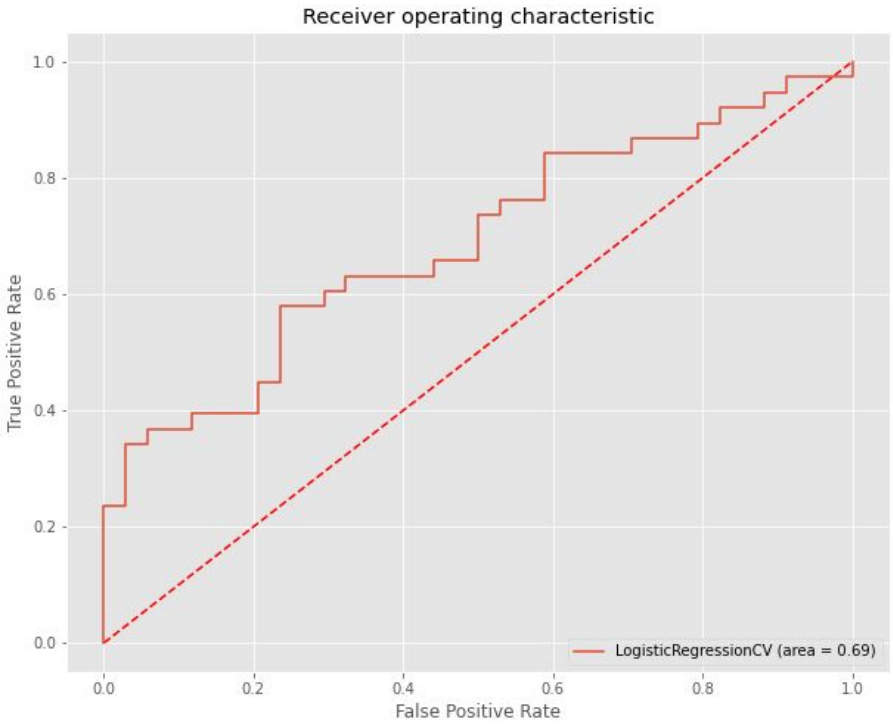
Precision Train: 0.8 **Precision Test: 0.6**

Train accuracy: 0.77 **Test Accuracy: 0.61**

Train AUCROC:



Test AUCROC:



	Precision	recall	f1-score	support
0	0.41	0.64	0.50	22
1	0.79	0.60	0.68	50
accuracy			0.61	72
Macro avg	0.60	0.62	0.59	72
Weighted avg	0.67	0.61	0.63	72

RandomForest Hyperparameter Tuned:

This param grid is what I felt would cover the global minima well enough.

```
param_grid = {
'n_estimators': np.arange(100, 300, 50),
'criterion': ['gini', 'entropy'],
'min_samples_split': np.arange(10, 100, 20),
'min_samples_leaf': np.arange(5, 20, 5),
'min_impurity_decrease': np.linspace(0, 2, 10).round(2)
}
```

The best params training,

```
{'criterion': 'gini',
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 5,
 'min_samples_split': 30,
 'n_estimators': 100}
```

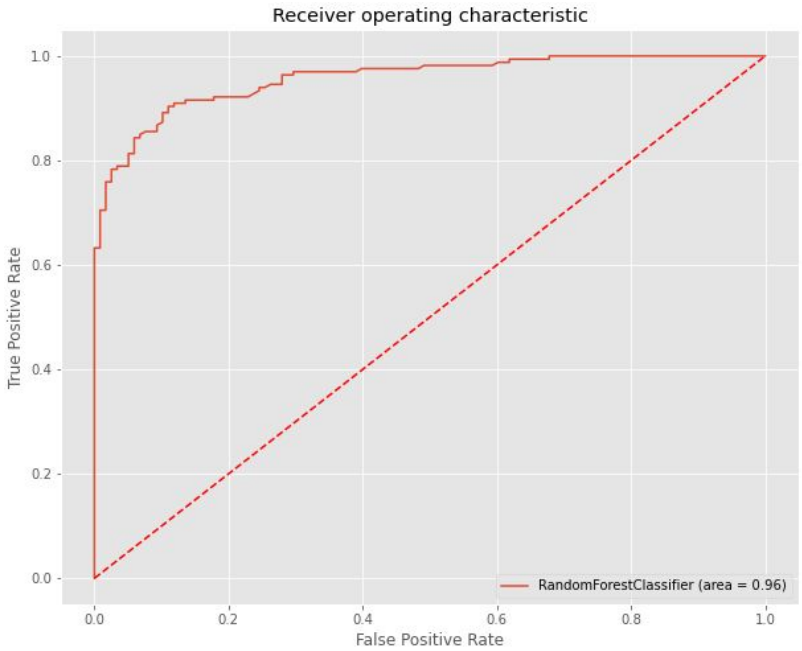
I ran the GridSearchCV with "`return_train_score = True`".

```
mean_test_accuracy : 0.75
mean_train_accuracy : 0.87
mean_test_auc : 0.75
mean_train_auc : 0.88
mean_test_precision : 0.81
mean_train_precision : 0.91
mean_test_recall : 1.0
mean_train_recall : 1.0
```

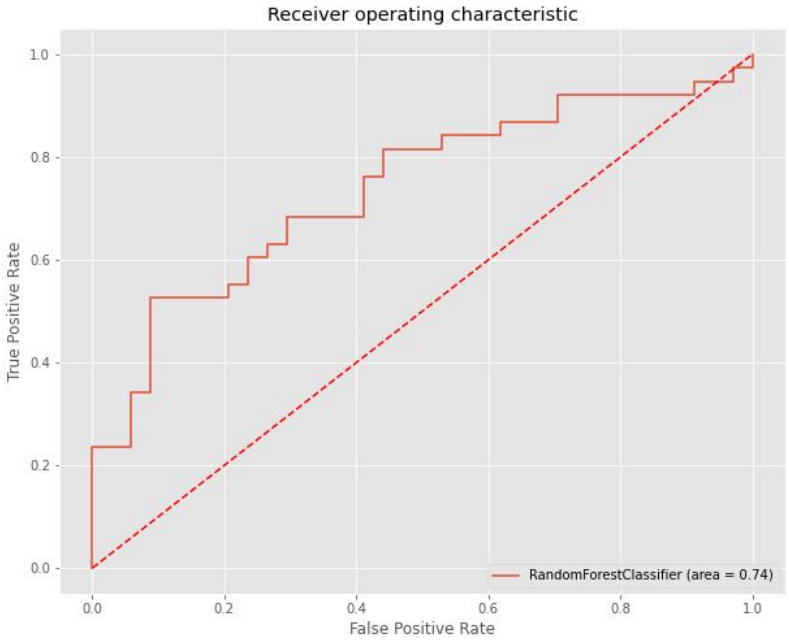
Both Train and Test accuracy have improved from the previous model.

	Precision	recall	f1-score	support
0	0.59	0.69	0.63	29
1	0.76	0.67	0.72	43
accuracy			0.68	72
Macro avg	0.68	0.68	0.68	72
Weighted avg	0.69	0.68	0.68	72

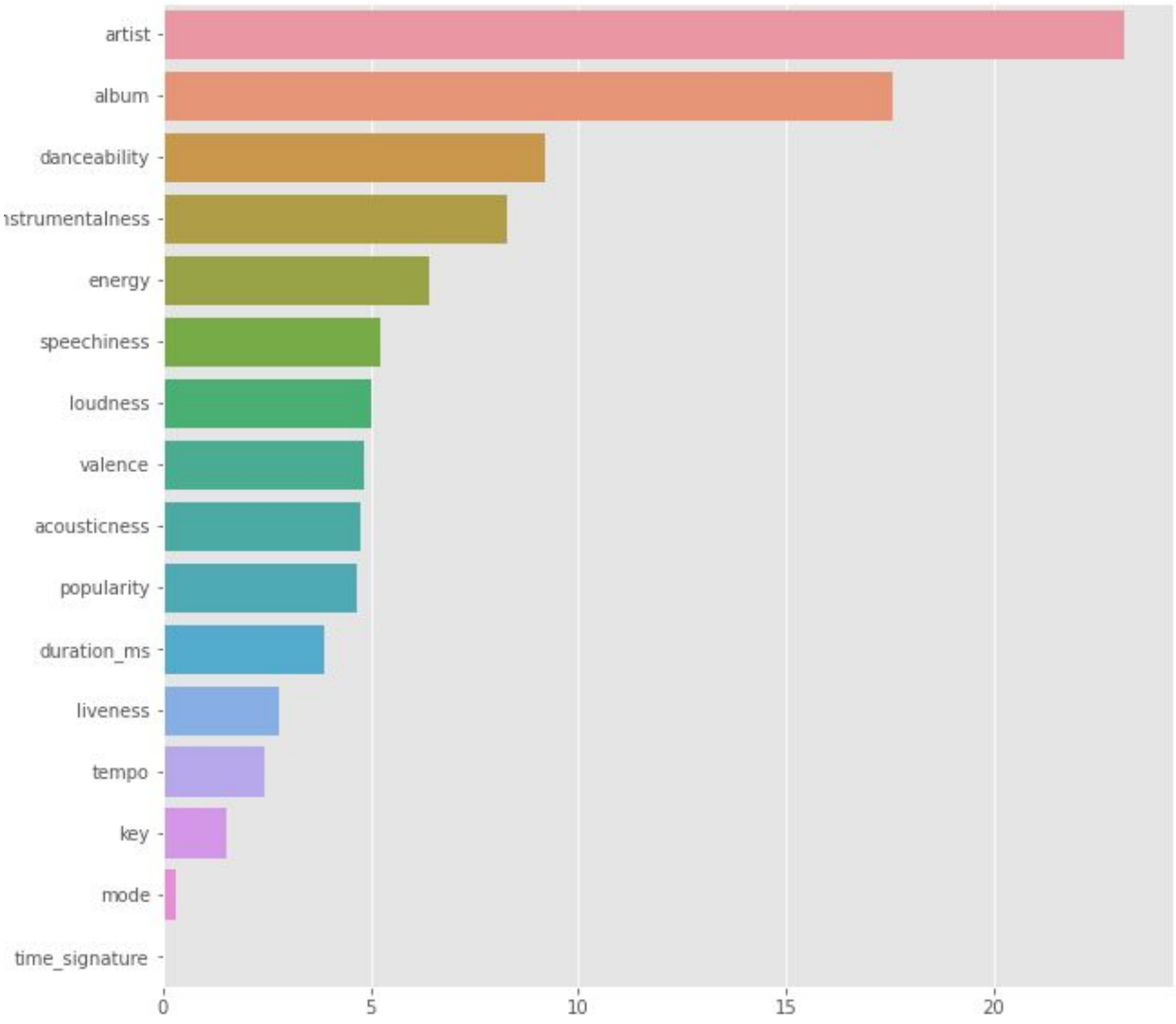
Train AUCROC:



Test AUCROC:


Receiver operating characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
RandomForestClassifier (area = 0.74)

Feature Importances:

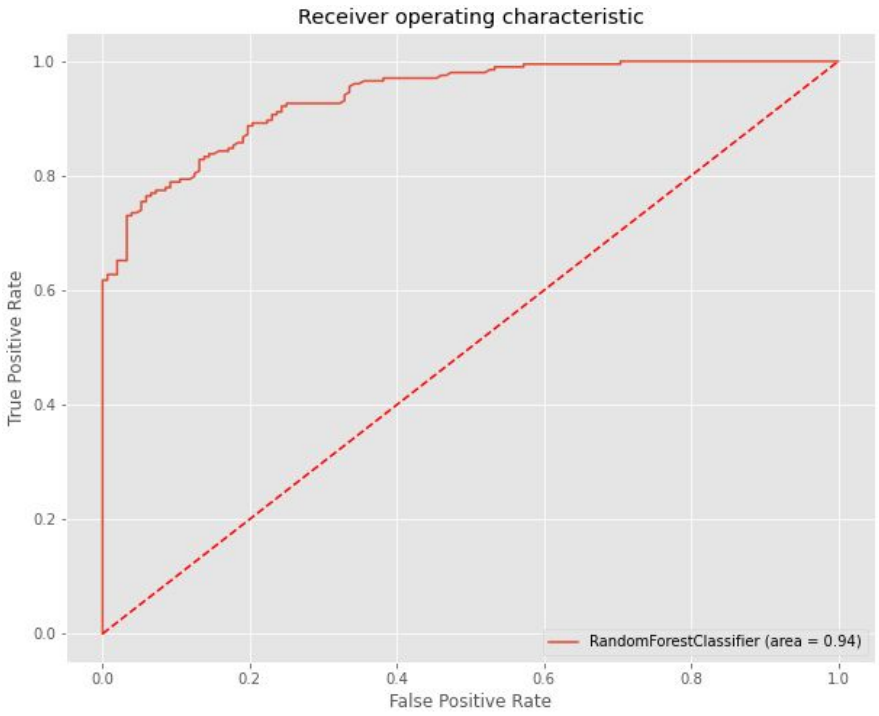


Recursive Feature Elimination

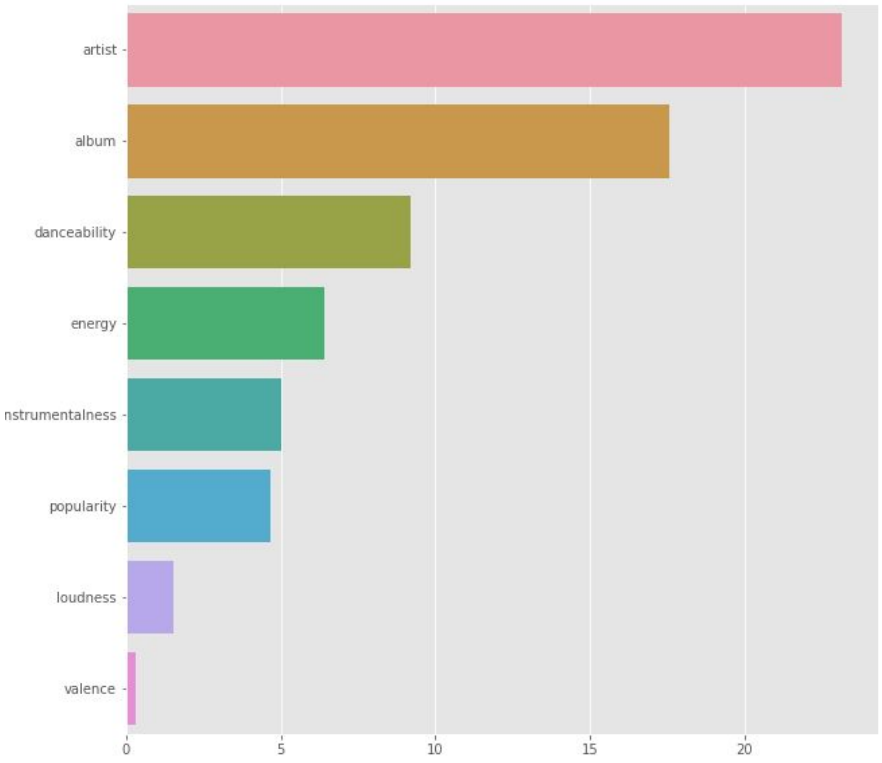
After getting the best estimator by running the GridSearchCV, I tried to reduce the number of features using the sklearn's RFE.

```
Model Accuracy:  0.85
```

AUCROC after removing half of the features.



Most Important Features



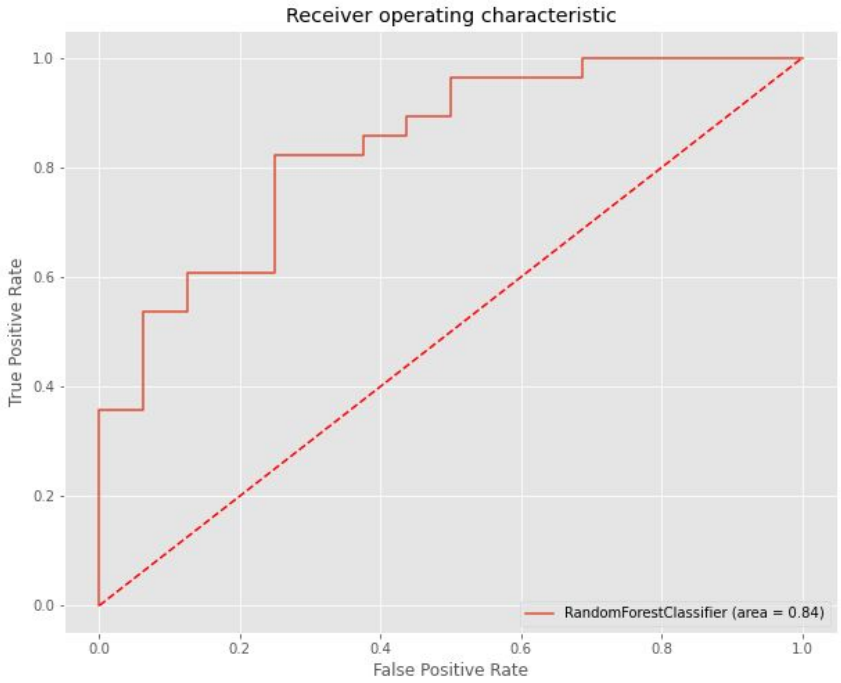
Test the model on new songs

I created another Test playlist with new songs to test how the model is performing. This is also a very laborious work because you'd have to manually label the songs as you like or dislike. Spotipy doesn't provide a way to get if you have liked the songs.

Here are the perf, numbers

	Precision	recall	f1-score	support
0	0.56	0.75	0.64	12
1	0.89	0.78	0.83	32

accuracy			**0.77**	**44**
Macro avg	**0.73**	**0.77**	**0.74**	**44**
Weighted avg	**0.80**	**0.77**	**0.78**	**44**



The Test dataset was of only 44 songs and you take songs from the weekly discover playlist or the daily-mix playlist, and the recall is satisfactory.

Conclusion:

I set out to create a model which could predict whether or not I would like a song, and was able to achieve satisfactory accuracy using RandomForestClassifier.

I tried to put in as many charts and visualizations as would be appropriate to make the report engaging to read.

The biggest challenge of this project was the data collection part, due to the large amount of manual work involved due to the need to manually label the songs. One thing that could make this a lot easier and meaningful would be to improve the Spotipy package to add an API which would grant access to the liked status of a song by the user.

Sources:

- Sklearn for creating an awesome ML library: https://scikit-learn.org/stable/
- Checkout the Spotify EDA here, It explains the audio features and the EDA is also beautifully done: https://rpubs.com/PeterDola/SpotifyTracks
- Music Taste Analysis Article on Medium, excellent EDA : https://towardsdatascience.com/a-music-taste-analysis-using-spotify-api-and-python-e52d186db5fc